

AutoCAD

VBA

Karsten Dummer, Michael Huth

AutoCAD VBA

Einführung, Anwendung, Referenz



ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <<http://dnb.ddb.de>> abrufbar.

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen. Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen oder sollten als solche betrachtet werden.

Umwelthinweis:

Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.

10 9 8 7 6 5 4 3 2 1

06 05 04

ISBN 3-8273-2105-0

© 2004 Addison Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH
Martin-Kollar-Straße 10-12, D-81829 München/Germany
Alle Rechte vorbehalten

Lektorat: Irmgard Wagner, Planegg, Irmwagner@t-online.de
Korrektorat: Petra Kienle, Fürstenfeldbruck
Herstellung: Philipp Burkart, pburkart@pearson.de
Satz: reemers publishing services gmbh, Krefeld – gesetzt aus der Sabon
Umschlaggestaltung: Hommer Grafik-Design, Haar bei München
Druck und Verarbeitung: Bercker Graphischer Betrieb, Kvelaer

Printed in Germany

Inhaltsverzeichnis

Vorwort	11
1 Erste Schritte mit AutoCAD-VBA	13
1.1 Steuerung der VBA-Umgebung	13
1.1.1 AutoCAD-Befehle zur Steuerung der VBA-Umgebung	13
1.1.2 Der VBA-Manager	14
1.1.3 Das Dialogfenster Makros	16
1.1.4 Makros in die AutoCAD-Menüstruktur einbinden	17
1.1.5 Automatisches Ausführen eines VBA-Makros	18
1.2 Die Benutzeroberfläche der VBA-Umgebung	18
1.2.1 Der Projektexplorer	19
1.2.2 Der Objektkatalog	20
1.2.3 Optionsdialog	25
1.2.4 Projekteigenschaften anpassen	26
1.2.5 Das Eigenschaftenfenster	27
1.3 Hilfsmittel für die Entwicklung in AutoCAD-VBA	27
1.3.1 Das Lokal-Fenster	30
1.3.2 Das Direktausgabe-Fenster	30
1.3.3 Das Überwachungsfenster	32
2 Einführung in VBA	33
2.1 Verwenden von Variablen	33
2.1.1 Festlegen der Variablennamen	33
2.1.2 Deklaration von Variablen	34
2.1.3 Vorgabedatentyp festlegen	36
2.1.4 Verfügbare Datentypen	37
2.1.5 Indizierte Variablen verwenden (Arrays)	38
2.1.6 Datenfelder	39
2.1.7 Dynamische Arrays verwenden (Arrays)	39
2.1.8 Feldanzahl von Arrays ermitteln	40
2.1.9 Datenfelder löschen	40
2.1.10 Deklarationen erzwingen	41
2.1.11 Prüfung von Datentypen	41
2.1.12 Datentypumwandlung	42
2.1.13 Einschränkungen bei der Verwendung von Zahlen	44
2.1.14 Benutzerdefinierte Datentypen	44
2.1.15 Verwenden von Konstanten und Konstantenauflistungen	46
2.2 Prozeduren in Modulen, Formularen und Klassenmodulen	47
2.2.1 Prozedurdefinition	47
2.2.2 Sub oder Function?	48
2.2.3 Aufruf und Reichweite von Prozeduren	49
2.2.4 Lebensdauer von Variablen und Konstanten	50
2.2.5 Prozeduren vorzeitig verlassen	53

2.2.6	Parameterübergabe an Prozeduren/Funktionen	53
2.2.7	Optionale Parameter	54
2.2.8	Übergabe von Parametern an Prozeduren	56
2.3	Schleifen, Abfragen und Verzweigungen	56
2.3.1	Verzweigungen mit If	56
2.3.2	Select Case-Verzweigung	57
2.3.3	For Next-Schleifen	58
2.3.4	Do Loop-Schleifen	60
2.4	Objekte	61
2.4.1	Objektverweise erstellen	61
2.4.2	Objekteigenschaften	62
2.4.3	Objektprozeduren	63
2.4.4	Objektauflistungen	64
2.4.5	Übersichtlicher Objektzugriff	66
2.4.6	Ereignisse	68
2.5	Entwicklung eigener Klassenmodule	70
2.5.1	Verwendung des Collection-Objekts	71
2.5.2	Eigenschaften in Klassenmodulen	75
2.5.3	Ereignisse in Klassenmodulen	76
2.5.4	Klassenableitung mit Implements	76
2.6	Operatoren in VBA	77
2.6.1	Operatorenübersicht	78
3	Arbeiten mit Formularen	81
3.1	Allgemeines zum Formularentwurf	81
3.1.1	Werkzeugsammlung	82
3.2	VBA-Standardsteuerelemente	83
3.2.1	Gemeinsame Eigenschaften	84
3.2.2	Gemeinsame Methoden	86
3.2.3	Gemeinsame Ereignisse	87
3.2.4	Bezeichnungsfeld (Label)	88
3.2.5	Textfeld (TextBox)	89
3.2.6	Listenfeld (ListBox)	90
3.2.7	Aufklappbares Listenfeld (ComboBox)	93
3.2.8	Kontrollkästchen (CheckBox)	94
3.2.9	Optionsfelder (OptionButton)	95
3.2.10	Umschaltfelder (ToggleButton)	96
3.2.11	Befehlsschaltflächen (CommandButton)	97
3.2.12	Rahmenfelder (Frames)	98
3.2.13	Multiseiten (MultiPage)	99
3.2.14	Bildlaufleisten (ScrollBar)	102
3.2.15	Drehfeld (SpinButton)	103
3.2.16	Anzeigefelder (Image)	104
3.3	Weitere Steuerelemente	106
3.3.1	Abbildungsliste (ImageList)	106
3.3.2	Listenansicht-Steuerelement (ListView)	108
3.3.3	Fortschrittsanzeige (ProgressBar)	117

3.3.4	Schieberegler (Slider)	118
3.3.5	Statusleiste (StatusBar)	121
3.3.6	Kombinationsfeld-Steuer-element (ImageCombo)	125
3.3.7	Symbolleiste-Steuer-element (Toolbar)	128
3.3.8	Strukturansicht-Steuer-element (TreeView)	138
3.3.9	Windows-Standarddialoge	147
3.4	Das Benutzerformular	157
3.4.1	Benutzerformulare verwalten	157
3.4.2	Steuer-elemente zur Laufzeit hinzufügen/entfernen	161
4	Vom Problem zur Lösung	165
4.1	Konzeption des Managers	165
4.2	Verwaltung der Gruppen	166
4.3	Verwaltung der Layer	178
4.4	Layer-Eigenschaften anpassen	191
4.5	Layer zurückspeichern	196
4.6	Optimierung des Projekts	199
4.7	Layer-Filter	206
5	Das Anwendungsobjekt	211
5.1	Zugriff auf die AutoCAD-Sitzung	211
5.1.1	Ereignisprozeduren des Anwendungsobjekts	212
5.1.2	Eigenschaften des Anwendungsobjekts	217
5.1.3	Prozeduren des Anwendungsobjekts	221
5.1.4	Zoomfunktionen des Anwendungsobjekts	223
5.2	Verwaltung der AutoCAD-Menüstruktur	225
5.2.1	Menügruppen	226
5.2.2	Einzelne Menügruppe bearbeiten	228
5.2.3	Verwaltung der Pull-down-Menüs in AutoCAD	230
5.2.4	Verwendung der Klasse AcadPopupMenu	231
5.2.5	Menüeinträge hinzufügen	233
5.2.6	Menü zum AutoCAD-Menü hinzufügen/entfernen	235
5.2.7	Verwendung der Klasse AcadPopupMenuItem	236
5.2.8	Beispielprojekt	238
5.2.9	Verwaltung der AutoCAD-Werkzeugkästen	242
5.2.10	Verwendung der Klasse AcadToolbar	243
5.2.11	Schaltflächen zum Werkzeugfenster hinzufügen	246
5.2.12	Verwendung des Schaltflächen-Objekts	247
5.3	AutoCAD-Einstellungen steuern	250
5.3.1	Anzeigeeinstellungen (Display-Objekt)	251
5.3.2	Entwurfseinstellungen (Drafting-Objekt)	258
5.3.3	Dateieinstellungen (Files-Objekt)	262
5.3.4	Einstellungen zum Öffnen und Speichern von Dateien (OpenSave-Objekt)	267
5.3.5	Ploteinstellungen (Output-Objekt)	272
5.3.6	Auswahleinstellungen (Selection-Objekt)	275

5.3.7	Systemeinstellungen (System-Objekt)	277
5.3.8	Benutzereinstellungen (User-Objekt)	279
5.3.9	Benutzerprofile (Profiles-Objekt)	284
6	Zeichnungsverwaltung in AutoCAD	289
6.1	Zeichnungsauflistung	289
6.1.1	Eigenschaften der Dokumentauflistung	290
6.1.2	Prozeduren der Dokumentauflistung	290
6.2	Auf einzelne Zeichnungen zugreifen	292
6.2.1	Ereignisprozeduren der Zeichnung	292
6.2.2	Eigenschaften der Zeichnung	298
6.2.3	Prozeduren der Klasse AcadDocument	311
6.3	Kommunikation des Benutzers mit der AutoCAD-Zeichnung	320
6.4	Der Modellbereich/Papierbereich	335
6.4.1	Zugriff auf den Modell-/Papierbereich	335
6.4.2	Grafische Objekte im Modell-/Papierbereich erstellen	336
6.4.3	Modell- und Papierbereich verwalten	402
7	Verwaltung der Zeichnungsstruktur	407
7.1	Bearbeiten von AutoCAD-Objekten	407
7.1.1	Eigenschaften aller AutoCAD-Objekte	407
7.1.2	Methoden aller AutoCAD-Objekte	409
7.1.3	Ereignisprozeduren aller AutoCAD-Objekte	413
7.1.4	Arbeiten mit Auflistungen	415
7.2	Verwendung von Gruppen	417
7.2.1	Gruppenauflistung	417
7.2.2	Einzelne Gruppen verarbeiten	419
7.3	Objektverbindungen (Hyperlinks)	426
7.3.1	Hyperlinks-Auflistung	426
7.4	Verwalten von Layern	434
7.4.1	Layer-Auflistung	434
7.4.2	Einzelne Layer verarbeiten	436
7.4.3	Der Layerstatusmanager	446
7.5	Linientypen verwalten	456
7.5.1	Linientypauflistung verwenden	456
7.5.2	Einzelne Linientypen verwenden	458
7.6	Layouts	468
7.6.1	Layoutverwaltung	468
7.6.2	Einzelne Layouts bearbeiten	469
7.7	Verwaltung der Textstile	498
7.7.1	Verwaltung der Stile in der Textstilauflistung	498
7.7.2	Einzelne Textstile verarbeiten	500
7.8	Benutzerkoordinatensysteme	516
7.8.1	Koordinatensystemauflistung	517
7.8.2	Einzelne Koordinatensysteme verwenden	522
7.9	Anwendungen in AutoCAD registrieren	528
7.9.1	Die Anwendungsauflistung	528
7.9.2	Einzelne Einträge der Anwendungsauflistung verarbeiten	529

7.10	Blockdefinitionen verwalten	532
7.10.1	Blockauflistung	532
7.10.2	Einzelne Blockdefinitionen verarbeiten	534
7.11	Benannte Ansichten	549
7.12	Benannte Ansichtsfenster	559
7.12.1	Ansichtsfenster verwalten	559
7.12.2	Ansichtsfenster im Papierbereich	570
7.13	Bemaßungsstile	576
7.13.1	Bemaßungsstilaufistung	576
7.13.2	Einzelne Bemaßungsstile verarbeiten	577
7.14	Erweiterte Daten (Erweiterungswörterbuch)	578
7.14.1	Wörterbuchsammlung	579
7.14.2	Einzelne Wörterbücher verwalten	581
8	Die Zeichnungsobjekte von AutoCAD	591
8.1	Allgemeines	591
8.1.1	Arbeiten mit Koordinaten	592
8.1.2	Arbeiten mit Vektoren	598
8.1.3	Arbeiten mit Winkeln	599
8.2	Zugriff auf vorhandene AutoCAD-Objekte	600
8.2.1	Grafische AutoCAD-Objekte wählen	600
8.3	AutoCAD-Objekte verarbeiten	615
8.3.1	Grundlegende Eigenschaften und Methoden grafischer Objekte	615
8.3.2	Grundlegende Objekte	632
8.3.3	3D-Flächenobjekt (Acad3DFace)	635
8.3.4	3D-Polylinien (Acad3DPolyline)	640
8.3.5	Kreisbögen (AcadArc)	646
8.3.6	Kreise (AcadCircle)	650
8.3.7	Ellipse (AcadEllipse)	655
8.3.8	Bemaßungsobjekte	660
8.3.9	Führungslinienobjekt (AcadLeader)	696
8.3.10	Schraffuren (AcadHatch)	704
8.3.11	Linie (AcadLine)	718
8.3.12	2D-Polylinien	724
8.3.13	Multilinie (AcadMLine)	737
8.3.14	Punktobjekt (AcadPoint)	743
8.3.15	Symbolobjekt (AcadShape)	745
8.3.16	Textobjekte	748
8.3.17	Strahl (AcadRay)	759
8.3.18	Konstruktionslinien (AcadXLine)	761
8.3.19	Band (AcadTrace)	763
8.3.20	2D-Fläche (AcadSolid)	765
8.3.21	Vielflächennetz (AcadPolyfaceMesh)	767
8.3.22	Polygonnetz (AcadPolygonMesh)	770
8.3.23	Rasterbild (AcadRasterImage)	782
8.3.24	Regionen (AcadRegion)	790

8.3.25	Spline (AcadSpline)	798
8.3.26	Geometrisches Toleranzobjekt (AcadTolerance)	809
8.3.27	Attributdefinitionen (AcadAttribute)	821
8.3.28	Attributreferenzen (AcadAttributeReference)	834
8.3.29	Blockreferenzen (AcadBlockReference)	836
8.3.30	Mehrfachblockreferenz (AcadMInsertBlock)	850
8.3.31	Externe Referenz (AcadExternalReference)	861
8.3.32	Volumenkörperobjekt (Acad3DSolid)	869
9	Plotten/Drucken mit VBA in AutoCAD	875
9.1	Ploteinstellungen	875
9.1.1	Einzelne Ploteinstellungen verarbeiten	876
9.2	Zeichnungen plotten	877
9.3	Drucken mit VBA	879
A	Anhang	891
A.1	Fehlerbehandlung	891
A.1.1	Laufzeitfehler abfangen	891
A.1.2	Das Err-Objekt	892
A.1.3	Auswertung von Laufzeitfehlern	893
A.2	Zugriff auf AutoCAD mit VB6	894
A.2.1	Das Zusatzprogramm AutoCAD Dockable ActiveX Control Container (accont)	895
A.3	Die Klasse MyUtility	897
A.4	Neuerungen in AutoCAD 2004	899
A.5	Infos zu AutoCAD-VBA im Internet	901
	Stichwortverzeichnis	903

Vorwort

Dieses Buch zu schreiben, war ein Wagnis. Ich bin es eingegangen, weil ich meine mehr als zehnjährige Erfahrung im Umgang mit der Programmiersprache Visual Basic und die Routine, die ich durch die Arbeit mit AutoCAD und dessen Anpassung habe, weitergeben möchte. Auch wenn ich zwischendurch öfter Zweifel hatte, ob sich all mein Wissen in diesem Buch zusammenfassen ließe, glaube ich im Endeffekt, dass es mir gelungen ist. Ich habe so viele Informationen wie möglich zusammengetragen, damit Sie problemlos Anwendungen in AutoCAD-VBA entwickeln können – angefangen von kleineren Anpassungen bis hin zu komplexen Anwendungen.

Nichtsdestotrotz kann auch ich eines nicht garantieren: Vollständigkeit. Zwar wurden alle Objekte berücksichtigt, die in der Standard-Objektbibliothek von AutoCAD verfügbar sind, so dass Sie jederzeit Ihr Wissen auffrischen oder erweitern können, wenn es darum geht, Anwendungen in AutoCAD zu erstellen. Allerdings können Sie problemlos über dieses Buch hinausgehen. Wenn Sie sich erst einmal mit den Standardobjekten von AutoCAD auskennen, wird es Ihnen sicherlich leicht fallen, die Zusatzbibliotheken von beispielsweise AutoCAD-MAP zu verwenden, da diese auf dem gleichen Prinzip aufbauen. Für den Fall, dass Sie dennoch unbeantwortete Fragen haben, werde ich leichtsinnigerweise meine E-Mail-Adresse veröffentlichen.

Bevor es nun losgehen kann, möchte ich mich an dieser Stelle noch bei einigen Menschen bedanken, die zur Entwicklung dieses Buchs beigetragen haben. Zunächst wären da natürlich der Verlag, insbesondere meine Lektorin Frau Irmgard Wagner, die mir mit Rat und Tat zur Seite stand. Dann wäre da auch Frau Petra Kienle, der ich dafür danke, dass sie mir einen großen Teil der Korrekturarbeiten abgenommen hat, da ich sonst sicher immer noch am Manuskript rumwurschteln würde. Zu erwähnen wären auch die Herren Michael Kofler (Autor von Büchern zu Linux, VB, VBA und anderen Themen) und Werner Sommer (Autor der Titel: „Schnellübersicht AutoCAD 2004 / LT 2004“, „AutoCAD 2004 / LT 2004, das Kompendium“ und „Jetzt lerne ich CAD, Zeichnen und Konstruieren mit AutoCAD 2004 / LT 2004“ und andere), die vorab bereits einen Blick in das Buch riskierten und mir ihre Meinung darüber mitgeteilt haben, so dass ich noch einige Verbesserungen einarbeiten konnte.

Zu guter Letzt möchte ich mich bei meinen Eltern bedanken, die mich in dieser langen Zeit unterstützt und mir viele Sachen abgenommen haben, so dass ich mich voll auf die Entwicklung dieses Buchs konzentrieren konnte.

Karsten Dummer

stralsunder@tiscali.de

Warum VBA benutzen?

Der erste Punkt, der für die Verwendung von VBA spricht, sind die offengelegten AutoCAD-Objekte, so dass Sie auch mit anderen Anwendungen wie z.B. Microsoft Word® oder auch Microsoft Access® auf die Daten einer Zeichnung zugreifen können. Andererseits besteht auch die Möglichkeit, aus AutoCAD-VBA diese Anwendungen wie Access zu steuern, da diese über die gleiche Entwicklungsumgebung verfügen – wenngleich diese auch etwas anders strukturiert ist.

Der zweite Punkt ist die leichte Erlernbarkeit der Programmiersprache Visual Basic. In die Grundkenntnisse haben Sie sich mit Sicherheit in wenigen Tagen eingearbeitet. Außerdem ist die Entwicklungsumgebung sehr benutzerfreundlich aufgebaut, so dass z.B. der Entwurf von Dialogfenstern sehr einfach von der Hand geht.

Ein weiterer Vorteil ist die flexible Verteilungsmöglichkeit der VBA-Projekte. Sie können diese als eigenständige Projektdateien erstellen oder, was ich persönlich als sehr gut empfinde, in Zeichnungen einbetten, was die Verteilung eines VBA-Projekts erheblich vereinfacht.

Konzeption des Buchs

Dieses Buch ist sowohl für VBA-Neulinge als auch für Umsteiger von z.B. AutoLISP gedacht. Es bezieht sich in erster Linie auf die Objektbibliothek von AutoCAD, die die Schnittstelle zwischen VBA und dem Anwendungsobjekt bildet. Nach einer kurzen Einführung in die Programmiersprache Visual Basic und den Umgang mit der Entwicklungsumgebung wendet sich das Buch voll und ganz der Beschreibung der Funktionen zur Kommunikation mit AutoCAD, den Zeichnungen und den darin enthaltenen Objekten zu. Ich habe versucht, dem Anwender das zu bieten, was dieser in der Online-Hilfe nicht findet:

- Komplette Anwendungsbeispiele zu jedem AutoCAD-Objekt, so dass Sie auch Zusammenhänge zwischen einzelnen Eigenschaften und Methoden der AutoCAD-Klassen erkennen können.
- Detaillierte Erklärungen der Klassen der Objektbibliothek von AutoCAD.
- Thematische Zusammenfassung der wichtigsten Eigenschaften und Methoden der einzelnen Klassen.

Konvention des Buchs

Im Verlauf dieses Buchs werden Sie immer wieder auf unterschiedliche Darstellungsarten des Textes stoßen. Jede Formatierung des Textes hat ihre eigene Bedeutung:

- Tastaturkürzel werden in Form von  + **Strg** + **F8** angegeben.
- Dateinamen mit „c:\vba\liesmich.txt“.
- Schaltflächen in Kapitälchen z.B. **ÖFFNEN** | **SCHLIESSEN** usw.
- Menübefehle ebenfalls in Kapitälchen, allerdings mit Bezug zueinander. Der Text **FORMAT→REIHENFOLGE→IN DEN VORDERGRUND** bezieht sich auf das Hauptmenü **FORMAT**, aus dem Sie im Untermenü **REIHENFOLGE** den Befehl **IN DEN VORDERGRUND** ausführen sollen.
- Schlüsselwörter sowie Variablen, Konstanten, Prozeduraufrufe usw. werden kursiv dargestellt, z.B. das *Application*-Objekt, das *ThisDrawing*-Klassenmodul oder die *ObjectID*-Eigenschaft.
- Werte bestimmter Eigenschaften werden im Text zusätzlich groß geschrieben. Beispielsweise erhält die *Visible*-Eigenschaft den Wert **TRUE**.

Beispielcode zu den einzelnen Objekten

Die Beispielcodes sind alle auf der CD zu finden und werden größtenteils komplett im Buch abgedruckt. Die einzelnen Projektdateien sind kapitelweise angeordnet, außerdem finden Sie einen Hinweis auf das dazugehörige Programm immer in der Einführungspassage zum Beispielcode.

Beispielprojekt: „applications.dvb“

1

Erste Schritte mit AutoCAD-VBA

Das erste Kapitel führt Sie in den Aufbau und die Bedienung der Entwicklungsumgebung von AutoCAD ein. Sie werden die Verwendung der einzelnen Komponentenfenster kennen lernen und so schon einmal einen Überblick über den riesigen Umfang dieser Programmierschnittstelle erhalten.

Kapitelübersicht

- 1.1 Steuerung der VBA-Umgebung
- 1.2 Die Benutzeroberfläche der VBA-Umgebung
- 1.3 Hilfsmittel für die Entwicklung in AutoCAD-VBA

1.1 Steuerung der VBA-Umgebung

In diesem Abschnitt befassen wir uns mit der Verwendung des VBA-Managers und den AutoCAD-Befehlen zur Steuerung der VBA-Projekte, wobei der VBA-Manager die übergeordnete Rolle spielen wird, da dieser alle AutoCAD-Befehle zum Ausführen, Laden, Speichern und Erstellen in sich vereinigt.

1.1.1 AutoCAD-Befehle zur Steuerung der VBA-Umgebung

`vbarun` – ruft ein Dialogfenster auf, das eine Liste mit vorhandenen Makros anbietet. Dort können Sie dann entweder eines der Makros ausführen, bearbeiten oder löschen. Der Befehl ist auch im Befehlszeilenmodus ausführbar. Dazu fügen Sie vor `vbarun` einen Bindestrich ein. Anschließend müssen Sie den Namen des Makros festlegen, das ausgeführt werden soll.

`vbaid` – wechselt in die VBA-Entwicklungsumgebung, in der Sie dann Codes, Formulare und Referenzen in den VBA-Projekten erstellen können. Wurde noch kein VBA-Projekt geladen oder erstellt, legt AutoCAD automatisch ein neues an, bevor die Entwicklungsumgebung gestartet wird.

`vbanan` – ruft ebenfalls den VBA-Manager auf. In diesem werden alle globalen oder in Zeichnungen eingebetteten VBA-Projekte aufgelistet. Sie können dort VBA-Projekte hinzufügen, entfernen, in Zeichnungen einbetten oder aus Zeichnungen extrahieren.

`vbalad` – fordert Sie in einem Dialogfenster auf, eine VBA-Projektdatei auszuwählen, die dann in die aktuelle AutoCAD-Sitzung geladen wird. Wenn das zu ladende Projekt andere VBA-Projekte referenziert, werden auch diese geladen (falls vorhanden). Sie können diesen Befehl

ebenfalls im Befehlszeilenmodus starten. Stellen Sie dem Kommando einfach einen Bindestrich voran (-vbalad). Sie müssten in diesem Fall den Namen der Projektdatei in der Befehlszeile eingeben, statt ihn im Dateidialog auszuwählen. Nachdem das Projekt erfolgreich geladen wurde, startet AutoCAD die VBA-Entwicklungsumgebung.

`vbaausf` – ist gleichzusetzen mit dem Befehl `vbarun`.

`vbaausdr` – führt einen einzeiligen VBA-Ausdruck aus. Bei diesen Ausdrücken handelt es sich um eine vollständige VB-Anweisung mit Schlüsselwörtern, Operatoren, Variablen, Konstanten und/oder Ausdrücken. Um mehrere Ausdrücke in eine Zeile aufzunehmen, müssen Sie diese durch einen Doppelpunkt (:) voneinander trennen.

Beispiel:

`vbaausdr`

Ausdruck: MsgBox "Ausdruck ausgeführt"

`vbaentf` – entfernt ein globales VBA-Projekt aus der aktuellen AutoCAD-Sitzung. Sie können den Namen des Projekts, das entfernt werden soll, in der nach Aufruf des Befehls angezeigten Eingabeaufforderung eingeben. Wenn Sie kein Projekt festlegen, wird das aktive globale Projekt entfernt.

1.1.2 Der VBA-Manager

Mithilfe des VBA-Managers können Sie die Projekte der aktuellen AutoCAD-Sitzung verwalten. Nachdem Sie den VBA-Manager gestartet haben, erscheint ein Dialogfenster, das alle bereits geladenen VBA-Projekte auflistet.

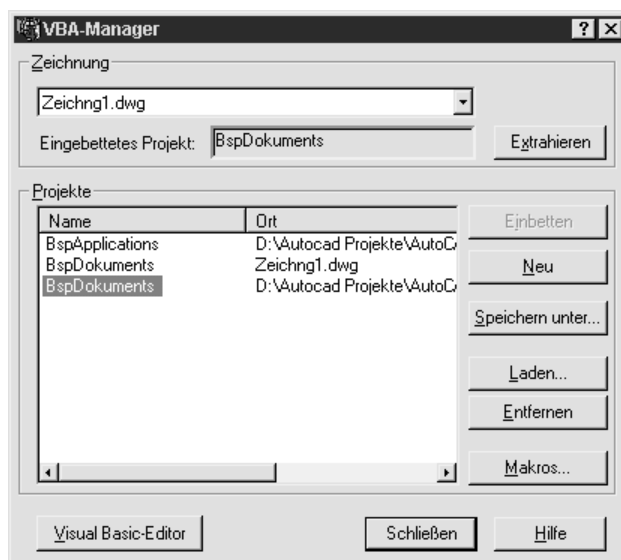


Abbildung 1.1: Der AutoCAD-VBA-Manager

VBA-Projekte verwalten

Um der AutoCAD-Sitzung ein neues VBA-Projekt hinzuzufügen, betätigen Sie lediglich die Schaltfläche **NEU**. AutoCAD erzeugt dann ein Projekt, das der Liste hinzugefügt wird. Dieses Projekt erhält zunächst den Namen „AcadProject“, den Sie in der VBA-Entwicklungsumgebung ändern können. Da das Projekt zu diesem Zeitpunkt noch nicht gespeichert wurde, steht im Abschnitt **ORT** der Eintrag **GLOBAL...** Erst nachdem Sie das Projekt in einer DVB-Datei gespeichert haben, wechselt der Eintrag auf den Speicherort des Projekts.

Wenn Sie bereits ein VBA-Projekt vorliegen haben, das Sie in der Entwicklungsumgebung bearbeiten wollen, können Sie es über die Schaltfläche **LADEN** dem Projekt-Manager hinzufügen. Damit ist es in der AutoCAD-Sitzung verfügbar und wird in der Projektliste mit Projekt- und Dateinamen angezeigt. Sie können eine unbegrenzte Anzahl an Projekten laden, allerdings nie eine Projektdatei zweimal. Projektnamen hingegen dürfen mehrfach vorkommen.

Um das Projekt aus der AutoCAD-Sitzung zu entladen, müssen Sie die Schaltfläche **ENTFERNEN** wählen. Sollte das Projekt Änderungen enthalten, die noch nicht gespeichert wurden, werden Sie durch AutoCAD darauf hingewiesen. Die Änderungen können dann immer noch gespeichert werden. Sie können gleichzeitig mehrere Projekte entfernen, indem Sie diese nach der Windows-Konvention selektieren (/**Strg**). Sie müssen allerdings darauf achten, dass Sie zuerst Projekte entfernen, die nicht von einem anderen Projekt referenziert werden. Wenn Sie versuchen, ein referenziertes, also in einem anderen geöffneten VBA-Projekt gebundenes, VBA-Projekt zu entfernen, weist AutoCAD Sie darauf hin, dass dies so nicht geht.

Eingebettete VBA-Projekte

Das Einbetten von VBA-Projekten kann in einigen Fällen recht hilfreich sein. Dabei wird das VBA-Projekt, das eingebettet ist oder eingebettet werden soll, direkt in die jeweilige Zeichnung integriert. Das heißt, dass alle Daten des Projekts, sowohl der Code als auch die eventuell vorhandenen Formulare, nicht mehr in einer Projektdatei (DVB), sondern in der jeweiligen Zeichnung gespeichert werden. Das bedeutet, dass das Projekt auch mit der Zeichnung geladen oder entfernt wird, also nur verfügbar ist, wenn auch das entsprechende Dokument in der AutoCAD-Sitzung geladen wurde. Wird die Zeichnung geschlossen, dann wird auch das Projekt entladen. Solch eine Einbettung kann nicht nur in Zeichnungs- oder Vorlagedateien durchgeführt werden, sogar DXF-Files können VBA-Projekte einbetten.

Um nun ein Projekt zu einer Zeichnung hinzuzufügen, müssen Sie lediglich das entsprechende Projekt in der Projektliste markieren und die Schaltfläche **EINBETTEN** betätigen. Die Projektdaten werden dann in der Zeichnung hinzugefügt, die gerade in der Liste der Zeichnungen ausgewählt wurde. Sie können lediglich ein VBA-Projekt pro Zeichnung einbetten, weshalb die entsprechende Schaltfläche auch deaktiviert wird, wenn bereits ein Projekt in der Zeichnung vorhanden ist.

Um ein eingebettetes Projekt wieder aus der Zeichnung zu entfernen, müssen Sie lediglich die Zeichnung aus der Liste auswählen und dann die Schaltfläche **EXTRAHIEREN** betätigen. Der VBA-Manager fragt Sie nun, ob Sie das Projekt exportieren möchten, bevor es entfernt wird. Damit das Projekt nicht verloren geht, sollten Sie den Exportvorschlag annehmen und das Projekt in einer Datei speichern. Danach wird das Projekt komplett aus der AutoCAD-Sitzung entfernt.

1.1.3 Das Dialogfenster Makros

Das Dialogfenster MAKROS dient zum Ausführen, Bearbeiten und Löschen von VBA-Makros. Des Weiteren können Sie in diesem Dialogfenster Makros erstellen, die VBA-Optionen festlegen und zum VBA-Manager wechseln. Nachdem Sie das Fenster gestartet haben, werden zunächst alle vorhandenen Makros in einem Listenfeld angezeigt, wie in Abbildung 1.2 zu sehen. Alle Makros werden dabei mit dem vollständigen Pfad, also dem Namen, dem Modul und dem Makro selbst, angezeigt.

<Makrodatei>!Modulname.Makro

Wird das gewünschte Makro bzw. Projekt nicht angezeigt, können Sie über die Schaltfläche VBA-MANAGER zu selbigem wechseln und dort das Projekt laden.

Im unteren Bereich finden Sie ein Textfeld, in dem Sie zu jedem Makro eine Kurzbeschreibung verfassen können. AutoCAD speichert diese Beschreibung in der Projektdatei.

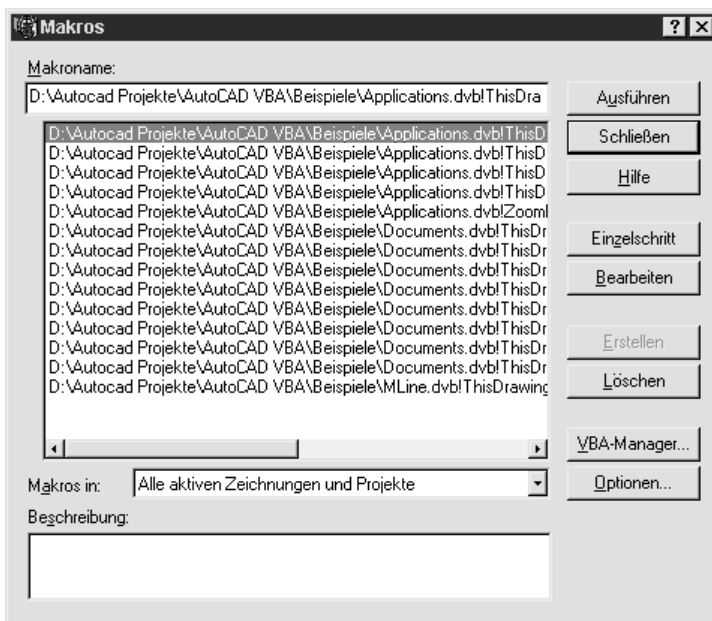


Abbildung 1.2: Dialogfenster Makros

AUSFÜHREN: Startet das ausgewählte Makro.

SCHLIESSEN: Beendet das Dialogfenster.

EINZELSCHRITT: Öffnet die VBA-Entwicklungsumgebung und startet das Makro im Einzelschrittmodus. Die Ausführung des Makros wird bei der ersten ausführbaren Codezeile angehalten.

BEARBEITEN: Öffnet die VBA-Entwicklungsumgebung und positioniert den Cursor in der ersten Zeile des Makros.

ERSTELLEN: Hiermit können Sie ein neues Makro in dem ausgewählten Projektmodul erstellen. Sollte der Makroname bereits vorhanden sein und Sie beantworten die Sicherheitsabfrage mit JA, dann wird der Code des vorhandenen Makros gelöscht.

LÖSCHEN: Löscht das ausgewählte Makro.

OPTIONEN: Ruft ein Dialogfenster auf, in dem Sie einige Einstellungen zu den Makros vornehmen können. Folgende Optionen können festgelegt werden:

- *Automatisches Einbetten aktivieren:* Bei Auswahl dieser Option legen Sie fest, dass beim Öffnen einer Zeichnung automatisch ein VBA-Projekt eingebettet wird.
- *Unterbrechung bei Fehlern zulassen:* Ist diese Option aktiviert, geht VBA in den Entwurfsmodus, sobald ein Fehler bei der Ausführung des Makros auftritt. Der Cursor wird an der Stelle positioniert, an der der Fehler aufgetreten ist, und die Ausführung wird unterbrochen. Sie können dann das Programm untersuchen und schrittweise ablaufen lassen oder fortsetzen. Die Ausführung wird allerdings nur dann unterbrochen, wenn der Fehler nicht abgefangen wird (On Error). Deaktivieren Sie diese Option, wird statt der Unterbrechung nur eine Fehlermeldung angezeigt und die Ausführung des Makros fortgesetzt.
- *Makro-Virenschutz aktivieren:* Die Aktivierung dieser Option bewirkt, dass beim Laden eines VBA-Projekts eine Warnung erscheint, dass eventuell Makroviren enthalten sein könnten.

Ausgewählte Makros anzeigen lassen

Da bei mehreren geladenen VBA-Projekten häufig eine Vielzahl an Makros aufgelistet werden, kann es oft ratsam sein, die Makros so zu filtern, dass nur bestimmte Makros angezeigt werden. Die jeweilige Quelle der Makros können Sie in der Liste unterhalb der Makroauflistung auswählen:

- Alle aktiven Zeichnungen und Projekte
- Alle aktiven Zeichnungen
- Alle aktiven Projekte
- Einzelne Zeichnung, die derzeit in AutoCAD geöffnet ist
- Einzelnes Projekt, das derzeit in AutoCAD geladen ist

Nachdem Sie einen der Einträge aus der Liste ausgewählt haben, wird die Makroliste aktualisiert. Es werden nur noch die Makros angezeigt, die der Auswahl entsprechen.

1.1.4 Makros in die AutoCAD-Menüstruktur einbinden

Sie müssen Makros nicht über die VBA-Umgebungsfenster ausführen, sondern können den Befehl auch an ein AutoCAD-Menü oder eine Werkzeugschaltfläche binden. Legen Sie dazu im Befehlstext des Menüeintrags oder der Schaltfläche den AutoCAD-Befehl `vbaausf` oder `vbarun`, gefolgt von dem entsprechenden Makro fest. Beispiel:

```
^C^C-vbarun ThisDrawing.Sample_Add3DFace
```

Wenn Sie nicht sicher sind, ob das Makro mit dem Namen auch in anderen VBA-Projekten vorhanden ist, können Sie noch den Namen der Projektdatei voranstellen, so wie er im Makrofenster aufgelistet wird.

```
^C^C-vbarun D:\Beispiele\ModelSpace.dvb!ThisDrawing.Sample_Add3DFace
```

Wenn nun der Menüpunkt oder die Schaltfläche gewählt wird, führt AutoCAD das entsprechende Makro aus.

1.1.5 Automatisches Ausführen eines VBA-Makros

Normalerweise wird ein Makro erst dann ausgeführt, wenn der Benutzer dieses über einen Menüpunkt, eine Schaltfläche oder die AutoCAD-Befehlszeile ausführt. Ein manuelles Eingreifen durch den Benutzer ist also erforderlich. Manchmal muss aber ein Makro gleich nach dem Programmstart ausgeführt werden. AutoCAD ermöglicht mit einigen Anpassungen die automatische Initialisierung der VBA-Umgebung und die automatische Ausführung von VBA-Makros.

Normalerweise wird die VBA-Umgebung immer erst dann initialisiert, wenn der Benutzer das erste Makro in AutoCAD ausführt. Damit die VBA-Umgebung beim Start von AutoCAD geladen wird, müssen Sie folgende Zeile in der Datei *acad.rx* einfügen:

```
acadvba.arx
```

Nachdem VBA automatisch geladen wird, könnten auch die Makros automatisch ausgeführt werden. Dazu müssen Sie die Datei *acad.dvb* in das AutoCAD-Verzeichnis kopieren, die Sie im Verzeichnis *Samples* unter *AutoCAD* finden, oder Sie legen in VBA ein neues Projekt an und speichern es dann unter dem Namen *acad.dvb*. Diesem Projekt fügen Sie im Klassenmodul *ThisDrawing* die Prozedur *ACADStartup()* hinzu. Diese Prozedur wird immer dann ausgeführt, wenn AutoCAD gestartet wurde. Sie können in dieser Prozedur die Aufrufe weiterer Prozeduren hinzufügen, die automatisch ausgeführt werden sollen.

1.2 Die Benutzeroberfläche der VBA-Umgebung

Nachdem Sie die VBA-Umgebung gestartet haben, erscheint die Benutzeroberfläche von VBA auf dem Bildschirm. Diese gleicht in der Bedienung und im Aussehen der von Microsoft Word® oder dem großen Bruder, der professionellen Umgebung von Visual Basic® 6 von Microsoft. Projekte, die Sie in dieser Umgebung entwickeln, können auch nur in AutoCAD ausgeführt werden und sind eigenständig nicht lauffähig.

Der Hauptbereich (1) der Benutzeroberfläche enthält die einzelnen Fenster des Projekts. Diese Fenster können Formulare, Module oder Klassenmodule sein. Sobald Sie ein Formularfenster aktivieren, wird das Steuerelementfenster (5) eingeblendet. Dieses Fenster enthält alle Steuerelemente, die Sie auf dem Formular positionieren können.

Im Projektexplorer (3) werden alle geladenen Projekte sowie deren einzelnen Objekte (Formulare, Module, Klassenmodule) in einer hierarchischen Darstellung angezeigt.

Haben Sie ein Formular zur Bearbeitung aktiviert, werden im Eigenschaftenfenster (4) alle Eigenschaften des Formulars bzw. Steuerelements, die zur Entwurfszeit verfügbar sind, angezeigt. Zusätzlich zu den Eigenschaften, wie den Namen des Objekts oder Formulars oder seine Farbe, finden Sie bei allen zur Laufzeit sichtbaren Steuerelementen (also nicht das ImageList) die Positionierungseigenschaften *Left*, *Top*, *Height* und *Width* für die linke obere Ecke, die Höhe und die Breite des Steuerelements. Diese Eigenschaften werden in VBA in der Einheit Punkt angegeben.

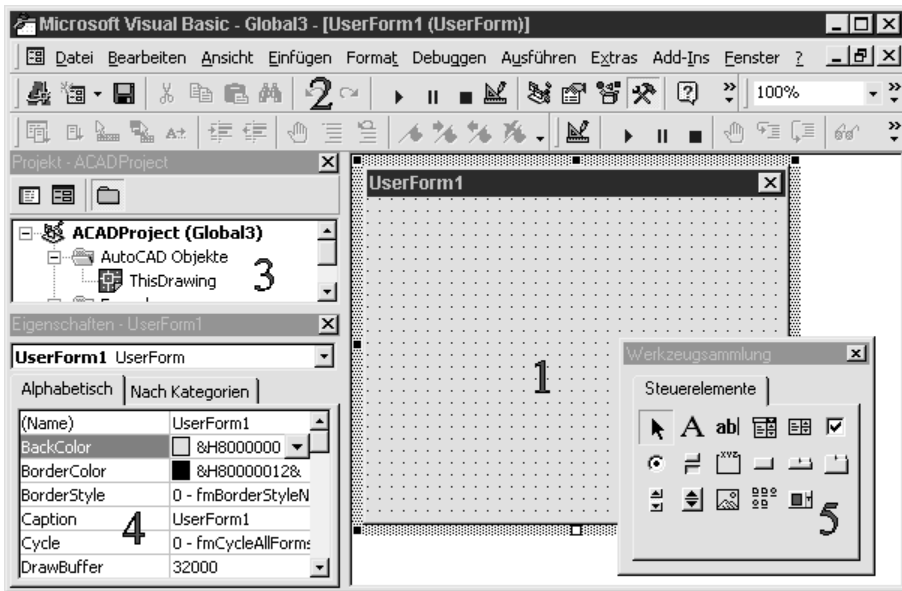


Abbildung 1.3: Die Entwicklungsoberfläche von AutoCAD-VBA

Im oberen Bereich der Benutzeroberfläche sehen Sie die Menüleiste mit den einzelnen Befehlen sowie die Toolbar-Leisten zur Bearbeitung der Projekte.

Über den Menüpunkt ANSICHT finden Sie eine Reihe von Menüpunkten, die die verschiedensten Komponentenfenster aktivieren. Sie sollten sich die dem jeweiligen Menüpunkt zugeordneten Tastenkürzel einprägen, da es während der Entwicklungsphase häufig zu einem Wechsel vom Formular oder Codefenster zu einer der zusätzlichen Komponenten und zurück kommen wird.

Zugriffstaste	Aktiviert
F7	das Codefenster
⇧ + F7	das Formular
F2	den Objektkatalog
Strg + G	das Direktfenster
Strg + R	den Projektextplorer
F4	das Eigenschaftsfenster

Tabelle 1.1: Zugriffstasten der Dialogfenster in der Entwicklungsumgebung

1.2.1 Der Projektextplorer

Im Projektextplorer werden alle Projekte und deren untergeordnete Formulare, Module und Klassenmodule aufgelistet. Sie können entweder eine hierarchische oder alphabetische Auflistung einstellen. Bei der hierarchischen Darstellung werden die einzelnen Komponenten in entsprechenden Ordnern, die dem Typ des jeweiligen Moduls entsprechen, angeordnet.

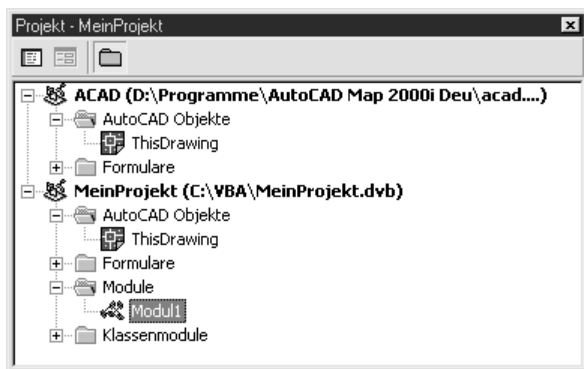


Abbildung 1.4: Projektextplorer

Das Klassenmodul *ThisDrawing* wird bei der Erstellung eines neuen Projekts automatisch zugeordnet. Alle anderen Projektobjekte müssen Sie manuell hinzufügen. Bewegen Sie hierzu den Mauszeiger auf den Projektextplorer und betätigen Sie die rechte Maustaste. Das eingeblendete Kontextmenü enthält Menüpunkte zum Einfügen eines Formulars, Moduls oder Klassenmoduls. Sie haben zusätzlich die Möglichkeit, Formulare, Module und Klassenmodule in Dateien zu exportieren oder aus Dateien wieder zu importieren. Diese Funktionen sind besonders wertvoll, um das Projekt zu optimieren. Ich habe im Laufe meiner Arbeit mit AutoCAD-VBA festgestellt, dass die Projektdatei *.dvb immer größer wird. Jedes Mal, wenn man das Projekt speichert, wird die Datei größer, egal ob Code hinzugefügt oder entfernt wird. Wenn Sie zum Abschluss Ihrer Entwicklungsarbeiten alle Formulare, Module und Klassenmodule in Dateien exportieren, dann ein neues Projekt erstellen und alle Dateien wieder importieren, werden Sie staunen, wie klein das Projekt eigentlich sein könnte.

1.2.2 Der Objektkatalog

Eines der wichtigsten Werkzeuge der VBA-Umgebung ist der Objektkatalog. Mithilfe des Objektkatalogs können Sie die Klassen, Eigenschaften, Methoden, Ereignisse und Konstanten anzeigen lassen, die in den Objektbibliotheken und Prozeduren Ihres Projekts zur Verfügung stehen. Sie können hier selbst erstellte Objekte oder Objekte aus anderen Anwendungen suchen und verwenden. Sie aktivieren den Katalog, indem Sie die Funktionstaste **[F2]** drücken oder den Menüpunkt **ANSICHT→OBJEKTKATALOG** wählen. Wenn Sie im Quellcode den Cursor auf ein Schlüsselwort positionieren und anschließend die Tastenkombination **[⇧]+[F2]** betätigen, wird ebenfalls der Objektkatalog aktiviert, nur dass in diesem Fall bereits das Schlüsselwort in der dazugehörigen Klasse markiert wird.

Im oberen Bereich des Objektkatalogfensters (2) befindet sich die Leiste zur Navigation durch die einzelnen Bibliotheken. Wenn Sie die Drop-down-Liste (5) ausklappen, so sehen Sie einige Standardeinträge, die bei der Erstellung eines neuen VBA-Projekts automatisch integriert werden. Der erste Listeneintrag mit der Bezeichnung *<Alle Bibliotheken>* stellt nichts anderes dar als die oberste Ebene der Hierarchie. Die Auswahl dieses Eintrags hat zur Folge, dass alle in dem Projekt integrierten Elemente im Listenfeld *Klassen* gleichzeitig angezeigt werden.

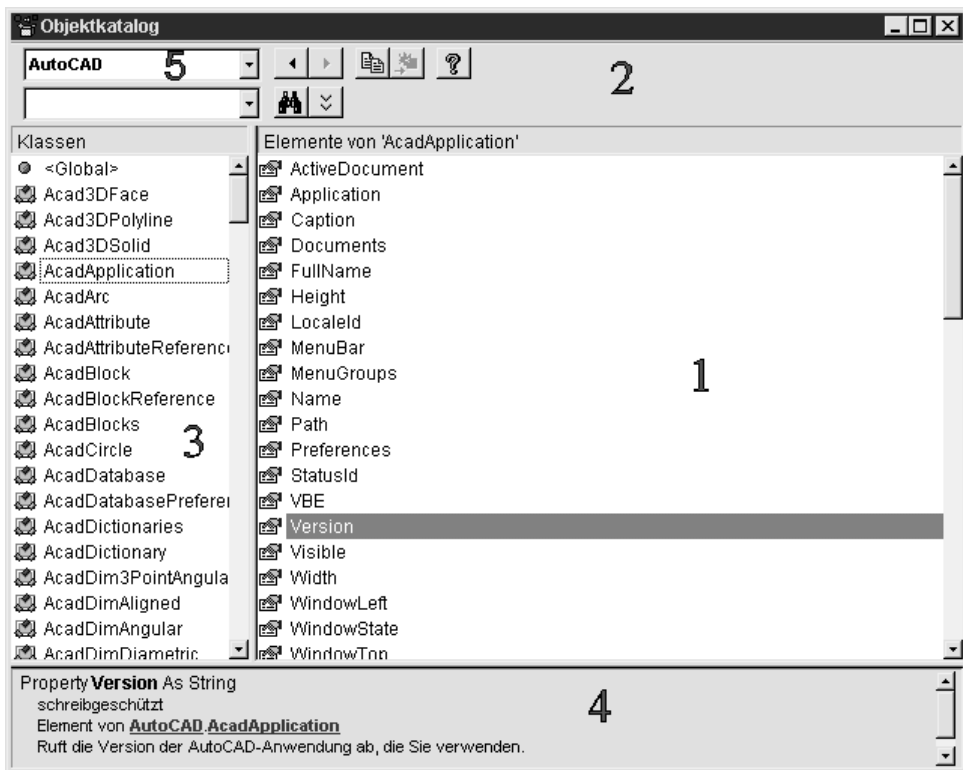


Abbildung 1.5: Der Objektkatalog von AutoCAD-VBA

Im Drop-Down-Listenfeld (5) finden Sie alle, dem aktuellen Projekt zugeordneten Bibliotheken. Um die Klassen einer bestimmten Bibliothek im Klassenlistenfeld darzustellen, wählen Sie hier die entsprechende Bibliothek aus. Anschließend werden im unteren Beschreibungsfeld (4) der Name sowie der Dateiname der Bibliothek angezeigt.

Nachdem Sie ein neues Projekt erstellt haben, werden bereits einige Bibliotheken automatisch dem Projekt hinzugefügt. Diese Bibliotheken bilden das Grundgerüst für die Arbeit mit AutoCAD-VBA:

ACADProjekt, oder wie auch immer der Name Ihres Projekts lautet, steht hier für Ihr VBA-Projekt.

AutoCAD ist die für die Arbeit in AutoCAD wichtigste Bibliothek, die Sie in Ihrem Projekt verwenden. Die Benutzung dieser Bibliothek wird im vorliegenden Buch noch hinreichend erläutert. Je nach verwendeter AutoCAD-Version können die einzelnen Klassen variieren.

MSForms enthält die Standardsteuerelemente für die Gestaltung der Formulare, z.B. Textfelder, Listenfelder oder auch Schaltflächen.

Die anderen beiden Bibliotheken enthalten verschiedene Standardfunktionen, die auch in VB6 verwendet werden.

Tipp

Wenn Sie im späteren Verlauf der Entwicklung eines Projekts die Members (Prozeduren, Variablen, Konstanten usw.) Ihrer Projektklasse betrachten, werden Sie feststellen, dass einige Elemente fettgedruckt dargestellt sind. Diese Einträge sind Elemente, die Sie in Ihrem Projekt selbst definiert haben, die also nicht standardmäßig zur Verfügung stehen.

Im linken Feld werden alle Klassen und Aufzählungen der aktuell gewählten Bibliothek aufgelistet. Wenn Sie eine bestimmte Klasse oder Aufzählung selektieren, werden im rechten Listefeld alle Elemente der entsprechenden Klasse angezeigt, die Sie verwenden können. Daneben enthalten die Klassen oft Schlüsselwörter, die nicht in der Elementliste erscheinen, da sie entweder intern verwendet werden oder eigentlich nicht mehr existieren, aus Kompatibilitätsgründen zu früheren Versionen aber noch enthalten sind. Ein gutes Beispiel sind die Eigenschaften *EntityName* und *EntityType*. Deren Aufgaben hat jetzt die Eigenschaft *ObjectName* übernommen. Die beiden Elemente wurden aber in der AutoCAD-Bibliothek belassen, damit Anwender, die bereits VBA-Projekte mit AutoCAD 14 erstellt haben, keine Probleme bei der Übernahme dieser Projekte in AutoCAD2000+ bekommen. Um diese Elemente sichtbar zu machen, müssen Sie im Kontextmenü den Punkt VERBORGENE ELEMENTE ANZEIGEN aktivieren. Die verborgenen Schlüsselwörter werden nun etwas abgeschwächt angezeigt. Außerdem finden Sie die Elemente jetzt auch bei der Entwicklung von Quellcode wieder, wenn die Elemente eines Objekts angezeigt werden.

Eine weitere Option, die bei der Suche im Objektkatalog nicht außer Acht zu lassen ist, ist die Anpassung der Anzeigenreihenfolge. Normalerweise sind die Elemente einer Klasse oder Konstantenaufzählung alphabetisch geordnet. Manchmal kann es aber ganz nützlich sein, die Elemente nach ihrer Funktion (Eigenschaft, Ereignis) zu sortieren. Dies können Sie veranlassen, indem Sie im Kontextmenü den Punkt ELEMENTE GRUPPIEREN aktivieren. Nun sind die Elemente in Gruppen zusammengefasst dargestellt – zunächst alle Eigenschaften, dann die Prozeduren, die Ereignisse und abschließend die Konstanten.

Wenn Sie ein Element einer Klasse auswählen, werden dessen Typ und die jeweiligen Parameter im Beschreibungsfeld angezeigt. Haben Sie z.B. die Funktion *ZoomScaled* der Klasse *AcadApplication* gewählt, wird folgende Definition angezeigt:

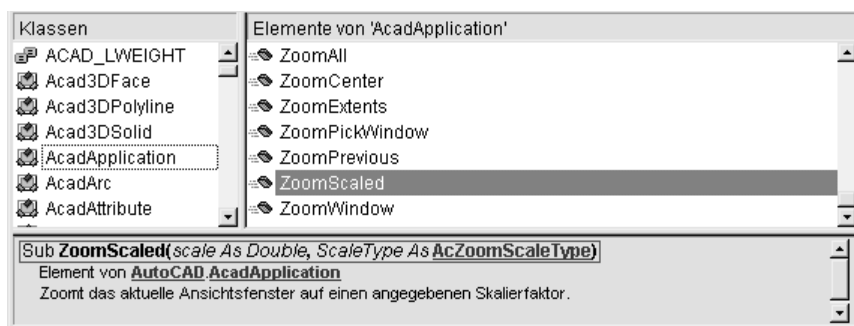


Abbildung 1.6: Ausgewähltes Element im Objektkatalog

Diese Definition entspricht der Deklarationszeile, mit der das Element erstellt wurde und welche Aufgabe diese übernehmen kann. Bei dem Element in Abbildung 1.6 wäre die Interpretation folgende: Es handelt sich um eine Prozedur mit dem Namen *ZoomScaled*, die die

Parameter *scale* und *ScaleType* erwartet. Der erste Parameter muss dem Datentyp *Double* entsprechen, der zweite dem Objekt- oder Konstantentyp *AcZoomScaled*.

Wird ein Parameter unterstrichen und fett dargestellt, so handelt es sich um einen Link auf eine andere Klasse im Objektkatalog. Sie gelangen dann über diesen Link auf die Definition der entsprechenden Klasse, indem Sie den gelinkten Text wählen.

Jede Klasse und deren Elemente werden durch folgende Symbole in den Listenfeldern dargestellt.

 Dieses Symbol stellt eine Auflistung im Objektkatalog dar. Eine Auflistung ist eine Liste von Konstanten, die unter einem Namen zusammengefasst werden. Eine Auflistung wird über die VB-Anweisung `Enum` deklariert

Beispiel:

```
Enum AcActiveSpace
    acModelSpace = 1
    acPaperSpace = 0
End Enum
```

Deklarien Sie eine Variable später mit dieser Aufzählung, werden alle Elemente der Aufzählung in einem Listenfeld angeboten.

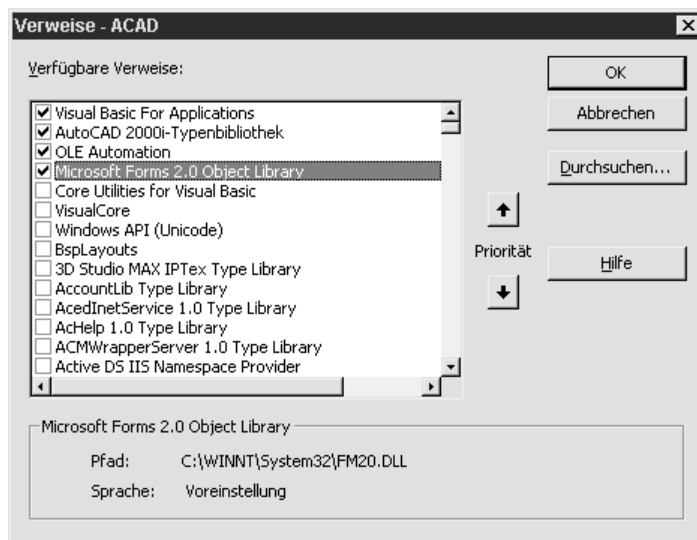


Abbildung 1.7: Verweisdialg

 Dieses Symbol stellt die formale Definition eines Objekts dar. Die Klasse dient als Vorlage, aus der eine Objektinstanz zur Laufzeit erzeugt wird. Die Klasse definiert die Eigenschaften des Objekts und die Methoden, die zur Steuerung des Objektverhaltens verwendet werden. Eine Klasse kann verschiedene Elemente enthalten.

-  Dieses Symbol stellt eine Eigenschaft eines Objekts dar. Einige Eigenschaften können Sie festlegen (lesen/schreiben), einige nur lesen (schreibgeschützt) und einige nur schreiben (nur schreiben). Wie Sie die Eigenschaft verwenden können, sehen Sie im Beschreibungsfeld. Können Sie die Eigenschaft nur lesen, steht die Zusatzinformation *schreibgeschützt* im Beschreibungsfeld.

Eigenschaften sind z.B. *Left*, *Top*, *Width* oder *Height*.

-  Dieses Symbol kennzeichnet ein Ereignis oder auch eine Aktion, die von einem Objekt erkannt wird. Ein Ereignis wird z.B. durch das Klicken mit der Maus oder das Drücken einer Taste ausgelöst. Sie können für dieses Ereignis Code schreiben, so dass das Objekt auf das Ereignis reagiert. Ereignisse können durch eine Benutzeraktion oder durch Programmcode hervorgerufen werden oder durch das System ausgelöst werden.

Ereignisse sind z.B. *MouseDown*, *EndSave* oder *ObjectModified*.

-  Dieses Symbol kennzeichnet eine Methode. Dabei handelt es sich um eine Prozedur oder Funktion, die auf ein Objekt angewendet wird.

Methoden sind z.B. *ZoomAll*, *LoadDVB*, *Quit* oder *RunMacro*.

Funktionen geben dabei, anders als Prozeduren, Werte als Ergebnis zurück.

Um dem Projekt eine weitere Objektbibliothek hinzuzufügen oder eine integrierte wieder zu entfernen, betätigen Sie die rechte Maustaste innerhalb des Objektkatalogs oder wählen den Menüpunkt EXTRAS→VERWEISE. Im folgenden Dialogfenster sind alle bereits integrierten Bibliotheken mit einem Häkchen versehen.

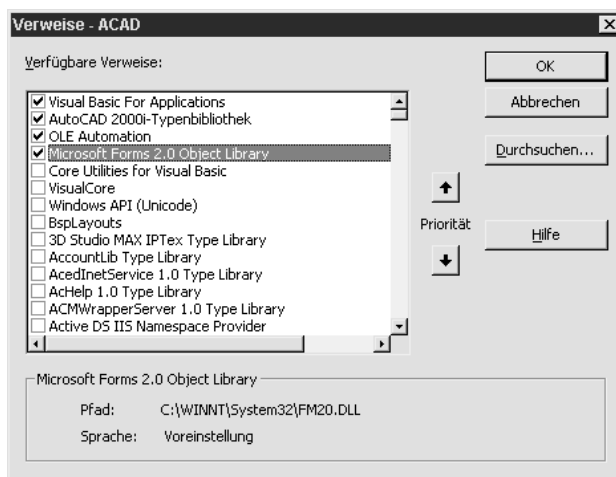


Abbildung 1.8: Verweisdialog

Sie können nun die gewünschte Bibliothek aus der Liste auswählen und dann durch Aktivierung des Hakens dem Projekt hinzufügen. Um eine Bibliothek zu entfernen, deaktivieren Sie einfach den Haken. Sie sollten allerdings darauf achten, dass diese nicht mehr verwendet wird, denn anders als bei der Auswahl von Steuerelementen weist AutoCAD Sie nicht darauf hin, dass die Bibliothek noch verwendet wird.

1.2.3 Optionsdialog

Über den Menüpunkt EXTRAS→OPTIONEN rufen Sie ein Dialogfenster auf, in dem Sie einige Einstellungen zur Arbeit in der Entwicklungsumgebung vornehmen können. Dieses Fenster ist in vier Kategorien unterteilt:

Reiter „Editor“

AUTOMATISCHE SYNTAXÜBERPRÜFUNG: Um bei der Eingabe von Codezeilen automatisch auf fehlerhafte Zeilen hingewiesen zu werden, müssen Sie diese Option aktivieren. Sobald Sie etwas fitter im Umgang mit VBA geworden sind, können Sie diese Option deaktivieren, da die Anzeige einer Fehlermeldung mit der Zeit störend wirken kann. Die fehlerhaften Zeilen werden trotzdem in roter Farbe angezeigt.

VARIABLENDEKLARATION ERFORDERLICH: Wenn Sie diese Option aktivieren, fügt VBA die Codezeile `Option Explicit` in jedes neue Modul ein. Damit werden Sie dazu gezwungen, jede verwendete Variable zu deklarieren (siehe „Deklarationen erzwingen“).

ELEMENTE AUTOMATISCH AUFLISTEN: Ist diese Option aktiviert, wird bei der Codeeingabe eine Liste mit Informationen angezeigt, die die Anweisung logisch vervollständigen würden. Diese Option sollte aktiviert sein, da die Codeeingabe dann flüssiger von der Hand geht.

QUICKINFOS: Bei Aktivierung dieser Option werden Informationen zu Funktionen und deren Parameter angezeigt.

DATENTIPPS: Wenn Sie im Haltemodus Informationen zum Inhalt von Variablen erhalten möchten, indem Sie nur den Mauszeiger auf die Variable bewegen, muss diese Option aktiviert werden.

Reiter „Editorformat“

In diesem Teil des Optionsdialogs nehmen Sie Anpassungen an Farbe, Schriftstil und -größe vor und bestimmen so, wie die einzelnen Schlüsselwörter im Codefenster dargestellt werden sollen. Normalerweise können Sie die Grundeinstellungen beibehalten, da die optische Abstimmung bereits gut gelungen ist.

Reiter „Allgemein“

EINSTELLUNGEN FÜR FORMULARRASTER: Hier können Sie die Positionierung von Steuerelementen auf Formularen beeinflussen. Je kleiner die Werte für die Rasterung sind, umso feiner wird die Positionierungsmöglichkeit der Steuerelemente.

BEARBEITEN UND FORTFAHREN: Wenn Sie während der Laufzeit Änderungen am Code vornehmen, werden in manchen Fällen, wie etwa die Deklaration neuer Variablen, die Inhalte aktueller Variablen gelöscht. Sie müssten dann den Code erneut ausführen, um den Variableninhalt wieder herzustellen, was bei weiten Programmverzweigungen oft die erneute Ausführung des Projekts nach sich zieht. Wenn Sie die Option **BENACHRICHTIGEN BEI ZUSTANDSÄNDERUNG** aktivieren, erhalten Sie eine Meldung, wenn Sie solche Änderungen am Code durchführen.

UNTERBRECHEN BEI FEHLERN: Diese drei Optionen nehmen Einfluss auf die Ausführung des Codes bei Auftreten von Fehlern. Die Option **BEI JEDEM FEHLER** schaltet sozusagen die Fehlerbehandlungsroutinen aus. Das Programm wird dann auch unterbrochen, wenn in der Prozedur eine `On Error`-Anweisung vorhanden ist. Für die Fehlersuche ist das oft recht praktisch.

Die anderen beiden Optionen führen nur dann zu unterschiedlichen Ergebnissen, wenn Sie in Ihrem Projekt mit Klassenmodulen arbeiten. Löst ein solches Klassenmodul einen Fehler aus, der nicht abgefangen wird, dann unterbricht VBA die Ausführung des Codes an unterschiedlichen Stellen. Ist die Option **IN KLASSENMODUL** aktiviert, wird die Ausführung an der Stelle unterbrochen, wo der Fehler tatsächlich auftrat. Die Option **BEI NICHT VERARBEITETEN FEHLERN** bewirkt, dass der Code an der Stelle unterbrochen wird, wo die den Fehler auslösende Prozedur aufgerufen wurde.

Reiter „Verankern“

Die Komponenten, wie das Eigenschaftsfenster, der Projektexplorer oder das Überwachungsfenster, lassen sich in der Entwicklungsumgebung verankern. Sie sind dann nicht mehr als eigenständiges Fenster zu erkennen, sondern bilden eine Einheit mit der Entwicklungsumgebung. Das erfordert natürlich einigen Platz auf dem Bildschirm, da die Fenster alle nebeneinander gelagert werden, weshalb die Verankerung nur bei entsprechender Bildschirmauflösung Sinn macht. Diese Technik fördert allerdings die Übersichtlichkeit der Entwicklungsumgebung. Die Verankerung kann hier für jedes einzelne Komponentenfenster aktiviert bzw. deaktiviert werden. Für den Objektkatalog empfehle ich die Deaktivierung, bei allen anderen bleibt es Anwendersache, ob lieber mit oder ohne Verankerung gearbeitet werden soll. Sie können die Verankerung auch in jedem Komponentenfenster direkt aus- und ebenso wieder einschalten, indem Sie im Kontextmenü des jeweiligen Fensters den Menüpunkt **VERANKERBAR** aktivieren oder deaktivieren.

1.2.4 Projekteigenschaften anpassen

Ein weiteres Dialogfenster, das zur Einstellung einiger Parameter dient, aktivieren Sie über den Menüpunkt **EXTRAS/EIGENSCHAFTEN VON ...** Die Einstellungen, die Sie hier vornehmen können, gelten allerdings ausschließlich für das gerade aktive VBA-Projekt. Dieser Dialog ist in zwei Kategorien unterteilt.

ALLGEMEIN/EIGENSCHAFTEN: Hier können Sie den Namen des Projekts festlegen sowie eine Kurzbeschreibung verfassen, die das Projekt genauer bezeichnet. Außerdem können Sie hier den Namen der Hilfedatei angeben, die mit dem Projekt verknüpft ist. Zusätzlich zur Hilfedatei kann noch eine Kontext-ID vergeben werden, die auf ein bestimmtes Hilfethema verweist, das aufgerufen wird, wenn die Hilfe zu diesem Projekt im Objektkatalog angefordert wird.

SCHUTZ: In diesem Teil des Dialogs können Sie ein Passwort festlegen, das Sie im zweiten Textfeld durch erneute Eingabe bestätigen müssen. Wenn Sie nun die Option **PROJEKT FÜR DIE ANZEIGE SPERREN** aktivieren, sollte eigentlich die Anzeige des Projekts geschützt sein. Wie sicher dieser Schutz ist, das vermag ich nicht zu beurteilen. Feststeht, dass es für die VBA-Umgebung anderer Anwendungen wie Word oder Excel bereits Programme gibt, die deren Schutz knacken können. Inwieweit das auch schon auf AutoCAD-VBA zutrifft, kann ich nicht sagen.

Tipp

Wenn Sie ein Projekt mit einem Passwort schützen und dann das Projekt und das Passwort an andere Anwender weitergeben, die mit einer früheren Version von AutoCAD arbeiten, dann hat der Benutzer der früheren AutoCAD-Version keine Möglichkeit, die Anzeige des Projekts zu aktivieren, obwohl er das Passwort kennt. AutoCAD bietet ihm gar nicht erst die Möglichkeit zur Eingabe eines Passworts, sondern zeigt lediglich die Meldung **PROJEKT FÜR ANZEIGE GESPERRT**. Der umgekehrte Weg funktioniert allerdings einwandfrei.

1.2.5 Das Eigenschaftsfenster

Über das Eigenschaftsfenster lassen sich bestimmte Anpassungen an Modulen, Formularen und Klassenmodulen vornehmen. Da ein Modul nur eine und ein Klassenmodul nur zwei Eigenschaften besitzt, fällt das Hauptaugenmerk auf die Anpassung von Formularen und der darin enthaltenen Steuerelemente.

Auch hier werden zwei unterschiedliche Anordnungen unterstützt, die auf zwei Reiter verteilt sind. Der linke Reiter zeigt eine alphabetische Auflistung der möglichen Eigenschaften an. Wenn Sie eine Unterteilung nach Kategorien bevorzugen, so schalten Sie auf den rechten Reiter NACH KATEGORIE um.

Siehe auch *Formularentwurf*.

1.3 Hilfsmittel für die Entwicklung in AutoCAD-VBA

Ihnen werden eine Reihe von Hilfsmitteln und Fenstern zur Verfügung gestellt, um Ihnen die Fehlersuche während der Entwicklung zu erleichtern. Im folgenden Abschnitt können Sie sich einen kurzen Überblick über die einzelnen Tools verschaffen. Beginnen wir zunächst einmal mit der schrittweisen Abarbeitung des Programmcodes.

Erzeugen Sie zunächst ein neues Projekt mithilfe des VBA-Managers. Fügen Sie diesem Projekt ein Formular und ein Modul hinzu. Platzieren Sie auf dem Formular zwei Schaltflächen. Lassen Sie sich das Codefenster des Formulars anzeigen und ergänzen Sie folgenden Code.

Formularcode:

```
Public OK As Boolean
```

```
Private Sub CommandButton1_Click()  
If MsgBox("Auswertung abbrechen?", vbYesNo) = vbYes Then  
    OK = True  
    Me.Hide  
End If  
End Sub
```

```
Private Sub CommandButton2_Click()  
If MsgBox("Auswertung abbrechen?", vbYesNo) = vbYes Then  
    OK = False  
    Me.Hide  
End If  
End Sub
```

Modulcode:

```
Private Sub CommandButton1_Click()  
UserForm1.Show  
End Sub
```

Zunächst einmal positionieren Sie einen Breakpoint im Modulcode. An dieser Marke wird die Ausführung des Codes unterbrochen. Solch eine Markierung wird durch einen roten Punkt am linken Rand des Fensters dargestellt. Zusätzlich ist die gesamte Codezeile markiert. Auf diese Weise ist es Ihnen möglich, das Programm an einer bestimmten Stelle anhalten zu lassen und gegebenenfalls eine Inspektion von Variablen durchzuführen. Um solch einen Breakpoint zu setzen, haben Sie mehrere Möglichkeiten. Zum einen können Sie den Cursor auf die entsprechende Zeile bewegen und einfach die Taste **[F9]** drücken oder den Menüpunkt **DEBUGGEN/HALTEPUNKT EIN/AUS** wählen. Zum anderen können Sie auch einfach auf den linken Rand, neben der Haltepunktzeile, mit der Maus klicken, um den Haltepunkt zu aktivieren oder wieder zu deaktivieren.

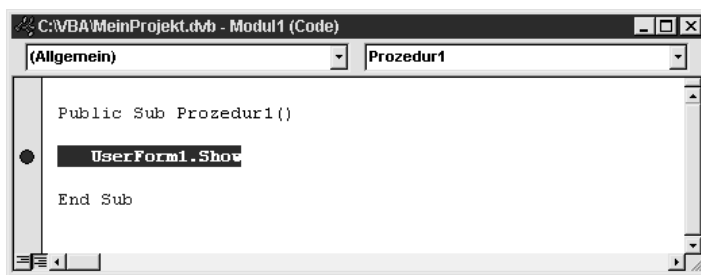


Abbildung 1.9: Haltepunkt markieren

Starten Sie nun die Prozedur über **AUSFÜHREN→SUB/USERFORM AUSFÜHREN** oder drücken Sie einfach die Taste **[F5]**. Die Ausführung des Programms wird nun wie erwartet an der vorgesehenen Stelle angehalten.

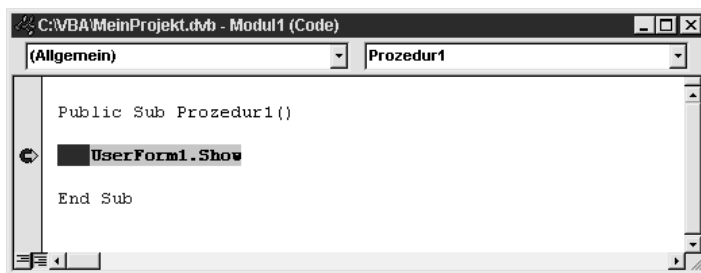


Abbildung 1.10: Ausführung am Haltepunkt gestoppt

Wie Sie erkennen können, ist ein Teil der Zeile gelb markiert und der rote Punkt wird durch einen gelben Pfeil überlagert. Sie können diesen gelben Pfeil mit der Maustaste erfassen, indem Sie den Mauszeiger auf diesen Pfeil bewegen. Der Mauszeiger ändert daraufhin seine Form. Drücken Sie nun die linke Maustaste und halten Sie diese gedrückt. Bewegen Sie den Mauszeiger an die nächste gültige Position und lassen Sie die Maustaste wieder los. Damit haben Sie den aktuellen Haltepunkt an eine andere Position verschoben und der Ablauf des Programms wird von dieser Stelle aus fortgesetzt.

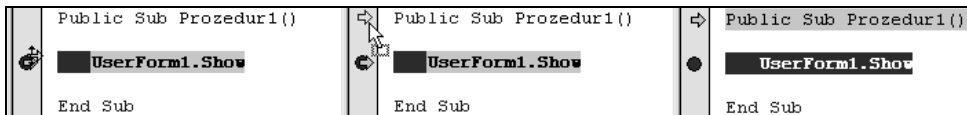


Abbildung 1.11: Haltepunkt auf eine andere Codezeile verschieben

Führen Sie nun das Programm fort, indem Sie erneut die Taste **[F5]** betätigen. Das Formular wird aufgerufen.

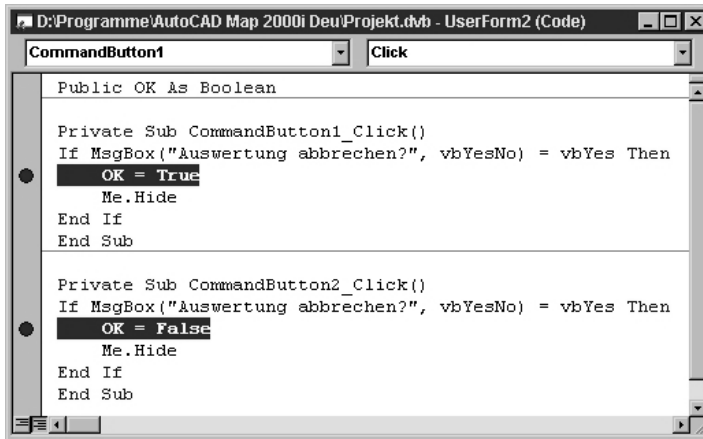


Abbildung 1.12: Weitere Haltepunkte setzen

Setzen Sie nun zwei weitere Haltepunkte wie in Abbildung 1.12 dargestellt. Sie brauchen hierfür die Ausführung des Programms nicht zu unterbrechen, sondern wechseln Sie lediglich in die Entwicklungsumgebung, um die entsprechenden Zeilen zu markieren, und wechseln Sie wieder zum Formular.

Wenn Sie jetzt die Schaltfläche WEITER betätigen, wird ein Abfragedialog eingeblendet. Beantworten Sie die gestellte Frage mit JA, wird die Ausführung am Haltepunkt unterbrochen.

Kontrollieren Sie doch einfach mal den Inhalt der Variablen OK vor und nach dem Durchlauf dieser Zeile. Bewegen Sie dazu den Mauszeiger auf die Variable und warten Sie einen Augenblick. Es wird deren aktueller Inhalt als Tooltip-Text angezeigt (Inhalt: Falsch). Wenn Sie die Funktionstaste **[F8]** betätigen, wird der Code um einen Programmschritt fortgeführt. Der aktuelle Haltepunkt befindet sich jetzt in der nächsten Programmzeile. Wenn Sie nun wieder den Mauszeiger auf die Variable OK bewegen und einen Moment warten, wird als aktueller Inhalt der Wert Wahr (TRUE) angezeigt.

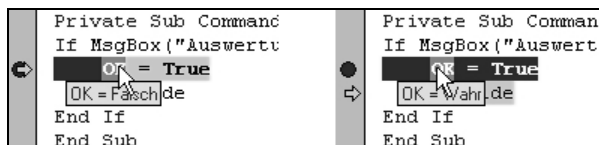


Abbildung 1.13: Variableninhalt vor und nach Durchlauf der Programmzeile

Sie können nun den Programmablauf fortsetzen, indem Sie die Taste **F5** betätigen.

Mit dieser Arbeitsweise verfügen Sie schon über einen guten Ansatz zur erfolgreichen Entwicklung von Programmen unter AutoCAD-VBA, aber es geht noch besser.

1.3.1 Das Lokal-Fenster

Im Lokal-Fenster werden alle deklarierten Variablen der aktuellen Prozedur und deren Werte angezeigt. Sie können dieses Fenster aktivieren, indem Sie den Menüpunkt **ANSICHT→LOKAL-FENSTER** ausführen.

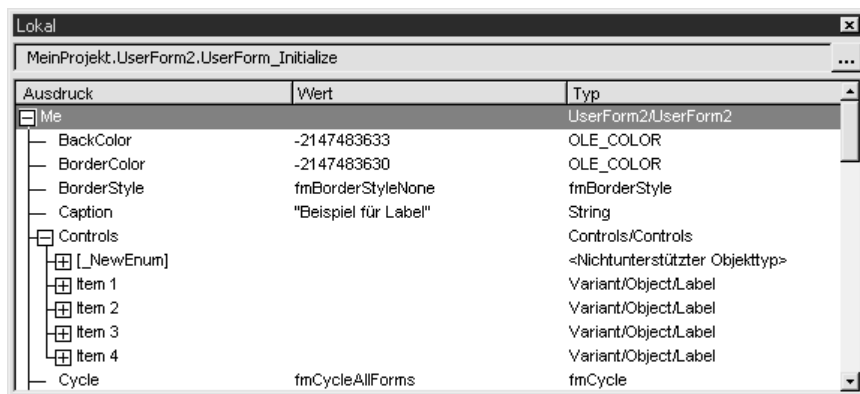


Abbildung 1.14: Das Lokal-Fenster

Der Inhalt des Fensters wird jedes Mal aktualisiert, nachdem ein Wechsel zwischen Laufzeit- und Haltemodus stattgefunden hat oder Sie einen anderen Eintrag in der Aufrufliste ausgewählt haben.

1.3.2 Das Direktausgabe-Fenster

Das Direktfenster der VBA-Umgebung ist ein besonderes Werkzeug beim Debugging Ihrer Anwendung. Wenn Sie sich im Debug-Modus befinden, können Sie dieses Fenster über den Menüpunkt **ANSICHT→DIREKTFENSTER** aktivieren. In diesem Fenster können Sie direkte Eingaben und Abfragen durchführen, ohne in den Programmablauf einzugreifen. Starten Sie einmal dieses Fenster und geben Sie folgenden Befehl ein:

```
? Me.Caption
```

Schließen Sie Ihre Eingabe mit der **↵**-Taste ab. Als Ergebnis müssten Sie jetzt die Caption-Eigenschaft des aktuell angezeigten Formulars erhalten.

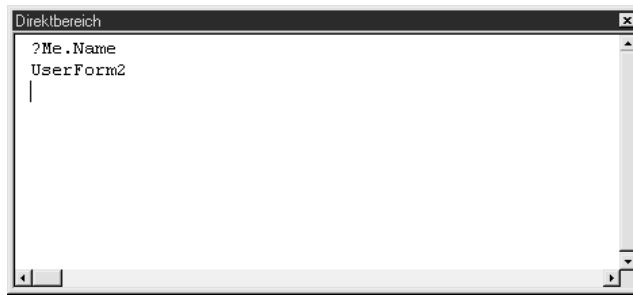


Abbildung 1.15: Direktausgabefenster

In der folgenden Tabelle sind alle Tastenkombinationen aufgelistet, die Sie im Direktfenster verwenden können.

Tastenkürzel	
Eingabetaste	Ausführen einer markierten Code-Zeile.
Strg + C	Kopieren des markierten Texts in die Zwischenablage.
Strg + V	Einfügen des Inhalts der Zwischenablage an der Einfügemarke.
Strg + X	Ausschneiden des markierten Texts in die Zwischenablage.
Strg + L	Anzeigen des Dialogfelds AUFRUFELISTE (nur im Haltemodus).
F5	Fortsetzen der Ausführung einer Anwendung.
F8	Zeilenweises Ausführen von Code (Einzelschritt).
↵ + F8	Prozedurweises Ausführen von Code (Prozedurschritt).
Entf oder ←	Löschen des markierten Texts, ohne ihn in die Zwischenablage zu kopieren.
F2	Anzeigen des Objektkatalogs.
Strg + ↶	Einfügen eines Wagenrücklaufzeichens.
Strg + Pos1	Positionieren des Cursors am Anfang des Direktfensters.
Strg + Ende	Positionieren des Cursors am Ende des Direktfensters.
↵ + F10	Anzeigen des Kontextmenüs.
Alt + F5	Ausführen des Fehlerbehandlungscodes oder Zurückgeben des Fehlers an die aufrufende Prozedur. Die Einstellung für die Fehlerbehandlung, die im Dialogfeld OPTIONEN auf der Registerkarte ALLGEMEIN festgelegt ist, ist davon nicht betroffen.
Alt + F8	Springt in den Fehlerbehandlungscodes oder gibt den Fehler an die aufrufende Prozedur zurück. Die Einstellung für die Fehlerbehandlung, die im Dialogfeld OPTIONEN auf der Registerkarte ALLGEMEIN festgelegt ist, ist davon nicht betroffen.

Tabelle 1.2: Tastaturkürzel des Direktfensters

Im Haltemodus wird eine Anweisung im Direktfenster in dem Kontext oder Bereich ausgeführt, der im Feld PROZEDUR angezeigt wird. Wenn Sie z.B. `Print Variablenname` eingeben, wird der Wert der lokalen Variable ausgegeben. Das ist dann so, als ob die `Print`-Methode in der Prozedur aufgerufen wurde, die beim Unterbrechen des Programms gerade aktiv war.

1.3.3 Das Überwachungsfenster

Das Überwachungsfenster wird automatisch eingeblendet, wenn Sie Überwachungsausdrücke in dem Projekt definiert haben. Markieren Sie hierfür zur Laufzeit eine Variable und betätigen Sie die Tastenkombination **⇧ + [F9]**. Es wird folgendes Dialogfenster angezeigt.

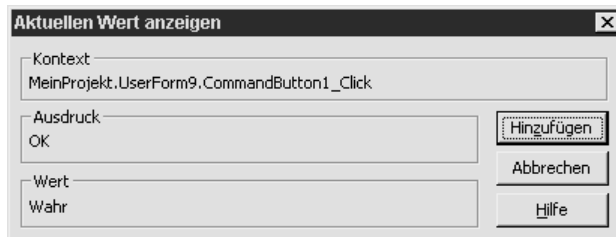


Abbildung 1.16: Überwachung hinzufügen

In diesem Dialogfenster sehen Sie schon einmal den Inhalt der Variablen **OK**. Betätigen Sie die Schaltfläche **HINZUFÜGEN**, um die Variable **OK** in das Überwachungsfenster einzutragen. Jetzt können Sie immer den aktuellen Inhalt der Variablen **OK** kontrollieren, vorausgesetzt Sie befinden sich im Gültigkeitsbereich dieser Variablen.

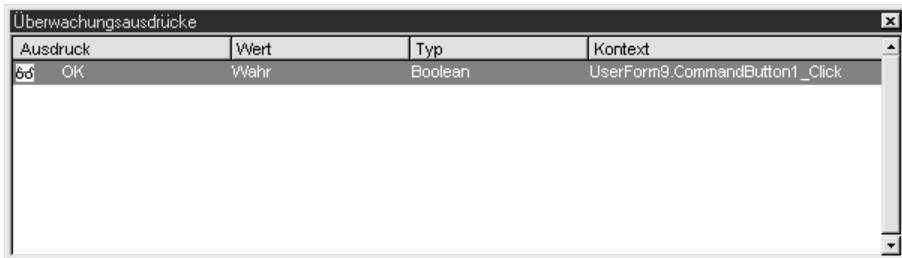


Abbildung 1.17: Das Überwachungsfenster

Für das Einfügen, Ändern und Löschen von Variablen im Überwachungsfenster gehen Sie wie folgt vor. Betätigen Sie im Überwachungsfenster die rechte Maustaste. Im darauf folgenden Kontextmenü können Sie zwischen drei verschiedenen Befehlen auswählen.

ÜBERWACHUNG BEARBEITEN: Diese Funktion ruft ein Dialogfenster zur Änderung der markierten Variablen im Überwachungsfenster auf.

ÜBERWACHUNG HINZUFÜGEN: Diese Funktion fügt dem Überwachungsfenster eine Variable zur Überwachung hinzu.

ÜBERWACHUNG LÖSCHEN: Diese Funktion entfernt die markierte Variable aus dem Überwachungsfenster. Sie können hierfür auch die Taste **[Entf]** betätigen.

Sie können die Variablennamen auch im Überwachungsfenster direkt ändern. Tippen Sie hierfür kurz auf den Namen der Variablen in der Spalte „Ausdruck“ und ändern Sie den Eintrag des Texts. Sie müssen hierbei lediglich darauf achten, dass die Variable, die zuvor dort eingetragen war, in derselben Prozedur vorhanden ist wie die neue.

2

Einführung in VBA

In diesem Kapitel erhalten Sie eine Einführung in die Konzeption der Programmiersprache Visual Basic. Angefangen beim Umgang mit Variablen, über die Verwendung von Schleifen und Verzweigungen bis hin zur Entwicklung von Klassen beinhaltet dieser Abschnitt alles, was Sie an Kenntnissen benötigen, um Anwendungen auf Modulebene zu erstellen.

Kapitelübersicht

2.1 Verwenden von Variablen

2.2 Prozeduren in Modulen, Formularen und Klassenmodulen

2.3 Schleifen, Abfragen und Verzweigungen

2.4 Objekte

2.5 Entwicklung eigener Klassenmodulen

2.6 Operatoren in VBA

2.1 Verwenden von Variablen

Bei der Programmierung ist es fast immer erforderlich, Zahlen, Zeichenfolgen oder Felder von einem Programmteil zu einem anderen zu transportieren. Diese Aufgabe übernehmen Variablen. Sie dienen sozusagen als Platzhalter für Werte, die gelagert werden sollen, um an anderer Stelle weiterverwendet zu werden.

2.1.1 Festlegen der Variablennamen

Für die Festlegung des Namens einer Variablen gibt es einige Einschränkungen, die Sie unbedingt beachten müssen. Zwar informiert Sie VBA darüber, wenn Sie einen fehlerhaften Variablennamen verwenden, aber es ist immer besser, solche Falscheingaben von vornherein auszuschließen. Wenn Sie einen Variablennamen definieren, müssen Sie also darauf achten, dass der Name einer Variablen nicht länger als 255 Zeichen sein darf, dass diese immer mit einem Buchstaben beginnt und dass Punkte, Leer- und bestimmte Sonderzeichen nicht enthalten sein dürfen. Obwohl die Verwendung von Umlauten, die ja spezielle deutsche Sonderzeichen darstellen, durchaus gestattet ist, habe ich mir von Anfang an angewöhnt, englische Bezeichnungen für die Variablennamen zu verwenden, da diese zumeist kürzer in der Schreibweise sind und bestimmte Bezeichnungen deutlicher auf den Punkt bringen, aber das bleibt natürlich Ihnen überlassen. Eine weitere Einschränkung, die Sie beachten müssen, betrifft die

Schlüsselwörter in VBA. Diese dürfen ebenfalls nicht als Variablennamen verwendet werden. Ein gutes Beispiel ist das Schlüsselwort `Lock`. Wenn Sie mit Layer-Objekten arbeiten, wird Ihnen unweigerlich dieser Begriff begegnen. Er definiert die Sperrung von Layern:

```
Layer.Lock = True
```

Wenn Sie aber versuchen, den Wert dieser Eigenschaft an eine Variable gleichen Namens zu übergeben, wird VBA Sie sofort bremsen und eine Fehlermeldung ausgeben. Der Name `Lock` wird nämlich auch als Schlüsselwort, das zur Sperrung von Dateien dient, verwendet. Sie müssten dann einen anderen Namen verwenden.

```
Locked = Layer.Lock
```

oder

```
bLock = Layer.Lock
```

Weitere Schlüsselwörter in VBA sind z.B. `Public`, `Dim`, `Sub`, `Function` usw.

2.1.2 Deklaration von Variablen

Deklarationen von Variablen werden normalerweise nur durch die Verwendung der Schlüsselwörter `Public` oder `Dim` vorgenommen. Zusätzlich kann der Datentyp angegeben werden, so dass folgende Schreibweise zustande kommt:

```
Dim LayerName As String
```

oder

```
Public LayerName As String
```

Der Unterschied zwischen den beiden Schlüsselwörtern `Public` und `Dim` ist lediglich die Reichweite der deklarierten Variable, aber dazu später mehr. Normalerweise können Sie in VBA Variablen ohne Deklaration verwenden. Diese Variablen sind dann immer vom Typ *Variant*.

```
iVal = 1
```

```
dVal = 0.005
```

```
sVal = "Zeichenfolge"
```

Dieser spezielle Datentyp passt sich automatisch an die übergebenen Daten an. Sie können diese Art der Zuweisung verwenden, vorausgesetzt, dass am Anfang des Modul- bzw. Formlarcodes nicht die Anweisung

```
Option Explicit
```

festgelegt wurde. Diese spezielle Anweisung zwingt den Entwickler, jede verwendete Variable auch zu deklarieren.

Warum sollte man aber diese Einschränkung in Kauf nehmen? Nun, einer der Gründe hierfür ist die Geschwindigkeit während der Programmausführung. Wenn Sie die folgenden Codezeilen einmal mit und einmal ohne Deklaration der entsprechenden Variablen durchführen, werden Sie feststellen, dass ein Durchlauf ohne Deklaration ungefähr doppelt solange benötigt.

Ohne Deklaration:

```
For X = 0 To 100000000
    Y = 3 * Z + X
Next
```

Mit Deklaration:

```
Dim X, Y, Z
For X = 0 To 100000000
    Y = 3 * Z + X
Next
```

Ein anderer guter Grund, alle Variablen zu deklarieren, ist, dass die Variablenamen nach der Eingabe automatisch in der Schreibweise angepasst werden. So können Schreibfehler von vornherein vermieden werden. Ein Beispiel: Sie deklarieren folgende Variable

```
Dim OutValue As VbMsgBoxResult
```

Geben Sie nun folgende Zeile ein:

```
outvalue = MsgBox("Test", vbYesNo)
```

Nachdem Sie die Eingabe der Zeile vollendet haben und in die nächste Zeile vorrücken, wird die Variable wie folgt angepasst:

```
OutValue = MsgBox("Test", vbYesNo)
```

Ein weiterer Vorteil ist, dass bei Deklarationen als thematisch zusammengefasste Konstante die entsprechenden verfügbaren Werte in einem Listefeld angeboten werden.

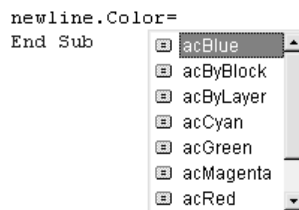


Abbildung 2.1: Anzeige der thematisch zusammengefassten Konstante acColor

Bei der Deklaration mehrerer Variablen brauchen Sie nicht jede in einer eigenen Zeile definieren, sondern Sie können diese auch in einer Zeile zusammenfassen.

```
Dim XCoord As Double
Dim YCoord As Double
Dim ZCoord As Double
```

Diese drei Deklarationen in einer Zeile zusammengefasst, ergibt Folgendes:

```
Dim XCoord As Double, YCoord As Double, ZCoord As Double
```

Sie müssen hier jeder Variablen den Datentyp zuweisen. Wenn Sie das nicht tun und die folgende Schreibweise verwenden

```
Dim XCoord, YCoord, ZCoord As Double
```

dann wird nur die Variable *ZCoord* mit dem Datentyp *Double* deklariert. Die anderen sind vom Datentyp *Variant*, da ja keine explizite Zuweisung des Datentyps erfolgte.

Tipp

Sie brauchen der deklarierten Variable keinen Datentyp zuzuweisen, da jede Variable intern als Datentyp *Variant* behandelt wird. Um aber Probleme in AutoCAD-VBA zu vermeiden, empfiehlt es sich, allen Variablen auch einen Datentyp zuzuordnen. Außerdem verwendet dieser Datentyp den meisten Speicherplatz.

2.1.3 Vorgabedatentyp festlegen

Wenn Sie später etwas umfangreichere Anwendungen in AutoCAD entwickeln, wird sich auch die Anzahl der verwendeten Variablen erhöhen. Sie können dann durch Verwendung einer Vorgabedefinition für Variablennamen bereits die Zuweisung eines Datentyps allein durch Verwendung des Variablennamens vorgeben. Zum Beispiel könnten alle booleschen Variablen mit einem *b* beginnen. Dies vorausgesetzt, können Sie durch die Verwendung des Schlüsselworts *DefBool* VBA mitteilen, dass alle Variablen, die mit einem *b* beginnen, als Variable des Datentyps *Boolean* anzusehen sind.

```
DefBool b
Dim bVisible
```

Die Variable *bVisible* bekommt nun automatisch den Datentyp *Boolean* verpasst. Das gilt für alle weiteren Variablen, die im Definitionsbereich des so festgelegten Datentyps liegt. Wenn Sie z.B. die Anweisung innerhalb einer Prozedur vornehmen und eine Variable außerhalb der Prozedur deklarieren,

```
Dim bVisible
Sub SetVisible()
DefBool b
End Sub
```

dann hat die Definition mit *Def...* keine Auswirkung auf die außerhalb der Prozedur deklarierten Variablen. Mehr zur Reichweite von Variablen in *Kapitel 2.2.3*.

Sie können für die Deklaration der Vorgabedatentypen auch ganze Buchstabenbereiche definieren, indem Sie die einzelnen Buchstaben durch Komma getrennt oder mit einem Bindestrich verbundene Buchstabenbereiche angeben:

```
DefBool b, l
```

oder

```
DefBool b - n
```

Der erste Fall deklariert alle Variablen, die mit einem *b* oder *l* beginnen, mit dem Datentyp *Boolean*. Im zweiten Fall werden alle Variablen, deren Namen mit einem Buchstaben im Bereich *b* bis *n* beginnen, mit dem Datentyp *Boolean* deklariert.

Schlüsselwort	Ergebnis
<i>DefBool</i>	Datentyp Boolean
<i>DefByte</i>	Datentyp Byte
<i>DefInt</i>	Datentyp Integer
<i>DefLng</i>	Datentyp Long
<i>DefCur</i>	Datentyp Currency
<i>DefSng</i>	Datentyp Single
<i>DefDbl</i>	Datentyp Double
<i>DefDec</i>	Datentyp Decimal
<i>DefDate</i>	Datentyp Date
<i>DefStr</i>	Datentyp String
<i>DefObj</i>	Datentyp Object
<i>DefVar</i>	Datentyp Variant

Tabelle 2.1: Schlüsselwörter für die Definition eines Vorgabedatentyps

2.1.4 Verfügbare Datentypen

Jede Variable, die Sie verwenden, entspricht einem bestimmten Datentyp. Die Datentypen haben aber nicht nur einen unterschiedlichen Verwendungszweck, sie benötigen auch unterschiedlich viel Speicherplatz. Darum ist es sinnvoll, wenn Sie die einzelnen Datentypen, die VBA zur Verfügung stellt, kennen lernen.

Datentyp	Größe	Wertebereich
<i>Byte</i>	1 Byte	Ganzzahlen im Bereich zwischen 0 und 255
<i>Boolean</i>	2 Byte	Logische Werte, die zur Prüfung zweier Zustände verwendet werden; <i>TRUE</i> (Wahr) oder <i>FALSE</i> (Falsch)
<i>Integer</i>	2 Byte	Ganzzahlen im Bereich zwischen –32.768 und 32.767
<i>Long</i>	4 Byte	Lange Ganzzahlen im Bereich zwischen –2.147.483.648 und 2.147.483.647
<i>Single</i>	4 Byte	Gleitkommazahlen mit einfacher Genauigkeit, –3,402823E38 bis –1,401298E-45 für negative Werte, 1,401298E-45 bis 3,402823E38 für positive Werte
<i>Double</i>	8 Byte	Fließkommazahl mit doppelter Genauigkeit, 1,79769313486231E308 bis –4,94065645841247E-324 für negative Zahlen, 4,94065645841247E-324 bis 1,79769313486232E308 für positive Zahlen
<i>Currency</i>	8 Byte	Währungsformat –922.337.203.685.477,5808 bis 922.337.203.685.477,5807

Tabelle 2.2: Datentypen in VBA

2.1.6 Datenfelder

Sie können Arrays auch mehrdimensional deklarieren, um z.B. eine Transformationsmatrix für die AutoCAD-Funktion `Object.TransformBy` zu erzeugen.

Transformationsmatrix					Das zur Transformationsmatrix deklarierte Array dimensionieren Sie danach auf folgende Weise:
	0	1	2	3	Dim transMat(3, 3) As Double
0	R00	R01	R02	T0	
1	R10	R11	R12	T1	
2	R20	R21	R22	T2	
3	0	0	0	1	

Um auf einen Wert in dieser Matrix zugreifen zu können oder einen Wert zu übergeben, verwenden Sie den entsprechenden Index getrennt durch Kommas.

Für den Wert der Zelle R21:

Wert lesen:

```
mtValue = transMat(2, 1)
```

Wert schreiben:

```
transMat(2, 1) = mtValue
```

2.1.7 Dynamische Arrays verwenden (Arrays)

Haben Sie einmal ein Array dimensioniert, können Sie während des Programmablaufs nichts mehr daran ändern, da eine Deklaration mit `Dim`, `Private`, `Public`, `Global` immer eine Konstante als Array-Deklaration benötigt. Es ist aber manchmal erforderlich, ein Array nachträglich auf eine bestimmte Größe zu dimensionieren. Nehmen wir z.B. einmal an, Sie möchten aus einer Polylinie bestimmte Koordinatenpaare entfernen oder der Polylinie hinzufügen. Dazu müssten Sie folgendermaßen vorgehen.

- Auslesen der vorhandenen Koordinaten über die Objektvariable *Coordinates* der entsprechenden Polylinie.

```
VarPoints = Pl.Coordinates
```

- Deklarieren eines neuen Array mit den entsprechenden Dimensionen, wobei jedes Feld dem angegebenen Datentyp entspricht

```
ReDim VertList(UBound(VarPoints) - 3) As Double
```

Die Funktion `UBound()` bestimmt dabei die obere Grenze der Koordinatenliste z.B. drei Koordinatenpaare ergeben einen Wert von:

3 Paare * 3 Werte pro Koordinate (X, Y, Z) – 1 Dimensionierung beginnt mit 0
ergibt

```
3 * 3 - 1 = 8                   -> VarPoints(0 To 8)
```

Im oberen Beispiel wurde ein Koordinatenpaar entfernt. Wollen Sie also ein Koordinatenpaar hinzufügen, müssen Sie die aktuelle Array-Größe um drei Felder erweitern.

Anschließend füllen Sie das neue Array mit den entsprechenden Werten und weisen die neue Koordinatenliste der Polylinie zu.

```
For x = 0 To UBound(VertList)
    VertList(x) = Wert
Next
Pl.Coordinates = VertList
```

Wenn Sie ein Array neu dimensionieren, sind alle zuvor belegten Inhalte verloren. Möchten Sie aber der Koordinatenliste `VarPoint` nur ein oder mehrere Koordinatenpaare hinzufügen, so ist es sinnvoll, den ursprünglichen Inhalt zu behalten. Dies erreichen Sie, indem Sie ein zusätzliches Schlüsselwort nach der Anweisung `ReDim` einfügen.

```
ReDim Preserve VarPoints(UBound(VarPoints) + 3)
```

Dadurch wird `ReDim` angewiesen, die bereits belegten Felder zu belassen und nur neue Grenzen für das Array festzulegen.

2.1.8 Feldanzahl von Arrays ermitteln

Vor allem bei der Verwendung dynamischer Arrays wird es oftmals erforderlich sein, die Anzahl der im Array enthaltenen Felder zu ermitteln. Zu diesem Zweck stellt VBA die Funktionen `LBound` und `UBound` zur Verfügung. `LBound` gibt den kleinsten, `UBound` den größten verwendeten Index des Arrays zurück. Sie können diese Funktionen z.B. in einer `For Next`-Schleife verwenden, um die einzelnen Felder nacheinander abzurufen:

```
For i = LBound(varPoints) To UBound(varPoints)
Next
```

Im zweiten Parameter beider Funktionen, die optional angegeben werden können, legen Sie die Dimension des Felds fest, dessen Index bei mehrdimensionalen Arrays bestimmt werden soll.

```
Dim varPoints(3, 10, 11)
Debug.Print UBound(varPoints, 1) 'gibt 3 aus
Debug.Print UBound(varPoints, 2) 'gibt 10 aus
Debug.Print UBound(varPoints, 3) 'gibt 11 aus
```

2.1.9 Datenfelder löschen

Für das Löschen von Datenfeldern stellt VBA die Anweisung `Erase` zur Verfügung. Bei statischen Arrays wird der Inhalt der einzelnen Felder zurückgesetzt, Zahlenwerte werden auf 0 und Zeichenfolgen auf eine Leerzeichenfolge (" ") gesetzt. *Variant*-Variablen erhalten den Wert *Empty*. Wenn Sie `Erase` auf dynamische Arrays anwenden, wird das gesamte Feld gelöscht, so dass Sie die Variable erneut mit einer `ReDim`-Anweisung dimensionieren müssten, falls Sie diese weiterverwenden möchten.

```
Erase VarPoints
```


2.1.10 Deklarationen erzwingen

Egal um welche Deklaration es geht, Sie sollten VBA zwingen, alle Variablen zu deklarieren. Zu diesem Zweck muss nur am Anfang jedes Formulars sowie in der ersten Zeile eines jeden Moduls der Ausdruck

Option Explicit

stehen. Sie können diese Anweisung automatisch bei jeder Erstellung neuer Formulare und Module durch das System hinzufügen lassen. Dazu müssen Sie nur die Option VARIABLEN-DEKLARATION ERFORDERLICH im Optionsdialog der Entwicklungsumgebung aktivieren.

2.1.11 Prüfung von Datentypen

Wenn Sie bei der Entwicklung von VBA-Anwendungen den Datentyp *Variant* verwenden, was ziemlich oft der Fall sein wird, dann ist es oftmals notwendig, die Variable auf den Datentyp zu prüfen. Ein Beispiel hierfür ist die Abfrage von Koordinaten zur Ermittlung von Eingabedaten grafischer Zeichnungsobjekte. Wenn Sie eine Variable mit dem Datentyp *Variant* deklarieren, diese aber noch nicht mit Daten gefüllt wurde, enthält sie den Wert *Empty* (Leer). Erst durch die Zuweisung eines Werts wird sie einem bestimmten Datentyp entsprechen. Wird z.B. die *GetPoint*-Funktion, die der Abfrage einer dreidimensionalen Koordinate dient, durch den Benutzer abgebrochen, bleibt der Inhalt der Variable *Empty*. Durch Prüfung des zurückgegebenen Datentyps könnte dann auf die Benutzereingabe reagiert und entsprechend verzweigt werden.

```
Dim varPoint As Variant
varPoint = ThisDrawing.Utility.GetPoint(, "Punkt wählen:")
If IsEmpty(varPoint) = True Then
Else
End If
```

In diesem Beispiel würde die Abfrage *IsEmpty()* den Wert *FALSE* zurückgeben, wenn der Benutzer eine Koordinate gewählt hat, oder den Wert *TRUE*, wenn der Benutzer die Auswahl der Koordinaten abgebrochen hat, da die Variable *varPoint* ja immer noch *Empty*, also leer ist. Sie können eine Variable auch manuell leeren, indem Sie ihr den Wert *Empty* zuweisen. Dies kann dann vorkommen, wenn Sie z.B. mehrere Koordinaten in einer Schleife ermitteln lassen, z.B. für die Koordinatenliste einer Polylinie:

```
Dim varPoint As Variant
Do
    varPoint = Empty
    varPoint = ThisDrawing.Utility.GetPoint(, "Punkt wählen:")
    If IsEmpty(varPoint) = True Then
        Exit Do
    Else
    End If
Loop
```

Die Abfrage `IsNull()` entspricht in etwa der `IsEmpty()`-Funktion. Allerdings gibt diese nur *TRUE* zurück, wenn die Variable überhaupt nicht belegt ist. Das ist der Fall, wenn Sie der Variablen den Wert Null zuweisen.

```
Dim varPoint As Variant
varPoint = Null
varPoint = ThisDrawing.Utility.GetPoint(, "Punkt wählen:")
If IsEmpty(varPoint) = True Then
Else
End If
```

Funktion	Beschreibung
<i>IsEmpty()</i>	Gibt <i>TRUE</i> zurück, wenn eine Variable deklariert wurde, aber keine Daten enthält.
<i>IsNull()</i>	Gibt <i>TRUE</i> zurück, wenn eine Variable den Wert <i>Null</i> enthält.
<i>VarType()</i>	Gibt einen Wert zurück, der den Datentyp der Variablen repräsentiert. Für die Rückgabe der Funktion sind einige Konstanten in der Konstantenauflistung <i>vbVarType</i> zusammengefasst.
<i>IsObject()</i>	Gibt <i>TRUE</i> zurück, wenn die Variable einen Objektverweis enthält.
<i>IsError()</i>	Gibt <i>TRUE</i> zurück, wenn es sich bei dem angegebenen Ausdruck um einen Fehlerwert handelt.
<i>IsNumeric()</i>	Gibt <i>TRUE</i> zurück, wenn es sich bei dem angegebenen Ausdruck um einen numerischen Wert handelt.
Beispiel für die Verwendung von <i>IsNumeric()</i>	
<i>IsNumeric</i> („123“)	Gibt <i>TRUE</i> zurück.
<i>IsNumeric</i> („A23“)	Gibt <i>FALSE</i> zurück.
<i>IsNumeric</i> (23)	Gibt <i>TRUE</i> zurück.
<i>A = 12.124</i>	
<i>IsNumeric</i> (A)	Gibt <i>TRUE</i> zurück.
<i>IsDate()</i>	Gibt <i>TRUE</i> zurück, wenn es sich bei dem angegebenen Ausdruck um einen Datumswert handelt.

Tabelle 2.3: Funktionen zur Prüfung von Variant-Variablen

2.1.12 Datentypumwandlung

Häufig werden Sie Daten einer Variablen in den Datentyp einer anderen Variablen konvertieren müssen, so z.B. bei der Übergabe von Parametern an bestimmte AutoCAD-Funktionen. Normalerweise geschieht dies durch VBA automatisch. Wenn Sie z.B. den Wert einer als Fließkommazahl deklarierten Variable in eine Variable des Typs Integer übergeben, was ja einer Ganzzahl entspricht, dann werden die Nachkommastellen entsprechend auf- oder abgerundet.

```
Dim I As Integer
I = 2.5
```

I enthält jetzt den Wert 2.

```
Dim I As Integer
I = 2.51
```

I enthält jetzt den Wert 3.

Wenn Sie Fließkommawerte an Zeichenfolgenvariablen übergeben, müssen Sie die Ländereinstellung des Systems berücksichtigen. Das Dezimalzeichen, das dort eingestellt wurde, wird auch als Dezimaltrennzeichen in der Variable benutzt. Da die Verwendung des Trennungszeichens in AutoCAD eine besondere Rolle spielt, sollten Sie auch nur die Umwandlungsfunktionen der AutoCAD-Objektbibliothek verwenden, um Fehler bei der Interpretation solcher Zahlen zu vermeiden (siehe auch *Kapitel 6.3*).

VBA stellt außerdem Umwandlungsfunktionen zur Verfügung, wodurch die Verwendung zusätzlicher Variablen entfallen kann. Wenn z.B. eine Funktion oder Prozedur die Übergabe eines Werts vom Typ *Long* verlangt und Sie bereits den Wert in einer *Integer*-Variablen gespeichert haben, können Sie mit der entsprechenden Umwandlungsfunktion den korrekten Datentyp übergeben.

```
Dim I As Integer
Sub EingabeLng(Zahl As Long)
End sub

EingabeLng i 'Löst einen Fehler aus
EingabeLng CLng(i) 'Diese Übergabe funktioniert
```

Für alle Datentypen existieren solche Umwandlungsfunktionen, den Datentyp *Object* einmal ausgenommen.

Funktion	Rückgabetyt	Datentypen des Parameters
<i>CBool</i>	<i>Boolean</i>	Als Eingabetyp der Funktionen können Sie Zeichenfolgen oder numerische Werte verwenden, die dem Bereich des zu erzeugenden Datentyps entsprechen.
<i>CByte</i>	<i>Byte</i>	
<i>CCur</i>	<i>Currency</i>	
<i>CDate</i>	<i>Date</i>	Beispielsweise.
<i>CDbl</i>	<i>Double</i>	<i>CBool</i> („Falsch“), <i>CBool</i> (-1) oder <i>CBool</i> („1“)
<i>CDec</i>	<i>Decimal</i>	<i>CInt</i> („100“)
<i>CInt</i>	<i>Integer</i>	<i>CDate</i> („07.01.200 3“), <i>CDate</i> („07.01.2003 12:47:11“)
<i>CLng</i>	<i>Long</i>	<i>Byte</i> („0“), <i>CByte</i> („255“)
<i>CSng</i>	<i>Single</i>	
<i>CVar</i>	<i>Variant</i>	
<i>CStr</i>	<i>String</i>	

Tabelle 2.4: Umwandlungsfunktion für Datentypen

2.1.13 Einschränkungen bei der Verwendung von Zahlen

Bei der Berechnung von Ganzzahlen gilt es einige Besonderheiten zu beachten. Wenn Sie z.B. zwei *Integer*-Werte miteinander multiplizieren und deren Ergebnis größer als der zulässige Wert für *Integer* ist, dann löst allein die Multiplikation einen Überlauffehler aus, auch wenn die Variable, die das Ergebnis aufnehmen soll, mit einem Datentyp deklariert wurde, dessen Wertebereich eigentlich groß genug sein sollte.

```
Dim Ergebnis As Long
Ergebnis = 250 * 300
```

Das liegt daran, dass die Operation in dem Wertebereich durchgeführt wird, die dem Term, mit dem größten Datentyp entspricht. Da beide Werte vom Typ *Integer* sind, wird die Multiplikation auch im *Integer*-Bereich durchgeführt. Um diesen Fehler zu umgehen, müssen Sie also nur einen der Terme dem Ausgabedatentyp anpassen. VBA stellt hierfür einige Datentypkennzeichner, die Sie der Zahl hinzufügen müssen, zur Verfügung.

Datentypkennzeichen	
&	Long
%	Integer
#	Double
@	Currency
!	Single

Tabelle 2.5: Datentypenkennzeichen

```
Dim Ergebnis As Long
Ergebnis = 250 * 300&
```

Sie könnten auch eine der Konvertierungsfunktionen verwenden, die zuvor besprochen wurden.

```
Dim Ergebnis As Long
Ergebnis = 250 * CLng(300)
```

2.1.14 Benutzerdefinierte Datentypen

Benutzerdefinierte Datentypen werden aus den Standarddatentypen von VBA definiert und kommen oft in Zusammenhang mit Datensätzen zum Einsatz. Diese Datentypen können nur auf Modulebene definiert werden. Wenn Sie Variablen mit einem benutzerdefinierten Datentyp deklarieren, gelten für diese Variablen die gleichen Bedingungen, was z.B. die Reichweite anbelangt, wie für andere, die einem Standarddatentyp entsprechen.

Die Definition eines benutzerdefinierten Datentyps wird immer mit dem Schlüsselwort *Type* eingeleitet und mit *End Type* abgeschlossen. Dem Schlüsselwort *Type* folgt der Name des neuen Datentyps. Innerhalb dieses Blocks können Sie die einzelnen Datenfelder mit den Standarddatentypen in Form von *Name As Datentyp* definieren. Die Anzahl der Datenfelder ist dabei unbegrenzt.

```
Type User
  UserName As String * 20
  UserID As Long
```

```
    Description As String
End Type

Sub user_macro()
Dim u As User
u.UserName = "Admin"
u.UserID = 200
u.Description = "Administrator"
Debug.Print u.UserName
End Sub
```

Wie bereits erwähnt, werden benutzerdefinierte Datentypen oft für Datensätze verwendet. Die einfachste Form eines solchen Datensatzes ist das Lesen und Schreiben in Dateien, die im Random-Modus geöffnet wurden. Allerdings müssen Sie hier beachten, dass nur Zeichenfolgen fester Länge verwendet werden können. Sie müssen also die Definition des benutzerdefinierten Datentyps folgendermaßen anpassen.

```
Type User
    UserName As String * 20
    UserID As Long
    Description As String * 255
End Type
```

Die erste Prozedur deklariert zunächst eine Variable mit dem benutzerdefinierten Datentyp. Als Nächstes wird eine Datei im Random-Modus geöffnet, wobei die Größe des zu schreiben- den Datensatzes durch die `Len`-Funktion ermittelt und als Parameter angehängt wird. Nachdem dann einige Daten der Variablen zugeordnet wurden, schreibt die `Put`-Anweisung die Daten der Variablen in die Datei.

```
Sub Write_User()
Dim u As User
Open "C:\VBA\Users.db" For Random As #1 Len = Len(u)
u.UserName = "Admin"
u.UserID = 200
u.Description = "Administrator"
Put #1, , u
Close
End Sub
```

Die zweiten Prozedur liest die Daten mithilfe der `Get`-Anweisung aus der Datei und gibt deren Inhalt im Debug-Fenster aus.

```
Sub Read_User()
Dim u As User
Open "C:\VBA\Users.db" For Random As #1 Len = Len(u)
Get #1, , u
Close
Debug.Print u.UserName
Debug.Print u.UserID
Debug.Print u.Description
End Sub
```

2.1.15 Verwenden von Konstanten und Konstantenauflistungen

Konstanten dienen der Aufnahme von Werten, die sich im Programmablauf nicht verändern. Sie können so bei immer wiederkehrenden Zahlen oder Zeichenfolgen den Konstantennamen zur Zuweisung eines Werts verwenden. Eine Konstante wird durch das Schlüsselwort `Const` definiert:

```
Const PI = 3.14159265358979
```

Statt jedes Mal den Wert für Pi einzugeben,

```
Umfang = 2 * 3.14159265358979 * 10
```

könnte eine Konstantendeklaration die Arbeit erleichtern:

```
Umfang = 2 * PI * 10
```

Sie können bei der Definition der Konstanten zusätzlich deren Datentyp festlegen.

```
Const PI As Double = 3.14159265358979
```

In der AutoCAD-Typenbibliothek sind bereits eine ganze Reihe von Konstanten definiert, wobei diese in Auflistungen zusammengefasst wurden. Diese Methode ist insofern von Nutzen, als dass sie eine gewisse Ordnung in die Zuweisung von Eigenschaften und Methoden bringt.

Lassen Sie uns das an einem kleinen Beispiel verdeutlichen. Bei der Zuordnung von Farben zu einem grafischen AutoCAD-Objekt erfolgt die Angabe der Farbe in Form einer Nummer. Zusätzlich sind die Grundfarben auch als Konstanten einer Auflistung vorhanden, so dass die Zuordnung der Farbwerte über einen Namen vorgenommen werden kann.

```
Object.Color = 6
```

Da die Konstante *acMagenta* den Wert 6 enthält, können Sie diese für die Farbzuzuweisung verwenden.

```
Object.Color = acMagenta
```

Solche Auflistungen lassen sich außerdem als Datentyp von Variablen, Parametern oder Rückgabewerten für Prozeduren deklarieren. Das hat den Vorteil, dass die Konstanten, die in der entsprechenden Auflistung definiert wurden, nach der Eingabe der Variablen aufgelistet werden.

Wenn Sie eigene Konstantenauflistungen definieren wollen, dann können Sie dafür die `Enum`-Anweisung benutzen. Diese beginnt, ähnlich wie die `Type`-Anweisung, mit `Enum Name` und endet mit `End Enum`. Innerhalb dieses Blocks können Sie die Konstanten definieren, die in der Auflistung zusammengefasst werden sollen.

```
Enum UserLevel  
    UserLevel0 = 0  
    UserLevel1 = 1  
    UserLevel2 = 2  
End Enum
```

Eine Einschränkung müssen Sie allerdings in Kauf nehmen. Die Konstanten, die mit der Konstantenauflistung definiert werden, sind immer vom Datentyp *Long*. Sie können also keine Zeichenfolgen oder Fließkommazahlen zuweisen.

Tipp

Wenn Sie den einzelnen Konstanten der Auflistung keine Werte zuweisen, nimmt `Enum` dies automatisch vor. Die erste Konstante würde den Wert 0 erhalten, alle weiteren immer den Wert des Vorgängers + 1. Wenn Sie beispielsweise für die erste Konstante den Wert 100 vergeben und für die zweite keinen, dann erhält die zweite Konstante automatisch den Wert 101.

2.2 Prozeduren in Modulen, Formularen und Klassenmodulen

Der Vorteil einer prozeduralen Programmiersprache wie VBA ist, dass Sie mehrere Anweisungen zu einer Prozedur zusammenfassen und diese Prozedur dann anhand eines Namens immer wieder ausführen lassen können. *Sub* oder *Function* sind Beispiele für solche Prozeduren. In jeder Prozedur können Sie beliebige Anweisungen zusammenstellen und zusätzlich weitere Prozeduraufrufe innerhalb einer Prozedur ausführen. Sogar so genannte Rekursionen (Aufrufe durch sich selbst) sind möglich. Mit Ausnahme der Ereignisprozeduren, die später noch besprochen werden, müssen die Prozeduren namentlich ausgeführt werden.

Der vorliegende Abschnitt befasst sich ausführlich mit dem Aufbau und der Aufgabe solcher Prozeduren.

2.2.1 Prozedurdefinition

Da Prozeduren zusammenhängende Blöcke von Anweisungen darstellen, müssen diese Blöcke auch begrenzt werden. Das funktioniert so ähnlich wie mit der *Type*-Anweisung oder *Enum*. Daraus lässt sich schließen, dass der Anweisungsblock mit *Sub...End Sub* oder *Function...End Function* zur Prozedur zusammengefasst wird.

```
Public Sub Prozedur1()
```

```
End Sub
```

```
Public Sub Prozedur2(Parameter1)
```

```
End Sub
```

```
Public Function Prozedur3()
```

```
End Function
```

```
Public Function Prozedur4() As Logical
```

```
End Function
```

Sie sehen hier vier verschiedene Arten von Prozeduren: eine Routine ohne jegliche Parameter, eine mit einem Parameter sowie zwei Funktionen. Mit diesen Beispielen soll Folgendes demonstriert werden: Sie haben jetzt zwar Ihrem Modul vier Prozeduren hinzugefügt, wenn Sie aber den VBA-Manager in AutoCAD starten (Befehl: `vbarun`), wird in der Makroliste lediglich ein Makro angezeigt und zwar die erste Routine `Prozedur1`. Das hat folgende Bewandnis:

- Es werden im VBA-Manager nur Sub-Routinen aufgelistet und keine Funktionen.
- Es werden keine Sub-Routinen angezeigt, die einen oder mehrere Parameter besitzen.

Wichtig hierbei ist wieder die Bestimmung des Namens. Für dessen Vergabe gelten dieselben Regeln wie für Variablennamen:

- Maximal 255 Zeichen lang
- Keine Sonderzeichen außer einem Unterstrich “_” oder den Umlauten
- Keine Verwendung von Schlüsselwörtern wie `Set`, `Sub`, `Function`, `Dim`, `Public` usw.

Außerdem muss der Name einer Prozedur innerhalb eines Containers eindeutig sein, was bedeutet, dass Sie innerhalb eines Moduls, Klassenmoduls oder Formulars einen Prozedurnamen nur einmal verwenden dürfen. Daraus wiederum folgt, dass Prozeduren gleichen Namens innerhalb unterschiedlicher Module durchaus gestattet sind. Beim Aufruf müssen die Container (Modulname) der Prozedur vorangestellt werden.

Tipp

Sie können für Ihre Prozeduren Kurzbeschreibungen verfassen, die im Informationsfeld des Objektkatalogs angezeigt werden. Wechseln Sie hierfür in den Objektkatalog (`[F2]`) und rufen Sie das Kontextmenü der entsprechenden Prozedur auf. Wenn Sie in diesem Menü den Punkt `EIGENSCHAFTEN` auswählen, können Sie im folgenden Dialog einen entsprechenden Beschreibungstext für die Prozedur festlegen.

2.2.2 Sub oder Function?

Nun stellt sich die Frage, wann verwende ich *Sub* und wann *Function*? Wie bereits angesprochen, unterscheiden sich die beiden Prozedurarten zunächst dadurch, dass eine *Sub*-Prozedur auch als Makro aufgerufen werden kann, wenn diese in einem Modul erstellt wurde, eine *Function* dagegen nicht. Der *Function*-Prozedurtyp wird eigentlich immer dann verwendet, wenn Sie ein Rückgabergebnis einer Prozedur erwarten. Zwar können Prozeduren ebenfalls Werte zurückliefern, sie tun dies aber auf eine andere Art und Weise, was Sie im Abschnitt 2.2.6 nachprüfen können. Das Ergebnis der Prozedur wird über den Namen der *Function* zurückgeliefert und muss innerhalb der Funktion dem Namen zugeordnet werden. Der Datentyp der dabei zum Einsatz kommt, kann wie bei Variablen zugewiesen werden – entweder durch die Benutzung der Datentypkennzeichnung

```
Function Add_Func#(A As Double, B As Double)
    Add_Func = A + B
End Function
```

oder durch das `As` Datentyp-Schlüsselwort


```
Function Add_Func(A As Double, B As Double) As Double
    Add_Func = A + B
End Function
```

Bei dieser einfachen Funktion werden die beiden Parameter A und B addiert und das Ergebnis an den Funktionsnamen übergeben. Da eine Funktion ein Ergebnis liefert, ergibt sich der Aufruf der Funktion aus anderen Prozeduren wie folgt:

```
Sub Add_Sub()
    Debug.Print Add_Func(17.5, 20.3)
End Sub
```

Diese Prozedur ruft die Funktion *Add_Func* auf und gibt das zurückgegebene Ergebnis im Debug-Fenster aus.

2.2.3 Aufruf und Reichweite von Prozeduren

Die Reichweite (ich nenne es mal so) von Prozeduren ist abhängig davon, welchen Zusatz Sie den Schlüsselwörtern *Sub* bzw. *Function* vorangestellt haben.

Zum einen gibt es das Wort *Private*, was nichts anderes bedeutet, als dass diese Prozedur nur in Ihrem Container (z.B. Formular) aufrufbar ist. Verwenden Sie allerdings das Schlüsselwort *Public*, können Sie diese Prozedur von jedem Ort aus aufrufen (nicht nur aus Ihrem Projekt, sondern sogar aus anderen Projekten).

Tipp

Um auf die Prozeduren anderer VBA-Projekte zugreifen zu können, müssen Sie einen Verweis auf das entsprechende VBA-Projekt in dem Objektkatalog F2 erstellen.

Lassen Sie uns das einmal anhand eines Beispiels nachprüfen. Um es recht anschaulich zu gestalten, eignet sich besonders eine Ereignisprozedur, die normalerweise nur im Formularcode aufrufbar ist. Fügen Sie zu diesem Zweck einmal einem neuen Projekt ein Formular hinzu und doppelklicken Sie auf das Formular. Es erscheint das Codefenster des Formulars, in dem bereits eine Prozedur existiert.

```
Private Sub UserForm_Click()
End Sub
```

Wechseln Sie nun in das Klassenmodul *ThisDrawing* und geben Sie die Zeile *UserForm1.use* ein. In der eingeblendeten Liste werden Sie die Prozedur *UserForm_Click* nicht finden.

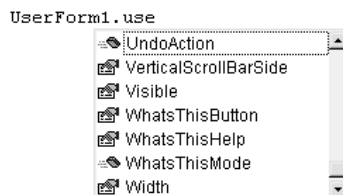


Abbildung 2.2: Lokale Deklaration

Im Listenfeld, das alle verfügbaren Eigenschaften und Methoden anzeigt, ist die Prozedur `UserForm_Click` nicht zu finden. Das liegt daran, dass deren Gültigkeit nur auf den Containercode begrenzt wurde, sozusagen privat ist. Kehren Sie nun in den Formularcode zurück und ändern Sie das Schlüsselwort `Private` in `Public`. Die Prozedur wird dadurch öffentlich und ist auch aus anderen Modulen heraus abrufbar.

```
Public Sub UserForm_Click
```

```
End Sub
```

Diese Vorgehensweise ist zwar nicht üblich, aber durchaus machbar. So haben Sie die Möglichkeit, die Ereignisprozedur von außerhalb „manuell“ auszulösen. Gehen Sie nun wieder zurück in das Klassenmodul *ThisDrawing* und geben Sie erneut die Zeile `UserForm1.Use` ein. Jetzt wird die Prozedur `UserForm_Click` auch in der eingblendeten Liste angezeigt.

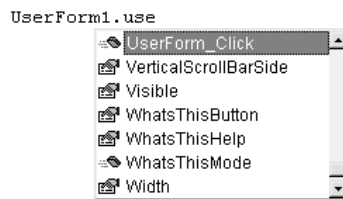


Abbildung 2.3: Globale Deklaration

Dass die Routine `UserForm_Click` im Listenfeld angeboten wird, bedeutet, dass sie jetzt von Modulebene aus erreichbar ist. Sie müssen lediglich den jeweiligen Container voranstellen und mittels eines Punkts trennen und falls erforderlich die notwendigen Parameter mit übergeben.

Tipp

Sie können diese Definitionsform auch mit Variablen vornehmen, so dass diese über Ihren Container erreichbar sind. Beispielsweise eine im Formular öffentlich deklarierte Variable `OK`, die das erfolgreiche Schließen des Formulars speichern könnte. Beispiele folgen!

2.2.4 Lebensdauer von Variablen und Konstanten

Entscheidend dafür, wo und wie lange Sie eine Variable am Leben halten möchten, ist, wie und wo Sie diese Variable deklarieren. Hierfür gibt es einige einfache Regeln, die Sie beachten müssen, damit Ihnen keine Variable unabsichtlich abhanden kommt.

Variablen und Konstanten, die innerhalb eines `Sub...End Sub`- oder eines `Function...End Function`-Bereichs deklariert wurden, gelten auch nur in diesem Block. Sie können auf diese Variablen von anderen Prozeduren aus nicht zugreifen.

```
Public Sub Prozedur1()
```

```
Dim X, Y, Z
```

```
Const A = 1
```

```
'Die Variablen X, Y, Z und A gelten nur innerhalb dieser Prozedur
```

```
End Sub
```

```
Public Sub Prozedur2()
'Die Variablen X, Y, Z und die Konstante A sind hier unbekannt
End Sub
```

Variablen und Konstanten, welche innerhalb eines Moduls (deklariert vor der ersten Prozedur) deklariert wurden, sind global in allen Modulen, Formularen und Klassenmodulen gültig.

```
{***** Modul1 *****}
```

```
Dim X, Y, Z
Const A = 1
Public Sub Prozedur1()
'Die Variablen X, Y, Z sowie A sind hier verfügbar
End Sub
}
```

```
{***** Modul2 *****}
```

```
Public Sub Prozedur1()
'Die Variablen X, Y, Z sowie A sind hier ebenfalls verfügbar
End Sub
}
```

```
{***** UserForm1 *****}
```

```
Private Sub TextBox1_Change()
'Die Variablen X, Y, Z und A sind hier ebenfalls verfügbar
End Sub
}
```

Variablen, die innerhalb eines Formulars (deklariert vor der ersten Prozedur) deklariert wurden, sind in diesem Formular und allen Routinen des Formulars verfügbar. Um die Variablen auch außerhalb des Formulars zur Verfügung zu stellen, müssen Sie die entsprechende Variable öffentlich deklarieren.

```
{***** UserForm1 *****}
```

```
Dim X, Y, Z
Public OK
Public Sub Prozedur1()
'Die Variablen X, Y, Z und OK sind hier verfügbar
End Sub
}
```

```
{***** Modul1 *****}
```

```
Public Sub Prozedur1()
'Die Variablen X, Y, Z sind hier nicht verfügbar
```

```
'Die Variable OK ist hier verfügbar, wenn sie entsprechend 'ausgewertet wird
A = UserForm1.OK
End Sub
```

```
}
```

```
{***** UserForm2*****}
```

```
Public Sub Prozedur1()
```

```
'Die Variablen X, Y, Z sind hier nicht verfügbar
```

```
'Die Variable OK ist hier verfügbar, wenn sie entsprechend 'ausgewertet wird
```

```
A = UserForm1.OK
```

```
End Sub
```

```
}
```

Prozeduren, die in Modulen deklariert wurden, sind automatisch aus allen anderen Prozeduren aufrufbar. In Formularen deklarierte Prozeduren sind nur dann zugänglich, wenn sie öffentlich deklariert wurden.

```
{***** UserForm1 *****}
```

```
Public Sub Prozedur1()
```

```
'Prozedur2 ist hier verfügbar
```

```
End Sub
```

```
Private Sub Prozedur2()
```

```
'Prozedur1 ist hier verfügbar
```

```
End Sub
```

```
}
```

```
{***** Modul1*****}
```

```
Public Sub Prozedur1()
```

```
'Prozedur2 des Formulars UserForm1 ist hier nicht verfügbar
```

```
'Prozedur1 ist hier verfügbar, wenn sie entsprechend angesprochen wird
```

```
UserForm1.Prozedur1
```

```
End Sub
```

```
}
```

```
{***** UserForm2*****}
```

```
Public Sub Prozedur1()
```

```
'Prozedur2 des Formulars UserForm1 ist hier nicht verfügbar
```

```
'Prozedur1 ist hier verfügbar, wenn sie entsprechend angesprochen wird
```

```
UserForm1.Prozedur1
```

```
End Sub
```

```
}
```

Tipp

Bei Konstanten, Variablen und Prozeduren, die in verschiedenen Modulen unter gleichem Namen vorkommen, hat immer die Prozedur der aktuellen Ebene den Vorrang.

2.2.5 Prozeduren vorzeitig verlassen

Manchmal kann es ganz nützlich sein, eine Prozedur vorzeitig zu verlassen, d.h. vor Erreichen von *End Sub* oder *End Function*. Sie können in diesem Fall die Anweisung *Exit Sub* für Unterprogramme und *Exit Function* für Funktionen verwenden.

```
Function Test_Func(A As Double, B As Double) As Double
    If A < 0 Then Test_Func = A: Exit Function
    Test_Func = B
End Function
```

Sie müssen lediglich darauf achten, dass beim vorzeitigen Verlassen von Funktionen das Ergebnis dem Namen zugewiesen wird, bevor Sie *Exit Function* aufrufen. Wenn Sie dies nicht tun, gibt die Funktion je nach Datentyp eine Leerzeichenfolge (" "), 0, *Empty* oder *FALSE* zurück. Für *Sub*-Prozeduren gilt dies nicht, da diese keinen Wert an sich zurückgeben.

2.2.6 Parameterübergabe an Prozeduren/Funktionen

Wie zuvor bereits angedeutet, können Prozeduren oder Funktionen ein oder mehrere Parameter mitgegeben werden. Diese Parameter, die nichts anderes als prozedurinterne Variablen darstellen, werden direkt nach der Deklaration der Prozedur bzw. Funktion in Klammern angegeben. Bei der Angabe mehrerer Parameter müssen diese durch ein Komma voneinander getrennt werden, wie Sie es bereits von der Deklaration von Variablen her kennen. Allerdings lassen Sie hier die Schlüsselwörter *Dim*, *Public* ... weg.

```
Public Sub Prozedur2(ByVal Par1 As String, Par2, _
    Par3 As AcadLayer, Optional Par4 As Double)
End Sub
```

Wie Sie bereits richtig beobachtet haben, habe ich alle Parameter unterschiedlich gestaltet, um Ihnen noch einige zusätzliche Schlüsselwörter näher zu bringen. Mit dem Schlüsselwort *ByVal* können Sie Variablen an die Prozedur übergeben, ohne dass der Wert der Variable zum Abschluss der Prozedur verändert zurückgegeben wird. Stellen Sie die Parametervariable ohne Datentyp zur Verfügung, dann entspricht diese *Variant* und Sie können Variablen jedes Datentyps an die Prozedur übergeben, andernfalls muss der übergebene Parameter exakt dem Datentyp entsprechen, der festgelegt wurde.

Übergabe von Arrays

An Prozeduren können aber auch Arrays übergeben werden, was für die Arbeit mit AutoCAD recht nützlich ist, da jedes Objekt in einem dreidimensionalen Raum liegt und somit immer X-, Y- und Z-Koordinaten besitzt. Diese in einem Array zusammenzufassen und der entsprechenden Prozedur zu übergeben, macht mehr Sinn, als für jede Koordinate eine eigene Variable festzulegen. Bei der Festlegung der Parameter stehen Ihnen nun zwei Möglichkeiten zur

Verfügung, diese Felder anzugeben. Entweder Sie kennzeichnen die Variable als Array, indem Sie ein Klammersymbol an den Variablenamen anhängen und dann den entsprechenden Datentyp festlegen:

```
Function Coord_Array(Coords() As Double)
End Function
```

Oder Sie deklarieren einfach die Variable mit dem Datentyp *Variant*:

```
Function Coord_Array(Coords)
End Function
```

Der Unterschied zwischen den beiden Formen besteht lediglich im Datentyp der Variablen. Während das Array im ersten Fall dem Datentyp *Double* entspricht, sind im zweiten Fall nur die einzelnen Felder vom Datentyp *Double*, die Variable selbst ist vom Datentyp *Variant*.

Um eine Liste (Array von drei Koordinaten) als Ergebnis einer Funktion zurückzugeben, deklarieren Sie das Array innerhalb der Funktion und füllen dieses mit entsprechenden Werten.

```
Public Function GetInsPoint(X, Y, Z) As Variant
Dim InsPoint(2) As Double
InsPoint(0) = X
InsPoint(1) = Y
InsPoint(2) = Z
GetInsPoint = InsPoint
End Function
```

Als Ergebnis der Funktion erhalten Sie nun ein Array, gefüllt mit den drei Koordinaten. Sie können natürlich auch ein oder mehrere Arrays als Parameter an eine Funktion übergeben. Als Beispiel soll hier die Ermittlung des Abstands zwischen zwei Punkten dienen.

```
Public Function GetDist(FromPt, ToPt) As Double
    'FromPt(0) = X1
    'FromPt(1) = Y1
    'ToPt(0) = X2
    'ToPt(1) = Y2
    GetDist = Sqr((FromPt(0) - ToPt(0))^ 2 + (FromPt(1) - ToPt(1))^ 2)
End Function
```

2.2.7 Optionale Parameter

Möglicherweise wollen Sie bei der Erstellung von Prozeduren mehrere Parameter angeben, obwohl Sie für die erfolgreiche Ausführung nicht unbedingt alle benötigen. Die restlichen Parameter sind eigentlich nur zur genaueren Angabe von bestimmten Eigenschaften innerhalb der Prozedur gedacht. Als Beispiel möchte ich die Speicherung der AutoCAD-Zeichnung unter einem bestimmten Namen anführen. Diese benötigt lediglich den Namen der Zeichnung zur korrekten Ausführung. Zusätzlich können Sie aber den Typ der Datei angeben. Wenn Sie diesen Parameter nicht belegen, dann wird die Funktion dennoch ausgeführt. Das liegt daran, dass der zusätzliche Parameter als optionaler (kann, muss aber nicht) Parameter angegeben wurde. Dies wird durch Verwendung des Schlüsselworts *Optional* erreicht. Jeder Parameter, dem Sie dieses Schlüsselwort voranstellen, muss beim Aufruf der Prozedur nicht angegeben werden.

```
Sub SaveFile(FileName As String, Optional FileType As Integer)
End Sub
```

Bleibt natürlich die Frage, wie man ermittelt, wann ein Parameter verwendet wurde und wann nicht. Die Überprüfung des Inhalts der Variablen macht ja keinen Sinn, da diese größtenteils mit Datentypen deklariert wurden und somit einen Initialisierungswert besitzen. Hier hilft die Funktion *IsMissing* weiter. Diese prüft, ob beim Aufruf der Prozedur der entsprechende Parameter angegeben wurde. Wurde der Parameter nicht verwendet, gibt die Funktion den Wert *TRUE* zurück, andernfalls den Wert *FALSE*.

```
Sub SaveFile(FileName As String, Optional FileType As Integer)
If IsMissing(FileType) = False Then
End If
End Sub
```

Tipp

Wenn Sie optionale Parameter verwenden, müssen Sie beachten, dass erst alle nicht optionalen Parameter und dann die restlichen optionalen in der Parameterliste angegeben werden. Sobald der erste optionale Parameter erscheint, müssen alle weiteren ebenfalls als optional angegeben werden.

Eine weitere Möglichkeit, Parameter optional anzugeben, ist die Anweisung *ParamArray* vor dem eigentlichen Parameter. Der Vorteil dieser Methode ist, dass nicht nur die optionalen Parameter, die Sie eigentlich nacheinander angeben müssten, zu einem Array zusammengefasst werden, sondern dass Sie eigentlich eine unbegrenzte Anzahl an Parametern festlegen können.

```
Sub Param_Array(ParamArray ParamList() As Variant)
End Sub
```

Die Festlegung des Parameters als *Variant*-Feld ist dabei zwingend erforderlich. Die einzelnen Parameter könnten dann mithilfe einer *For Next*-Schleife ausgelesen werden. Die Obergrenze des Arrays ermittelt dabei wieder die Funktion *UBound()*, womit Sie die Anzahl der Parameter erhalten und so jeden einzelnen auslesen können.

```
Sub Param_Array(ParamArray ParamList() As Variant)
For I = 0 To Ubound(ParamList)
    Parameter = ParamList(i) 'Parameter enthält jetzt den n'ten Parameter der Liste
Next
End Sub
```

oder:

```
Sub Param_Array(ParamArray ParamList() As Variant)
For Each Parameter In ParamList
    'Parameter enthält jetzt den n'ten Parameter der Liste
Next
End Sub
```

Eine gleichzeitige Verwendung beider Methoden der optionalen Parameter ist nicht gestattet, weshalb Sie entweder Parameter mit dem Schlüsselwort *Optional* festlegen oder die Parameterliste mit dem Schlüsselwort *ParamArray* verwenden müssen. Außerdem können die Werte, die Sie mit *ParamArray* übernehmen, nicht verändert werden.

2.2.8 Übergabe von Parametern an Prozeduren

Wenn Sie Prozeduren mit vielen optionalen Parametern aufrufen, dann können Sie die Parameter, die Sie nicht vergeben wollen, direkt mit ihrem Namen ansprechen, statt sie durch Kommas abzutrennen. Daraus ergeben sich zwei verschiedene Varianten des Prozeduraufrufs. Ein kleines Beispiel wird dies am besten verdeutlichen. Angenommen, Sie wollen den Inhalt von Attributen einer Blockreferenz anpassen. Die einzelnen Attribute des Blocks beinhalten den Raumnamen, die Raumnummer und den Verwendungszweck. Dazu könnten Sie folgende Prozedur definieren:

```
Sub Change_Attributes(BlockRef, Optional Name$, Optional Nummer&, Optional Beschreibung$)
End Sub
```

Wenn Sie z.B. den zweiten und dritten Parameter beim Ausführen der Prozedur nicht belegen wollen, sähe der Aufruf folgendermaßen aus:

```
Change_Attributes BlockRef, , , "Wohnzimmer"
```

Die zweite Möglichkeit wäre, den Parameter `Beschreibung` direkt anzusprechen:

```
Change_Attributes BlockRef, Beschreibung:="Wohnzimmer"
```

Sobald Sie den ersten benannten Parameter verwenden, müssen alle nachfolgenden ebenfalls benannt werden.

2.3 Schleifen, Abfragen und Verzweigungen

Der folgende Abschnitt geht etwas näher auf die Verwendung von Schleifen, Abfragen und Verzweigungen ein. Mit diesen Methoden haben Sie die Möglichkeit, über den Code Entscheidungen treffen zu lassen.

2.3.1 Verzweigungen mit If

Typische Entscheidungen sind dabei „Wenn ... dann ... sonst“. Eine einfache Abfrage würde dabei folgendermaßen aussehen:

```
If BedingungIstErfüllt Then
    Anweisungen für diesen Weg
End If
```

Sie können die gesamte Anweisung auch in einer Zeile schreiben, müssen dann aber die Anweisung `End If` weglassen:

```
If BedingungIstErfüllt Then Anweisungen für diesen Weg
```

Die Bedingung muss dabei einen booleschen Wert erfüllen, also wahr (*TRUE*) oder falsch (*FALSE*) sein. In Frage kommen also Bedingungen wie z.B. `TextBox1.Text = "ACAD"`, `X = 0` oder vielleicht `EOF(1) = True`. Wird die entsprechende Bedingung erfüllt, wird der dazugehörige Bedingungscode ausgeführt.

Eine erweiterte Art der `If`-Anweisung ist der weiterführende Vergleich mittels `Else` bzw. `ElseIf`. Damit sind Sie in der Lage, weitere Abfragen durchzuführen, wenn die erste Bedingung nicht erfüllt wurde.

```
If Bedingung1IstErfüllt Then
    Anweisungen für diesen Weg
ElseIf Bedingung2IstErfüllt Then
    Anweisungen für diesen Weg
ElseIf Bedingung3IstErfüllt Then
    Anweisungen für diesen Weg
Else
    Keine der Bedingungen wurde erfüllt
End If
```

In diesem Beispiel werden alle Bedingungen nacheinander geprüft, bis eine den Status „Bedingung ist erfüllt“ erreicht. Dann wird der entsprechende Code ausgeführt. Sollte keine der Bedingungen erfüllt werden, wird die Anweisung der `Else`-Verzweigung ausgeführt.

Sie können aus dieser Abfrage-Anweisung den Teil mit der `Else`-Anweisung herauslassen, wenn Sie keine abschließende Auswertung vornehmen wollen, oder die `ElseIf`-Anweisungen, wenn Sie nur zwei Bedingungen (entweder – oder) prüfen lassen möchten.

Eine weitere Möglichkeit, einen Vergleich auszuführen, stellt die `IIf`-Funktion dar. Diese Funktion enthält sowohl einen `True`- als auch einen `False`-Part. Das bedeutet, dass der Wert als Ergebnis der Abfrage zurückgegeben wird, der erfüllt wurde.

```
Ergebnis = IIf(Abfrage, BedingungIstErfüllt, BedingungIstNichtErfüllt)
```

Tipp

Da die Funktion `IIf` immer sowohl den Teil `TRUE` als auch den Teil `FALSE` auswertet, kann es zu unerwünschten Nebeneffekten kommen. Tritt z.B. in einer der beiden Bedingungen ein Fehler auf, wird durch die Funktion generell ein Fehler zurückgegeben – auch dann, wenn der erfüllte Teil diesen Fehler nicht erzeugen sollte.

2.3.2 Select Case-Verzweigung

Eine weitere hervorragende Möglichkeit, Abfragen komplexer zu gestalten, ist die `Select Case`-Anweisung. Mit dieser Anweisung können Sie verschiedene Abfragebedingungen kombinieren, ohne den Umweg der logischen Verknüpfung in der `If`-Anweisung gehen zu müssen.

```
Select Case Object.Color
    Case acBlue
        ` Die Objektfarbe ist Blau
    Case acRed
        ` Die Objektfarbe ist Rot
    Case acByLayer, acByBlock
        ` Die Objektfarbe ist VonBlock oder VonLayer
    Case 1 to 7
        ` Die Objektfarbe ist eine der Grundfarben
```

```
Case Is > 7
    ` Die Objektfarbe ist eine Farbe größer 7
Case Else
    ` Die Objektfarbe ist eine andere als die vorhergehenden
End Select
```

Im obigen Beispiel wird die Farbe des AutoCAD-Objekts ausgewertet. Ich habe dabei sowohl die Konstanten (*acBlue*, *acRed* ...) als auch den Farbindex verwendet. Im Prinzip werden die AutoCAD-Farben als Indexwerte im Bereich von 0 bis 256 verwendet. Die Grundfarben wurden schon in der Konstanten-Auflistung *ACAD_COLOR* zusammengefasst.

Die *Select*-Anweisung beginnt immer mit dem *Select Case*-Vergleichswert. Anschließend wird dieser Wert mit den entsprechenden *Case*-Abschnitten verglichen. Wird dieser Vergleich erfüllt, wird der Code, der auf die *Case*-Anweisung folgt, ausgeführt und anschließend mit *End Select* beendet.

2.3.3 For Next-Schleifen

Anders als bei einer Verzweigung wird bei Schleifen ein Code so lange wiederholt, bis die Bedingung für das Beenden der Schleife erfüllt wurde oder die Schleife durch Programmcode vorzeitig beendet wird. Man unterscheidet dabei drei Arten von Schleifen:

- Zählschleifen
- Kopfgesteuerte Schleifen
- Fußgesteuerte Schleifen

Bei Zählschleifen handelt es sich um einen Ablauf, bei dem die maximale Anzahl der Durchläufe vordefiniert ist. Das bedeutet nichts anderes als „nimm jeden Wert von X nach N und addiere eine Zahl hinzu“.

```
For x = 0 To 100
    ` Anweisung
Next
```

Dies ist die einfachste Form einer *For Next*-Schleife. Sie können diese Schleife auch so erweitern, dass der Wert X um zwei Zähler erhöht wird (also 0, 2, 4 ... 100) oder die Schleife rückwärts abläuft.

Zähler um 3 erhöhen:

```
For x = 0 To 100 Step 3
    ` Anweisung
Next
```

Schleife läuft rückwärts:

```
For x = 100 To 0 Step -1
    ` Anweisung
Next
```

Schleife läuft rückwärts und wird um jeweils 3 vermindert:

```
For x = 100 To 0 Step -3
    ` Anweisung
Next
```

Um die Schleife vorzeitig zu verlassen, sobald eine bestimmte Bedingung erfüllt ist, benutzen Sie die Anweisung *Exit For*. Wenn Sie Schleifen ineinander verschachteln, gilt die *Exit For*-Anweisung immer für die aktuelle Schleife. Der Code wird dann in der nächsthöheren Schleife fortgesetzt. Sie müssen allerdings darauf achten, dass Sie keine Zählvariable innerhalb der Schleife unabsichtlich ändern, da sonst die Schleife vorzeitig verlassen werden könnte. Außerdem ist es bei Verschachtelungen von Schleifen nicht erlaubt, eine Zählvariable mehrfach zu verwenden (VBA würde dies auch nicht zulassen).

Verschachtelte Schleife vorzeitig verlassen:

```
For x = 0 To 100
    For i = 0 To 100
        If BedingungIstErfüllt Then
            `Code wird in der For x Schleife fortgesetzt
            Exit For
        End If
    Next
    If BedingungIstErfüllt Then Exit For
Next
```

Eine besondere Form stellt die *For Each Next*-Schleife dar. Diese wiederholt eine Gruppe von Anweisungen für jedes Element eines angegebenen Datenfelds oder einer Auflistung. Diese Schleifen werden Sie häufig in Zusammenhang mit den Objektauflistungen von Layern, Linientypen oder Textstilen usw. verwenden. Angenommen, möchten alle Layer der aktuellen Zeichnung nacheinander abrufen. Für eine *For Next*-Schleife würde das wie folgt aussehen:

```
For i = 0 To ThisDrawing.Layers.Count
    Set ActLayer = ThisDrawing.Layers(i)
Next
```

Ein Verweis auf das jeweilige Layer-Objekt müsste bei jedem Durchlauf an die Objektvariable *ActLayer* übergeben werden. Mit einer *For Each Next*-Schleife hingegen wäre der Ablauf nicht nur etwas kürzer, sondern auch eleganter:

```
For Each ActLayer In ThisDrawing.Layers.Count
Next
```

Wieder enthält die Variable *ActLayer* einen Verweis auf den jeweiligen Layer, nur dass hier nicht extra eine explizite Zuweisung erfolgen muss. Die Schleife wird so lange wiederholt, wie sich Elemente in der Auflistung oder im Datenfeld befinden. Bei jedem Durchlauf wird immer das nächstfolgende Element abgerufen und an die Variable übergeben, die dem Schlüsselwort *Each* unmittelbar folgt. Innerhalb des Schleifenblocks gelten dieselben Regeln wie für eine *For Next*-Schleife.

2.3.4 Do Loop-Schleifen

Nun zur kopfgesteuerten Schleife. Diese Form der Schleife prüft vor jedem Durchlauf des Schleifenkörpers (definiert den Code zwischen Schleifenbeginn und -ende), ob eine bestimmte Bedingung wahr (*TRUE*) oder falsch (*FALSE*) ist. Die Anzahl der Durchläufe ist vor dem Start nicht festgelegt, sowohl bei der kopf- als auch der fußgesteuerten Schleife. Die Schleife wird einfach so oft durchlaufen, bis eine Bedingung zum Beenden der Schleife erfüllt wurde oder diese durch die Anweisung `Exit Do` beendet wurde.

Für diesen Typ Schleife gibt es folgende Formen:

```
While Bedingung
    `Anweisungen
Wend
```

oder:

```
Do While Bedingung
    `Anweisungen
Loop
```

In jeder Schleife wird der Code so lange ausgeführt, bis die Bedingung den Wert *FALSE* erreicht hat. Hat die Bedingung bereits beim Eintritt in die Schleife den Wert *FALSE*, wird diese überhaupt nicht ausgeführt. Ein Beispiel aus der Praxis für die kopfgesteuerte Schleife ist ein Durchlauf durch Dateien oder Datenbanken. Im Folgenden sehen Sie ein Beispiel für den groben Durchlauf durch eine Datei:

```
Open Datei For Input As #1
Do While Not EOF(1)
    Line Input #1, Daten
Loop
Close
```

Sollte die Datei noch leer sein, also das Dateiende ist nach Öffnung der Datei bereits erreicht, wird die Schleife gar nicht durchlaufen. Der Codeablauf wird dann direkt hinter der Schleife fortgesetzt.

Anders verhält es sich bei den fußgesteuerten Schleifen. Diese Schleifen werden erst einmal durchlaufen, bevor die Bedingung zur Beendigung der Schleife geprüft wird. Sie hätten also, entsprechend des vorherigem Beispiels, bereits einmal versucht, eine Zeile aus der Datei zu lesen, bevor geprüft wird, ob der Zeiger bereits am Ende der Datei angelangt ist.

Für diesen Typ Schleife gibt es folgende Formen, die sich nur in der Art der Formulierung unterscheiden:

```
Do
    `Anweisungen
Loop While Bedingung
```

oder:

```
Do
    `Anweisungen
Loop Until Bedingung
```

Bei der ersten Schleifenform wird die Schleife ausgeführt, solange die Bedingung wahr (*TRUE*) ist. In der zweiten Form wird die Schleife ausgeführt, bis die Bedingung wahr (*TURE*) ist.

Sie können die Schleife auch ohne Bedingungen zusammenstellen, wenn Sie die Bedingungen innerhalb der Schleife prüfen und entsprechend die Schleife mit `Exit Do` verlassen.

```
Do
    If BedingungIstErfüllt Then
        `Schleife wird beendet
    Exit Do
End If
Loop
```

Tipp

Da der Benutzer bei jeder Abarbeitung von Schleifen keinerlei Möglichkeit hat, auf andere Ereignisse (Tastendruck, Abbrechen) zu reagieren, ist es erforderlich, den Code innerhalb der Prozedur kurzzeitig zu unterbrechen und dem System die Steuerung zu übergeben. Diese Möglichkeit bietet uns die `DoEvents`-Anweisung. `DoEvents` übergibt die Ablaufsteuerung an das Betriebssystem. Das Programm erhält erst dann die Steuerung wieder zurück, wenn das Betriebssystem alle wartenden Ereignisse verarbeitet hat und alle wartenden Tastenanschläge in der `SendKeys`-Warteschlange abgearbeitet sind.

2.4 Objekte

Die Arbeit mit Objekten stellt den eigentlichen Kernpunkt einer objektorientierten Programmiersprache wie VBA dar. Jedes AutoCAD-Element, angefangen von den einfachsten grafischen Elementen wie Punkte bis hin zur gesamten AutoCAD-Sitzung, kann als Objekt abgerufen werden. Zu verdanken ist dies der AutoCAD-Objektbibliothek, die bereits in Zusammenhang mit dem Objektkatalog (*Kapitel 1.2.2*) erwähnt wurde. Diese Bibliothek stellt für jedes AutoCAD-Objekt eine Klasse zur Verfügung. AutoCAD-Objekte sind *Application* (eine einzelne AutoCAD-Sitzung), *Document* (eine einzelne Zeichnung), *Preferences* (steuert die Einstellung von AutoCAD), *ModelSpace* (verwaltet die Daten des Modellbereichs), *Layer* (einzelner Layer mit allen dazugehörigen Daten), *Polyline* (eine Polylinie) usw.

Alle diese Elemente sind in einer Hierarchie angeordnet, wobei *Application* deren oberstes Objekt darstellt. Über dieses Objekt kann auf die AutoCAD-Sitzung aus allen Modulen, Formularen und Klassenmodulen zugegriffen werden. *Application* enthält wiederum Eigenschaften und Methoden, die es dem Anwender ermöglichen, weitere Objekte der AutoCAD-Sitzung abzurufen oder zu erstellen. Mehr zum Thema *Application*-Objekt finden Sie in *Kapitel 5*.

2.4.1 Objektverweise erstellen

Es gibt zwei Möglichkeiten, eine Objektvariable zu erstellen – entweder durch Erzeugung einer neuen Instanz eines Objekts oder durch Erstellung eines Verweises auf ein bereits vorhandenes Objekt. In beiden Fällen spielt die `Set`-Anweisung die entscheidende Rolle.

Folgende Anweisung erstellt z.B. einen Verweis auf die aktive Zeichnung der AutoCAD-Sitzung:

```
Set ActDoc = Application.ActiveDocument
```

Durch die zusätzliche Verwendung des Parameters *New* wird eine neue Instanz der Klasse *AcadDocument* erzeugt, was gleichzeitig eine neue Zeichnung in AutoCAD erstellt.

```
Dim ActDoc As AcadDocument  
Set ActDoc = New AcadDocument
```

Der Verweis auf das entsprechende Objekt bleibt so lange bestehen, bis entweder die Variable zurückgesetzt wird:

```
Set ActDoc = Nothing
```

ein Verweis auf ein anderes Dokument der Variable zugeordnet wird:

```
Set ActDoc = Application.ActiveDocument
```

oder das Objekt, auf das verwiesen wurde, nicht mehr existiert:

```
Set ActDoc.Close
```

In diesem Fall können Sie nicht mehr auf die Eigenschaften, Methoden und dem Objekt untergeordnete Elemente zugreifen. Die Objektvariable enthält dann lediglich die übergeordneten Verweise, z.B. den Verweis auf das *Application*-Objekt.

Tipp

Die Anweisung `Set` erzeugt *keine* Kopie des Objekts, sondern lediglich einen Bezug zum Ursprungsobjekt. Das bedeutet, dass eine Änderung am Ursprungsobjekt sich auch auf die Objektvariable, die den Verweis gespeichert hat, auswirkt, da es sich ja um dasselbe Objekt handelt.

2.4.2 Objekteigenschaften

Fast jedes AutoCAD-Objekt besitzt bestimmte Eigenschaften, mit denen die Merkmale dieser Objekte (etwa die Zuordnung zu einem Layer, die Farbe oder die ID, mit der es in AutoCAD identifiziert wird) wiedergeben und zum größten Teil geändert werden können. Eigenschaften ähneln vom Gebrauch her Variablen. Sie entsprechen einem bestimmten Datentyp und werden über ihren Container (das Objekt selbst) abgerufen.

Ausgabe des Anwendungsnamens inklusive Verzeichnis im Debug-Fenster:

```
Debug.Print Application.FullName
```

Die Änderung der Eigenschaft eines Objekts wird meist sofort auf dem Bildschirm sichtbar. Wenn Sie z.B. die Farbeigenschaften einer Linie ändern, können Sie das Ergebnis dieser Änderung im Zeichnungsfenster sofort erkennen. Außerdem kann die Änderung einiger Eigenschaften unweigerlich Auswirkungen auf andere Eigenschaften haben.

Eine Änderung des Status des AutoCAD-Anwendungsfensters hat wiederum Auswirkungen auf die Eigenschaften *Height* und *Width*.

```

Debug.Print Application.Height
Debug.Print Application.Width
Application.WindowState = acNorm
Debug.Print Application.Height
Debug.Print Application.Width

```

Objektvariablen

Eine besondere Eigenschaft von Objekten ist der Verweis auf ein weiteres Objekt. Zur übersichtlicheren Erstellung des Quellcodes ist es oftmals besser, den Inhalt dieser Eigenschaft in Objektvariablen zu speichern, statt immer wieder endlose Verschachtelungen zu verwenden, um auf die Eigenschaften und Methoden des entsprechenden Objekts, dessen Verweis in der Eigenschaft gespeichert ist, zugreifen zu können. In diesem Fall reicht es bei der Übergabe an eine Variable nicht, einfach nur den Inhalt der Eigenschaft an eine Variable zu übergeben. In diesem Fall wird immer das Schlüsselwort *Set* verwendet, um den Verweis zu erstellen.

Da es sich bei folgendem Beispiel um ein Objekt und nicht „nur“ um eine „simple“ Eigenschaft handelt, müssen Sie statt

```
ActLayer = ThisDrawing.ActiveLayer
```

die Anweisung

```
Set ActLayer = ThisDrawing.ActiveLayer
```

verwenden, um eine Objektvariable zu erstellen. Der umgekehrte Weg erfolgt allerdings ohne das Schlüsselwort *Set*. Wenn Sie z.B. den aktuellen Layer ändern möchten, dann müssen Sie den Verweis auf einen Layer der Layer-Auflistung in die Eigenschaft *ActiveLayer* zurückschreiben.

```
ThisDrawing.ActiveLayer = ThisDrawing.Layers("0")
```

2.4.3 Objektprozeduren

Objektprozeduren oder auch Methoden genannt, führen bestimmte Anweisungen mit dem aufrufenden Objekt aus. Diese Anweisung könnte z.B. das Zoomen im Zeichnungsfenster sein.

```
Application.ZoomAll
```

Im Prinzip sind die Objektprozeduren nichts anderes als in eine Objektklasse eingebettete *Function* oder *Sub*. Diese Methoden können jedoch nur über ihren Container aufgerufen werden. Ansonsten gelten dieselben Regeln, was die Parameterübergabe oder die Rückgabe von Ergebnissen betrifft. Wann es sich um eine *Sub* oder *Function* handelt, können Sie schon allein daran erkennen, ob ein Rückgabetyt angeboten wird

```

Application.GetInterfaceObject |
GetInterfaceObject(ProgID As String) As Object

```

Abbildung 2.4: Mit Rückgabetyt

oder nicht

```
Application.ZoomCenter
ZoomCenter(Center, Magnify As Double)
```

Abbildung 2.5: Ohne Rückgabetyt

2.4.4 Objektauflistungen

Zu den besonderen Objekten der AutoCAD-Objektbibliothek zählen die Auflistungen. Wenn Sie den Objektkatalog aktivieren und sich die einzelnen Klassen von AutoCAD anschauen, dann werden Sie feststellen, dass es häufig Elemente gibt, die sich bezüglich ihres Namens nur in der Endung „s“ unterscheiden, beispielsweise *AcadLayer* und *AcadLayers* oder *AcadBlock* und *AcadBlocks*.

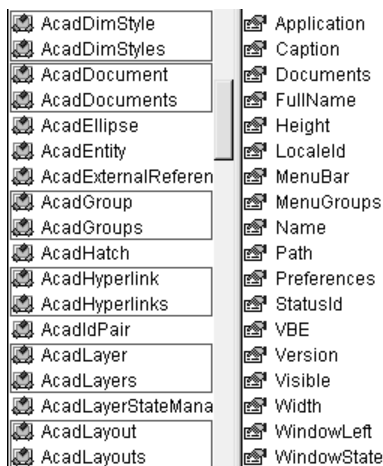


Abbildung 2.6: Auflistungsklassen und dazugehörige Objektklassen

Die Objekte, die mit dem Plural-„s“ enden, verwalten gleichnamige Objekte, die in der Einzahl angegeben sind, also *Documents* → *Document*, *Layers* → *Layer* oder *DimStyles* → *DimStyle* usw. Alle diese Auflistungsobjekte enthalten zu diesem Zweck Eigenschaften und Methoden, die immer demselben Zweck dienen, egal auf welches Objekt sie angewendet werden. Die *Count*-Eigenschaft gibt immer die Anzahl der in der Auflistung enthaltenen Objekte zurück, *Add* fügt der Auflistung ein neues Element hinzu und *Item* ruft ein Element aus der Auflistung ab.

Tipp

Obwohl die meisten Auflistungen auch eine `Delete`-Anweisung enthalten, löst deren Ausführung einen Fehler aus. Sie müssen vielmehr die `Delete`-Anweisung der Objekte in der Auflistung benutzen, um die entsprechenden Objekte aus der Auflistung zu entfernen.

Add-Anweisung

Wie schon erwähnt, fügt die *Add*-Anweisung ein Element der Auflistung, die diese Methode aufgerufen hat, hinzu. Die Syntax ist dabei immer dieselbe. Im Parameter von *Add* legen Sie einen eindeutigen Namen fest. Eindeutig heißt, dass noch kein Element mit demselben Namen in der Auflistung geführt wird, da die *Name*-Eigenschaft gleichzeitig als Schlüssel verwendet wird, mit dem sich das Element wieder abrufen lässt. Wurde das Element erfolgreich zur Auflistung hinzugefügt, erhalten Sie als Rückgabe einen Verweis auf das erstellte Objekt, das der Klasse entspricht, mit der die Objekte der Auflistung definiert sind. Sie können auch das Element der Auflistung hinzufügen, ohne das Rückgabeobjekt zu übernehmen. In dem Fall rufen Sie die Anweisung auf, ohne die Parameter in Klammern einzuschließen und ohne Zuweisung an eine Objektvariable

Layer hinzufügen mit Erstellung eines Verweises auf das neue Layer-Objekt:

```
Sub AddLayer()  
Dim ActLayer As AcadLayer  
Set ActLayer = ThisDrawing.Layers.Add("Layer1")  
End If
```

Nur den Layer hinzufügen:

```
Sub AddLayer()  
ThisDrawing.Layers.Add "Layer1"  
End If
```

Count-Eigenschaft

Die *Count*-Eigenschaft gibt die Anzahl der in der Auflistung befindlichen Objekte zurück. Sie können somit die Obergrenze der Auflistung bestimmen und dementsprechend die *Item*-Anweisung benutzen.

Die folgende Zeile gibt die Anzahl der Layer der aktuellen Zeichnung im Debug-Fenster aus:

```
Debug.Print ThisDrawing.Layers.Count
```

Wenn Sie Elemente aus der Auflistung entfernen oder mit der *Add*-Anweisung Objekte hinzufügen, ändert sich logischerweise auch die *Count*-Eigenschaft.

Item-Anweisung

Die *Item*-Anweisung kann sozusagen als Standardanweisung der Auflistungen angesehen werden. Das bedeutet, dass Sie den Namen der Anweisung auch weglassen können und stattdessen den erforderlichen Parameter direkt mit dem Auflistungsobjekt aufrufen. Der folgende Aufruf

```
ThisDrawing.Layers.Item(0)
```

hat beispielsweise die gleiche Wirkung wie

```
ThisDrawing.Layers(0)
```

Im Parameter wird der Index des gesuchten Objekts gefordert. Dieser Index kann eine Zahl sein, die die direkte Position des Objekts innerhalb der Auflistung bestimmt, oder der Name des Objekts, der wiederum mit dessen *Name*-Eigenschaft übereinstimmt. Wenn Sie die Posi-

tion des Objekts angeben, müssen Sie darauf achten, dass der Index des ersten Elements einer Auflistung immer mit 0 beginnt, was zufolge hat, dass das letzte Element den Index `Count - 1` besitzt. Folgende Prozedur ruft nacheinander alle Layer der Zeichnung ab und gibt deren Namen im Debug-Fenster aus.

```
Sub ShowLayers()
For i = 0 To ThisDrawing.Layers.Count - 1
    Debug.Print ThisDrawing.Layers(i).Name
Next
End Sub
```

Tip

Sie brauchen nicht unbedingt einen Verweis auf ein Objekt zu erstellen, sondern können sich durch die Hierarchie des entsprechenden Objekts zum Ziel vorarbeiten. Da die `Item`-Anweisung bereits einen Verweis auf das gesuchte Objekt enthält, können Sie die Eigenschaften und Methoden direkt nach dem Parameter angeben.

`ThisDrawing.Layers(i)` enthält bereits den Verweis auf das *n*-te Element der Layer-Auflistung, weshalb Sie anschließend sofort die Eigenschaften und Methoden dieses Objekts verwenden können.

Das gilt übrigens für alle Objekte, die untergeordnete Objekte enthalten.

Wenn Sie das Objekt direkt mit Namen ansprechen, müssen Sie im Parameter eine Zeichenfolge angeben. Diese Zeichenfolge muss mit der *Name*-Eigenschaft übereinstimmen. Diese Form der Abfrage ist schneller, als alle Elemente der Auflistung zu durchlaufen und jedes einzelne Element zu kontrollieren,

```
Function GetLayer(LayerName As String) As AcadLayer
For i = 0 To ThisDrawing.Layers.Count - 1
    If ThisDrawing.Layers(i).Name = LayerName Then
        Set GetLayer = ThisDrawing.Layers(i)
        Exit Function
    End If
Next
End Function
```

Das macht allerdings nur Sinn, wenn der aktuelle Name bekannt ist.

```
Function GetLayer(LayerName As String) As AcadLayer
Set GetLayer = ThisDrawing.Layers(LayerName)
End Function
```

2.4.5 Übersichtlicher Objektzugriff

Bei der Arbeit mit Objekten führt der Aufruf einer Eigenschaft oder Prozedur innerhalb des Objekts oft zu vielfachen Verschachtelungen. Nehmen wir als Beispiel die Ausgabe der verschiedenen Verzeichnisse des *Preferences*-Objekts. Zunächst ist dieses Objekt Eigentum des *Application*-Objekts. Um es anzusprechen, muss also das *Application*-Objekt vorangestellt werden:

Application.Preferences

Da auch das *Preferences*-Objekt verschiedene Unterobjekte umfasst, folgt die nächste Unterteilung:

Application.Preferences.Files

Erst das Objekt *Files* enthält die gewünschten Eigenschaften zur Ermittlung der einzelnen Verzeichnisse. Wenn Sie nicht extra eine Variable erstellen wollen, die einen Verweis auf Files aufnimmt, würde der Aufruf jedes Mal folgendermaßen aussehen:

```
Debug.Print Application.Preferences.Files.AltFontFile
Debug.Print Application.Preferences.Files.AltTabletMenuFile
Debug.Print Application.Preferences.Files.AutoSavePath
```

usw.

VBA bietet mit dem `With End With`-Block die Möglichkeit, den Abruf der Eigenschaften und Methoden auf ein Ausgangsobjekt zu beziehen. Das Objekt, auf das sich die innerhalb des Blocks aufgerufenen Eigenschaften und Methoden beziehen, folgt einfach der `With`-Anweisung. Beendet wird der Block mit `End With`. Sobald Sie sich innerhalb des Blocks mit einer Eigenschaft oder Methode auf das angegebene Objekt beziehen, stellen Sie der Eigenschaft oder Methode einfach einen Punkt (.) voran – ganz so, als würde das Objekt davor stehen.

```
With Application.Preferences.Files
    Debug.Print .AltFontFile
    Debug.Print .AltTabletMenuFile
    Debug.Print .AutoSavePath
End With
```

`With End With`-Blöcke lassen sich auch verschachteln. Sie müssen dabei lediglich darauf achten, auf welches Objekt sich der entsprechende Block gerade bezieht.

```
Public Sub Width_Example()
    Dim Menu As AcadPopupMenu
    With Application.MenuBar
        For i = 0 To .Count
            Set Menu = .Item(i)
            With Menu
                Debug.Print Menu.Name
            End With
        Next
    End With
End Sub
```

2.4.6 Ereignisse

Eine besondere Form der Methoden, die ein Objekt ausführen kann, ist die Ereignisprozedur. Diese Prozeduren werden selbständig durch das Objekt aufgerufen, wenn eine bestimmte Reaktion auf ein Ereignis, das dieses Objekt beeinflusst, auftritt. Ein solches Ereignis könnte eine Änderung an den Eigenschaften eines Objekts oder die Änderung der Größe des Zeichnungsfensters sein. Da Ereignisse objektspezifisch sind, also abhängig von dem auslösenden Objekt, gibt es für die verschiedensten Objekte auch die unterschiedlichsten Ereignisse. Eine Linie löst z.B. kein *NewDrawing*-Ereignis (eine neue Zeichnung wird erstellt) aus, da dieses Ereignis nur in Zusammenhang mit dem *Application*-Objekt steht. Die beiden Klassen mit den mit Abstand meisten Ereignisprozeduren sind *Application* und *Document*. Alle anderen Objekte enthalten nur das *Modified*-Ereignis, das ausgelöst wird, wenn eine Änderung am entsprechenden Objekt durchgeführt wurde.

Tipp Die Bedeutung der Eigenschaften der Objekte *Application* und *Document* wird in den entsprechenden Kapiteln genau erklärt.

Ereignisprozeduren eines Objekts aktivieren

Die Ereignisse der einzelnen Objekte sind nicht automatisch in das VBA-Projekt integriert. Das bedeutet, dass Ereignisprozeduren erst dann ausgelöst werden, wenn Sie eine Objektvariable explizit darauf vorbereiten. Eine Ausnahme gibt es allerdings: das Klassenmodul *ThisDrawing*, das ja bereits bei der Erstellung des VBA-Projekts erstellt wurde und einen Verweis auf das aktuelle Dokument speichert. Dieses Klassenmodul enthält bereits alle Ereignisprozeduren, die ein Dokumentobjekt auslösen kann. Das können Sie kontrollieren, indem Sie in das Klassenmodul wechseln und die Objektvariable in der Objektliste auswählen. Daraufhin erscheinen in der Prozedurliste alle Ereignisprozeduren, die ein Dokumentobjekt auslösen kann. Nun gilt das Klassenmodul *ThisDrawing* nur für das aktuelle Dokument und außerdem fehlen noch die Ereignisprozeduren des *Application*-Objekts. Um diese zu aktivieren, müssen Sie eine spezielle Objektvariable deklarieren und anschließend einen Verweis darauf erstellen. Dazu verwenden Sie das Schlüsselwort *WithEvents*, das der *Public*- oder *Dim*-Anweisung folgt. Für eine Objektvariable des Typs *AcadApplication* würde dies wie folgt aussehen:

```
Public WithEvents AcadApp As AcadApplication
```

Tipp Das Schlüsselwort *WithEvents* kann nur innerhalb von Klassenmodulen verwendet werden, weshalb die Deklaration entweder im Klassenmodul *ThisDrawing* oder innerhalb eines eigenen Klassenmoduls erfolgen muss.

Die Zuweisung eines Datentyps ist dabei zwingend erforderlich. Sonst wäre nicht klar, um welche Ereignisse von welcher Objektklasse es sich handeln soll. Die Objektvariable *AcadApp* entspricht nun dem Objektmodell der Klasse *AcadApplication*, genau wie die Objektvariable *Application*. Was jetzt noch bleibt, ist die Erstellung eines Verweises auf *Application*, damit die Ereignisse auch ausgelöst werden können. Die Initialisierung erfolgt wie gehabt:

```
Set AcadApp = Application
```

Hier stellt sich allerdings die Frage, wo diese Anweisung im Quellcode positioniert wird. In AutoCAD-VBA besteht das Problem darin, dass während der Entwicklung von VBA-Projekten die initialisierten Objektvariablen entladen werden, so dass die Ereignisse dann erst wieder verfügbar werden, wenn Sie einen erneuten Verweis auf das entsprechende Objekt erstellt haben.

Das ist für einfache Objekte nicht ganz so schlimm, da ein Verweis auf ein bestimmtes Objekt normalerweise immer erst im Verlauf eines Programms erfolgt. Wenn Sie z.B. ein bestimmtes Objekt überwachen wollen, beispielsweise eine Linie, dann wird sicherlich erst eine Auswahl der entsprechenden Linien erfolgen. Daraufhin wird der Verweis erstellt und erst dann werden die kommenden Ereignisse abgewartet. Beim *Application*-Objekt ist das etwas anders. Dieses Objekt enthält Ereignisprozeduren, die auch ausgelöst werden sollen, ohne dass erst ein Makro zur Erstellung eines Verweises auf *Application* ausgeführt wird. Die Zuordnung des Verweises sollte also möglichst nach dem Laden des Projekts erfolgen. Zu diesem Zweck müssten Sie die Ereignisse des Klassenmoduls *ThisDrawing* verwenden, das ja permanent verfügbar ist. Wenn der Anwender das VBA-Projekt über einen AutoCAD-Befehl lädt, löst das z.B. die Ereignisprozedur *EndCommand* aus, auf die entsprechend reagiert werden kann.

```
Private Sub AcadDocument_EndCommand(ByVal CommandName As String)
Set AcadApp = Application
End Sub
```

Wenn das Projekt bereits beim Start von AutoCAD geladen werden soll, empfiehlt es sich, den Ladevorgang nicht über den Menüpunkt EXTRAS→ANWENDUNGEN des AutoCAD-Anwendungsmanagers durchzuführen, sondern die Datei *acad.dvb* zu nutzen, die über eine Startprozedur verfügt (*Kapitel 1.1.5*). Die Abfolge könnte wie folgt aussehen:

```
Sub ACADStartup()
On Local Error Resume Next
ThisDrawing.Application.LoadDVB "C:\VBA\MeinProjekt.dvb"
RunMacro "MeinProjekt.ThisDrawing.initAcadApp"
End Sub
```

Zunächst wird das entsprechende Projekt geladen und anschließend ein Makro im geladenen Projekt ausgeführt, das speziell die Objektvariable *AcadApp* des neuen Projekts initialisiert. Da die Objektvariable *AcadApp* im Klassenmodul *ThisDrawing* deklariert wurde und der Aufruf des Makros sich ebenfalls auf *ThisDrawing* bezieht, muss die Prozedur dort enthalten sein.

```
Public WithEvents AcadApp As AcadApplication
Public Sub initAcadApp()
Set AcadApp = Application
End Sub
```

Auf Ereignisse reagieren

Um auf bestimmte Ereignisse zu reagieren, die ein Objekt auslösen kann, müssen Sie die entsprechende Ereignisprozedur aktivieren. Da durch die Deklaration einer Objektvariablen mit dem Schlüsselwort *WithEvents* die Ereignisse im Codefenster verfügbar werden, können Sie einfach die entsprechende Prozedur aus der Liste auswählen. VBA erstellt daraufhin den Prozedurkörper mit den jeweiligen Parametern, die zu der Prozedur gehören. Lassen Sie uns die genaue Abfolge nachvollziehen. Nehmen wir einmal an, Sie möchten immer informiert werden, wenn ein Objekt in AutoCAD verändert wird – sei es die optische Darstellung eines grafischen Objekts oder der Inhalt einer Auflistung. Dafür stellt die Dokumentklasse das Ereignis *ObjectModified* zur Verfügung. Wechseln Sie nun in das Quellcodefenster des Klassenmoduls *ThisDrawing* und wählen Sie das Objekt *AcadDocument* aus der Objektliste aus.



Abbildung 2.7: Objektliste

In der Prozedurliste können Sie nun die Ereignisprozedur *ObjectModified* aktivieren, woraufhin die Ereignisprozedur automatisch angelegt wird.

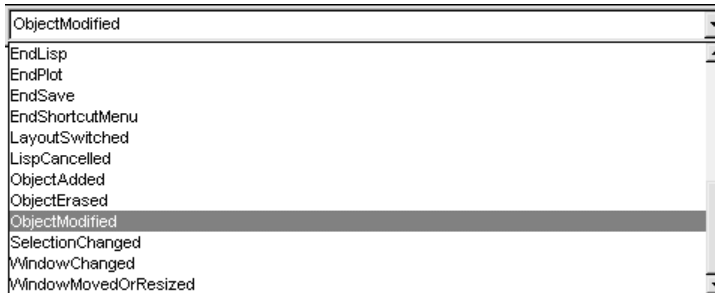


Abbildung 2.8: Prozedurliste

```
Private Sub AcadDocument_ObjectModified(ByVal Object As Object)
End Sub
```

Dieser Prozedur müssen Sie nun nur noch den entsprechenden Code hinzufügen, der ausgeführt werden soll, wenn ein Objekt in der aktuellen Zeichnung geändert wird.

2.5 Entwicklung eigener Klassenmodule

Neben den Modulen und Formularen können Sie in Ihren Projekten auch Klassenmodule verwenden. Ein sehr aussagekräftiges Beispiel, was Klassenmodule eigentlich sind, ist das Objekt *ThisDrawing*, das AutoCAD ja bekanntlich schon bei der Erstellung Ihres Projekts automatisch hinzufügt. Klassenmodule unterscheiden sich von den Standardmodulen darin, wie sie die Daten speichern. Von Daten in Standardmodulen wird immer nur eine Kopie erstellt. Bei einer Änderung des Inhalts einer öffentlichen Variable in einem Teil des Programms weist diese Variable dementsprechend in einer anderen Prozedur ebenfalls den geänderten Wert auf.

Klassenmodule hingegen existieren jeweils in jeder mit diesem Klassenmodul erstellten Instanz. Unter „Instanz“ soll dabei die Deklaration eines Objekts mit dem jeweiligen Klassenmodul verstanden werden. Außerdem kann auf öffentliche Variablen eines Klassenmoduls nur über eine Objektvariable, die als eine Instanz des Klassenmoduls erstellt wurde, zugegriffen werden.

```
Dim Object1 As Klasse1
Dim Object2 As Klasse1

Set Object1 = New Klasse1
Set Object2 = New Klasse1
```

Beide Objekte enthalten nun die definierten Eigenschaften und Methoden des Klassenmoduls, können aber verschiedene Daten aufnehmen, da es sich um zwei eigenständige Instanzen der Klasse1 handelt.

Das soll einmal anhand eines kleinen Beispiels veranschaulicht werden. Fügen Sie dazu einem Projekt ein neues Klassenmodul hinzu und ergänzen Sie den Quellcode nur um die Deklaration einer einfachen Variablen.

```
Public Kommentar As String
```

Erstellen Sie nun eine Instanz der Klasse im Modul *ThisDrawing* mittels folgender Prozedur:

```
Sub KlassenBeispiel  
Dim FirstClass As Klasse1  
Dim SecondClass As Klasse1  
  
Set FirstClass = New Klasse1  
Set SecondClass = New Klasse1  
  
FirstClass.Kommentar = "Die erste Klasse wurde geändert"  
  
MsgBox FirstClass.Kommentar  
MsgBox SecondClass.Kommentar  
  
End Sub
```

Wenn Sie diese Prozedur ausführen, erscheinen zwei Meldungsfenster, mit denen demonstriert wird, dass jede Instanz eigene Daten speichert.

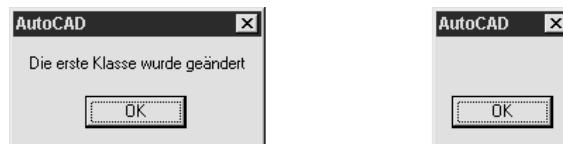


Abbildung 2.9: Meldungen zweier Instanzen

Klassenmodule enthalten aber nicht nur Eigenschaften, sondern können komplexe Prozeduren enthalten und zur Verwaltung ganzer Datenreihen verwendet werden. Bleiben wir bei dem komplexen Klassenmodul *ThisDrawing*. Dieses verwaltet alle Objekte der Zeichnung. Dies wäre nur mit Arrays wohl schlecht zu programmieren, weshalb ein Objekt ins Spiel kommt, das diese Aufgabe viel besser lösen kann. Und damit sind wir auch schon beim Kernpunkt der Anwendung von Klassenmodulen angekommen – dem *Collection*-Objekt.

2.5.1 Verwendung des Collection-Objekts

Das *Collection*-Objekt bringt einige Grundfunktionen zur Verwaltung von Daten mit. Dabei ist es nicht einmal erforderlich, dass Elemente, die in die Auflistung aufgenommen werden, vom gleichen Datentyp sind. Elemente einer Auflistung stehen eigentlich nur dadurch in Beziehung, dass sie Inhalt des *Collection*-Objekts sind.

Erstellt wird eine Auflistung wie alle anderen Objekte auch. Zunächst die Deklaration einer Variablen vom Datentyp *Collection*:

```
Dim MyCol As Collection
```

Und anschließend die Erstellung der Objektvariablen:

```
Set MyCol = New Collection
```

Das so erstellte Objekt stellt nun die Eigenschaften der Auflistung zur Verfügung, mit denen Sie Daten hinzufügen, abrufen oder entfernen können.

```
Add Item, [Key], [Before], [After]
```

Diese Anweisung fügt der Auflistung ein neues Element hinzu. Die Vergabe des ersten Parameters ist zwingend erforderlich. Er enthält das Element, das zur Auflistung hinzugefügt werden soll. Mit dem zweiten Parameter (*Key*) können Sie ein eindeutiges Schlüsselwort mitgeben, das das Element, anstatt der Positionsangabe, in der Auflistung identifiziert. Die Argumente *Before* und *After* bezeichnen die Position, an der das Element in der Auflistung hinzugefügt werden soll. Sie können für diese beiden Werte entweder die Positionsangabe oder eine Zeichenfolge, die dem Schlüssel eines bereits vorhandenen Eintrags entspricht, verwenden. Das angegebene Element wird entweder vor (*Argument Before*) oder nach (*Argument After*) dem angegebenen Element eingefügt. Wenn Sie für die Position eine Zahl angeben, muss diese in einem Bereich von 1 bis Anzahl der bereits in der Auflistung befindlichen Elemente angesiedelt sein.

```
Dim MyCol As Collection
Set MyCol = New Collection
MyCol.Add Object1, "Objekt1"

Item Index
```

Um ein bestimmtes Element aus der Auflistung abzurufen, müssen Sie die *Item*-Anweisung verwenden. Als Parameter dieser Anweisung verwenden Sie die Positionsangabe oder das Schlüsselwort, mit dem das Element hinzugefügt wurde. Die *Item*-Anweisung ist die Standardprozedur des *Collection*-Objekts und muss somit nicht explizit angegeben werden.

```
Set MyObject = MyCol.Item("Objekt1")
```

oder

```
Set MyObject = MyCol.Item(1)
```

oder

```
Set MyObject = MyCol("Objekt1")
```

oder

```
Set MyObject = MyCol(1)
```

Remove Index

Mit dieser Anweisung entfernen Sie das Element aus der Auflistung, dessen Position oder Schlüsselwort im Parameter *Index* angegeben wurde:

```
Set MyObject = MyCol.Remove("Objekt1")
```

oder

```
Set MyObject = MyCol.Remove(1)
```

Count

Als Letztes gibt es noch die *Count*-Eigenschaft zur Ermittlung der genauen Anzahl der Elemente, die in die Auflistung aufgenommen wurden.

Ein praktisches Anwendungsbeispiel

Diese Grundfunktionen bilden das Gerüst für die Entwicklung komplexer Klassenmodule. Gehen wir dazu etwas ins Detail. Angenommen, Sie möchten ein Objekt erstellen, das die verschiedensten Benutzer verwalten soll. Dazu ist ein Klassenmodul hervorragend geeignet. Genauer gesagt werden zwei Klassenmodule benötigt. Das eine wird das einzelne Benutzerobjekt definieren, das andere die Objekte verwalten. Legen Sie zunächst ein neues Projekt an und fügen Sie diesem Projekt zwei Klassenmodule hinzu. Das erste Klassenmodul benennen Sie in *Benutzer*, das zweite in *BenutzerGruppe* um.

Quellcode des Klassenmoduls „Benutzer“

Diesem Klassenmodul müssen Sie lediglich zwei Variablen hinzufügen. Die erste wird den Namen des Benutzers aufnehmen, die zweite eine Kurzbeschreibung.

```
Public Benutzername As String
Public Beschreibung As String
```

Quellcode des Klassenmoduls „BenutzerGruppe“

Das Klassenmodul *BenutzerGruppe* wird die oberste Hierarchiestufe bilden, genauso wie Sie es bereits von der Layer-Auflistung und den darin enthaltenen Layern kennen. Als Erstes wird eine lokale Objektvariable benötigt, die die einzelnen Benutzer aufnehmen soll.

```
Dim mCol As Collection
```

Diese Objektvariable muss natürlich wieder initialisiert werden. Dazu können Sie das Ereignis *Initialize* verwenden, das jedes Klassenmodul auslöst, wenn eine neue Instanz der Klasse erzeugt wird. In dieser Ereignisprozedur erzeugen wir nun eine neue Instanz von *Collection* in der Objektvariablen *mCol*. Beachten Sie dabei die Verwendung von *New*, damit auch wirklich eine neue Instanz erzeugt wird. Diese Instanz ist nun so lange verfügbar, wie die Instanz der Objektvariablen, die später aus dem Klassenmodul *Benutzergruppe* erzeugt wird, existiert – oder einfacher ausgedrückt: so lange, bis das Ereignis *Terminate* ausgelöst wird.

```
Private Sub Class_Initialize()
Set mCol = New Collection
End Sub
```

Da diese Klasse die Verwaltung der einzelnen Benutzer übernehmen soll, müssen bestimmte Prozeduren dem Klassenmodul hinzugefügt werden, die diese Aufgabe übernehmen. Als Erstes muss natürlich eine Funktion zur Verfügung gestellt werden, die es ermöglicht, neue Benutzer zur Auflistung hinzuzufügen.

Add-Prozedur

Zunächst wird eine neue Instanz eines Objekts der Klasse *Benutzer* erstellt, die die Daten der Parameter *Benutzer* und *Beschreibung* aufnimmt. Anschließend wird nur noch das neue Objekt der Auflistung *mCol* hinzugefügt.

```
Public Function Add(Name As String, Optional Beschreibung As _String)
Dim NewUser As Benutzer
Set NewUser = New Benutzer
NewUser.Benutzername = Name
```

```
If IsMissing(Beschreibung) = False Then
    NewUser.Beschreibung = Beschreibung
End If
mCol.Add NewUser
End Function
```

Item-Prozedur

Die Item-Prozedur dient dem Abruf eines vorhandenen Benutzers aus der Auflistung. Eigentlich sollte die Item-Prozedur keine Prozedur, sondern eine Eigenschaft sein, aber dazu mehr im nächsten Kapitel. Hier soll die Item-Prozedur zunächst als Prozedur aufgefasst werden. Der übergebene Parameter wird dabei als Positionsangabe in der Auflistung *mCol* verwendet. Wenn Sie auch einen Schlüssel zum Abruf des entsprechenden Benutzerobjekts verwenden wollen, müssen Sie diesen bei der Add-Anweisung als zweiten Parameter angeben.

```
Public Function Item(Index) As Benutzer
    On Local Error Resume Next
    Set Item = mCol.Item(Index)
End Function
```

Remove-Prozedur

Die Remove-Anweisung enthält ebenfalls den Parameter *Index*, nur wird hier der Index an die Remove-Anweisung der Auflistung übergeben, so dass das entsprechende Benutzerobjekt entfernt wird.

```
Public Function Remove(Index)
    On Local Error Resume Next
    mCol.Remove Index
End Function
```

Count-Prozedur

Auch hier sollte es sich eigentlich um eine Eigenschaft handeln. Zunächst verwenden wir die Count-Prozedur aber als Prozedur.

```
Public Function Count() As Integer
    On Local Error Resume Next
    Count = mCol.Count
End Function
```

Clear-Prozedur

Bei der Clear-Prozedur, für die ja kein Gegenstück in dem Auflistungsobjekt existiert, muss lediglich eine neue Instanz des Objekts *mCol* erstellt werden. Dadurch wird einfach erreicht, dass die vorhandene Benutzerliste sozusagen geleert wird. Da durch die Anwendung von *New* die vorherige Instanz von *mCol* freigegeben wird, fällt die Zurücksetzung mit *Nothing* weg.

```
Public Function Clear() As Integer
    On Local Error Resume Next
    Set mCol = New Collection
End Function
```

2.5.2 Eigenschaften in Klassenmodulen

Bei Eigenschaften handelt es sich ebenfalls um Prozeduren, die allerdings etwas anders gestaltet werden und sich beim Aufruf wie Variablen verhalten. Nehmen wir als Beispiel gleich die zuvor behandelte `Item`-Prozedur und wandeln diese in eine „richtige“ Eigenschaft um. Gegenüber normalen Prozeduren bieten die Eigenschaften gleich zwei Anweisungen mit gleichem Namen. Die eine dient dem Lesen des Inhalts der Eigenschaft und beginnt mit `Property Get`:

```
Public Property Get Item(Index) As Benutzer
Set Item = mCol(Index)
End Property
```

Genauso könnte nun auch `Count` angepasst werden:

```
Public Property Get Count() As Integer
Count = mCol.Count
End Property
```

Die andere `Property`-Prozedur dient dem Schreiben neuer Eigenschaftswerte und wird mit `Property Let` gekennzeichnet. Für das Beispiel wird keine `Property Let`-Anweisung verwendet, da die Eigenschaften `Item` und `Count` schreibgeschützt sind.

Gehen wir nun etwas näher auf die beiden Funktionen ein. Der erste Schritt ist die Festlegung einer lokalen Variable, die als Kopie der Eigenschaft dienen soll. Als praktisches Beispiel werden wir gleich das Klassenmodul *Benutzer* entsprechend anpassen.

```
Private mvarBeschreibung As String
```

Als Nächstes folgt die Prozedur zum Schreiben des Eigenschaftswerts. Dieser wird in der Variablen im Parameter der Prozedur übergeben und dann einfach der lokalen Variable zugeordnet.

```
Public Property Let Beschreibung(ByVal vData As String)
mvarBeschreibung = vData
End Property
```

Das Lesen der Eigenschaft erfolgt wieder durch Ermittlung des aktuellen Werts der lokalen Variable.

```
Public Property Get Beschreibung() As String
Beschreibung = mvarBeschreibung
End Property
```

Derselbe Vorgang dient der Erstellung der Benutzernamen-Eigenschaft: zunächst Deklaration der lokalen Kopie, dann Erstellung der Prozeduren `Property Let` und `Property Get`, wobei sich diese auf die lokale Variable *mvarBenutzerName* beziehen.

```
Private mvarBenutzerName As String
Public Property Let BenutzerName(ByVal vData As String)
mvarBenutzerName = vData
End Property
```

```
Public Property Get BenutzerName() As String
BenutzerName = mvarBenutzerName
End Property
```

Wenn Sie einer Eigenschaft Objekte zuweisen wollen, dann müssen Sie statt `Property Let` die Prozedur `Property Set` verwenden. Die lokale Kopie wird weiterhin deklariert, nur dass dort der entsprechende Datentyp angegeben wird.

```
Private mvarMyObject As MyObjectClass
```

Bei der Erstellung der Prozeduren ist zu beachten, dass die Objekte den Variablen mit der `Set`-Anweisung zugeordnet werden.

```
Public Property Set MyObject(ByVal vData As MyObjectClass)
Set mvarMyObject = vData
End Property

Public Property Get MyObject() As MyObjectClass
Set MyObject = mvarMyObject
End Property
```

2.5.3 Ereignisse in Klassenmodulen

Um die Funktionalität der Klassenmodule zu komplettieren, fehlen natürlich noch Ereignisse. Diese werden ähnlich wie Variablen oder Konstanten im Deklarationsteil des Klassenmoduls erstellt. Das Schlüsselwort, das hierfür zum Einsatz kommt, lautet `Event`. Wenn Sie folgende Zeile der Klasse *BenutzerGruppe* hinzufügen, erhält sie ein Ereignis:

```
Public Event UserLocked(ByVal UserName As String)
```

Dieses Ereignis muss nur noch in einer bestimmten Prozedur innerhalb des Klassenmoduls ausgeführt werden, z.B. in der `Add`-Funktion. Die Anweisung `RaiseEvent` übernimmt diesen Part.

```
RaiseEvent UserLocked
```

Sie müssen nun nur noch ein Objekt mit `WithEvents` deklarieren, das eine neue Instanz auf Basis des Klassenmoduls erstellt, und die Ereignisprozedur mit dem entsprechenden Code belegen.

```
Dim WithEvents MyUsers As BenutzerGruppe
Private Sub MyUsers_UserLocked(ByVal UserName As String)
End Sub
```

2.5.4 Klassenableitung mit Implements

In AutoCAD gibt es einige Klassen, die als Grundlage für die Definition vieler anderer Klassen dienen. Ein Beispiel hierfür ist die Klasse *AcadObject*, die die Grundlage aller AutoCAD-Objekte bildet. Dieses Prinzip lässt sich auch auf die Klassenmodule übertragen.

Stellen Sie sich vor, Sie haben eine solche Grundlagenklasse erstellt. Diese enthält verschiedene Eigenschaften und Methoden, die Sie auf andere Klassenmodule übertragen wollen. Nun könnten Sie diese auf die anderen Klassenmodule übertragen, indem Sie die Ursprungsklasse Prozedur für Prozedur und Eigenschaft für Eigenschaft manuell nachgestalten oder Sie übertragen sie einfach mit `Implements` in das neue Klassenmodul. Um z.B. die Definition der Eigenschaften und Methoden der Klasse *AcadDatabase* zu übertragen, müssten Sie folgende Zeile in das Klassenmodul einbinden.

```
Implements AcadDatabase
```

Daraufhin erscheint *AcadDatabase* in der Objektliste des Codefensters. Da Sie nun die gesamte Struktur der implementierten Klasse *AcadDatabase* nachbilden müssten (Auswahl der einzelnen Eigenschaften und Methoden aus der Prozedurliste), wird der Code bei Klassen mit dieser Vielzahl an Eigenschaften und Methoden recht unübersichtlich.

2.6 Operatoren in VBA

Zu den Operatoren muss man normalerweise nicht viele Worte verlieren. Deshalb fasse ich dieses Thema nur kurz zusammen und erkläre einige Besonderheiten im Umgang mit Operatoren. Operatoren lassen sich in vier Hauptkategorien unterteilen: arithmetische, Vergleichs-, Verkettungs- und logische Operatoren.

Vergleichsoperatoren dienen, wie der Name bereits deutlich macht, dem Vergleich zweier Werte oder Zeichenketten. Neben den „normalen“ Vergleichsoperatoren wie größer, gleich oder kleiner gibt es noch zwei zusätzliche Operatoren. Der *Is*-Operator vergleicht zwei Objektvariablen miteinander und gibt, falls diese derselben Quelle entspringen, *TRUE* zurück. Der *Like*-Operator stellt eine besondere Form der Vergleichsoperatoren dar. Die Verwendung dieses Operators veranlasst VBA, einen Vergleich zwischen einer Zeichenkette und einer Mustervorlage vorzunehmen. Wenn das angegebene Muster in der Zeichenfolge wiederzufinden ist, gibt der Vergleich den Wert *TRUE* zurück. Im Muster können folgende Platzhalter verwendet werden:

Musterzeichen	Steht in der Zeichenfolge für
?	Ein beliebiges Zeichen
*	Kein oder mehrere Zeichen
#	Eine beliebige Ziffer
[]	Die eckigen Klammern legen den Anfang und das Ende einer Liste der Zeichen fest, die in der Zeichenfolge vorhanden sind.
[!]	Wenn der ersten eckigen Klammer ein Ausrufezeichen folgt, legt diese den Anfang und das Ende einer Liste der Zeichen fest, die <i>nicht</i> in der Zeichenfolge vorhanden sein dürfen.

Tabelle 2.6: Platzhalter des *Like*-Operators

Mehr Informationen zur Verwendung des *Like*-Operators finden Sie in der VBA-Online-Hilfe unter *Like*-Operator.

Logische Operatoren dienen der Verknüpfung zweier oder mehrerer Bedingungen. Ein typisches Anwendungsbeispiel ist z.B. die Kontrolle eines gesetzten Bits bei der Auswertung bestimmter Einstellungen. Nehmen Sie z.B. die Speicherung des Layer-Status. Dieser Status wird anhand einer speziellen Maske gespeichert, die durch das Setzen einiger Bits der Eigenschaft *Mask* bestimmt wird. Zur Kontrolle, welche der Bits gesetzt wurden, kann eine logische Verknüpfung verwendet werden. Um beispielsweise zu ermitteln, ob das Bit, das die Speicherung der Frieren-Eigenschaft aktiviert, gesetzt wurde, vergleicht man den Wert der *Mask*-Eigenschaft mit dem Wert, der das Frieren-Bit setzt. Ist das entsprechende Bit gesetzt, wird als Ergebnis der Wert des Bits zurückgegeben.

```
IF (Mask And acLsFrozen) = acLsFrozen Then
End If
```

Die folgende Tabelle zeigt eine kurze tabellarische Übersicht mit den Ergebnissen eines bitweisen Vergleichs.

Mask-Eigenschaft	Wert von acLsFrozen	Ergebnis
0	0	0
1	0	0
0	1	0
1	1	1

2.6.1 Operatorenübersicht

Vergleichsoperatoren	
=	Gleich
<>	Ungleich
<	Kleiner
<=	kleiner gleich
>	Größer
>=	größer gleich
Is	Gibt <i>TRUE</i> zurück, wenn beide Vergleichsvariablen auf dasselbe Objekt verweisen.
	Folgende Anweisung gibt <i>TRUE</i> zurück:
	Set ObjVar = Application.Documents.Application
	If Application Is ObjVar Then
	...
	Die Anweisung
	Set ObjVar = Application.Documents
	If Application = ObjVar Then
	...
	gibt hingegen <i>FALSE</i> zurück.

<i>Like</i>	Dient zum Vergleich zweier Zeichenfolgen, wobei die zweite Vergleichszeichenfolge ein Muster definiert. Wenn die Zeichenfolge und das Muster übereinstimmen, gibt der Vergleich den Wert <i>TRUE</i> zurück.

Logische Operatoren

<i>And</i>	Logische Konjunktion
<i>Or</i>	Logische Disjunktion
<i>Not</i>	Logische Negation
<i>XOr</i>	Logische Exklusion
<i>Imp</i>	Logische Implikation
<i>Eqv</i>	Logische Äquivalenz

Arithmetische Operatoren

<i>^</i>	Potenzierung
<i>*</i>	Multiplikation
<i>/</i>	Division zweier Zahlen mit dem Ergebnis einer Fließkommazahl
<i>\</i>	Division zweier Zahlen mit dem Ergebnis einer Ganzzahl
<i>Mod</i>	Gibt den Rest einer ganzzahligen Division zweier Zahlen zurück
<i>+</i>	Addition
<i>-</i>	Subtraktion

Verkettungsoperatoren

<i>+</i>	Verkettet zwei Ausdrücke miteinander, wobei einer der beiden eine Zeichenfolge sein muss. Der zweite Ausdruck kann ein beliebiger Wert vom Typ <i>Variant</i> sein, außer <i>Null</i> . Ist keiner der beiden eine Zeichenfolge, handelt es sich um eine arithmetische Operation.
<i>&</i>	Verkettet zwei Zeichenfolgen miteinander

3

Arbeiten mit Formularen

Eine wirklich hervorragende Eigenschaft von VBA ist die visuelle Entwicklungsumgebung. Sie können darin Ihre Formulare, ohne zusätzlichen Code, nach dem Baukastenprinzip zusammenstellen. Im folgenden Kapitel erfahren Sie, wie Sie solche Formulare entwerfen, was dabei zu beachten ist und wie Sie schließlich diese Formulare in den Programmcode einbinden.

Kapitelübersicht

- 3.1 Allgemeines zum Formularentwurf
- 3.2 VBA-Standardsteuerelemente
- 3.3 Weitere Steuerelemente
- 3.4 Das Benutzerformular

3.1 Allgemeines zum Formularentwurf

Nachdem Sie ein Formular Ihrem Projekt hinzugefügt haben, erscheint es in folgender Form in der Entwicklungsumgebung. Bis auf den Titel, den Sie natürlich anpassen können, ist das Formular noch leer.

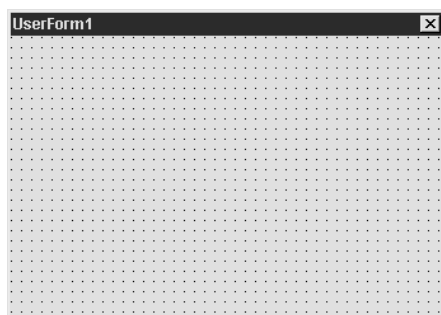


Abbildung 3.1: Neues Formular

Ein Formular erwacht erst durch die Kommunikation mit dem Benutzer zu neuem Leben. Da ein leeres Formular wenig Sinn macht, müssen bestimmte Eingabeelemente vorhanden sein, mit denen der Benutzer spezielle Eingaben tätigen kann. Dazu gehören z. B. Schaltflächen zur Ausführung von Befehlen, Textfelder für variable Eingaben oder auch Listfelder zur Auswahl einer oder mehrerer Vorgaben. Diese Eingabeelemente werden auch Steuerelemente genannt.

3.1.1 Werkzeugsammlung

Wenn Sie ein Formular im Entwurfsmodus zum aktiven Formular machen, erscheint ein kleines Dialogfenster. In diesem Fenster, der Werkzeugsammlung, sind alle Steuerelemente aufgeführt, die Sie momentan in einem Formular platzieren können.

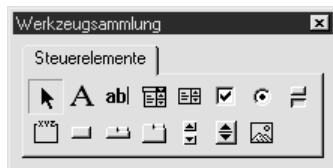


Abbildung 3.2: Formular Werkzeugsammlung

Nach dem ersten Start bietet die Werkzeugsammlung bereits eine gewisse Grundausstattung an Steuerelementen (MSForms-Bibliothek) in Form von kleinen Symbolen. Diese Steuerelemente werden im Verlauf dieses Kapitels noch näher erläutert. Aber bereits jetzt lässt sich sagen, dass diese Grundausstattung bei weitem nicht ausreichen wird. Glücklicherweise haben Sie die Möglichkeit, weitere Steuerelemente hinzuzufügen. Dazu müssen Sie nur mit der rechten Maustaste in das Werkzeugfenster klicken und dann aus dem Kontextmenü den Punkt ZUSÄTZLICHE STEUERELEMENTE aufrufen. Im darauf folgenden Dialogfenster können Sie nun eines oder mehrere Steuerelement(e) markieren, die Sie benötigen.

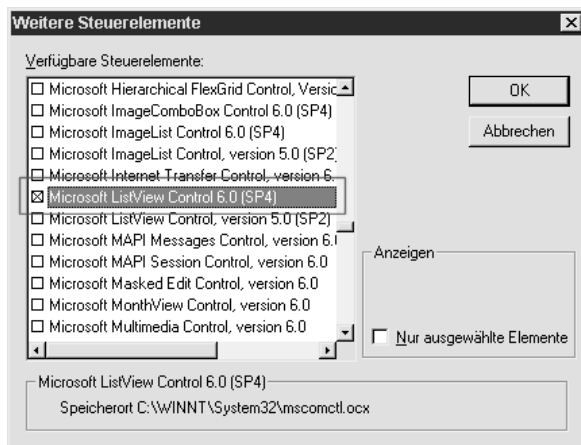


Abbildung 3.3: Auswahldialog für weitere Steuerelemente

Sobald Sie Ihre Auswahl mit OK bestätigen, werden die entsprechenden Steuerelemente Ihrer Werkzeugsammlung als Symbol hinzugefügt. Im vorliegenden Fall wäre dies das Steuerelement *ListView* – ein Listenansicht-Steuerelement, das dem Anzeigen von Elementen unter Verwendung einer von vier unterschiedlichen Ansichten dient.

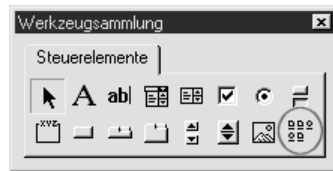


Abbildung 3.4: Werkzeugsammlung mit neuem Steuerelement

Neben diesem Listenansicht-Steuerelement gibt es weitere Steuerelemente, die für das Design und eine komfortable Arbeitsweise Ihres VBA-Projekts sehr wichtig sein können. Mit der Zeit kann dadurch die Sammlung der Steuerelemente recht umfangreich werden. VBA bietet in diesem Zusammenhang die Möglichkeit, die Steuerelemente auf mehrere Reiter zu verteilen. Dazu müssen Sie nur mit der rechten Maustaste auf den Reiter klicken, in diesem Fall also STEUERELEMENTE, und aus dem folgenden Kontextmenü den Punkt NEUE SEITE aufrufen. Dem Fenster wird ein neuer Reiter hinzugefügt und die dazugehörige Seite wird aktiviert. Sie können nun dieser Seite Ihre ganz persönliche Sammlung von Steuerelementen hinzufügen.

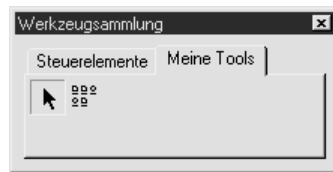


Abbildung 3.5: Neue Seite im Werkzeugfenster

Tipp

Wenn Sie bereits Steuerelemente auf einer anderen, z.B. der ersten, Seite platziert haben, diese aber gerne auf der eigenen Seite ablegen möchten, dann müssen Sie einfach den Mauszeiger über das Symbol bewegen, dann die linke Maustaste betätigen und das Symbol mit gedrückter Maustaste auf die entsprechende Seite schieben.

Je nachdem, ob Sie die **[Strg]**-Taste gedrückt halten oder nicht, wird eine Kopie des Symbols erstellt oder das Symbol einfach nur verschoben.

3.2 VBA-Standardsteuerelemente

Wie bereits erwähnt, stellt VBA für den Formularentwurf bereits eine ganze Reihe von Steuerelementen zur Verfügung. Diese sind alle in der Steuerelementbibliothek Microsoft Forms 2.0 (FM20.DLL) enthalten. Diese Datei wird bereits bei der Installation von AutoCAD in Ihr System integriert. Im Folgenden erfahren Sie Genaueres über die Funktion und die Verwendung dieser Standardelemente.

3.2.1 Gemeinsame Eigenschaften

Diese MSForm-Steuerelemente besitzen alle eine Anzahl an gemeinsamen Merkmalen, die vorab kurz vorgestellt werden sollen, ohne jedoch zu sehr ins Detail zu gehen. Mithilfe dieser Eigenschaften lässt sich die Darstellung der einzelnen Steuerelemente erheblich beeinflussen.

Farbeigenschaften

Jedes Steuerelement besitzt einige Eigenschaften zur Anpassung der farblichen Gestaltung. Die wichtigsten sind *BackColor* und *ForeColor*. Die *BackColor*-Eigenschaft setzt dabei die Hintergrundfarbe, *ForeColor* dementsprechend die Farbe des Vordergrunds. Der Vordergrund entspricht dem angezeigten Element, das abhängig vom Zweck des Steuerelements ist. Bei einem Label, einem Textfeld oder Listenfeld wäre dies der angezeigte Text, bei einer Bildlaufleiste die beiden Pfeile und bei einem Rahmen der Titeltext des Rahmens. Bei Steuerelementen, die andere Elemente aufnehmen können, beispielsweise *Rahmen*, beeinflusst die eingestellte Vorder- und Hintergrundfarbe zusätzlich die Farbeigenschaften der Steuerelemente, die neu in dem entsprechenden Container platziert werden. Die Farbe selbst wird durch eine Ganzzahl dargestellt. Jede Zahl entspricht dabei einem bestimmten Farbwert. Dieser Farbwert setzt sich aus unterschiedlichen Teilen zusammen. Der erste entspricht dem Farbtyp, mit dem zwischen RGB-Farben und den Systemfarben unterschieden wird. In den folgenden zwei Tabellen sind einige Beispiele für die Zusammensetzung des Farbwerts aufgeführt.

Farbtyp	Blau	Grün	Rot	entspricht der Farbe
&H00	FF	FF	FF	Weiß
&H00	00	00	00	Schwarz
&H00	FF	00	00	Blau
&H00	00	FF	FF	Gelb

Tabelle 3.1: Beispiele für die Verwendung von RGB-Farben

Wenn Sie mit den Systemfarben arbeiten, werden die Farben verwendet, die in der Systemsteuerung unter ANZEIGE→DARSTELLUNG eingestellt wurden. Jede Farbe entspricht dabei einer Farbnummer.

Farbtyp	Farbnummer	entspricht der Farbe
&H80	000002	der Aktiven Titelleiste
&H80	000004	der Menüleiste
&H80	00000F	einer Schaltfläche
&H80	000012	eines Schaltflächentextes

Tabelle 3.2: Beispiele für die Verwendung von Systemfarben

Tipp

Für die Umwandlung von RGB-Farben in eine Ganzzahl können Sie die Funktion *RGB* verwenden. Diese erzeugt aus den Angaben Rot, Grün, Blau den entsprechenden Farbwert.

Anpassung des Mauszeigers

Mit diesen Eigenschaften steuern Sie die Darstellung des Mauszeigers, sobald dieser über dem entsprechenden Steuerelement positioniert wird. Die *MousePointer*-Eigenschaft bestimmt, welcher Mauszeiger zum Einsatz kommen soll. Sie können entweder einen Mauszeiger wählen, der in der Systemsteuerung unter **MAUS**→**ZEIGER** eingerichtet ist, oder einen benutzerdefinierten Mauszeiger einstellen. Wenn Sie den benutzerdefinierten gewählt haben, wird der Zeiger verwendet, den Sie in der *MouseIcon*-Eigenschaft festlegen können.

Spezielle Darstellung des Steuerelements

Bei einigen Steuerelementen können Sie eine zusätzliche optische Anpassung vornehmen. Das Steuerelement lässt sich so einstellen, dass es z.B. flach, hervorgehoben oder in das Formular eingelassen erscheint. Diese Eigenschaft gilt allerdings nicht für alle Steuerelemente und ist bei einigen etwas eingeschränkt. Die zulässigen Darstellungsmöglichkeiten sind in der Tabelle 3.3 aufgeführt.

Wert	Darstellung
0 – <i>fmSpecialEffectFlat</i>	
1 – <i>fmSpecialEffectRaised</i>	
2 – <i>fmSpecialEffectSunken</i>	
3 – <i>fmSpecialEffectEtched</i>	
6 – <i>fmSpecialEffectBump</i>	

Tabelle 3.3: Konstanten der Eigenschaft *SpecialEffect*

Steuerelementposition und -größe

Die Größe eines Steuerelements, also die Ausdehnung in X und Y wird durch die Eigenschaften *Width* (Breite) und *Height* (Höhe) bestimmt. Diese Werte werden von der linken oberen Ecke des Steuerelements gemessen.

Die Position des Steuerelements ergibt sich aus den Angaben *Left* und *Top*. Sie geben die Position vom linken (*Left*) und oberen (*Top*) Rand des Containers an, in dem sie abgelegt wurden. Wenn Sie z.B. Steuerelemente innerhalb eines Rahmens einfügen, werden die Abstandswerte vom Rand des Rahmens und nicht vom Formular aus gemessen. Um die absolute Position innerhalb des Formulars zu erhalten, müssten Sie die Abstände des Rahmens und die des Steuerelements zusammenfassen.

`AbsLeft = Frame1.Left + TextBox1.Left`

Die Werte, die Sie für diese Eigenschaften verwenden, werden in der Einheit *Punkt* gemessen, wobei ein *Punkt* etwa 0,35 mm entspricht.

Tabulatoreigenschaften

Mit der Eigenschaft *TabStop* können Sie festlegen, ob ein Steuerelement den Fokus erhalten soll, wenn der Anwender sich mithilfe der Tabulatortaste von Steuerelement zu Steuerelement bewegt. Die *TabIndex*-Eigenschaft legt dafür zusätzlich die Reihenfolge der Ansteuerung der einzelnen Steuerelemente fest. Zuerst erhält das Steuerelement den Fokus, dessen *TabIndex*-

Eigenschaft den niedrigsten Wert besitzt. Bei Betätigung der Tabulatortaste wird dann immer das Steuerelement mit dem nächsthöheren Wert angesteuert, sofern dessen *TabStop*-Eigenschaft auf *TRUE* gesetzt wurde. Zu beachten ist dabei noch, dass jeder Container eigene *TabIndex*-Werte verwaltet. Sie könnten also innerhalb eines Rahmens wieder mit dem Index 0 beginnen.

Font-Eigenschaft

Diese Eigenschaft gilt für alle Steuerelemente, die Text darstellen, und bestimmt das Font-Objekt. Dieses Objekt enthält alle Informationen wie Schriftart, -höhe, -schnitt usw. zur Darstellung des Texts. Wenn Sie für diese Eigenschaft *Null* festlegen, dann wird die Standardschrift des Systems verwendet.

Steuerelemente sperren

VBA stellt zwei Möglichkeiten zur Verfügung, Steuerelemente für den Anwender unzugänglich zu machen, ohne diese auszublenden. Mit der Eigenschaft *Enabled* lässt sich das Steuerelement so einstellen, dass es nicht mehr den Fokus erhalten kann. Wenn Sie diese Eigenschaft auf *FALSE* setzen, dann wird es abgeschwächt dargestellt und das Element kann durch den Anwender nicht mehr aktiviert werden. Die zweite Eigenschaft *Locked* sperrt das Steuerelement für Benutzereingaben, gilt allerdings nicht für alle Steuerelemente. Wenn Sie diese Eigenschaft auf *TRUE* setzen, dann kann der Anwender zwar das Steuerelement aktivieren (den Fokus darauf setzen), die Änderung des jeweiligen Werts ist aber nicht mehr möglich.

Visibile-Eigenschaft

Diese Eigenschaft steuert die Sichtbarkeit des Steuerelements. Ist diese Eigenschaft auf *TRUE* gesetzt, erscheint das Steuerelement auf dem Formular. Ist sie hingegen *FALSE* eingestellt, verschwindet das Steuerelement zur Laufzeit vom Formular, es wird unsichtbar. Zur Entwurfszeit hingegen bleibt es sichtbar. Auch wenn das Steuerelement unsichtbar ist, so bleibt es dennoch aus dem Code heraus ansprechbar.

Tag-Eigenschaft

Jedes Steuerelement besitzt eine *Tag*-Eigenschaft, in der zusätzliche Informationen abgespeichert werden können.

HelpContextID

Diese Eigenschaft legt die ID fest, die auf ein Thema der Hilfedatei verweist. Wenn ein Steuerelement den Fokus erhält und der Anwender anschließend F1 betätigt, wird das entsprechende Hilfethema aufgerufen.

3.2.2 Gemeinsame Methoden

SetFocus

Mit dieser Anweisung können Sie zur Laufzeit den Eingabefokus auf das Steuerelement setzen, mit dem diese Anweisung ausgeführt wird.

`TextBox1.SetFocus`

Move

Diese Anweisung verschiebt das Steuerelement auf die angegebenen Positionswerte und stellt seine Breite bzw. Höhe ein. Die Verwendung dieser Anweisung macht allerdings nur dann Sinn, wenn Sie die komplette Position und/oder Größe des Steuerelements anpassen wollen. Da jeder Wert optional angegeben werden kann, könnten Sie hiermit auch nur die Breite bzw. nur den Abstand vom Rand festlegen (siehe auch *Steuerelementposition* und *-größe*).

3.2.3 Gemeinsame Ereignisse

MouseMove

Wenn der Anwender den Mauszeiger über dem Steuerelement bewegt, dann wird dieses Ereignis ausgelöst. Die Parameter nennen die Position (X, Y), die der Mauszeiger zum Zeitpunkt des Ereignisses hatte, sowie die Taste der Maus (Parameter *Button*) und der Tastatur (Parameter *Shift*), die gedrückt wurde.

Maustastenwerte des Parameters *Button*

1 – <i>fmButtonLeft</i>	Linke Maustaste wurde gedrückt.
2 – <i>fmButtonRight</i>	Rechte Maustaste wurde gedrückt.
4 – <i>fmButtonMiddle</i>	Mittlere Maustaste wurde gedrückt.

Wurden zwei oder drei Tasten gleichzeitig gedrückt, dann beinhaltet der Parameter eine Kombination der Maustastenwerte, z.B. linke und rechte Maustaste → *Button* = 3. Auswerten könnten Sie den Parameter mit einem logischen Vergleich (*Kapitel 2.6*):

```
If (Button And 2) = 2 Then MsgBox "rechte Taste gedrückt"
```

Die folgende Tabelle enthält alle Werte, die der Parameter *Shift* zurückgeben kann.

Tastenwerte des Parameters *Shift*

1	 wurde gedrückt.
2	 wurde gedrückt.
3	 +  wurden gedrückt.
4	 wurde gedrückt.
5	 +  wurden gedrückt.
6	 +  wurden gedrückt.
7	 +  +  wurden gedrückt.

MouseDown/MouseUp

Sobald der Benutzer eine der Maustasten drückt, wird das *MouseDown*-Ereignis ausgelöst. Wenn er dann die Taste wieder loslässt, tritt das *MouseUp*-Ereignis auf. Die Parameter, die dieses Ereignis liefern kann, entsprechen denen des Ereignisses *MouseMove*.

Enter/Exit

Wenn ein Steuerelement den Fokus erhalten kann, dann können auch diese Ereignisse ausgelöst werden. *Enter* wird ausgelöst, sobald ein Steuerelement den Fokus erhält, *Exit* wird aktiv, wenn es den Fokus wieder verliert.

3.2.4 Bezeichnungsfeld (Label)

Das Bezeichnungsfeld dient in erster Linie der Darstellung von beschreibendem Text für Elemente, die diese Möglichkeit selbst nicht besitzen. In Bezeichnungsfeldern können aber auch Bilder dargestellt werden. Der beschreibende Text wird in der Eigenschaft *Caption* festgelegt, für ein darzustellendes Bild dient die Eigenschaft *Picture*. Darüber hinaus enthält dieses Steuerelement weitere Eigenschaften zur Darstellung auf dem Formular. Der anzuzeigende Text kann z. B. wahlweise linksbündig, rechtsbündig oder zentriert dargestellt werden, was Sie über die Eigenschaft *TextAlign* festlegen. Außerdem ist es möglich, verschiedene optische Formatierungen über die Eigenschaft *SpecialEffect* vorzunehmen. Zusätzlich besitzt dieses Steuerelement die Eigenschaft *AutoSize* zur automatischen Anpassung der Größe an den Inhalt des Labels und *Picture* und *PicturePosition* zur Darstellung eines Bilds im Label. Wenn Sie der Eigenschaft *WordWrap* den Wert *TRUE* zuweisen, wird der Text auch mehrzeilig angezeigt.

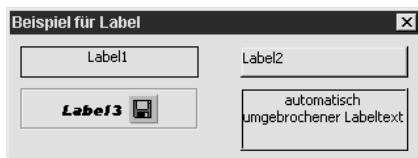


Abbildung 3.6: Bezeichnungsfelder

Tipp

Sie können die Größe des Labels zur Entwurfszeit auch automatisch anpassen lassen, ohne die Eigenschaft *AutoSize* zu verwenden, indem Sie einen Doppelklick auf die Marken zur Einstellung der Größe durchführen. Diese Methode funktioniert bei fast allen Steuerelementen.

Eigenschaften	
<i>Accelerator</i>	Legt eine Zugriffstaste fest. Den Fokus erhält allerdings nicht das Label sondern das Steuerelement, das dem Label folgt.
<i>AutoSize</i>	Passt die Größe an den Text an
<i>Caption</i>	Beschriftung des Bezeichnungsfeldes
<i>ControlTipText</i>	Gelber InfoText
<i>Picture</i>	Ordnet dem Bezeichnungsfeld ein Bild zu
<i>PicturePosition</i>	Position des Bildes
<i>SpecialEffect</i>	Verschiedene Formen der 3D-Darstellung
<i>TextAlign</i>	Textausrichtung des Bezeichnungsfeldes
Ereignisse	
<i>Click</i>	Das Bezeichnungsfeld wurde angeklickt.

3.2.5 Textfeld (TextBox)

Das Textfeld dient der Anzeige von Daten, die beispielsweise durch den Benutzer eingegeben wurden oder aus einer Datenquelle stammen. Dabei kann das Textfeld an die Datenquelle gebunden werden, so dass Änderungen am Text sofort an die Datenquelle übertragen werden. Textfelder bieten zwei Möglichkeiten, den im Textfeld dargestellten Text abzurufen oder zu schreiben. Die standardmäßig dafür vorgesehene Eigenschaft ist *Text*. Sie können die gleiche Wirkung aber auch mit *Value* erzielen. Eine Besonderheit von Textfeldern ist die mögliche Darstellung als Passworteingabefeld. Wenn Sie in der Eigenschaft *PasswordChar* ein bestimmtes Zeichen festlegen, vorzugsweise könnte dies das Multiplikationszeichen (*) sein, dann werden statt des eingegebenen Texts immer nur diese Zeichen sichtbar sein. Der eigentliche Text lässt sich aber trotzdem über die *Text/Value*-Eigenschaft abrufen. Für die mehrzeilige Darstellung in der Textbox müssen Sie die Eigenschaft *MultiLine* auf *TRUE* setzen und gegebenenfalls die Bildlaufleisten über die Eigenschaft *ScrollBars* einstellen.

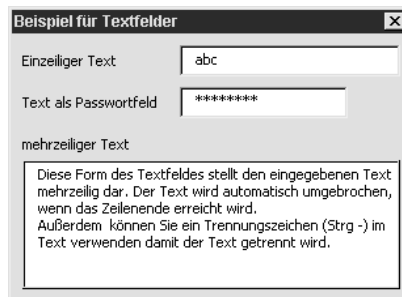


Abbildung 3.7: Textfelder

Mit der Eigenschaft *AutoWordSelect* legen Sie fest, ob die Auswahl auf das gesamte Wort erweitert werden soll, wenn der Anwender die Selektierung z.B. in der Mitte des Worts beginnt. Für einzeilige Texte können Sie die Eigenschaft *SelectionMargin* auf *FALSE* setzen. Dadurch verschwindet der Zwischenraum, der zwischen Textfeldanfang und dem ersten Buchstaben angezeigt wird.

Zusätzlich verfügt das Textfeld über Eigenschaften, die nur zur Laufzeit verfügbar sind und dementsprechend nur über den Quellcode angesprochen werden können. Hierbei handelt es sich um Eigenschaften, die eine Selektierung des eingegebenen Texts vornehmen oder den selektierten Text verarbeiten. Mit *SelStart* kann der Startpunkt der Selektierung ausgelesen bzw. festgelegt werden. *SelLength* legt die Länge des selektierten Texts fest. *SelText* beinhaltet den Text, der markiert wurde. Die Eigenschaften werden häufig kombiniert, um den eingegebenen Text zu markieren und/oder anzupassen. Im Folgenden sehen Sie einige Zeilen, die die Verwendung der verschiedenen *Sel*-Eigenschaften demonstrieren.

```
With TextBox1
    .SelStart = 8
    .SelLength = 1
    .SelText = "eingefügter Text"
    .SelStart = 0
    .SelLength = Len(.Text)
    .SelText = "überschriebener Text"
End With
```

Um den markierten Text über den Code in die Zwischenablage zu kopieren oder aus der Zwischenablage herauszuholen, können Sie die Methoden *Copy*, *Cut* und *Paste* verwenden.

```
With TextBox1
    .SelStart = 0
    .SelLength = Len(.Text)
    .Copy
End With
With TextBox2
    .SelStart = 0
    .Paste
End With
```

Das Standardereignis einer *TextBox* ist *Change*. Da dieses Ereignis immer dann ausgelöst wird, wenn der Benutzer eine Änderung am Text vornimmt, also bei Eingabe oder Löschung eines einzelnen Zeichens, können Sie auch die Ereignisse *AfterUpdate* oder *BeforeUpdate* verwenden, um auf Änderungen am Text zu reagieren.

Das *KeyPress*-Ereignis wird immer dann ausgelöst, wenn der Benutzer eine alphanumerische Taste sowie Esc oder Kombinationen mit Strg drückt. Der ANSI-Code der gedrückten Taste wird im Parameter der Prozedur übermittelt. Da dieses Ereignis nicht alle Tasten abdeckt, müssen Sie zur Überwachung der Tastatur die Ereignisse *KeyDown* und *KeyUp* verwenden. Diese reagieren ebenfalls auf die Betätigung von Tasten, *KeyDown* auf das Herunterdrücken und *KeyUp* auf das Loslassen, nur dass bei diesen Ereignissen alle Tasten, bis hin zu den Tasten Strg und ⇧, berücksichtigt werden. Als Parameter werden der Tastaturcode der gedrückten Taste sowie der Zustand der ⇧-, Strg- und Alt-Taste übermittelt.

Weitere Eigenschaften	
<i>AutoSize</i>	Passt die Größe an den Text an.
<i>EnterKeyBehavior</i>	Legt die Wirkung der Eingabetaste im mehrzeiligen Textfeld fest.
<i>HideSelection</i>	Legt fest, ob der hervorgehobene Text dies auch bleibt, wenn das Textfeld den Fokus verliert.
<i>IntegralHeight</i>	Steuert die Anpassung der Größe der Textbox, damit die Zeilen vollständig in der Höhe angezeigt werden.
<i>MaxLength</i>	Legt die maximale Anzahl an Zeichen fest, die ein Benutzer im Textfeld eingeben kann.
<i>TabKeyBehavior</i>	Legt fest, ob Tabulatoren im Textfeld erlaubt werden, wenn die Eigenschaft <i>MultiLine</i> auf <i>TRUE</i> gesetzt wurde.
<i>WordWrap</i>	Steuert den Umbruch des Texts bei Erreichen des Zeilenendes.

3.2.6 Listenfeld (ListBox)

Dieses Steuerelement dient der Darstellung einer Liste von Werten, aus der der Benutzer dann einen oder mehrere Einträge auswählen kann. Aufgrund der verschiedenen Darstellungsarten, die über die Eigenschaften der *ListBox* festgelegt werden können, ist dieses Steuerelement viel-

seitig verwendbar. Ein großer Nachteil gegenüber Listenfeldern in VB6 ist die fehlende Möglichkeit zur Sortierung der Listeneinträge. Außerdem können Listenfeldeinträge nur zur Laufzeit hinzugefügt werden.

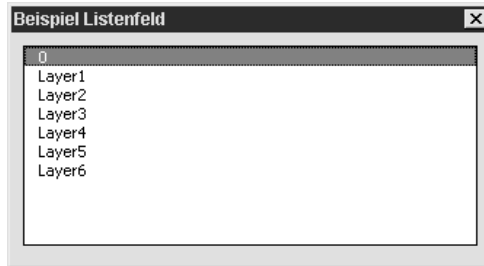


Abbildung 3.8: Listenfeld

Wenn die Einträge nicht mehr vollständig im Anzeigebereich zu sehen sind, werden die entsprechenden Bildlaufleisten automatisch eingeblendet.

Listeneinträge verwalten

Die Einträge des Listenfelds werden in der Eigenschaft *List(n)* verwaltet. Zum Hinzufügen eines Eintrags dient die Anweisung *AddItem*. Im ersten Parameter legen Sie die Zeichenfolge fest, die dem Listenfeld hinzugefügt werden soll. Mit dem zweiten Parameter können Sie eine Position bestimmen, wo der neue Eintrag in der Liste eingefügt werden soll.

```
Private Sub UserForm_Initialize()
Dim ActLayer As AcadLayer
For Each ActLayer In ThisDrawing.Layers
    ListBox1.AddItem ActLayer.Name
Next
End Sub
```

Wenn Sie die Elemente eines Arrays hinzufügen wollen, müssen Sie nicht die *AddItem*-Anweisung verwenden. Sie können in diesem Fall einfach das Array an die Eigenschaft *List()* übergeben.

```
Private Sub UserForm_Initialize()
Dim ActLayer As AcadLayer
Dim LayerList
ReDim LayerList(ThisDrawing.Layers.Count - 1) As String
For Each ActLayer In ThisDrawing.Layers
    LayerList(i) = ActLayer.Name
    i = i + 1
Next
ListBox1.List = LayerList
End Sub
```

Der Abruf und die Änderung der Einträge erfolgt ebenfalls über die *List()*-Eigenschaft.

Abruf eines Eintrags:

```
EintragX = ListBox1.List(x)
```

Änderung eines Eintrags:

```
ListBox1.List(x) = EintragX
```

Die Anzahl der im Listenfeld enthaltenen Einträge können Sie über die Eigenschaft *ListCount* ermitteln. Mit *RemoveItem* kann unter Angabe der Position ein einzelner Eintrag aus der Liste entfernt werden. Die Anweisung *Clear* hingegen löscht den gesamten Listeninhalt.

Die Nummerierung der Listenelemente beginnt mit dem Index 0 und endet mit *ListCount - 1*. Der Eintrag des aktuell markierten Listeneintrags lässt sich über zwei verschiedene Methoden abrufen. Die erste ist eine Kombination der Eigenschaften *List* und *ListIndex* (legt die Position des gewählten *List*-Eintrags fest).

```
SelectedItem = ListBox1.List(ListBox1.ListIndex)
```

Sie können den Eintrag aber auch einfach über die *Text*-Eigenschaft auslesen.

Mehrfachauswahl

Durch Umstellen der Eigenschaft *MultiSelect* ermöglichen Sie dem Anwender eine Mehrfachauswahl von Listeneinträgen. Diese Eigenschaft bietet hierfür zwei verschiedene Versionen. Wenn Sie die Eigenschaft auf die normale Mehrfachauswahl (*fmMultiSelectMulti*) stellen, dann werden die Einträge durch Mausklick oder Betätigung der Leertaste ausgewählt bzw. abgewählt. Hierbei lohnt sich eventuell die Einstellung der Kontrollkästchen, indem Sie für die *ListStyle*-Eigenschaft den Wert *fmListStyleOption* festlegen. Die zweite Version arbeitet bei der Auswahl der Einträge nach dem Windows-Standard (Auswahl durch gleichzeitiges Betätigen der Maustaste und ☐ bzw. ☐ *Strg*). Um diese Listenform zu erhalten, müssen Sie *MultiSelect* auf den Wert *fmMultiSelectExtended* einstellen.

Die zuvor besprochenen Eigenschaften *ListIndex* und *Text* behalten ihre Gültigkeit. Um die selektierten Einträge zu ermitteln, müssen Sie die gesamte Liste Eintrag für Eintrag durchgehen und die Einstellung der *Selected()*-Eigenschaft überprüfen. Gibt diese den Wert *TRUE* zurück, wurde der Eintrag ausgewählt. Der umgekehrte Weg funktioniert natürlich entsprechend. Durch Zuweisung des Werts *TRUE* bzw. *FALSE* an die *Selected()*-Eigenschaft legen Sie fest, welcher Eintrag selektiert werden soll und welcher nicht.

```
For I = 0 To ListBox1.ListCount - 1
    If ListBox1.Selected(i) = True Then
        'Eintrag wurde ausgewählt
    End If
Next
```

Weitere Eigenschaften

*BoundColumn**ColumnCount**ColumnHeads**ColumnWidths**IntegralHeight**MatchEntry*

Ereignisse	
<i>Click</i>	Ein Listeneintrag wurde ausgewählt.
<i>DblClick</i>	Doppelklick auf einen Listeneintrag.

3.2.7 Aufklappbares Listenfeld (ComboBox)

Die ComboBox ist sozusagen die einzeilige Version des Listenfelds. Die Einträge werden auf die gleiche Weise hinzugefügt, abgerufen und entfernt wie beim Listenfeld. Die ComboBox stellt zwei verschiedene Typen zur Verfügung, die durch die entsprechende Einstellung der *Style*-Eigenschaft festgelegt werden. Der erste Typ (*Style=fmStyleDropDownCombo*) ermöglicht die Eingabe eines beliebigen Texts im Eingabefeld der *ComboBox*. Der zweite Typ (*Style=fmStyleDropDownList*) vervollständigt die Eingabe des Texts durch einen bereits vorhandenen Eintrag. Die Eingabe eines nicht im Listenfeld enthaltenen Texts ist mit dieser Einstellung nicht möglich. Eine optische Veränderung der Listeneinträge erreichen Sie durch Anpassung der Eigenschaft *ListStyle*. Mithilfe des Werts *fmListStyleOption* wird jeder Eintrag wie ein Optionsfeld dargestellt.

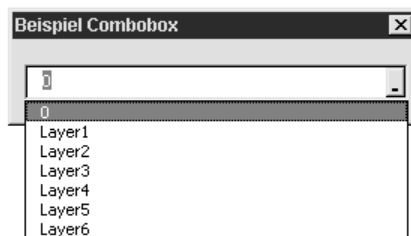


Abbildung 3.9: Aufklappbares Listenfeld

Die Breite der ausgeklappten Liste lässt sich mit *ListWidth* anpassen. Der Wert, den Sie hier festlegen, wird in der Einheit Punkt gemessen. Wenn Sie die Breite auf den Wert 0 einstellen, dann wird das Listenfeld immer genauso breit dargestellt wie das Textfeld inklusive der Schaltfläche.

Die Darstellung der Schaltfläche zum Aufklappen der Liste lässt sich ebenfalls anpassen. Mit *ShowDropButtonWhen* können Sie z.B. festlegen, wann die Schaltfläche angezeigt werden soll.

Steuerung der Sichtbarkeit der Schaltfläche	
<i>fmShowDropButtonWhenNever</i>	Die Schaltfläche wird gar nicht angezeigt. Diese Einstellung sollten Sie nur verwenden, wenn die <i>Style</i> -Eigenschaft den Wert <i>fmStyleDropDownList</i> erhalten hat. Andernfalls hat der Benutzer keine Möglichkeit, die Liste aufzuklappen.
<i>fmShowDropButtonWhenFocus</i>	Die Schaltfläche wird nur angezeigt, wenn die ComboBox den Fokus erhält.
<i>fmShowDropButtonWhenAlways</i>	Die Schaltfläche wird immer angezeigt (Standardeinstellung).

Zusätzlich können Sie noch die optische Darstellung der Schaltfläche festlegen. Durch Einstellung der Eigenschaft *DropButtonStyle* lassen sich folgende Ergebnisse erzielen:

<i>fmDropButtonStylePlain</i>		<i>fmDropButtonStyleEllipsis</i>	
<i>fmDropButtonStyleArrow</i>		<i>fmDropButtonStyleReduce</i>	

Eigenschaften und Ereignisse
Siehe Steuerelement Listenfeld

3.2.8 Kontrollkästchen (CheckBox)

Das Kontrollkästchen-Steuerelement dient dazu, dem Anwender eine Auswahl zwischen zwei Werten nach dem Prinzip Ja/Nein oder Ein/Aus zu ermöglichen. Wenn das Kontrollkästchen aktiviert wurde, dann wird ein Häkchen dargestellt und die *Value*-Eigenschaft besitzt den Wert *TRUE*. Wird kein Häkchen angezeigt, besitzt *Value* den Wert *FALSE*. Diese beiden Einstellungsmöglichkeiten sind bereits zur Entwurfszeit verfügbar. Es gibt aber noch eine dritte Form der Darstellung. Diese könnten Sie z.B. dann verwenden, wenn der Anwender aus einer Liste mehrere Einträge (z.B. Layer-Namen) ausgewählt hat und die Eigenschaft (z.B. frieren/tauen), die durch das Kontrollkästchen dargestellt wird, bei den Einträgen verschiedene Zustände besitzt. Folgt man dem Windows-Standard für diesen Fall, dann muss das Kästchen in einen undefinierten Zustand versetzt werden. Dazu muss nur der Wert *Null* an die *Value*-Eigenschaft vergeben werden.



Abbildung 3.10: Kontrollkästchen

Tip

Da *Value* drei Zustände enthalten kann, bietet diese Eigenschaft keine Auswahlliste, sondern ein Textfeld. Sie müssen die Werte direkt im Eingabefeld der Eigenschaft eingeben (*TRUE*, *FALSE* oder *Null*).

Durch die Einstellung der *TrippleState*-Eigenschaft auf *TRUE* kann auch der Anwender den undefinierten Zustand des Kontrollkästchens zur Laufzeit durch Klicken auf das Kästchen aktivieren.

Eigenschaften	
<i>Accelerator</i>	Legt eine Zugriffstaste für das Optionsfeld fest.
<i>Alignment</i>	Text links oder rechts neben dem Auswahlshalter.
<i>AutoSize</i>	Passt die Größe an den Text an.

Eigenschaften	
<i>Caption</i>	Beschriftung des Optionsfelds.
<i>ControlTipText</i>	Gelber InfoText.
<i>GroupName</i>	Name der Gruppe, der das Optionsfeld zugeordnet werden soll.
<i>Locked</i>	Sperrt das Optionsfeld für Benutzereingaben.
<i>Picture</i>	Ordnet dem Optionsfeld ein Bild zu.
<i>PicturePosition</i>	Position des Bilds.
<i>SpecialEffect</i>	Verschiedene Formen der 3D-Darstellung.
<i>TextAlign</i>	Ausrichtung des Schaltflächentexts.
<i>TrippleState</i>	Einstellung des undefinierten Zustands durch den Anwender zulassen.
<i>Value</i>	Aktueller Zustand.

Ereignisse	
<i>AfterUpdate</i>	Nach Änderung des Zustands des Umschaltfelds.
<i>BeforeUpdate</i>	Bevor der Zustand des Umschaltfelds sich ändert.
<i>Click</i>	Umschaltfeld wird angeklickt.

3.2.9 Optionsfelder (OptionButton)

Optionsfelder ähneln in der Darstellung und Verwendung den Kontrollkästchen. Auch hier können drei verschiedene Zustände eingestellt werden (Ein, Aus oder *Null*). Der Unterschied besteht darin, dass der Benutzer das Feld ausschließlich aktivieren kann. Eine Deaktivierung erfolgt nur, wenn ein anderes Optionsfeld, das zur selben Gruppe gehört, aktiviert wird. Von Optionsfeldern, die zu einer Gruppe gehören, kann der Anwender immer nur eines aktivieren. Alle Optionsfelder, die in ein und demselben Container platziert werden, z.B. direkt im Formular oder in einem Rahmen, gehören derselben Gruppe an. Diesen Zustand können Sie umgehen, indem Sie die *GroupName*-Eigenschaft verwenden. In dieser definieren Sie den Namen der Gruppe, der das Optionsfeld angehören soll. Alle Optionsfelder mit demselben Gruppennamen gehören zusammen.



Abbildung 3.11: Optionsfelder

Die *TrippleState*-Eigenschaft ist für Optionsfelder relativ sinnlos, da der Anwender sowieso nur den Zustand *aktiviert* (*TRUE*) festlegen kann.

Eigenschaften und Ereignisse
Siehe Steuerelement Kontrollkästchen

3.2.10 Umschaltfelder (ToggleButton)

Umschaltfelder werden, genau wie Kontrollkästchen, für die Auswertung eines Zustands verwendet. Der Unterschied ist, dass Umschaltfelder wie Schaltflächen dargestellt werden. Sie können über die *Caption*-Eigenschaft einen Schaltflächentext und zusätzlich ein Schaltflächensymbol (Eigenschaft *Picture*) festlegen. Die Position des Bilds innerhalb der Schaltfläche wird dann über die Eigenschaft *PicturePosition* eingestellt. Die häufigste Anwendungsmethode wird allerdings, wie in der Abbildung zu sehen, die Darstellung des jeweiligen Zustands durch zwei unterschiedliche Bilder sein. Die *Value*-Eigenschaft speichert den aktuellen Zustand der Schaltfläche (*TRUE* = gedrückt, *FALSE* = nicht gedrückt, *Null* = undefiniert). Beim Auswerten oder Setzen dieser Eigenschaft gelten Konventionen der Kontrollkästchen. Wenn Sie mit den Umschaltfeldern nur Symbole verwenden, sollten Sie zumindest eine Beschriftung in der Eigenschaft *ControlTipText* (gelber Infotext) eintragen.



Abbildung 3.12: Umschaltfelder

Wenn Sie zwei Symbole pro Schaltfläche verwenden wollen, um die beiden Zustände optisch zu unterscheiden, dann können Sie das *ImageList*-Steuerelement zur Verwaltung der Symbole benutzen. Die Verwendung dieses speziellen Steuerelements wird in *Kapitel 3.3.2* genauer beschrieben.

Eigenschaften	
<i>Accelerator</i>	Legt eine Zugriffstaste für das Umschaltfeld fest.
<i>AutoSize</i>	Passt die Größe an den Text/die Grafik an.
<i>Caption</i>	Schaltflächentext.
<i>ControlTipText</i>	Gelber InfoText.
<i>Locked</i>	Sperrt die Schaltfläche für Benutzereingaben.
<i>Picture</i>	Schaltflächensymbol.
<i>PicturePosition</i>	Position des Schaltflächensymbols.
<i>TextAlign</i>	Ausrichtung des Schaltflächentexts.
<i>TrippleState</i>	Einstellung des undefinierten Zustands durch den Anwender zulassen.
<i>Value</i>	Aktueller Zustand.

Ereignisse	
<i>AfterUpdate</i>	Nachdem der Zustand des Umschaltfelds sich geändert hat.
<i>BeforeUpdate</i>	Bevor der Zustand des Umschaltfelds sich ändert.
<i>Click</i>	Umschaltfeld wird angeklickt.

3.2.11 Befehlsschaltflächen (CommandButton)

Die Befehlsschaltflächen ähneln in der Darstellung den Umschaltfeldern. Allerdings speichern diese Steuerelemente nicht den Zustand, sondern sie sollen lediglich den Start, das Beenden oder den Abbruch bestimmter Aktionen ausführen. Im Normalfall müssen Sie dabei nur auf das *Click*-Ereignis reagieren, das die Schaltfläche auslöst, wenn der Anwender die Schaltfläche betätigt. Die Verwendung von Schaltflächensymbolen wird genauso über die Eigenschaften *Picture* und *PicturePosition* gesteuert wie bei Umschaltfeldern. Eine Besonderheit weist dieses Steuerelement allerdings noch auf: Sie können über die Eigenschaften *Cancel* und *Default* bestimmen, ob die Schaltfläche durch Betätigen der - und/oder der -Taste ausgeführt werden soll, auch wenn diese nicht den aktuellen Fokus besitzt.

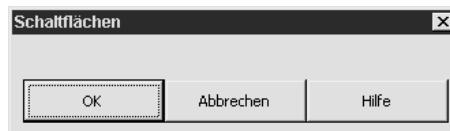


Abbildung 3.13: Schaltflächen

Normalerweise bleibt der Fokus auf der Schaltfläche, wenn der Anwender diese mit der Maus angeklickt hat. Diesen Zustand können Sie ebenfalls beeinflussen, indem Sie die Eigenschaft *TakeFocusOnClick* anpassen. Wenn Sie diese auf den Wert *FALSE* einstellen, dann bleibt der Fokus auf dem Steuerelement, das diesen zuletzt besessen hatte.

Eigenschaften	
<i>AutoSize</i>	Passt die Größe an den Text/die Grafik an.
<i>Cancel</i>	Auswahl durch die  -Taste.
<i>Caption</i>	Schaltflächentext.
<i>ControlTipText</i>	Gelber InfoText.
<i>Default</i>	Auswahl durch die  -Taste.
<i>Locked</i>	Sperrt die Schaltfläche für Benutzereingaben.
<i>Picture</i>	Schaltflächensymbol.
<i>PicturePosition</i>	Position des Schaltflächensymbols.
<i>TakeFocusOnClick</i>	<i>FALSE</i> – Fokus wird nicht auf die Schaltfläche gesetzt. <i>TRUE</i> – Schaltfläche erhält durch Anklicken den Fokus.
Ereignisse	
<i>Click</i>	Schaltfläche wird angeklickt.

3.2.12 Rahmenfelder (Frames)

Mit diesem Steuerelement können Sie eine Gruppe von Steuerelementen optisch zusammenfassen oder einzelne Steuerelemente, z.B. Listenfelder, isolieren. Wenn Sie einen Rahmen verschieben, dann werden alle darin befindlichen Steuerelemente mitverschoben. Der Text, der im Allgemeinen den Inhalt des Rahmens beschreibt, wird in der Eigenschaft *Caption* gespeichert. Zusätzlich lässt sich im Rahmen ein Bild anzeigen, das dann über die *Picture*-Eigenschaft zugeordnet wird. Die Position dieses Bilds ergibt sich wieder aus der Eigenschaft *PicturePosition*. Die Darstellung des Bilds kann aber noch weiter beeinflusst werden. Durch die Änderung der Eigenschaft *PictureSizeMode* können Sie drei verschiedene Anpassungen der Bildgröße vornehmen. Die erste Einstellung (*fmPictureSizeModeClip*) zeigt das Bild in seiner Originalgröße an, durch die zweite Einstellung (*fmPictureSizeModeStretch*) wird das Bild auf die Ausdehnung des Rahmens skaliert und die dritte Einstellung (*fmPictureSizeModeZoom*) passt die Größe des Bilds an die minimal mögliche Ausdehnung des Rahmens an, wobei das Seitenverhältnis des Bilds erhalten bleibt.

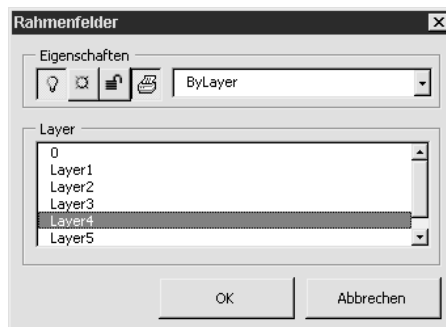


Abbildung 3.14: Rahmenfelder

Eine wirklich gelungene Funktion, die ein Rahmen bietet, ist die Möglichkeit, den Rahmeninhalt zu scrollen. Dazu müssen Sie zunächst die Bildlaufleisten mit der Eigenschaft *ScrollBars* aktivieren. Je nachdem, welche Bildlaufleiste Sie benötigen, können Sie die Konstanten *fmScrollBarsNone* (=0) für keine, *fmScrollBarsHorizontal* (=1) für die horizontale, *fmScrollBarsVertical* (=2) für die vertikale oder *fmScrollBarsBoth* (=3) für beide verwenden. Welche der Bildlaufleisten permanent angezeigt werden soll, lässt sich zusätzlich mit der Eigenschaft *KeepScrollBarsVisible* einstellen. Damit der Rahmen auch weiß, wie weit in den entsprechenden Richtungen gescrollt werden kann, müssen Sie dies dem Rahmen in den Eigenschaften *ScrollHeight* und *ScrollWidth* mitteilen. Die Eigenschaften *ScrollTop* und *ScrollLeft* bestimmen die aktuelle Position des Rahmeninhalts.

Auf die Elemente, die innerhalb des Rahmens positioniert wurden, können Sie entweder über deren Namen direkt oder über die *Controls*-Eigenschaft des Rahmens zugreifen. Der Zugriff über diese Auflistung ist von Vorteil, wenn Sie Elemente zur Laufzeit dem Rahmen hinzufügen. Angenommen, Sie wollen alle Attributwerte eines Blocks anzeigen lassen, dann können Sie die Anzahl der benötigten Textfelder nicht im voraus bestimmen. Sie müssten für jedes Attribut ein eigenes Textfeld zur Laufzeit hinzufügen und nach Beendigung auch wieder entfernen. Durch den Zugriff über die *Controls*-Auflistung gestaltet sich die Programmierung dann sehr flexibel (Beispiel in Kapitel 3.4.2).

Eigenschaften	
<i>ActiveControl</i>	Aktives Steuerelement innerhalb des Rahmens (nur zur Laufzeit verfügbar).
<i>Caption</i>	Rahmenbeschriftung.
<i>Controls</i>	Auflistung aller Rahmenelemente.
<i>InsideHeight/-Width</i>	Höhe und Breite des sichtbaren Rahmenbereichs.
<i>KeepScrollBars Visible</i>	Bildlaufleisten bei Bedarf oder immer anzeigen.
<i>ScrollBars</i>	Steuert die Verwendung der Bildlaufleisten.
<i>ScrollHeight/-Width</i>	Legt die Größe des verschiebbaren Bereichs fest.
<i>ScrollLeft/-Top</i>	Linke obere Ecke des sichtbaren Bereichs.
<i>SpecialEffect</i>	Verschiedene Formen der 3D-Darstellung.
<i>Zoom</i>	Legt einen Vergrößerungsfaktor für die Anzeige des Rahmeninhalts fest.
Methoden	
<i>SetDefaultTabOrder</i>	Legt eine Standardsortierreihenfolge (<i>TabIndex</i>) für den Rahmeninhalt fest. Die Steuerelemente werden danach von oben nach unten und von links nach rechts durchnummeriert.
Ereignisse	
<i>AddControl</i>	Steuerelement wurde hinzugefügt.
<i>RemoveControl</i>	Steuerelement wurde entfernt.
<i>Scroll</i>	Die Bildlaufleisten wurden neu positioniert.

3.2.13 Multiseiten (MultiPage)

Mit den Multiseiten können Sie verschiedene Steuerelemente zu Gruppen zusammenfassen und für jede Gruppe sozusagen einen eigenen Dialogbereich festlegen (siehe AutoCAD-Optionsdialog). Wenn Sie die Steuerelemente auf ein Multiseiten-Steuerelement verteilen, sollten Sie einige Grundregeln beachten, damit das Formular bedienerfreundlich bleibt.

1. Sie sollten es bei der Verwendung dieses Steuerelements nicht mit der Anzahl an Reitern übertreiben. Ansonsten könnte der Benutzer die Übersicht über den Inhalt des Formulars verlieren.
2. Sie sollten die Steuerelemente in den Seiten so anordnen, dass sie inhaltlich zusammenpassen.
3. Die Reiter sollten mit einem kurzen, aber aussagekräftigen Text beschriftet werden, der alles über den Reiterinhalt wiedergibt.
4. Die Schaltflächen zum Verlassen des Formulars sollten immer außerhalb der Multiseiten positioniert werden, damit der Benutzer nicht erst lange suchen muss.



Abbildung 3.15: Multiseiten

Sollten die Reiter nicht in eine Zeile passen, blendet das Multiseiten-Steuerelement automatisch zwei Schaltflächen zur Weiterschaltung auf den nächsten bzw. vorhergehenden Reiter ein. Sie können die Seiten auch so verteilen, dass sie alle sichtbar sind (*MultiRow = TRUE*). Passt ein Reiter dann nicht mehr in die Zeile, wird er in einer neuen Zeile angezeigt.

Seiten zur Entwurfszeit verwalten

Um der Multiseite einen weiteren Reiter hinzuzufügen, klicken Sie mit der rechten Maustaste auf einen vorhandenen Reiter (nicht auf die Seitenfläche) und wählen im Kontextmenü **NEUE SEITE**. Den Titel des Reiters legen Sie mit der *Caption*-Eigenschaft fest. Löschen können Sie eine Seite über die Option **SEITE LÖSCHEN** im Kontextmenü. Die Position, die der Reiter innerhalb der Tab-Auflistung einnehmen soll, lässt sich über *Index* festlegen. Der erste Reiter erhält den Index 0, der zweite den Index 1 usw.

Wenn Sie optische Spielereien mögen, dann können Sie auch die Überblendungseffekte der Multiseite verwenden. Standardmäßig sind diese ausgeschaltet. Sie können über die Eigenschaft *TransitionEffect* festgelegt werden. Die Dauer des festgelegten Effekts wird in der Eigenschaft *TransitionPeriod* eingestellt. Gültig sind Werte zwischen 0 und 100000 Millisekunden.

Nachdem Sie alle gewünschten Reiter festgelegt haben, können Sie die Steuerelemente auf den einzelnen Seiten positionieren. Es werden immer nur die Steuerelemente des aktivierten Reiters angezeigt, so dass die Zusammenstellung der einzelnen Bereiche flott von der Hand geht.

Seiten zur Laufzeit verwalten

Obwohl die Gestaltung des Steuerelements oft im Entwurfsmodus abgeschlossen werden kann, könnte ein Zugriff auf einzelne Seiten zur Laufzeit erforderlich werden, um beispielsweise Seiten in bestimmten Fällen auszublenden oder Steuerelemente hinzuzufügen bzw. zu entfernen. Die einzelnen Blätter werden in der *Pages*-Auflistung verwaltet. Diese Auflistung bietet nun die Methoden *Add* zum Hinzufügen, *Remove* zum Entfernen und *Item* zum Abrufen von einzelnen Blättern. Zusätzlich könnten Sie auch alle Blätter auf einmal entfernen, indem Sie die *Clear*-Anweisung ausführen.

Der Zugriff erfolgt wie gewohnt über die Angabe des Index der entsprechenden Seite.

```
Set Page1 = MultiPage1.Pages(0)
```

Da *Pages* die Standardeigenschaft des Steuerelements ist, können Sie *Pages* auch weglassen.

```
Set Page1 = MultiPage1(0)
```

In der Objektvariablen *Page1* sind nun alle Eigenschaften und Steuerelemente enthalten, die das erste Blatt umfasst. Um auf das aktuell gewählte Blatt zuzugreifen, bedienen Sie sich der Eigenschaft *Value*, die dessen Index gespeichert hat.

Über die *Controls*-Auflistung des einzelnen Blatts kann nun wieder auf die einzelnen Steuerelemente zugegriffen werden, ganz genau wie bei einem Rahmenfeld.

Wo gerade die Rahmen erwähnt werden, auch das Multiseiten-Steuerelement bietet die Möglichkeit der Einblendung der Bildlaufleisten und damit des Verschiebens des Inhalts eines Blatts. Die Eigenschaften, die hierfür zum Einsatz kommen, entsprechen denen des Rahmenfelds.

Tipp

Wenn Sie zur Entwurfszeit auf das Multiseiten-Steuerelement zugreifen wollen (nicht die einzelnen Seiten), dann müssen Sie entweder auf den markierten Randbereich des Steuerelements klicken oder die Multiseite im Eigenschaftenfenster wählen.

Multiseiten – Eigenschaften

<i>MultiRow</i>	Legt fest, ob die Reiter auf mehrere Zeilen verteilt werden können.
<i>Style</i>	Legt die Darstellung der Reiter fest (Tab oder Button).
<i>Pages</i>	Auflistung aller Rahmenelemente.
<i>TabOrientation</i>	Legt fest, ob die Reiter oberhalb, unterhalb, links oder rechts von der Seite angezeigt werden.

Multiseiten – Ereignisse

<i>AddControl</i>	Steuerelement wurde hinzugefügt.
<i>RemoveControl</i>	Steuerelement wurde entfernt.
<i>Scroll</i>	Die Bildlaufleisten wurden neu positioniert.

Eigenschaften

<i>Accelerator</i>	Legt eine Zugriffstaste der Seite fest.
<i>Caption</i>	Seitentitel.
<i>Controls</i>	Auflistung aller Seitenelemente.
<i>InsideHeight/Width</i>	Höhe und Breite des sichtbaren Seitenbereichs.
<i>KeepScrollBarsVisible</i>	Bildlaufleisten bei Bedarf oder immer anzeigen.
<i>ScrollBars</i>	Steuert die Verwendung der Bildlaufleisten.
<i>ScrollHeight/Width</i>	Legt die Größe des verschiebbaren Bereichs fest.
<i>ScrollLeft/Top</i>	Linke obere Ecke des sichtbaren Bereichs.
<i>SpecialEffect</i>	Verschiedene Formen der 3D-Darstellung.
<i>Zoom</i>	Legt einen Vergrößerungsfaktor für die Anzeige des Rahmeninhalts fest.

Seiten – Methoden

<i>SetDefaultTabOrder</i>	Legt eine Standardsortierreihenfolge (<i>TabIndex</i>) für den Seiteninhalt fest. Die Steuerelemente werden danach von oben nach unten und von links nach rechts durchnummeriert.
<i>Accelerator</i>	Legt eine Zugriffstaste für das Umschaltfeld fest.

Die Bibliothek enthält zwar noch das *TabStrip*-Steuerelement, was aber hier außer Acht gelassen werden soll, da dieses Steuerelement der Multiseite sehr ähnlich ist, dem gleichen Zweck dient, aber in der Verwendung viel umständlicher zu handhaben ist. Sie sollten dem *TabStrip*-Steuerelement immer die Multiseite vorziehen.

3.2.14 Bildlaufleisten (ScrollBars)

Bildlaufleisten werden im Allgemeinen dafür verwendet, dem Anwender die Steuerung eines bestimmten Wertebereichs zu ermöglichen. Solche Bildlaufleisten haben Sie ja bereits bei den Rahmenfeld- oder MultiSeiten-Steuerelementen kennen gelernt. Wenn Sie die Bildlaufleiste auf dem Formular positionieren, ist die Eigenschaft *Orientation* auf den Wert *fmOrientationAuto* eingestellt. Dadurch passt die Bildlaufleiste die Ausrichtung (horizontal/vertikal) automatisch an. Ist die Breite größer als die Höhe, wird sie horizontal dargestellt, andernfalls vertikal. Um eine konkrete Ausrichtung (horizontal/vertikal) zu erhalten, können Sie die Werte *fmOrientationHorizontal* und *fmOrientationVertical* verwenden.

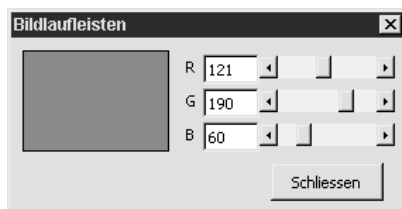


Abbildung 3.16: Bildlaufleisten

Die wichtigsten Eigenschaften, die dieses Steuerelement zur Verfügung stellt, sind *Min* und *Max*, für den minimalen und den maximalen Wert, sowie *SmallChange* und *LargeChange*, die den Umfang der Bewegungen des Durchlaufpfeils steuern. Der aktuelle Wert, der sich zwischen den Werten der Eigenschaften *Min* und *Max* bewegt, ist über die Eigenschaft *Value* abrufbar. Sie können außerdem eine Verzögerung für die *Change*-Ereignisse über die *Delay*-Eigenschaft festlegen. Diese beeinflusst die Dauer zwischen den einzelnen Klickereignissen, wenn der Benutzer die Maustaste gedrückt hält. Der Standardwert dieser Eigenschaft beträgt 50 Millisekunden. Dieser Wert bewirkt, dass bis zum zweiten Auslösen des Ereignisses fünfmal so viel Zeit vergehen würde, wie eingestellt wurde, und ab da immer der eingestellte Wert bis zum wiederholten Auslösen des *Click*-Ereignisses. Vereinfacht lässt sich dies wie folgt formulieren: Je höher der Wert eingestellt wird, umso langsamer läuft das Bildlauffeld durch die Bildlaufleiste.

Mit der *ProportionalThumb*-Eigenschaft können Sie außerdem die Größe des Bildlauffelds steuern. Wenn Sie den Wert *TRUE* einstellen, passt sich die Höhe bzw. Breite proportional den Werten der Eigenschaften *Min* und *Max* an.

Die beiden wichtigsten Ereignisse in Zusammenhang mit diesem Steuerelement sind *Change* und *Scroll*. Das *Change*-Ereignis wird immer dann ausgelöst, wenn sich der Wert der *Value*-Eigenschaft ändert. Da dies nicht geschieht, solange Sie das Bildlauffeld bewegen, müssen Sie auf das *Scroll*-Ereignis zurückgreifen. Dieses Ereignis wird aber nicht ausgelöst, wenn Sie auf die Schaltflächen klicken. Deshalb müssen Sie die beiden Ereignisse kombinieren, wie das im Beispiel zur Abbildung 3.16 demonstriert wird. Der Code der einzelnen Bildlaufereignisse ist nahezu identisch, so dass ein Blick auf den Code einer Bildlaufleiste reichen sollte.

Im folgenden die Ereignisprozeduren der Bildlaufleiste die die rote Farbe steuert:

```
Private Sub ScrBarRed_Change()
    TBarRed = ScrBarRed

    Image1.BackColor = RGB(ScrBarRed, ScrBarGreen, ScrBarBlue)
End Sub

Private Sub ScrBarRed_Scroll()
    ScrBarRed_Change
End Sub

Private Sub TBarRed_Change()
    If TBarRed < 0 Then TBarRed = 0
    If TBarRed > 255 Then TBarRed = 255
    ScrBarRed = TBarRed
End Sub
```

Eigenschaften	
<i>ControlTipText</i>	Gelber Infotext.
<i>Delay</i>	Verzögerung zwischen den <i>Click</i> -Ereignissen.
<i>LargeChange</i>	Schrittweite für die seitenweise Änderung.
<i>Min</i>	Minimum des Wertebereichs.
<i>Max</i>	Maximum des Wertebereichs.
<i>Orientation</i>	Ausrichtung der Bildlaufleiste.
<i>ProprtionalThumb</i>	Steuert die Größe des Bildlauffelds.
<i>SmallChange</i>	Schrittweite beim Anklicken eines Buttons.
<i>Value</i>	Aktueller Wert.
Ereignisse	
<i>Change</i>	Der Wert der <i>Value</i> -Eigenschaft wurde geändert.
<i>Scroll</i>	Das Bildlauffeld wurde bewegt.

3.2.15 Drehfeld (SpinButton)

Das Drehfeld-Steuerelement ähnelt in der Verwendung und Bedienung den Bildlaufleisten. Der Unterschied dabei ist, dass ein Drehfeld kein Bildlauffeld besitzt. Die Steuerung erfolgt somit nur über die beiden Schaltflächen. Das Drehfeld lässt sich ebenfalls wahlweise auf eine horizontale oder vertikale Richtung festlegen.

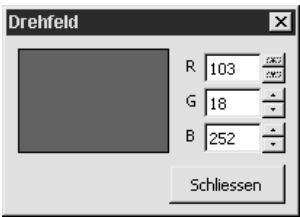


Abbildung 3.17: Beispiel Drehfeld

Da das Drehfeld über kein Bildlauffeld verfügt, besitzt es auch kein Scroll-Ereignis. Dafür können Sie mit den Ereignissen *SpinDown* und *SpinUp* zusätzlich auf die Betätigung der Schaltflächen reagieren. Bei horizontaler Ausrichtung tritt das *SpinUp*-Ereignis auf, wenn der Benutzer auf die rechte Schaltfläche klickt, *SpinDown* kommt bei Betätigung der linken Schaltfläche zum Einsatz.

Das *Change*-Ereignis wird, wie bei den Bildlaufleisten, durch die Änderung der *Value*-Eigenschaft ausgelöst.

Eigenschaften	
<i>ControlTipText</i>	Gelber Infotext.
<i>Delay</i>	Verzögerung zwischen den <i>Click</i> -Ereignissen.
<i>Min</i>	Minimum des Wertebereichs.
<i>Max</i>	Maximum des Wertebereichs.
<i>Orientation</i>	Ausrichtung des Drehfelds.
<i>SmallChange</i>	Schrittweite beim Anklicken eines Buttons.
<i>Value</i>	Aktueller Wert.

Ereignisse	
<i>Change</i>	Der Wert der <i>Value</i> -Eigenschaft wurde geändert.
<i>SpinDown</i>	Die untere bzw. linke Schaltfläche wurde betätigt.
<i>SpinUp</i>	Die obere bzw. rechte Schaltfläche wurde betätigt.

3.2.16 Anzeigefelder (Image)

Anzeigefelder werden zur Darstellung von Bildern verwendet, die mit der *Picture*-Eigenschaft zugeordnet werden. Wenn Sie die Bilder nicht über das Eigenschaftsfenster, sondern erst zur Laufzeit laden möchten, dann verwenden Sie die *LoadPicture*-Methode:

```
Image1.Picture = LoadPicture("Bilddatei")
```

Alternativ können Sie die *Picture*-Eigenschaft eines anderen Steuerelements (z.B. eines aus der Bildsammlung von *ImageList*) zuweisen.

```
Image1.Picture = Image2.Picture
```


Mithilfe einiger Eigenschaften lässt sich die Anzeige des Bilds noch ein wenig anpassen. *PictureAlignment* z.B. legt die Position (oben links, oben rechts, zentriert usw.) des Bilds fest, wenn dieses kleiner als das Anzeigefeld ist. *PictureSizeMode* bestimmt den Anzeigemodus des Bilds. Sie können für diese Eigenschaft drei verschiedene Einstellungsmöglichkeiten wählen:

- *fmPictureSizeModeClip*: In diesem Modus wird das Bild in seiner Originalgröße angezeigt. In diesem Fall können Sie die *AutoSize*-Eigenschaft dafür verwenden, die Größe des Anzeigefelds der des Bilds anzupassen.
- *fmPictureSizeModeStretch*: In diesem Modus wird das Bild der Größe des Anzeigefelds angepasst. Wenn das Anzeigefeld und das Bild unterschiedliche Seitenverhältnisse aufweisen, werden die Proportionen des Bilds angepasst.
- *fmPictureSizeModeZoom*: In diesem Modus wird das Bild auf die kleinste Anzeigefeldausdehnung angepasst. Die Proportionen des Bilds bleiben erhalten.

Mit der *PictureTiling*-Eigenschaft können Sie außerdem eine Mehrfachanzeige des Bilds aktivieren. Durch die Einstellung des Werts auf *TRUE* erreichen Sie, dass das Bild mehrfach neben- und untereinander angezeigt (gekachelt) wird.

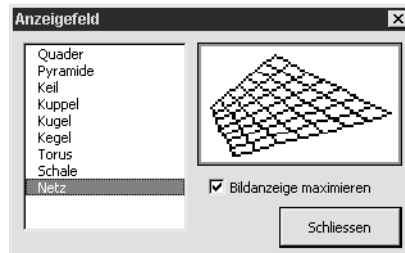


Abbildung 3.18: Beispiel Anzeigefeld

Tip

Wenn Sie zur Entwurfszeit ein zugeordnetes Bild entfernen möchten, dann müssen Sie im Wertefeld der *Picture*-Eigenschaft die Entf-Taste betätigen. Um ein Bild während der Laufzeit zu entfernen, weisen Sie der *Picture*-Eigenschaft den Wert *Null* zu.

Eigenschaften	
<i>AutoSize</i>	Passt die Größe des Anzeigefelds an die Originalgröße des Bilds an.
<i>BorderStyle</i>	Legt die Anzeige einer Umrandung fest.
<i>ControlTipText</i>	Gelber Infotext.
<i>Picture</i>	Hier ist das Bild gespeichert.
<i>PictureAlignment</i>	Bildposition.
<i>PictureSizeMode</i>	Skalierung des Bilds.
<i>PictureTiling</i>	Einzelbild oder Kachelung.
<i>SpecialEffect</i>	Verschiedene Formen der 3D-Darstellung.

3.3 Weitere Steuerelemente

Neben den bereits beschriebenen Steuerelementen der MSForms-Bibliothek gibt es noch eine unzählige Menge an Steuerelementen, die Sie ebenfalls verwenden könnten. Es existieren eine ganze Reihe von Internetseiten, die sich mit diesem speziellen Thema befassen und von denen Sie auch Steuerelemente herunterladen können.

Einige dieser zusätzlichen Steuerelemente möchte ich an dieser Stelle nicht unerwähnt lassen. Es handelt sich dabei um die Microsoft-Windows-Standardsteuerelemente 6.0, die alle in der Bibliothek mscomctl.ocx enthalten sind. Diese können Ihnen bei der Entwicklung von VBA-Anwendungen hilfreiche Dienste leisten. Im folgenden Abschnitt werde ich die, meiner Meinung nach, wichtigsten kurz beschreiben.

3.3.1 Abbildungsliste (ImageList)

Das *ImageList*-Steuerelement kann eine Auflistung von Bildern enthalten, die wiederum anderen Steuerelementen zugeordnet werden. Da dieses Steuerelement zur Laufzeit nicht sichtbar ist und nur zur „Lagerung“ von Bilddaten verwendet wird, kommt für die Abbildungsliste nur eine Verwendung in Zusammenhang mit anderen Steuerelementen, die diese Bilddaten wiedergeben können, in Frage.

Nachdem Sie die Abbildungsliste hinzugefügt haben, erscheint ein Symbol auf dem Formular. Dieses Symbol stellt das Steuerelement dar und ist nur zur Entwurfszeit verfügbar. Wenn Sie das Symbol anklicken, dann erscheinen im Eigenschaftsfenster die Daten der Abbildungsliste. In diesem Fenster sehen Sie zusätzlich den Eintrag (BENUTZERDEFINIERT). Auf diese Zeile können Sie klicken, um das Einstellungsfenster des Steuerelements aufzurufen.

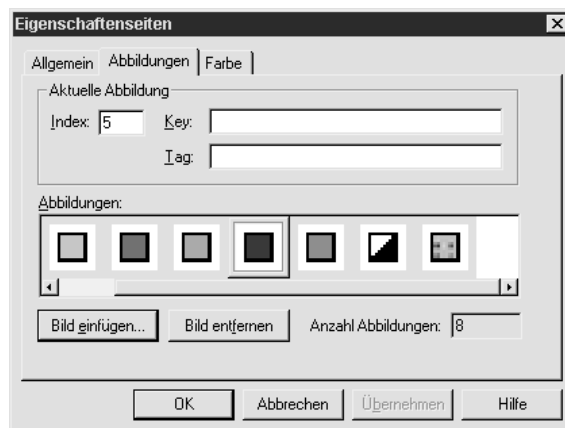


Abbildung 3.19: Eigenschaftseite der Abbildungsliste

In diesem Fenster können Sie nun alle Einstellungen zur Abbildungsliste vornehmen.

Reiter „Allgemein“

In diesem Bereich des Dialogs legen Sie die Größen der Bilder der Auflistung fest. Die hier eingestellte Größe gilt für alle Bilder, die der Abbildungsliste hinzugefügt werden. Wenn erst einmal ein Bild in der Liste vorhanden ist, dann können Sie diese Eigenschaften nicht mehr ändern. Sie müssten also die gesamte Liste leeren, um neue Größen festzulegen. Die Werte in den Textfeldern *Height* und *Width* werden in den Eigenschaften *ImageHeight* und *ImageWidth* gespeichert.

Mit *UseMaskColor* können Sie festlegen, ob die Farbe, die Sie in der Eigenschaft *MaskColor* festgelegt haben, genutzt werden soll. Die Verwendung von *MaskColor* bewirkt, dass die Farbe, die Sie dort einstellen, als transparente Farbe bei der Darstellung der Bilder auf anderen Steuerelementen verwendet werden soll.

Reiter „Abbildungen“

In diesem Abschnitt des Dialogs können Sie Bilder zur Abbildung hinzufügen oder entfernen und außerdem den einzelnen Bildern Schlüssel (*Key*-Eigenschaft) zuordnen. Dieser Schlüssel wird verwendet, um das entsprechende Bild anhand des Schlüsselwerts statt des Index aus der Abbildungsliste abzurufen.

Im Bereich *Abbildungen* sind die einzelnen Bilder nacheinander aufgelistet. Um ein Bild hinzuzufügen, wählen Sie die Schaltfläche BILD EINFÜGEN. Beachten Sie dabei die Position, an der das Bild eingefügt werden soll. Wenn sich z.B. bereits vier Bilder in der Liste befinden und Sie gerade das zweite Bild markieren, dann wird das Bild an der dritten Position eingefügt.

Um ein Bild aus der Liste zu entfernen, markieren Sie das entsprechende Bild und betätigen danach die Schaltfläche BILD ENTFERNEN.

Reiter „Farbe“

In diesem Dialogbereich können Sie sowohl die Hintergrundfarbe als auch die Farbe für die Eigenschaft *MaskColor* festlegen.

ListImages-Auflistung

Die wichtigste Eigenschaft, die Sie zur Laufzeit mit der Abbildungsliste verwenden werden, ist *ListImages*. Diese gibt eine Auflistung mit allen vorhandenen Bildern zurück. Sie können über diese Auflistung Bilder abrufen oder neue Bilder hinzufügen. Zu diesem Zweck besitzt das Auflistungsobjekt verschiedene Methoden:

Mit der *Add*-Methode fügen Sie ein Bild der Auflistung hinzu. Sie können im ersten Parameter die Position des hinzuzufügenden Bilds festlegen, mit dem zweiten den Schlüssel und mit dem dritten das Bild selbst. Alle drei Parameter dieser Anweisung sind optional.

```
Set NewImage = ImageList1.ListImages.Add
```

Tipp

Das Bild, das in der *ListImages*-Auflistung verwaltet wird, entspricht nicht der *Picture*-Eigenschaft anderer Steuerelemente. Vielmehr handelt es sich um ein Objekt, das neben *Picture* noch andere Eigenschaften enthält.

Mit der *Remove*-Anweisung können Sie ein Bild aus der Auflistung entfernen. Im Parameter müssen Sie lediglich den Index des Bilds angeben. Das kann die direkte Position sein oder der Schlüssel des Bilds, falls Sie einen vergeben haben. Die *Clear*-Anweisung entfernt alle Bilder aus der Auflistung.

ImageList1.ListImages.Remove 1

Die *Item*-Eigenschaft liefert einen Verweis auf das angegebene Bild. Im Index können Sie wieder die Position oder den Schlüssel (falls vergeben) angeben.

Set ActImage = ImageList1.ListImages(1)

Tip

Wenn Sie die Abbildungsliste an ein anderes Steuerelement gebunden haben, dann können Sie keine Bilder aus der Liste entfernen. Außerdem lassen sich Bilder nur am Ende der Liste hinzufügen. Die *Clear*-Anweisung ist ebenfalls blockiert.

Einzelne Bilder verwenden

Bei Steuerelementen, an die die Abbildungsliste gebunden werden kann, reicht die Angabe der Positionsnummer oder der Schlüsselname, um das entsprechende Bild dort anzeigen zu lassen. Bei den anderen, die nicht über diese Eigenschaft verfügen, müssen Sie das Bild direkt zuweisen. Verwenden Sie hierfür die *Picture*-Eigenschaft des einzelnen Bilds der Abbildungsliste.

Image1.Picture = ImageList1.ListImages(1).Picture

Zu beachten ist hierbei noch, dass diese Übergabe keinen Verweis auf das Bild der Abbildungsliste darstellt.

3.3.2 Listenansicht-Steuerelement (ListView)

(Im Verlaufe des Buchs werden Sie eine Reihe vollständig beschriebener Beispiele finden, die dieses Steuerelement verwenden.)

Damit Sie gleich ein praktisches Anwendungsbeispiel zum Thema „Abbildungsliste“ bekommen, wird im Folgenden das Listenansicht-Steuerelement beschrieben. Dieses besitzt die Möglichkeit, Abbildungslisten zu binden, so dass die darin enthaltenen Bilder im Steuerelement direkt verwendet werden können. Dieses Steuerelement stellt vier verschiedene Möglichkeiten der Ansicht von Elementen zur Verfügung, darunter auch eine Listenansicht, wie Sie sie bereits vom Layermanager her kennen.

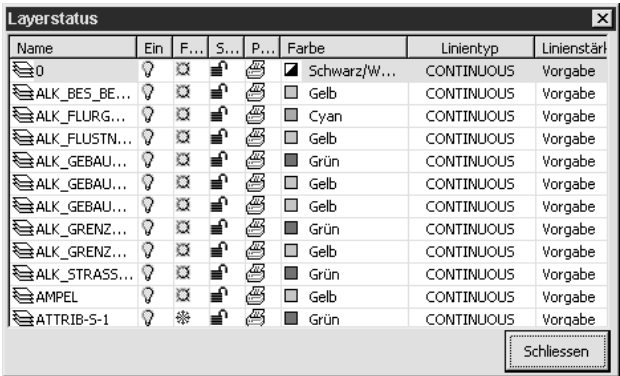


Abbildung 3.20: Listenansicht-Steuerelement

Nachdem Sie das Steuerelement auf dem Formular platziert haben, sehen Sie dort nur eine weiße Fläche, beschriftet mit dem Namen der Listenansicht (z.B. ListView1). Diese Darstellung wird sich zur Entwurfszeit auch nicht ändern, da die Listenelemente erst zur Laufzeit hinzugefügt werden. Aus diesem Grund werden Sie das Formular öfter starten müssen, um die Entwicklung zu kontrollieren.

Beschäftigen wir uns zunächst mit den Eigenschaften und deren Wirkung auf das Steuerelement. Genau wie die Abbildungsliste können Sie auch hier eine Eigenschaftsseite aktivieren, die die Einstellung der einzelnen Eigenschaften erleichtert.

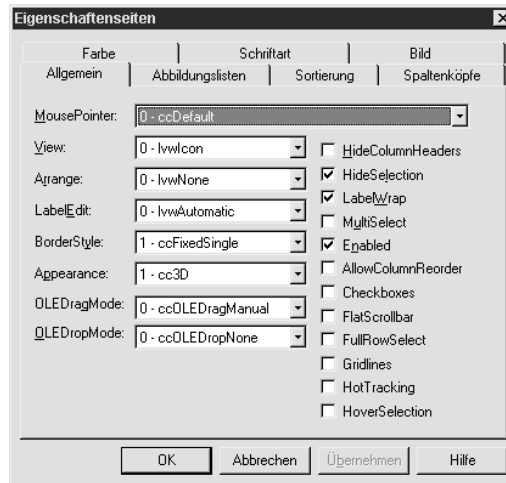


Abbildung 3.21: Eigenschaftsseite des Listenansicht-Steuerelements

Reiter „Allgemein“

In diesem Abschnitt finden Sie alle Eigenschaften, mit denen Sie Einstellungen tätigen können, die sich auf das gesamte Listenansicht-Steuerelement auswirken. Gehen wir diese Schritt für Schritt durch.

MousePointer: Hier bestimmen Sie den Mauszeiger, der angezeigt werden soll, wenn sich der Mauszeiger über dem Steuerelement befindet.

View: Legen Sie hier eine der vier verschiedenen Ansichten des Steuerelements fest. Sie haben dabei die Auswahl zwischen:

- **lvwIcon** – für die Anzeige der Einträge als große Symbole mit darunter befindlichem Text.
- **lvwSmallIcon** – für die Anzeige der Einträge als Text mit einem kleinen Symbol.
- **lvwList** – für die spaltenförmige Anzeige der Einträge als Text mit einem kleinen Symbol.
- **lvwReport** – bei dieser Ansichtsform wird jeder Eintrag in einer eigenen Zeile dargestellt. Zusätzlich können dem Eintrag weitere Informationen zugeordnet werden, die dem ersten Eintrag spaltenweise folgen (siehe Layermanager).

Arrange legt die Anordnung der Symbole in den Ansichten fest.

LabelEdit: Wenn Sie nicht möchten, dass der Benutzer den Text des Listeneintrags automatisch ändern kann, dann müssen Sie diese Eigenschaft auf den Wert *lvwManual* einstellen. Andernfalls kann der Benutzer durch einfaches Anklicken des entsprechenden Texts eine Eingabeaufforderung einblenden, in der er dann den Text ändert (gilt im Report-Modus nur für die erste Spalte).

Appearance/BorderStyle: Die *BorderStyle*-Eigenschaft wird nur verwendet, wenn Sie die Eigenschaft *Appearance* auf 2D einstellen. Dann können Sie mit *BorderStyle* festlegen, ob ein Rand um das Steuerelement gezogen werden soll oder nicht.

OleDragMode/OleDropMode: Diese beiden Eigenschaften legen das Verhalten für Ziehen- und Ablegen-Vorgänge fest. *OleDragMode* bestimmt, ob das Steuerelement als Quelle solcher Vorgänge dienen kann und ob dies manuell oder automatisch erfolgen soll. *OleDropMode* bestimmt, ob auf der Liste ein Objekt abgelegt werden kann.

HideColumnHeaders: (nur Report-Modus) Legt fest, ob die Spaltenköpfe der Listenansicht sichtbar sein sollen oder ausgeblendet werden. Die Listeneinträge sind von der Einstellung dieser Eigenschaft allerdings nicht betroffen.

HideSelection: Mit dieser Eigenschaft können Sie steuern, ob die Auswahl eines oder mehrerer Listeneinträge auch angezeigt werden soll, wenn das Steuerelement nicht mehr den Fokus besitzt.

■ *HideSelection* = *TRUE* → Auswahl wird nicht mehr angezeigt.

■ *HideSelection* = *FALSE* → Auswahl wird angezeigt.

LabelWrap: (nur Symbol-Modus) Legt fest, ob der Text umgebrochen werden soll (*TRUE*), wenn der Text eine bestimmte Länge erreicht hat. Die Länge des Texts einer Zeile kann in der Systemsteuerung eingestellt werden.

MultiSelect: Wenn Sie diese Eigenschaft auf den Wert *TRUE* einstellen, dann ermöglicht dies dem Benutzer eine Mehrfachauswahl von Listeneinträgen.

AllowColumnReorder: (nur Report-Modus) Mit dieser Eigenschaft können Sie festlegen, ob der Benutzer die Anordnung der Spalten verändern kann (*TRUE*) oder nicht (*FALSE*).

CheckBoxes: Wenn Sie diese Eigenschaft auf den Wert *TRUE* einstellen, dann wird vor jedem Listeneintrag ein Kontrollkästchen dargestellt.

FlatScrollBar: Mit dieser Eigenschaft können Sie zwischen einer 3D- und einer flachen Darstellung der Bildlaufleiste des Steuerelements umschalten.

FullRowSelect: (nur Report-Modus) Sollten Sie mehrere Spalten pro Listeneintrag verwenden, dann können Sie mit dieser Eigenschaft festlegen, ob bei Auswahl des Benutzers die gesamte Zeile (*TRUE*) oder nur die erste Spalte markiert werden soll.

GridLines: (nur Report-Modus) Durch Aktivierung dieser Eigenschaft erreichen Sie eine zeilenweise Darstellung der Listenansicht.

HotTracking: Mit dieser Eigenschaft legen Sie fest, ob mit der Maus angezeigt werden soll, dass sich der Mauszeiger über einem Eintrag befindet. Der Mauszeiger wechselt dann auf das Handsymbol wie bei einem Hyperlink.

HoverSelection: Diese Eigenschaft sollten Sie nur aktivieren, wenn die Mehrfachauswahl abgeschaltet wurde. Wurde *HoverSelection* aktiviert, dann wird ein Listeneintrag markiert, sobald sich der Mauszeiger darüber befindet.

Reiter „Abbildungslisten“

Dieser Bereich bietet normalerweise die Möglichkeit, den verschiedenen Symboldarstellungen Abbildungslisten bereits zur Entwurfszeit zuzuordnen. Leider besteht diese Möglichkeit nicht in VBA.

Reiter „Sortierung“

Hier können Sie verschiedene Sortierungsmöglichkeiten für die Anordnung der Listenelemente festlegen. Die Eigenschaft *Sorted* bestimmt, ob die Listeneinträge überhaupt sortiert werden sollen. Mit der *SortKey*-Eigenschaft legen Sie fest, welche Spalte für die Sortierung verantwortlich ist, wenn sich das *ListView*-Steuerelement im Report-Modus befindet. Der Wert 0 legt fest, dass zum Sortieren der Listeneintrag verwendet wird. Ein Wert größer 0 bestimmt die entsprechend untergeordnete Spalte des Listeneintrags zur Sortierung. Ob eine auf- (*lvwAscending*) oder absteigende (*lvwDescending*) Sortierreihenfolge verwendet wird, können Sie über die Eigenschaft *SortOrder* festlegen.

Reiter „Spaltenköpfe“

Bei der Darstellung der Liste im Report-Modus lassen sich, wie bereits erwähnt, verschiedene Unterelemente zusätzlich zum Listeneintrag selbst anzeigen. Jedes Element, angefangen beim eigentlichen Listeneintrag, bis hin zum letzten Unterelement, wird in einer eigenen Spalte angezeigt. Diese können Sie in diesem Bereich des Dialogs zur Entwurfszeit hinzufügen. Zur Laufzeit sind die Spalten in der Auflistung *ColumnHeaders* gespeichert.

Abbildung 3.22: Festlegung der Spaltenköpfe

Mit den Schaltflächen SPALTE EINFÜGEN und SPALTE ENTFERNEN fügen Sie entweder eine neue Spalte hinzu oder Sie entfernen die gerade angezeigte Spalte. Über die beiden Schalter neben dem Index können Sie zwischen den einzelnen Spalten hin- und herschalten. Der Index gibt dabei die Position der aktuell angezeigten Spalte wieder.

Nun zu den einzelnen Eigenschaften, die Sie hier für jede einzelne Spalte festlegen können.

Text: Legt die Spaltenüberschrift fest.

Alignment: Bestimmt die Ausrichtung der in der Spalte angezeigten Texte. Sie können zwischen einer linksbündigen, rechtsbündigen oder zentrierten Darstellung wählen. Die erste Spalte akzeptiert allerdings nur den Wert *lvwColumnLeft*.

Width: Hiermit können Sie einen Wert für die Breite der Spalte festlegen. Beachten müssen Sie dabei, dass die Einheit Punkt verwendet wird. Da die Einstellung der richtigen Breite immer eine recht diffizile Angelegenheit ist, kann ich nur empfehlen, die Spalten einfach anzulegen, das Formular zu starten und die Spalten auf die richtige Breite zu ziehen und sich dann die entsprechenden Werte anzeigen oder ausgeben zu lassen. Außerdem ist es immer sinnvoll, eine Breitenänderung durch den Benutzer zu speichern und beim nächsten Programmstart wiederherstellen zu lassen.

Key: Hier können Sie wieder einen eindeutigen Schlüssel für die jeweilige Spalte festlegen.

Tag: In dieser Eigenschaft können Sie, wenn Sie wollen, zusätzliche Informationen zur aktuellen Spalte speichern.

Symbolindex (Icon-Eigenschaft): Dieser Eigenschaft können Sie zur Entwurfszeit keinen Wert zuordnen, da zunächst eine Abbildungsliste zugeordnet werden muss, was in VBA ja nur zur Laufzeit möglich ist. Die Eigenschaft selbst legt die Nummer des Symbols fest, das im jeweiligen Spaltenkopf angezeigt werden soll. Diese Nummer entspricht dem Index der Bilder, die in der zugeordneten Abbildungsliste enthalten sind.

Reiter „Farbe/Schriftart/Bild“

In den restlichen drei Abschnitten der Eigenschaftsseite können Sie noch die Hintergrund- und die Vordergrundfarbe des Steuerelements, die Schriftart der Listeneinträge sowie ein Hintergrundbild und einen Mauszeiger einstellen.

Abbildungslisten zuordnen

Was zur Entwurfszeit leider nicht funktioniert, können Sie zur Laufzeit natürlich nachholen, nämlich die Zuweisung der Abbildungslisten. Insgesamt können Sie dem Steuerelement drei verschiedene Abbildungslisten zuweisen. Jede dieser Listen dient einem eigenen Darstellungszweck. In der *Icons*-Eigenschaft wird die Abbildungsliste zugewiesen, deren Bilder für die großen Symbole verwendet werden sollen (Icon-Modus), *SmallIcons* bekommt die Abbildungsliste, die die normalen (kleinen) Symbole enthält und *ColumnHeaderIcons* erhält die Symbole für die Spaltenköpfe. Die Zuweisung erfolgt immer auf die gleiche Weise:

1. Abbildungsliste für die großen Symbole festlegen:

```
Set ListView1.Icons = ImageList1
```

2. Abbildungsliste für die kleinen Symbole festlegen:

```
Set ListView1.SmallIcons = ImageList2
```

3. Abbildungsliste für die Spaltenköpfe festlegen:

```
Set ListView1.ColumnHeaderIcons = ImageList2
```

Spaltenköpfe zur Laufzeit verwalten

Der Zugriff auf die Spaltenköpfe erfolgt über die *ColumnHeaders*-Auflistung. Wenn Sie die Spaltenköpfe erst zur Laufzeit hinzufügen wollen, dann verwenden Sie wie gehabt die *Add*-Anweisung von *ColumnHeaders*. Dieser Anweisung können Sie in den einzelnen Parametern die Eigenschaften der Spalten mitteilen. Im Normalfall reicht es aber, die Spaltenüberschrift und eventuell noch die Breite der Spalte festzulegen.


```

ListView1.ColumnHeaders.Add Text:="Spalte1"
ListView1.ColumnHeaders.Add Text:="Spalte2", Width:=30
ListView1.ColumnHeaders.Add Text:="Spalte3", Width:=80

```

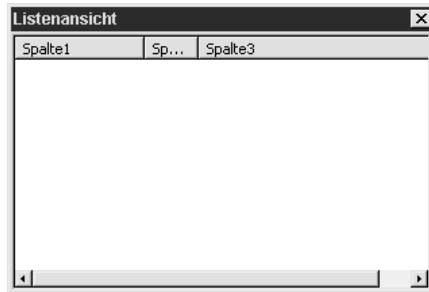


Abbildung 3.23: Drei Spalten mit unterschiedlicher Breite hinzugefügt

Um eine vorhandene Spalte abzurufen, können Sie wie üblich die *Item*-Eigenschaft verwenden. Zum Löschen einzelner oder aller Spalten besitzt die *ColumnHeaders*-Auflistung die *Remove*- und *Clear*-Anweisungen.

Listeneinträge verwalten

Die Verwaltung der Listeneinträge scheint aufgrund der Vielfalt an Darstellungsmöglichkeiten auf den ersten Blick recht kompliziert. Wenn das Prinzip aber erst einmal verstanden wurde, dann werden Sie den Umgang mit diesem Steuerelement sehr schnell erlernen. Zunächst einmal möchte ich ein wenig auf die Struktur der Listenansicht eingehen. An oberster Stelle stehen die Listeneinträge. Diesen ist wieder jeweils ein Auflistungsobjekt untergeordnet, das alle Unterelemente aufnimmt. Die Listeneinträge werden in jedem Fall in der Listenansicht angezeigt, egal welche Ansicht Sie eingestellt haben. Die Unterelemente werden nur in der Report-Ansicht angezeigt und das auch nur, wenn eine Spalte zur Darstellung angelegt wurde. Existieren nicht genügend Spalten, dann werden nur so viele Unterelemente angezeigt, wie Spalten verfügbar sind. Die Eigenschaft, über die Sie auf die Listeneinträge zugreifen können, heißt *ListItems*. Das Hinzufügen, Entfernen und Abrufen einzelner Einträge funktioniert wie gewohnt. In den Parametern können Sie, wie bei den Spaltenköpfen, so ziemlich alle Eigenschaften mitgeben.

Die folgende Prozedur fügt die Namen aller in der Zeichnung verfügbaren Layer der Listenansicht hinzu:

```

For Each ActLayer In ThisDrawing.Layers
    ListView1.ListItems.Add Text:=ActLayer.Name
Next

```

Der nächste Schritt wäre das Hinzufügen der einzelnen Unterelemente. Diese werden in der *ListSubItems*-Auflistung verwaltet. Über diese Auflistung können Sie Unterelemente mit den üblichen Anweisung hinzufügen, entfernen oder abrufen. Die Texte, die die Daten des Listenelements darstellen, können außerdem über die Eigenschaft *SubItems* als Array abgerufen oder gesetzt werden. Wenn wir uns wieder am Beispiel Layermanager orientieren, dann besteht die zweite Spalte lediglich aus der Darstellung eines Symbols für ein- bzw. ausgeschaltet. Also muss dem Unterelement kein Text, sondern nur das entsprechende Symbol zugeordnet werden. Nehmen wir einmal an, dass das Symbol zur Darstellung der Eigenschaft LAYER

EINGESCHALTET das erste Symbol der Abbildungsliste ist, während das für LAYER AUSGESCHALTET das zweite Symbol ist. Da die Unterelemente zum aktuell hinzugefügten Listeneintrag gehören, können die vorherigen Codezeilen erweitert werden:

```
For Each ActLayer In ThisDrawing.Layers
    Set LV = ListView1.ListItems.Add(Text:=ActLayer.Name)
    LV.ListSubItems.Add ReportIcon:=IIf(ActLayer.LayerOn = True, _
    1, 2)
Next
```

Einzelne Listeneinträge bearbeiten

Nachdem Sie nun einzelne Listeneinträge hinzufügen, entfernen oder abrufen können, möchte ich noch einige Worte zu den Eigenschaften und Methoden einzelner Listeneinträge verlieren.

Ein Verweis auf den aktuell gewählten Listeneintrag ist in der Eigenschaft *SelectedItem* gespeichert und somit über diese erreichbar. Mit der bis dato noch nicht erwähnten Eigenschaft *Ghosted* können Sie festlegen, ob das Symbol eines Listeneintrags abgeschwächt dargestellt werden soll (*FALSE*) oder nicht. Diese Eigenschaft erscheint besonders dann sinnvoll, wenn Sie einem Benutzer anzeigen wollen, dass das Element nicht verfügbar ist, oder um einen Ausschneidevorgang darzustellen. Mit den Eigenschaften *Left*, *Top*, *Height* und *Width* können Sie die momentane Position und Größe des Listeneintrags ermitteln oder festlegen. Die Position bezieht sich immer auf den Randbereich des Steuerelements. Wenn Sie den Inhalt mit der Bildlaufleiste verschieben, dann ändern sich auch die Eigenschaften *Left* und/oder *Top*.

Die *Selected*-Eigenschaften können Sie verwenden, um zu kontrollieren, ob ein Eintrag gewählt wurde (*TRUE*) oder nicht (*FALSE*). Sie können der Eigenschaft auch einen Wert zuweisen, um das Element zu selektieren oder die Auswahl aufzuheben.

Sollten Sie die Listenansicht für den DragandDrop-Modus vorbereitet haben, dann könnte die Anweisung *CreateDragImage* recht nützlich werden. Diese erzeugt ein Icon, das sowohl den Text als auch das Symbol des Eintrags darstellt. Dieses Icon kann dann bei Drag&Drop-Operationen als Verschiebesymbol verwendet werden.

```
DragPic = ListView1.SelectedItem.CreateDragImage
```

Manchmal kann es notwendig werden, einen Bildlauf automatisch durchzuführen, um sicherzustellen, dass der Listeneintrag sichtbar ist. Stellen Sie sich vor Sie nehmen einen Eintrag zum aktiven Element vor und der Benutzer muss erst durch die Listenansicht scrollen, um ihn zu sehen. Mit der *EnsureVisible*-Funktion können Sie dies automatisch tun. Einfach die Funktion mit dem Element aufrufen und es wird gegebenenfalls ein Bildlauf durchgeführt, damit das Element sofort sichtbar wird.

```
ListView1.ListItems(n).EnsureVisible
```

Ereignisse und Methoden des Listenansicht-Steuerelements

Nun noch einige Anmerkungen zu speziellen Methoden und Ereignissen, die für die Verwendung dieses Steuerelements wichtig sind.

Über die *FindItem*-Anweisung können Sie ein bestimmtes Listenelement durch Angabe einer Zeichenfolge finden lassen. Falls ein Eintrag gefunden wird, gibt die Funktion einen Verweis auf das Listenelement zurück. Durch Festlegung zusätzlicher Parameter lässt sich die Suche noch präzisieren.

Parameter Wert	
Dieser Parameter legt fest, in welchem Teil des Listeneintrags nach der angegebenen Zeichenfolge gesucht werden soll. Sie können folgende Konstanten verwenden:	
<i>lvwText</i>	Sucht die Zeichenfolge in der <i>Text</i> -Eigenschaft des Listeneintrags (Voreinstellung).
<i>lvwSubitem</i>	Sucht die Zeichenfolge in der <i>Text</i> -Eigenschaft der Unterelemente des Listeneintrags.
<i>lvwTag</i>	Sucht die Zeichenfolge in der <i>Tag</i> -Eigenschaft des Listeneintrags.

Parameter Index	
Wenn Sie einen Index festlegen, dann wird ab der Position gesucht, die dem Index oder Schlüssel (Key-Eigenschaft) eines Listeneintrags entspricht.	

Parameter Treffer	
Dieser Parameter legt fest, ob der gesuchte Wert vollständig oder partiell mit der <i>Text</i> -Eigenschaft übereinstimmen soll.	
<i>lvwWholeWord</i>	Der Text muss vollständig mit der gesuchten Zeichenfolge übereinstimmen.
<i>lvwPartial</i>	Es reicht, wenn der Text mit der gesuchten Zeichenfolge beginnt.

Die Suche nach einem Layer, der mit der Zeichenfolge „Attrib“ beginnt, könnte folgendermaßen aussehen.

```
Set fItem = ListView1.FindItem("Attrib", , , lvwPartial)
```

Die Anweisung `GetFirstVisible` gibt einen Verweis auf das Listenelement zurück, das im sichtbaren Bereich ganz oben steht.

Sie kennen sicher die Eigenschaft des Layermanagers, den Layernamen eines gerade hinzugefügten Layers zur Eingabe in der Layerliste anzubieten. Dieses Steuerelement kann das auch. Sie müssen lediglich die `StartLabelEdit`-Anweisung ausführen und schon wird das aktive Element zur Bearbeitung angeboten.

```
Set ListView1.SelectedItem = ListView1.ListItems.Add( _
    Text:="neuer Layer")
ListView1.StartLabelEdit
```

Aufgrund seiner Vielseitigkeit bietet dieses Steuerelement natürlich auch eine ganze Reihe von Ereignisprozeduren. Die wichtigsten habe ich einmal für Sie zusammengestellt:

AfterLabelEdit: Nachdem der Benutzer die Bearbeitung eines Texts abgeschlossen hat, wird dieses Ereignis ausgelöst. Sie machen die Änderung des Benutzers rückgängig, indem Sie den Wert der Variablen *Cancel* auf *TRUE* setzen. Der Wert des neuen Eintrags wird in der Variablen *NewString* übergeben.

```
Private Sub ListView1_AfterLabelEdit(Cancel As Integer, _
    NewString As String)
If NewString = "0" Then
    MsgBox "Änderung des Layers 0 nicht gestattet!", vbCritical
```

```

    Cancel = True
End If
End Sub

```

BeforeLabelEdit: Dieses Ereignis wird ausgelöst, wenn der Benutzer versucht, eine Änderung am Listeneintrag vorzunehmen. Sie können so die automatische Änderung für ganz bestimmte Element sperren, obwohl die *LabelEdit*-Eigenschaft auf *lvwAutomatic* eingestellt wurde, indem Sie die *Cancel*-Variable auf den Wert *TRUE* setzen.

```

Private Sub ListView1_BeforeLabelEdit(Cancel As Integer)
If ListView1.SelectedItem = "0" Then
    Cancel = True
End If
End Sub

```

ColumnClick: (nur Report-Modus) Dieses Ereignis könnten Sie z.B. dafür nutzen, eine Sortierung der betreffenden Spalte vorzunehmen, da es immer dann ausgelöst wird, wenn der Benutzer einen Spaltenkopf anklickt. Das entsprechende Spaltenkopfbjekt wird im Parameter *ColumnHeader* mitgegeben.

```

Private Sub ListView1_ColumnClick(ByVal ColumnHeader As _
MSComctlLib.ColumnHeader)
ListView1.SortKey = ColumnHeader.Index - 1
ListView1.SortOrder = lvwAscending
ListView1.Sorted = True
End Sub

```

ItemCheck: Dieses Ereignis findet statt, wenn der Benutzer auf das Kontrollkästchen neben dem List-Eintrag klickt. Es kann also nur ausgelöst werden, wenn die Kontrollkästchen aktiviert wurden (Eigenschaft *CheckBoxes* = *TRUE*). Der Parameter *Item* enthält das betroffene Listenelement und Sie könnten beispielsweise verhindern, dass es ausgewählt wird.

```

Private Sub ListView1_ItemCheck(ByVal Item As _
MSComctlLib.ListItem)
If Item = "0" Then
    Item.Checked = False
End If
End Sub

```

ItemClick: Dieses Ereignis wird ausgelöst, nachdem der Benutzer einen Listeneintrag angeklickt hat. Da auch hier das betroffene Element im Parameter *Item* mitgeliefert wird, können Sie feststellen, welcher Eintrag das Ereignis ausgelöst hat. Das Element ist danach auch über die *SelectedItem*-Eigenschaft der Listenansicht abrufbar. (Dieses Ereignis wird noch vor *Click* ausgelöst.)

Tipp

Dieses Steuerelement kann insgesamt sechs verschiedene OLE-Ereignisse auslösen. Alle diese Ereignisse stehen in Zusammenhang mit Drag&Drop-Aktionen des Benutzers. Am Ende dieses Kapitels finden Sie eine kurze Einführung zu diesem Thema.

3.3.3 Fortschrittsanzeige (ProgressBar)

Ein sehr einfaches und doch gelungenes Steuerelement ist die *ProgressBar*. Damit lassen Sie den Verlauf eines Vorgangs anzeigen, um zu verhindern, dass der Benutzer im Dunklen tappt und nicht genau weiß, läuft der Vorgang noch oder hängt die Kiste.



Abbildung 3.24: ProgressBar

Die wichtigsten Eigenschaften dieses Steuerelements sind, wie bei einer Bildlaufleiste, *Min*, *Max* und *Value*. *Min* und *Max* bestimmen den Bereich, *Value* den aktuellen Ablaufwert. Sie können zusätzlich noch einige Eigenschaften zur Anpassung der Darstellung der Statusanzeige festlegen. Dazu müssen Sie nur die Eigenschaftsseite des Steuerelements aktivieren.

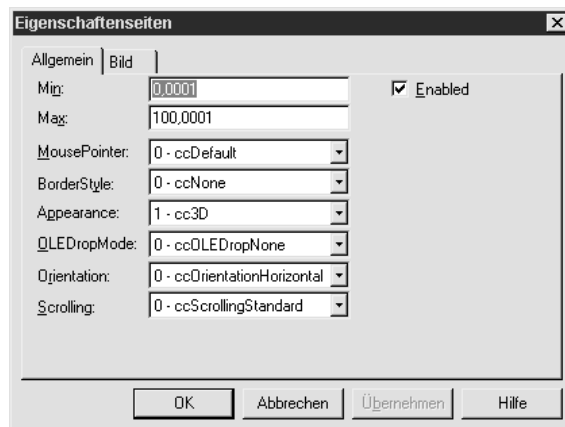


Abbildung 3.25: ProgressBar-Eigenschaftsseite

BorderStyle: Diese Eigenschaft aktiviert einen Rand um die Statusanzeige. Wenn Sie den Rand im 3D-Modus darstellen, dann sieht das Steuerelement etwas gewöhnungsbedürftig aus. Sie sollten ihn also nur im 2D-Modus verwenden.

Appearance: Schaltet zwischen 2D- und 3D-Darstellung um.

OleDropMode: Legt fest, ob auf der Statusanzeige OLE-Objekte abgelegt werden dürfen.

Orientation: Legt für die Statusanzeige eine horizontale oder vertikale Ausrichtung fest.

Scrolling: Mit dieser Eigenschaft können Sie zwischen zwei verschiedenen Darstellungsmöglichkeiten des Anzeigebalkens umschalten.

Die Bedienung der Statusanzeige funktioniert im Wesentlichen durch das Setzen der *Value*-Eigenschaft, was folgendes Beispiel beweist. Hier wird lediglich der Bereich von *Min* bis *Max* durchlaufen:

```

Do
  If ProgressBar1 + 1 > ProgressBar1.Max Then Exit Do
  ProgressBar1 = ProgressBar1 + 1
Loop

```

Tipp

Bei der Verwendung einer Statusanzeige sollten Sie dem Benutzer die Möglichkeit bieten, den Vorgang abubrechen. Dazu müssen Sie die Anweisung `DoEvents` in die Schleife mit aufnehmen, damit das Betriebssystem die Anforderung der Schaltflächenbetätigung verarbeiten kann. Diese Methode ist im Beispielformular zu diesem Thema bereits eingearbeitet.

3.3.4 Schieberegler (Slider)

Das Schieberegler-Steuerelement erinnert in der Wirkungsweise an Bildlaufleisten oder Drehfelder, nur dass hier der Benutzer über einen einzelnen Schieber die *Value*-Eigenschaft steuern kann. Dieses Steuerelement verfügt zusätzlich über die Möglichkeit, einen Auswahlbereich statt eines einzelnen Werts festzulegen.

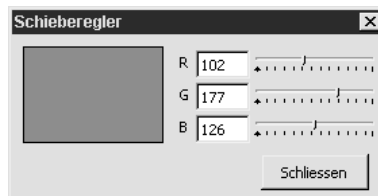


Abbildung 3.26: Schieberegler

Da es sich erneut um ein Steuerelement handelt, dessen Aufgabe die Verwaltung eines Wertebereichs ist, sind die wichtigsten Eigenschaften wieder *Min*, *Max* und *Value*. Zusätzlich bietet dieses Steuerelement weitere Eigenschaften, die Sie über die Eigenschaftsseiten anpassen können.

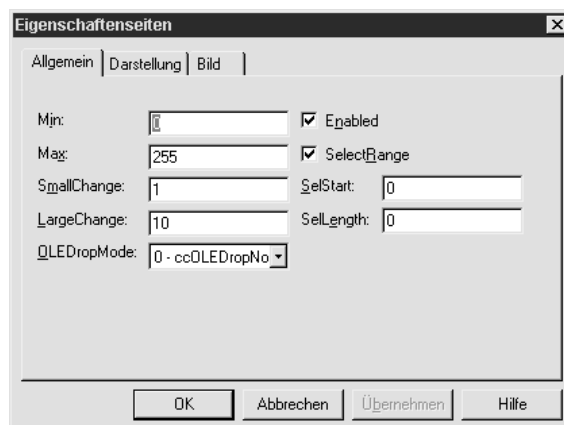


Abbildung 3.27: Eigenschaftsseite des Schiebereglers

Auch dieser Dialog verfügt über mehrere Abschnitte, die Sie jeweils über die entsprechenden Reiter aktivieren können.

Reiter „Allgemein“

Min/Max: Diese beiden Eigenschaften legen den unteren bzw. den oberen Wert des Bereichs fest.

SmallChange: Diese Eigenschaft bestimmt die Schrittweite, mit der die *Value*-Eigenschaft erhöht bzw. verringert werden soll, wenn der Benutzer den Schieber verwendet.

LargeChange: Diese Eigenschaft legt die Schrittweite fest, mit der die *Value*-Eigenschaft erhöht bzw. verringert werden soll, wenn der Benutzer in den End- bzw. Anfangsbereich des Schiebereglers klickt.

Festlegen eines Auswahlbereichs

Wie bereits erwähnt, können Sie mit diesem Steuerelement einen Auswahlbereich innerhalb der Grenzen *Min* und *Max* festlegen, der dann optisch gekennzeichnet wird.

SelectRange: Wenn Sie diese Eigenschaft aktivieren, dann werden die Werte der Eigenschaften *SelStart* und *SelLength* zur Festlegung eines Auswahlbereichs verwendet. Andernfalls entspricht *SelStart* der *Value*-Eigenschaft und *SelLength* wird ignoriert.

SelStart: Diese Eigenschaft legt die Startposition des Auswahlbereichs fest

SelLength: Diese Eigenschaft legt die Länge des Auswahlbereichs fest. Hinweis: Bei dem gewählten Bereich muss es sich immer um eine positive Zahl handeln, da sonst ein Fehler ausgelöst wird.

Im folgenden Beispiel wird die Bereichsauswahl aktiviert, wenn der Benutzer während der Betätigung des Schiebereglers die -Taste gedrückt hält.

```
Private Sub Slider1_MouseDown(ByVal Button As Integer, _
    ByVal Shift As Integer, ByVal x As stdole.OLE_XPOS_PIXELS, _
    ByVal y As stdole.OLE_YPOS_PIXELS)
    If Shift = 1 Then
        Slider1.SelectRange = True
        Slider1.SelStart = Slider1
    End If
End Sub

Private Sub Slider1_MouseUp(ByVal Button As Integer, _
    ByVal Shift As Integer, ByVal x As stdole.OLE_XPOS_PIXELS, _
    ByVal y As stdole.OLE_YPOS_PIXELS)
    If Shift = 1 Then
        'Gewählten Bereich auswerten
    Else
        Slider1.SelectRange = False
    End If
End Sub

Private Sub Slider1_Scroll()
    If Slider1.SelectRange = True Then
        On Error Resume Next
        Slider1.SelLength = Slider1 - Slider1.SelStart
    End If
End Sub
```

Reiter „Darstellung“

In diesem Bereich der Eigenschaftsseite können Sie einige optische Anpassungen am Schieberegler vornehmen.

Orientation: Mit dieser Eigenschaft legen Sie die Ausrichtung des Reglers fest. Anders als bei Bildlaufleisten passt sich dieses Steuerelement nicht automatisch an. Sie müssen also selbst bestimmen, ob der Regler in horizontaler oder vertikaler Ausrichtung dargestellt werden soll.

TickStyle: Mit dieser Eigenschaft können Sie vorgeben, auf welcher Seite des Reglers die Einteilungsstriche dargestellt werden sollen. Wenn Sie die Einteilung auf beiden Seiten anzeigen lassen, dann sollte das Steuerelement die entsprechende Höhe bzw. Breite besitzen, um beide Einteilungen darstellen zu können.

Konstante	Beschreibung	Darstellung
<i>sldBottomRight</i>	Die Einheiteneinteilung wird je nach Ausrichtung unterhalb oder rechts vom Regler angezeigt.	
<i>sldTopLeft</i>	Die Einheiteneinteilung wird je nach Ausrichtung oberhalb oder links vom Regler angezeigt.	
<i>sldBoth</i>	Die Einheiteneinteilung wird auf beiden Seiten angezeigt.	
<i>sldNoTicks</i>	Die Einheiteneinteilung wird nicht dargestellt.	

Tabelle 3.4: Konstanten der Eigenschaft *TickStyle*

TickFrequency: Mit dieser Eigenschaft können Sie die Häufigkeit der Einheitenmarkierungen festlegen. Wenn Sie z.B. einen Wertebereich von z.B. 100 gewählt haben (*Min/Max*) und für *TickFrequency* den Wert 2, dann wird alle zwei Einheiten ein Strich angezeigt, insgesamt also 50 Striche. Sie sollten diesen Wert so einstellen, dass eine vernünftige Einteilung herauskommt (z.B. 2er-, 5er- oder 10er-Einteilung) und die einzelnen Striche trotzdem genügend Abstand haben, damit der Benutzer eine logische und auch gut sichtbare Einteilung erhält.

TextPosition: Mit dieser Eigenschaft können Sie die Position des Infotexts festlegen, der normalerweise den aktuellen Wert der *Value*-Eigenschaft anzeigt. Sie können die Position auf rechts oder unter dem Regler bzw. links oder oberhalb des Schiebers einstellen.

Weitere Eigenschaften und Methoden

Neben den bereits erwähnten Eigenschaften bietet dieses Steuerelement weitere Eigenschaften an, die nicht in der Eigenschaftsseite eingestellt werden können.

Mit der Eigenschaft *GetNumTicks* können Sie die Anzahl der dargestellten Einheitenmarkierungen ermitteln, die sich aus dem Wertebereich und *TickFrequency* ergeben. Mit der *Text*-Eigenschaft können Sie einen anderen Infotext festlegen, der beim Verschieben des Reglers angezeigt werden soll.

Die *ClearSel*-Anweisung löscht den in den Eigenschaften *SelStart* und *SelLenght* festgelegten Auswahlbereich.

Ereignisse

Auf die Änderung der Reglerposition können Sie wie bei Bildlaufleisten mit den Ereignisprozeduren *Change* und *Scroll* reagieren. *Change* wird ausgelöst, wenn sich der Inhalt der *Value*-Eigenschaft ändert, und *Scroll*, wenn der Benutzer den Regler verschiebt. Im Folgenden ein

kurzer Auszug aus dem Quellcode des Beispielformulars, der den roten Farbanteil der Hintergrundfarbe des Anzeigefelds ändert. Den gesamten Code des Beispiels finden Sie im Formular *Regler*.

```
Private Sub SliderRed_Change()  
    TRed = SliderRed  
    Image1.BackColor = RGB(SliderRed, SliderGreen, SliderBlue)  
End Sub  
  
Private Sub SliderRed_Scroll()  
    SliderRed_Change  
End Sub
```

3.3.5 Statusleiste (StatusBar)

Dieses Steuerelement soll für gewöhnlich dafür verwendet werden, zusätzliche Informationen anzuzeigen, vorzugsweise im unteren Bereich eines Formulars. Bei diesen Informationen handelt es sich beispielsweise um Statusdaten des Programms, Informationen zu einem gewählten Objekt oder einfach nur Datum, Uhrzeit oder der Zustand bestimmter Tasten (*Einfg*, *Scroll* usw.). Zusätzlich zu den einzelnen Texten können Sie auch Symbole darstellen. Die Statusleiste selbst besteht aus einer Zeile, die sich in bis zu 16 Einzelfelder unterteilen lässt.



Abbildung 3.28: Anzeige von Informationen in der Statuszeile

Die einzelnen Felder der Statuszeile sind in der *Panels*-Auflistung gespeichert und können bereits zur Entwurfszeit hinzugefügt und angepasst werden. Dazu müssen Sie nur die Eigenschaftsseite des Steuerelements aufrufen. Diese unterteilt sich wiederum in verschiedene Abschnitte.

Reiter „Allgemein“

Hier geben Sie die Eigenschaften vor, die die Darstellung der gesamten Statuszeile betreffen. Mit der Eigenschaft *Style* können Sie zwischen zwei verschiedenen Zuständen wählen. Die erste Darstellungsmöglichkeit (*sbrNormal*), die auch die Standardeinstellung ist, legt fest, dass alle Felder der Statuszeile angezeigt werden sollen. Die zweite Möglichkeit (*sbrSimple*) schaltet die Zeile auf ein einziges Feld um, das sich über die gesamte Breite der Zeile ausdehnt. Wenn Sie nur einen Statustext anzeigen wollen (z.B. die Linientypbeschreibung), dann reicht diese Einstellung völlig aus. Die Felder, die Sie eventuell angelegt haben, bleiben in der *Panels*-Auf-

listung erhalten. Der anzuzeigende Text wird in diesem Fall über die Eigenschaft *SimpleText* festgelegt. Um einen Infotext anzuzeigen, wenn der Benutzer den Mauszeiger über einem der Statusfelder platziert, müssen Sie die Eigenschaft *ShowTips* auf den Wert *TRUE* einstellen. Sie können für jedes Feld einen eigenen Infotext festlegen.

Reiter „Grundflächen“ (Panels-Auflistung)

In diesem Bereich der Eigenschaftsseite können Sie die einzelnen Statusfelder bereits zur Entwurfszeit festlegen. Das erste Feld wird schon erzeugt, wenn Sie das Steuerelement auf dem Formular das erste Mal positionieren.

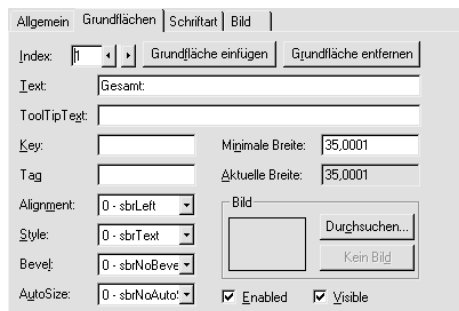


Abbildung 3.29: Festlegen der einzelnen Statusfelder

Mit den Schaltflächen GRUNDFLÄCHE EINFÜGEN und GRUNDFLÄCHE ENTFERNEN werden die einzelnen Statusfelder zur *Panels*-Auflistung hinzugefügt bzw. aus dieser wieder entfernt. Der Index zeigt die jeweilige Position innerhalb der Auflistung an. In dieser Reihenfolge werden die Felder später auch in der Statuszeile angezeigt.

Tipp

Leider hat diese Eigenschaftsseite eine Macke. Nachdem Sie eine Änderung an der Auflistung durchgeführt haben, z.B. ein Feld hinzugefügt bzw. entfernt oder die Änderungen an den Eigenschaften mit der Schaltfläche ÜBERNEHMEN bestätigt haben, positioniert sich die Eigenschaftsseite automatisch auf das erste Feld. Dieser Umstand kann mit der Zeit ganz schön nervig werden, weshalb Sie darauf achten sollten, erst die Einstellung zu tätigen und dann die Änderungen mit ÜBERNEHMEN zu bestätigen.

Für die einzelnen Felder können Sie folgende Einstellungen vornehmen:

Text: Diese Eigenschaft enthält den Text, der im Statusfeld angezeigt werden soll, wenn die Eigenschaft *Style* auf *sbrText* eingestellt worden ist.

ToolTipText: Wenn Sie die Eigenschaft *ShowTips* aktiviert haben, dann können Sie hier den Infotext festlegen, der angezeigt werden soll. Diese Eigenschaft ist insofern ganz nützlich, als das Sie statt eines Texts auch nur Symbole anzeigen lassen können, die dann durch diesen Infotext näher erklärt werden.

Key: Legt den Schlüssel des Felds fest. Diesen Schlüssel können Sie im Quellcode statt des Index benutzen, um das Feld aus der *Panels*-Auflistung abzurufen.

Tag: Hier lassen sich zusätzliche Informationen zum Statusfeld abspeichern.

Minimale Breite (*MinWidth*): Da sich die einzelnen Felder so anpassen lassen, dass sie sich über die gesamte Breite der Statuszeile erstrecken, muss man auch eine Mindestbreite für die einzelnen Felder bestimmen können. Besitzt das Feld eine feste Breite, dann entspricht der Wert dieser Eigenschaft der Feldbreite. Haben Sie das Feld auf eine variable Breite eingestellt, dann entspricht der Wert dieser Eigenschaft der Mindestbreite. Die aktuelle Breite können Sie in dem Fall der Eigenschaft *Width* entnehmen.

AutoSize: Diese Eigenschaft hängt mit den Breiteeigenschaften zusammen. Sie können über diese Eigenschaft drei verschiedene Zustände definieren.

Konstante	Beschreibung
<i>sbrNoAutoSize</i> (Standardwert)	Bei dieser Einstellung erhält das Feld eine feste Breite, die dem Wert der Eigenschaft <i>MinWidth</i> entspricht.
<i>sbrSpring</i>	Wenn sich die Breite der gesamten Statuszeile ändert und dadurch zusätzlicher Raum frei wird, dann wird der entstandene Freiraum unter allen Feldern aufgeteilt, deren <i>AutoSize</i> -Eigenschaft diesen Wert besitzt. Für die Felder gilt allerdings: Die Felder werden nur so weit verkleinert, bis der Wert <i>MinWidth</i> erreicht ist.
<i>sbrContents</i>	Die Breite des Felds passt sich automatisch der Breite des Inhalts an, der in dem Feld angezeigt wird. Allerdings kann das Feld nie kleiner werden, als in der Eigenschaft <i>MinWidth</i> eingestellt wurde.

Tabelle 3.5: Mögliche Werte der Eigenschaft *AutoSize*

Alignment: Legt die Ausrichtung des Texts innerhalb des Statusfelds fest. Der Text kann entweder linksbündig, rechtsbündig oder zentriert ausgerichtet werden.

Style: In dieser Eigenschaft können Sie den Typ des angezeigten Feldwerts festlegen. Die Standardeigenschaft ist zwar *Text*, aber Sie könnten auch das aktuelle Datum oder die Uhrzeit in einem Statusfeld anzeigen lassen.

Konstante	Beschreibung
<i>sbrText</i> (Standardwert)	Zeigt im Statusfeld den Text und/oder das Bild an, den/das Sie in den entsprechenden Eigenschaften festgelegt haben.
<i>sbrCaps</i>	Zeigt im Statusfeld den Zustand der  -Taste an.
<i>sbrNum</i>	Zeigt im Statusfeld den Zustand der  -Taste an.
<i>sbrIns</i>	Zeigt im Statusfeld den Zustand der  -Taste an.
<i>sbrScrl</i>	Zeigt im Statusfeld den Zustand der  -Taste an.
<i>sbrTime</i>	Zeigt im Statusfeld die aktuelle Uhrzeit im eingestellten Systemformat an.
<i>sbrDate</i>	Zeigt im Statusfeld das aktuelle Datum im eingestellten Systemformat an.
<i>sbrKana</i>	Zeigt im Statusfeld die Buchstaben  an. Wenn die  -Taste aktiviert wurde, ist dieser Text fett formatiert, andernfalls abgeschwächt.

Tabelle 3.6: Mögliche Werte der Eigenschaft *Style*

Wenn Sie für diese Eigenschaft einen anderen Wert als *sbrText* (0) festlegen, dann wird der Wert der Eigenschaft *Text* ignoriert. Die einzelnen Felder aktualisieren sich bei Änderungen selbst.

Bevel: Mit dieser Eigenschaft können Sie die einzelnen Felder entweder flach (*sbrNoBevel*), erhaben (*sbrRaised*) oder eingedrückt (*sbrInset*) darstellen.

Visible: Um ein Statusfeld vorübergehend auszublenden, können Sie den Wert dieser Eigenschaft auf den Wert *FALSE* umstellen. Um das Feld wieder sichtbar zu machen, brauchen Sie dieser Eigenschaft nur den Wert *TRUE* zuzuweisen.

Bild (Picture): Da Sie an dieses Steuerelement keine Abbildungsliste binden können, müssen Sie den Feldern, in denen Sie ein kleines Symbol darstellen wollen, über die *Picture*-Eigenschaft ein Bild zuweisen. Wenn Sie dies zur Laufzeit erledigen, dann können Sie entweder die *LoadPicture*-Anweisung verwenden oder einzelne Symbole einer Abbildungsliste direkt übergeben.

```
StatusBar1.Panels(1).Picture = LoadPicture("Bilddatei")
```

oder

```
StatusBar1.Panels(1).Picture = ImageList1.ListImages(1).Picture
```

Verwaltung der einzelnen Felder zur Laufzeit

Wie im Vorfeld bereits mehrfach angeführt, können Sie zur Laufzeit auf die Statusfelder über die *Panels*-Auflistung zugreifen. Zum Hinzufügen von Feldern verwenden Sie wie immer die *Add*-Anweisung. Bei der Verwendung dieser Anweisung können Sie einige Eigenschaften bereits im Parameter festlegen. Zurückgegeben wird ein Verweis auf das neue Statusfeld, so dass Sie die Eigenschaften auch über diesen Verweis festlegen können.

```
Set NewPanel = StatusBar1.Panels.Add(Text:="Beschreibung")
NewPanel.Bevel = sbrNoBevel
```

Entfernen können Sie die Statusfelder mit der *Remove*-Anweisung unter Angabe der Position oder des Schlüssels. Die *Clear*-Anweisung entfernt hingegen alle Felder.

Der Abruf erfolgt über die *Item*-Eigenschaft unter Angabe des Index. Das erste Feld besitzt den Index 1 und das letzte den der *Count*-Eigenschaft. Wenn Sie für die Felder Schlüssel vergeben haben, können Sie diesen statt der Position verwenden.

```
Set ActPanel = StatusBar1.Panels(1)
ActPanel.Visible = False
```

Sie müssen nicht unbedingt einen Verweis auf das Statusfeld an eine Variable übergeben, sondern können auch direkt mit dem Statusfeld arbeiten.

```
StatusBar1.Panels(1).Visible = True
```

Ereignisse der Statusanzeige

Die wichtigsten Ereignisse, auf die Sie reagieren müssen, sind *PanelClick* und *PanelDbClick*. Beide werden durch das Klicken des Benutzers auf ein Statusfeld ausgelöst. *PanelClick* folgt einem einfachen Klicken, *PanelDbClick* einem Doppelklick. In beiden Fällen wird ein Verweis auf das Statusfeld, das das Ereignis ausgelöst hat, im Parameter *Panel* mitgeliefert. Sie können so auf eine Auswahl des Benutzers reagieren und z.B. die *Bevel*-Eigenschaft umschalten.

```
Private Sub StatusBar1_PanelClick(ByVal Panel As _
MSComctlLib.Panel)
If Panel = "LST" Then
ThisDrawing.Preferences.LineWeightDisplay = IIf( _
```

```

ThisDrawing.Preferences.LineWeightDisplay = True, _
False, True)
End If
StatusBar1.Panels("Lst").Bevel = IIf( _
ThisDrawing.Preferences.LineWeightDisplay = True, sbrInset, _
sbrRaised)
End Sub

```

3.3.6 Kombinationsfeld-Steuerelement (ImageCombo)

Die *ImageCombo* stellt eine Erweiterung der einfachen *ComboBox* dar. Mit diesem Steuerelement können Sie jedem Listeneintrag ein eigenes Symbol zuordnen. Beispiele für die Anwendung finden sich bei der Auswahl von Farben oder Linienstärken. Die einzelnen Einträge werden auch nicht als Listarray, sondern als Objekte vom Typ *ComboItem* geführt, die wiederum über die Auflistung *ComboItems* abrufbar sind.

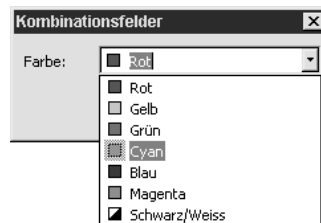


Abbildung 3.30: Kombinationslistenfeld

Die Bilder, die jeden Eintrag untermalen, werden über eine an das Steuerelement gebundene Abbildungsliste (*Kapitel 3.3.1*) verwaltet. Die Listeneinträge lassen sich bei diesem Steuerelement nicht vorab festlegen, sondern Sie müssen das zur Laufzeit tun. Ansonsten können die Einstellungen wieder über die Eigenschaftsseite vorgenommen werden.

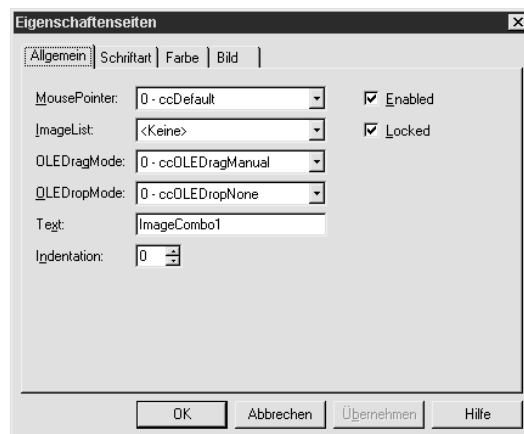


Abbildung 3.31: Eigenschaftsseite der ImageCombo

Reiter „Allgemein“

ImageList: In dieser Eigenschaft wird die Abbildungsliste an das Steuerelement gebunden. Leider können Sie die Zuweisung, anders als in VB6, nur zur Laufzeit vornehmen.

OleDragMode/OleDropMode: Diese beiden Eigenschaften legen das Verhalten für Ziehen- und Ablegenvorgänge fest. *OleDragMode* bestimmt, ob das Steuerelement als Quelle solcher Vorgänge dienen kann und ob dies manuell oder automatisch erfolgen soll. *OleDropMode* gibt vor, ob auf der Liste ein Objekt abgelegt werden kann.

Text: Diese Eigenschaft legt die Zeichenfolge fest, die im Textfeld des Steuerelements angezeigt werden soll. Zur Laufzeit können Sie auch die Eigenschaften *SelStart*, *SelLength* und *SelText* benutzen, um Teile bzw. die komplette Zeichenfolge zu selektieren.

Indentation: Diese Eigenschaft legt die Anzahl der Leerzeichen fest, um die die einzelnen Listeneinträge einschließlich Symbol eingerückt werden sollen, wenn *Indentation* nicht explizit für das einzelne Listenelement festgelegt ist.

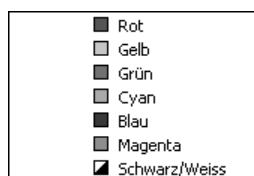


Abbildung 3.32: Einrückung um fünf Zeichen

Locked: Damit der Benutzer die Zeichenfolge des Textfelds nicht ändern kann, müssen Sie die Eigenschaft *Locked* auf den Wert *TRUE* setzen. Das Textfeld wird dann für die Eingabe gesperrt.

Reiter „Schriftart/Farbe/Bild“

Über die restlichen Reiter können Sie die Eigenschaften für den Schriftstil, die Vorder- und Hintergrundfarbe sowie den Mauszeiger festlegen.

Listeneinträge verwalten

Wenn Sie zusätzlich zu den Texten auch Symbole festlegen wollen – was ja der Fall ist, da Sie sonst ja auch die *ComboBox* verwenden könnten –, dann müssen Sie zunächst die Abbildungsliste an das Steuerelement binden. Hierfür besitzt das *ImageCombo*-Steuerelement die Eigenschaft *ImageList*.

```
ImageCombo1.ImageList = ImageList1
```

Nun können Sie damit beginnen, die Liste mit Elementen zu füllen. Nehmen wir an, Sie wollen eine Farbauswahlliste mit den AutoCAD-Grundfarben erstellen. Dazu müssten die einzelnen Farben, sieben an der Zahl, als einzelne Symbole in der Abbildungsliste vorhanden sein. Die Reihenfolge können Sie zwar selbst festlegen, vorzugsweise ist aber eine Reihenfolge nach den Farbwerten zu wählen, zuerst Rot, dann Gelb, dann Grün usw.

Jetzt kann damit begonnen werden, die einzelnen Listeneinträge zu erstellen. Das erste Element wird die Farbe Rot mit dem entsprechenden Index des Symbols in der Abbildungsliste.

```
ImageCombo1.ComboItems.Add Text:"Rot", Image:=1
```

Alle anderen Farben folgen entsprechend, so dass Sie eine vollständige Auflistung der einzelnen Farben erhalten.

```
ImageCombo1.ComboItems.Add Text:="Gelb", Image:=2
ImageCombo1.ComboItems.Add Text:="Grün", Image:=3
ImageCombo1.ComboItems.Add Text:="Cyan", Image:=4
ImageCombo1.ComboItems.Add Text:="Blau", Image:=5
ImageCombo1.ComboItems.Add Text:="Magenta", Image:=6
ImageCombo1.ComboItems.Add Text:="Schwarz/Weiss", Image:=7
```

Die Liste wäre jetzt mit den einzelnen Elementen gefüllt, im Textfeld wird allerdings immer noch die Zeichenfolge angezeigt, die Sie in der *Text*-Eigenschaft festgelegt haben. Das aktuell gewählte Listenelement ist in der Eigenschaft *SelectedItem* des *ImageCombo*-Objekts gespeichert. Zu diesem Zeitpunkt gibt es noch den Wert *Nothing* zurück. Erst wenn der Benutzer einen Listeneintrag auswählt oder Sie einen Verweis auf einen Listeneintrag an die *SelectedItem*-Eigenschaft übergeben, ist ein Eintrag gewählt worden, was hiermit erledigt wird.

Auswahl der Farbe Rot:

```
ImageCombo1.SelectedItem = ImageCombo1.ComboItems(1)
```

Tipp

Verwechseln Sie nicht die Texteigenschaft mit der Auswahl eines Listenelements. Die *Text*-Eigenschaft enthält lediglich die Zeichenfolge des Textfelds. Der Verweis auf das ausgewählte Element (Symbol + Text) ist in der *SelectedItem*-Eigenschaft gespeichert.

Die Auswertung des ausgewählten Elements erfolgt ebenfalls über die *SelectedItem*-Eigenschaft. Angenommen, Sie wollen das Symbol, das durch die Auswahl in dem Steuerelement angezeigt wird, auf ein Anzeigefeld (Abbildung 3.33) übertragen. Dann müssten Sie in der *Click*-Ereignisprozedur, die auf die Auswahl des Benutzers reagiert, folgende Zeile eintragen:

```
Image1.Picture = ImageList1.ListImages( _
    ImageCombo1.SelectedItem.Index).Picture
```

Der Trick bei der ganzen Sache ist, dass für die *Picture*-Eigenschaft des Anzeigefelds die Bilder der Abbildungsliste mitbenutzt werden. Da der Index der Elemente genau mit der Positionsangabe des Bilds in der Abbildungsliste übereinstimmt, ist die Übergabe ganz einfach. Wenn Sie für die Listeneinträge dieselben Schlüssel verwenden wie für die Bilder der Abbildungsliste, dann können Sie auch die *Key*-Eigenschaft zur Identifikation verwenden. Der Effekt ist derselbe.

```
Image1.Picture = ImageList1.ListImages( _
    ImageCombo1.SelectedItem.Key).Picture
```



Abbildung 3.33: Auswahl der Farbe Blau

Wenn Sie für ein ausgewähltes Element ein anderes Symbol verwenden wollen als das in der Liste angezeigte, dann können Sie in der *SellImage*-Eigenschaft den Index eines anderen Bilds festlegen. Dieses Bild muss allerdings auch in der Abbildungsliste, die an das *ImageCombo*-Steuerelement gebunden wird, enthalten sein.

3.3.7 Symbolleiste-Steuerelement (Toolbar)

(Beispielformular: 03\Steuerelemente.dvb!SymbolleisteFrm)

Das Symbolleisten-Steuerelement kennen Sie bereits in abgewandelter Form aus der Arbeit in AutoCAD. Es kann eine Reihe benutzerdefinierter Schaltflächen aufnehmen, die Sie unterschiedlich gestalten können, und eignet sich somit hervorragend als Ersatz für die in VBA fehlende Möglichkeit, Menüs in Formularen zu definieren. Aufgrund der Vielseitigkeit dieses Steuerelements erscheint dessen Verwendung auf den ersten Blick recht kompliziert. Wenn Sie allerdings das Prinzip der unterschiedlichen Schaltflächendefinitionen einmal verstanden haben, wird Ihnen dieses Steuerelement gute Dienste leisten.

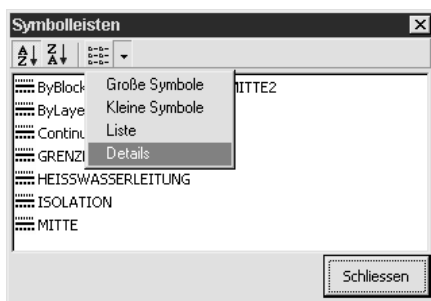


Abbildung 3.34: Symbolleisten-Steuerelement

Alle Schaltflächen, die dieses Steuerelement aufnimmt, werden in der *Buttons*-Auflistung verwaltet. Eine einzelne Schaltfläche wird durch das *Button*-Objekt dargestellt. Wenn Sie einer Schaltfläche den Stil *tbrDropDown* zuordnen, können Sie der Schaltfläche zusätzlich ein Menü zuordnen. Dieses Menü wird durch die *ButtonMenus*-Auflistung dargestellt und ist wiederum über das *Button*-Objekt abrufbar. Die Speicherung der einzelnen Menüpunkte erfolgt in der *ButtonMenus*-Auflistung.

Sie können auch bei diesem Steuerelement „fast“ alle Einstellungen zur Entwurfszeit vornehmen, indem Sie die Eigenschaftsseite aufrufen. Diese ist in insgesamt drei Bereiche unterteilt, wobei der letzte lediglich das Symbol für den benutzerdefinierten Mauszeiger einstellt. Beschäftigen wir uns darum mit den ersten beiden Reitern, die jeder für sich bereits recht umfangreich sind.

Reiter „Allgemein“

In diesem Bereich des Dialogfelds passen Sie die Eigenschaften des Steuerelements selbst an. Diese Eigenschaften beeinflussen größtenteils die Darstellung der Symbolleiste:

MousePointer: Legt das Symbol fest, das der Mauszeiger annehmen soll, wenn er sich über dem Steuerelement befindet.

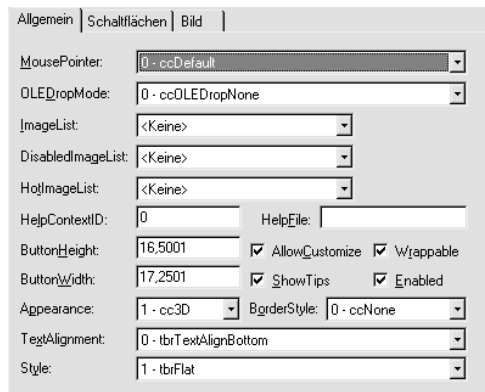


Abbildung 3.35: Der Abschnitt „Allgemein“ der Eigenschaftsseite

OLEDropMode: Mit dieser Eigenschaft können Sie festlegen, ob auf der Symbolleiste ein Objekt abgelegt werden kann.

Abbildungslisten: Normalerweise können Sie hier die drei Abbildungslisten an die Symbolleiste binden, so dass sich die Symbole den Schaltflächen zuordnen lassen. Leider funktioniert diese Methode auch bei diesem Steuerelement unter VBA nicht. Sie müssen also die Zuweisung zur Laufzeit vornehmen, aber dazu komme ich später. Zunächst klären wir die Bedeutung der einzelnen Abbildungslisten.

ImageList: Die Abbildungsliste, die Sie dieser Eigenschaft zuordnen, enthält die Symbole der Schaltflächen im Normalfall, wenn also die Schaltfläche weder gesperrt ist noch der Mauszeiger sich darüber befindet.

DisabledImageList: Die Abbildungsliste, die Sie dieser Eigenschaft zuordnen, enthält die Symbole der deaktivierten Schaltflächen (*Enabled* = *FALSE*).

HotImageList: Die Abbildungsliste, die Sie dieser Eigenschaft zuordnen, enthält die Symbole für die aktiven Schaltflächen. Eine Schaltfläche wird aktiv, wenn sich der Mauszeiger über einer ausführbaren Schaltfläche befindet.

Sie müssen nicht alle Abbildungslisten belegen. Für den Zustand, für den kein Symbol existiert, wird das Bild der Abbildungsliste *ImageList* angezeigt.

HelpFile/HelpContextID: In diesen Eigenschaften können Sie sowohl die Hilfedatei als auch eine ID des Hilfethemas der Symbolleiste zuordnen. Mit diesen beiden Eigenschaften hat es eine ganz besondere Bewandnis: Symbolleisten lassen sich zur Laufzeit durch den Benutzer anpassen, ohne dass Sie eine einzige Codezeile dafür schreiben, allerdings nur, wenn Sie dies mit der Eigenschaft *AllowCustomize* zulassen. Wenn Sie einmal das Beispielformular starten und einen Doppelklick auf einen leeren Bereich der Symbolleiste ausführen, dann wird ein Einstellungsdialog gestartet. In diesem Dialogfenster kann der Benutzer die Schaltflächen der Symbolleiste neu anordnen.

Die Hilfedatei und die ContextID, die Sie der Symbolleiste zugeordnet haben, kann dann durch den Benutzer in diesem Dialogfenster aufgerufen werden. Wenn Sie dem Benutzer die automatische Schaltflächenanpassung nicht gestatten möchten, dann deaktivieren Sie diese Funktion, indem Sie die Eigenschaft *AllowCustomize* auf *FALSE* setzen.

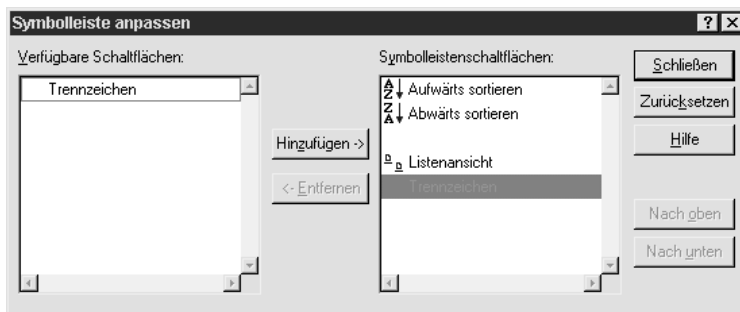


Abbildung 3.36: Anpassungsfenster für Symbolleisten

Tip

Da der Benutzer die Möglichkeit hat, die Schaltflächen über ein Dialogfenster neu zu ordnen, ist es wichtig, dass Sie für alle Schaltflächen die *Key*-Eigenschaft (Schlüssel) vergeben. Damit lassen sich die einzelnen Schaltflächen besser identifizieren.

ButtonHeight/ButtonWidth: Die Werte dieser Eigenschaften können Sie nicht ändern. Sie geben lediglich die aktuelle Höhe und Breite einer einzelnen Schaltfläche wieder. Diese Werte sind hauptsächlich davon abhängig, ob und welche Symbole Sie verwenden und ob und wo Sie einen Schaltflächentext festlegen.

Appearance: Legt fest, ob die Symbolleiste mit einem 3D-Effekt oder flach dargestellt werden soll.

TextAlignment: Mit dieser Eigenschaft bestimmen Sie, ob die Schaltflächentexte unterhalb oder rechts vom Schaltflächensymbol dargestellt werden sollen.

Style: Diese Eigenschaft gibt vor, ob die Schaltflächen mit einem 3D-Effekt oder flach dargestellt werden sollen.

AllowCustomize: Legt fest, ob der Benutzer die Symbolleiste anpassen kann.

Wrappable: Mit dieser Eigenschaft können Sie festlegen, ob die Symbolleiste umgebrochen werden kann, falls nicht alle Schaltflächen in eine Zeile passen. Wenn Sie diese Eigenschaft aktivieren (*Wrappable=TRUE*), dann wird die Toolbar-Leiste, falls erforderlich, in der nächsten Zeile fortgesetzt.

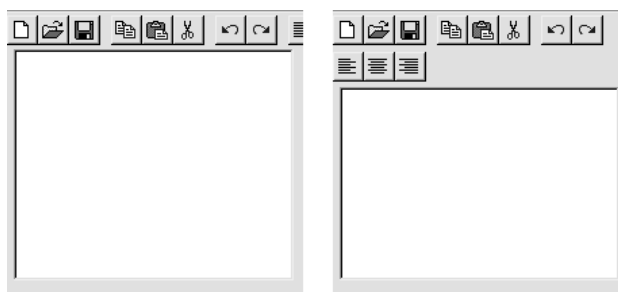


Abbildung 3.37: Links wurde die Eigenschaft deaktiviert, rechts aktiviert

ShowTips: Wenn Sie den Wert dieser Eigenschaft auf *TRUE* setzen, dann werden die Infotexte der einzelnen Schaltflächen, falls diese belegt wurden, angezeigt.

BorderStyle: Mit dieser Eigenschaft können Sie einen Rahmen darstellen, der um die gesamte Symbolleiste gezogen wird. Diese Eigenschaft ist allerdings nur wirksam, wenn die Eigenschaft *Appearance* auf den Wert *ccFlat* eingestellt wurde.

Die Anpassung bzw. den Abruf der aktuellen Eigenschaftswerte können Sie natürlich auch zur Laufzeit durchführen, indem Sie die angesprochenen Eigenschaften mit dem Steuerelement ansprechen.

Die Anpassung der *Style*-Eigenschaft könnte beispielsweise wie folgt aussehen:

```
ToolBar1.Style = tbrStandard
```

Reiter „Schaltflächen“

Um die Schaltflächen, die die Symbolleiste anzeigen soll, zur Entwurfszeit festzulegen, müssen Sie in diesen Bereich der Eigenschaftsseite wechseln. Zunächst ist nur der obere Bereich dieser Ansicht interessant. Hier können Sie die einzelnen Schaltflächen hinzufügen bzw. entfernen und zwischen den vorhandenen hin- und herwechseln, um eventuell verschiedene Eigenschaften anzupassen.

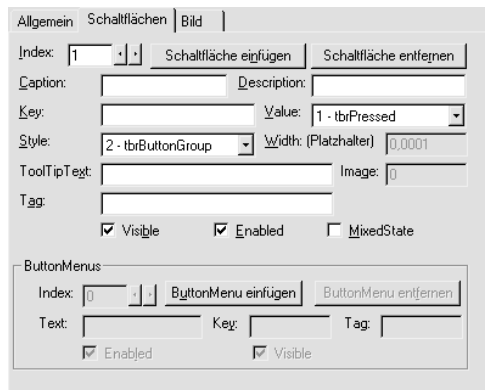


Abbildung 3.38: Der Abschnitt „Schaltflächen“ der Eigenschaftsseite

Alle Schaltflächen, die Sie hier bearbeiten, können Sie zur Laufzeit über die *Buttons*-Auflistung abrufen.

Caption: Diese Eigenschaft legt die Schaltflächenbeschriftung fest. Je nach Einstellung der Eigenschaft *TextAlignment* wird der Text unterhalb oder rechts neben dem Schaltflächensymbol angezeigt.

Tipp

Wenn Sie Schaltflächen beschriften, dann sollten die Zeichenfolgen kurz und knapp ausfallen, da alle Schaltflächen dieselbe Breite erhalten. Stellen Sie sich vor, Sie würden für eine einzige Schaltfläche z.B. die Beschriftung „Zeichnung exportieren“ vergeben. Dann würden alle anderen Schaltfläche eine Breite erhalten, als würden sie ebenfalls diese Beschriftung besitzen.

Description: Über diese Eigenschaft können Sie einen Beschreibungstext für die Schaltfläche festlegen. Dieser Text wird dann neben dem Schaltflächensymbol im Anpassungsfenster für Symbolleisten (Abbildung 3.36) angezeigt.

Key: Legt den Schlüssel für die Schaltfläche fest. Die Vergabe eines Schlüssels für einzelne Schaltflächen kann ich für dieses Steuerelement nur empfehlen.

Value: Diese Eigenschaft speichert den Zustand der Schaltfläche – entweder gedrückt (*tbrPressed*) oder normal (*tbrUnpressed*). Das Festlegen dieser Eigenschaft zur Entwurfszeit hängt mit der *Style*-Eigenschaft der Schaltflächen zusammen. Sie können so bereits die Schaltfläche in den gedrückten Zustand versetzen, wenn Sie *Style* auf *tbrCheck* oder *tbrButtonGroup* eingestellt haben. Bei allen anderen Stilen macht das wenig Sinn.

Style: Mit dieser Eigenschaft legen Sie den zuvor bereits erwähnten Stil der Schaltfläche fest. Jeder einzelne Stil erzielt nicht nur eine andere Darstellungsform der Schaltfläche, sondern definiert dadurch auch deren Verwendungszweck.

tbrDefault (Standardeinstellung): Durch diese Einstellung legen Sie die Schaltfläche als ganz normale Befehlsschaltfläche fest.

tbrCheck: Wenn Sie diese Einstellung verwenden, dann kann die Schaltfläche als Umschalter verwendet werden, ähnlich einem Umschaltfeld (*Kapitel 3.2.10*).

tbrButtonGroup: Diese Einstellung definiert die Schaltfläche als Mitglied einer Schaltflächengruppe. Innerhalb einer Gruppe kann nur jeweils eine Schaltfläche aktiviert werden. Als Beispiel für solch eine Gruppierung könnten die Schaltflächen LINKSBÜNDIG, RECHTSBÜNDIG, ZENTRIERT und BLOCKSATZ bei einem Textprogramm dienen. Alle Schaltflächen dieses Stils, die nebeneinander liegen, gehören derselben Gruppe an. Um eine neue Gruppe zu beginnen, müssen Sie die Gruppen mit einer Trennungslinienschaltfläche voneinander abgrenzen.

tbrSeparator: Mit dieser Einstellung degradieren Sie die Schaltfläche zu einer einfachen Trennungslinie mit fester Breite. Diese kann z.B. als optische Abgrenzung zwischen Schaltflächen unterschiedlicher Themen dienen.

tbrPlaceholder: Dieser Stil gleicht von der Darstellung her einer Trennungslinie, allerdings ist deren Breite variabel.

tbrDropDown: Hierbei handelt es sich um einen ganz besonderen Stil. Der Schaltfläche wird ein DropDown-Button zugewiesen, über den der Benutzer ein Menü aktivieren kann, das Sie unter *ButtonMenus* der Schaltfläche zugeordnet haben.

Width: Diese Eigenschaft wird nur für Schaltflächen des Typs *tbrPlaceholder* verwendet. Hier bestimmen Sie die Breite der Platzhalterschaltfläche.

ToolTipText: In dieser Eigenschaft können Sie den gelben Infotext für eine einzelne Schaltfläche festlegen. Dieser wird immer dann angezeigt, wenn sich der Mauszeiger über der entsprechenden Schaltfläche befindet – vorausgesetzt, die Eigenschaft *ShowTips* der Symbolleiste wurde aktiviert. Die Anzeige eines Infotexts ist vor allem dann wichtig, wenn Sie lediglich Symbole in den Schaltflächen darstellen.

Tag: In dieser Eigenschaft können Sie zusätzliche Informationen zu jeder Schaltfläche ablegen.

Visible: Mit dieser Eigenschaft steuern Sie die Sichtbarkeit der Schaltflächen. Wenn Sie dieser Eigenschaft den Wert *FALSE* zuweisen, wird die entsprechende Schaltfläche ausgeblendet. Der Wert *TRUE* blendet sie hingegen wieder ein. Die aktuelle Reihenfolge der Schaltflächen bleibt von diesem Vorgang unbeeinflusst.

Enabled: Mit dieser Eigenschaft sperren Sie Schaltflächen, indem Sie ihnen den Wert *FALSE* zuweisen. Eine solche Schaltfläche bleibt zwar sichtbar, wird aber abgeschwächt dargestellt. Wenn Sie der Eigenschaft *DisabledImageList* eine Abbildungsliste zugeordnet haben, dann wird das Symbol dieser Liste statt des Symbols der Eigenschaft *ImageList* angezeigt.

Image: Diese Eigenschaft können Sie zur Entwurfszeit nicht belegen, da eine Zuordnung der Abbildungslisten erst zur Laufzeit möglich ist. Aber so viel vorab: Hier wird die Position des Symbols der Abbildungsliste gespeichert, das auf der Schaltfläche dargestellt werden soll. Solange diese Eigenschaft den Wert 0 besitzt, wird kein Symbol angezeigt.

Zuordnung der Schaltflächenmenüs

Wie bereits erwähnt, können Sie bestimmten Schaltflächen Menüs zuordnen, das heißt, zuordnen können Sie diese Menüs allen Schaltflächen. Angezeigt wird das jeweilige Menü erst, wenn Sie der Schaltfläche den Stil *tbrDropDown* zugewiesen haben und der Benutzer dann auf die DropDown-Schaltfläche drückt.

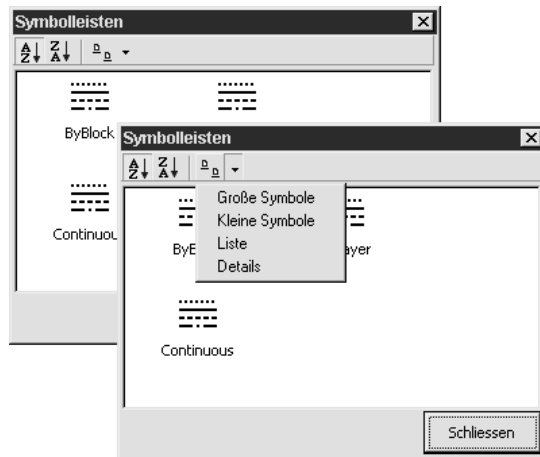


Abbildung 3.39: Schaltflächenmenü in normalem und danach in aktiviertem Zustand

Die einzelnen Menüpunkte werden in der Schaltflächeneigenschaft *ButtonMenus* verwaltet, wobei die Reihenfolge in dieser Auflistung der Anzeigereihenfolge im Menü entspricht. Sie können diese Menüs bereits zur Entwurfszeit festlegen, indem Sie zur entsprechenden Schaltfläche wechseln und die Menüpunkte nach dem gleichen Prinzip erstellen wie die Schaltflächen. Dazu finden Sie im unteren Bereich des Abschnitts *Schaltflächen* den abgegrenzten Teil *ButtonMenus*.



Abbildung 3.40: Schaltflächenmenüs entwerfen

Das Hinzufügen eines Menüpunkts erfolgt mit der Schaltfläche `BUTTONMENU EINFÜGEN`, das Entfernen mit `BUTTONMENU ENTFERNEN`. Jedem Menüpunkt können Sie jetzt noch ganz spezielle Eigenschaften zuordnen, die sowohl die Verwaltung als auch die optische Darstellung des einzelnen Eintrags beeinflussen.

Index: Zeigt die Position innerhalb der *ButtonMenus*-Auflistung an und somit die Position innerhalb des Menüs.

Text: Mit dieser Eigenschaft bestimmen Sie den Text des Menüpunkts.

Key: Hier können Sie einen eindeutigen Schlüssel für den Menüpunkt festlegen. Auch hier ist die Festlegung von eminenter Bedeutung, da hierdurch die Identifizierung des Menüpunkts besser erfolgen kann. Sie sollten sich von vornherein angewöhnen, für jeden Menüpunkt des Symbolleisten-Steuerelements einen Schlüssel zu vergeben, der sich von allen anderen unterscheidet.

Tag: In dieser Eigenschaft können Sie zusätzliche Informationen zum Menüpunkt ablegen.

Enabled: Mit dieser Eigenschaft sperren Sie den Menüpunkt für die Benutzerauswahl, indem Sie den Wert *FALSE* zuweisen. Der Menüpunkt wird dann abgeschwächt dargestellt. Um die Sperrung wieder aufzuheben, weisen Sie der Eigenschaft den Wert *TRUE* zu.

Visible: Diese Eigenschaft steuert die Sichtbarkeit des Menüpunkts. Wenn Sie den Wert *FALSE* zuweisen, wird die Schaltfläche ausgeblendet, sie bleibt aber weiterhin über den Programmcode abrufbar. Durch Zuweisung des Werts *TRUE* wird der Menüpunkt wieder sichtbar.

Tipp

Sie können die Darstellung der Schaltflächenmenüs bereits zur Entwurfszeit kontrollieren, indem Sie zweimal auf das Steuerelement klicken (keinen Doppelklick). Dadurch wird die Symbolleiste aktiviert und Sie können die entsprechende DropDown-Schaltfläche betätigen.

Zusammenfassend kann man sagen, dass die Erstellung der Schaltflächen und Schaltflächenmenüs zur Entwurfszeit wesentlich komfortabler ist und Sie viel schneller vorankommen, wie wenn Sie diesen Vorgang erst zur Laufzeit durchführen. Die Einschränkung, die Schaltflächensymbole erst zur Laufzeit zuordnen zu können, kann man, denke ich, durchaus in Kauf nehmen.

Verwaltung der Schaltflächen zur Laufzeit

Die Verwaltung der Schaltflächen zur Laufzeit ist genauso einfach wie zur Entwurfszeit. Sie müssen lediglich die Abläufe, die Sie in der Eigenschaftsseite vornehmen, in eine entsprechende Code-Reihenfolge bringen. Die Funktionen zum Hinzufügen und Entfernen sind wie gehabt *Add*, *Remove* und *Clear*. Der Zugriff auf die Schaltflächenauflistung erfolgt über die *Toolbar*-Eigenschaft *Buttons*.

Die Abfolge in der Eigenschaftsseite würde in etwa wie folgt aussehen:

- Schaltfläche hinzufügen
- Schaltflächeneigenschaften festlegen
 - Eventuell Menüpunkt hinzufügen
 - Menüpunkteigenschaften festlegen
 - Den Vorgang so lange wiederholen, wie Menüpunkte hinzugefügt werden sollen
- Und wieder von vorn beginnen

Die Erstellung über den Programmcode läuft nach demselben Muster ab.

```
Dim TBtn As MSComctlLib.Button
Dim TBtnMenu As MSComctlLib.ButtonMenu
ToolBar1.ImageList = ImageList1
```

1. Schaltfläche:

```
Set TBtn = ToolBar1.Buttons.Add
TBtn.Style = tbrButtonGroup
TBtn.ToolTipText = "Aufwärts sortieren"
TBtn.Image = 1 'Erstes Symbol der Abbildungsliste zuordnen
TBtn.Value = tbrPressed
```

2. Schaltfläche:

```
Set TBtn = ToolBar1.Buttons.Add
TBtn.ToolTipText = "Abwärts sortieren"
TBtn.Image = 2 'Zweites Symbol der Abbildungsliste zuordnen
TBtn.Style = tbrButtonGroup
```

3. Schaltfläche:

```
Set TBtn = ToolBar1.Buttons.Add
TBtn.Style = tbrSeparator
```

4. Schaltfläche:

```
Set TBtn = ToolBar1.Buttons.Add
TBtn.ToolTipText = "Ansichten"
TBtn.Image = 3 'Drittes Symbol der Abbildungsliste zuordnen
TBtn.Style = tbrDropdown
```

Menü der vierten Schaltfläche erstellen:

1. Menüpunkt

```
Set TBtnMenu = TBtn.ButtonMenus.Add(Text:="Große Symbole")
TBtnMenu.Key = "lrgIcons"
```

2. Menüpunkt

```
Set TBtnMenu = TBtn.ButtonMenus.Add(Text:="Kleine Symbole")
TBtnMenu.Key = "smIcons"
```

3. Menüpunkt

```
Set TBtnMenu = TBtn.ButtonMenus.Add(Text:="Liste")
TBtnMenu.Key = "List"
```

4. Menüpunkt

```
Set TBtnMenu = TBtn.ButtonMenus.Add(Text:="Details")
TBtnMenu.Key = "Details"
```

5. Schaltfläche:

```
Set TBtn = ToolBar1.Buttons.Add
TBtn.ToolTipText = "Hilfe"
```

usw.

Bei der Erstellung der Schaltflächen bzw. deren Menüpunkte können Sie der *Add*-Anweisung bereits diverse Eigenschaften in den Parametern mitgeben. Um die Schaltflächen wieder zu entfernen, verwenden Sie wieder die *Remove*- oder die *Clear*-Methode.

```
Toolbar1.Buttons.Remove Index
```

Der Zugriff auf die einzelnen Schaltflächen erfolgt wie gehabt über die *Item*-Eigenschaft unter der Angabe der Position oder des Schlüssels (wenn Sie einen vergeben haben) der Schaltfläche.

```
Set TBtn = Toolbar1.Buttons(n)
```

Auf die Menüs der Schaltfläche können Sie über die *ButtonMenus*-Auflistung zugreifen. Dieser Zugriff erfolgt über eine Objektvariable, die einen Verweis auf die entsprechende Schaltfläche gespeichert hat:

```
Set TBtn = Toolbar1.Buttons(n)
Set TBtnMenu = TBtn.ButtonMenus(n)
```

Oder Sie benutzen den direkten Weg über die *Buttons*-Auflistung:

```
Set TBtnMenu = Toolbar1.Buttons(n).ButtonMenus(n)
```

Symbolleisten-Ereignisse

Interessant bei der Verwendung von Symbolleisten sind vor allem drei Ereignisprozeduren: *ButtonClick* wird ausgelöst, wenn der Benutzer auf eine Schaltfläche klickt. Da dieses Steuerelement nicht in der Lage ist, den Fokus zu erhalten, ist dies nur mit der Maus möglich. Bei Schaltflächen vom Typ *tbrSeparator* und *tbrPlaceholder* wird kein Ereignis ausgelöst. Im Parameter dieser Prozedur erhalten Sie das Schaltflächenobjekt, das das Ereignis ausgelöst hat. Sie brauchen dann im Quellcode nur zu kontrollieren, welche Schaltfläche es war, und können entsprechend reagieren.

```
Private Sub Toolbar1_ButtonClick(ByVal Button As _
MSComctlLib.Button)
Select Case Button.Key
Case "sortup"
    ListView1.SortOrder = lvwAscending
Case "sortdown"
    ListView1.SortOrder = lvwDescending
End Select
End Sub
```

Das *ButtonDropDown*-Ereignis wird ausgelöst, wenn der Benutzer auf den DropDown-Pfeil einer Schaltfläche des Typs *tbrDropDown* klickt, noch bevor das dazugehörige Menü eingeblendet wird. Über den Parameter der Prozedur können Sie die Schaltfläche identifizieren, deren DropDown-Schaltfläche betätigt wurde.

```
Private Sub Toolbar1_ButtonDropDown(ByVal Button As _
MSComctlLib.Button)
End Sub
```

Das dritte Ereignis, das die Schaltflächen direkt betrifft, ist *ButtonMenuClick*. Dieses Ereignis wird immer dann ausgelöst, wenn der Benutzer einen Menüpunkt aus dem Schaltflächenmenü ausgewählt hat. Da diese Prozedur im Parameter lediglich den Menüpunkt als Objekt mitgibt

und so kein Bezug zur übergeordneten Schaltfläche besteht, haben Sie zwei Möglichkeiten, den richtigen Menüpunkt zu identifizieren. Ein kleines Beispiel: Stellen Sie sich vor, Sie haben zwei *ListView*-Steuerelemente. Für beide Listen wollen Sie jeweils eine DropDown-Schaltfläche anlegen, z.B. zum Einstellen der entsprechenden *ListView*-Ansicht (Große Symbole, Kleine Symbole, Liste und Detail). Da beide denselben Aufbau haben (*Item(1)*-*Item(4)*), kommen Sie mit der Auswertung des Index des geklickten Menüpunkts nicht weiter, denn sowohl im ersten als auch im zweiten DropDown-Menü gibt es eine Schaltfläche mit z.B. dem Index 1. Also bleibt in nächster Konsequenz die Zuordnung eines Schlüssels, womit wir bei der ersten Identifikationsmöglichkeit wären. Wenn Sie für jeden Menüpunkt einen anderen Schlüsselnamen vergeben, können Sie den Menüpunkt korrekt identifizieren.

```
Private Sub Toolbar1_ButtonMenuClick(ByVal ButtonMenu As _
MSComctlLib.ButtonMenu)
Select Case ButtonMenu.Key
Case "lrgIconsLV1"
    ListView1.View = lvwIcon
Case "smIconsLV1"
    ListView1.View = lvwSmallIcon
Case "ListLV1"
    ListView1.View = lvwList
Case "DetailsLV1"
    ListView1.View = lvwReport
Case "lrgIconsLV2"
    ListView2.View = lvwIcon
Case "smIconsLV2"
    ListView2.View = lvwSmallIcon
Case "ListLV2"
    ListView2.View = lvwList
Case "DetailsLV2"
    ListView2.View = lvwReport
End Select
End Sub
```

Die zweite Möglichkeit wäre die Benutzung der Eigenschaft *Parent*, die jedes Menüobjekt besitzt. In dieser Eigenschaft ist ein Verweis auf die übergeordnete Schaltfläche gespeichert, womit die Identifizierung wieder ganz einfach wird.

```
Private Sub Toolbar1_ButtonMenuClick(ByVal ButtonMenu As _
MSComctlLib.ButtonMenu)
If ButtonMenu.Parent.Key = "LView1" Then
    Select Case ButtonMenu.Key
    Case "lrgIconsLV1"
        ListView1.View = lvwIcon
    Case "smIconsLV1"
        ListView1.View = lvwSmallIcon
    Case "ListLV1"
        ListView1.View = lvwList
    Case "DetailsLV1"
        ListView1.View = lvwReport
    End Select
Else
    Select Case ButtonMenu.Key
    Case "lrgIcons"
```

```

        ListView2.View = 1vwIcon
    Case "smIcons"
        ListView2.View = 1vwSmallIcon
    Case "List"
        ListView2.View = 1vwList
    Case "Details"
        ListView2.View = 1vwReport
    End Select
End If
End Sub

```

Symbolleisten-Methoden

Das Symbolleisten-Steuerelement stellt noch einige andere Methoden zur Verfügung, die vielleicht interessant für Sie sein könnten. Mit der ersten Anweisung `Customize` können Sie das Dialogfenster „Symbolleisten anpassen“ (Abbildung 3.36) manuell aufrufen. Das ist sinnvoll, wenn Sie steuern möchten, wann der Benutzer dieses Dialogfenster benutzen darf und wann nicht. Außerdem können Sie mit dieser Vorgehensweise dem Benutzer eine Schaltfläche für diese Funktion zur Verfügung stellen. (Wenn Sie die `Customize`-Anweisung verwenden, können Sie `AllowCustomize` auf `FALSE` setzen.) Damit die Anpassungen, die der Benutzer an der Symbolleiste vorgenommen hat, auch beim nächsten Aufruf des Programms vorhanden sind, stellt das Steuerelement zwei Anweisungen zur Verfügung:

SaveToolBar: Mit dieser Anweisung können Sie den aktuellen Zustand der Symbolleiste in der Windows-Registrierung speichern.

RestoreToolBar: Mit dieser Anweisung können Sie den Zustand der Symbolleiste, der mit der `SaveToolBar`-Anweisung gespeichert wurde, wiederherstellen.

Für beide Anweisungen sind gewisse Parameterangaben erforderlich, die die Position der Daten in der Registrierung bestimmen. Wenn Sie beispielsweise die Speicherung mit folgenden Parametern ausführen,

```
ToolBar1.SaveToolBar "MyApp", "UserForm1", "ToolBar1"
```

dann finden Sie den Eintrag in der Registrierung unter folgendem Schlüssel wieder:

```
HKEY_CURRENT_USER\MyApp\UserForm1
```

Zur Wiederherstellung verwenden Sie dann dieselbe Positionsangabe:

```
ToolBar1.RestoreToolBar "MyApp", "UserForm1", "ToolBar1"
```

3.3.8 Strukturansicht-Steuerelement (TreeView)

(Beispielformular: 03\Steuerelemente.dvb!TreeViewFrm)

Dieses Steuerelement dient der hierarchischen Darstellung von Texten, die zusätzlich durch ein Symbol gekennzeichnet sein können. Solche Strukturansichten kennen Sie bereits aus dem AutoCAD-DesignCenter. Dort wird diese Ansicht z.B. zur Anzeige der Zeichnungsstruktur verwendet.

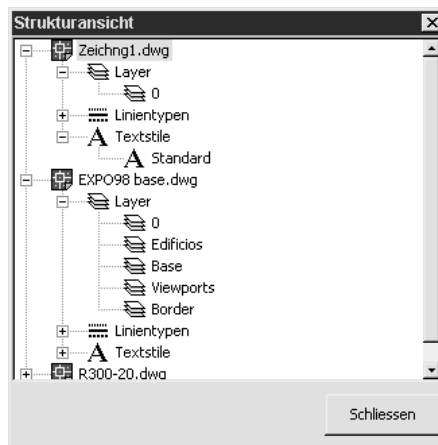


Abbildung 3.41: Strukturansicht der AutoCAD-Sitzung

Alle Elemente der Strukturansicht werden in der *Nodes*-Auflistung gespeichert. Entscheidend bei der Verwendung der Strukturansicht ist das Verständnis dafür, wie dieses Steuerelement aus den Elementen eine hierarchische Darstellung erzeugt. Eigentlich müssen Sie sich nur an eine Vorgabe halten. Betrachten Sie die Einträge nicht als Struktur, sondern als eine Auflistung von *Node*-Objekten, die aufeinander verweisen. Bevor wir uns allerdings mit der Erstellung einer Ansicht wie in Abbildung 3.41 befassen, sollten Sie sich mit den Eigenschaften des Steuerelements befassen. Starten Sie hierfür wieder die Eigenschaftsseite des Steuerelements.

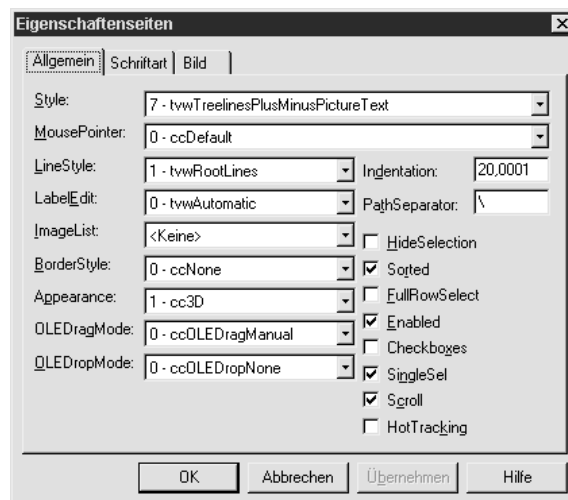


Abbildung 3.42: Eigenschaftsseite des Strukturansicht-Steuerelements

Die Eigenschaftsseite ist wieder in verschiedene Bereiche unterteilt. Unter dem Reiter „Schriftart“ nehmen Sie wieder die Einstellungen des Schriftstils vor, im Reiter „Bild“ wird der benutzerdefinierte Mauszeiger festgelegt.

Reiter „Allgemein“

In diesem Bereich der Eigenschaftsseite können Sie wieder die Werte der Eigenschaften zur Entwurfszeit festlegen und die Darstellung bzw. Funktion des Steuerelements bestimmen.

Style: Mit dieser Eigenschaft legen Sie die Art der Darstellung der einzelnen Einträge fest. Diese Eigenschaft beeinflusst nicht die Node-Objekte an sich, sondern betrifft nur die optische Darstellung und die Art der Bedienung des Steuerelements. Sie können aus insgesamt acht verschiedenen Darstellungsformen wählen.

Konstante	Beschreibung
<i>twwTextOnly</i>	Es wird lediglich der Wert der <i>Text</i> -Eigenschaft angezeigt.
<i>twwPictureText</i>	Es wird sowohl der Wert der <i>Text</i> -Eigenschaft als auch das dazugehörige Symbol angezeigt.
<i>twwPlusMinusText</i>	Zusätzlich zum Text wird auch ein Plus oder Minus vor dem Element angezeigt, das Unterelemente besitzt.
<i>twwPlusPictureText</i>	Es werden der Text, das Symbol und ein Plus-/Minus-Zeichen für Einträge, die Unterelemente besitzen, angezeigt.
<i>twwTreelinesText</i>	Es wird der Text angezeigt und zusätzlich in Form einer Linie eine Verbindung zwischen den Objekten, die einander zugeordnet sind.
<i>twwTreelinesPictureText</i>	Wie <i>twwTreelinesText</i> , nur dass zusätzlich ein Symbol angezeigt werden kann.
<i>twwTreelinesPlusMinusText</i>	Zusätzlich zur Linienverbindung und zum Text werden noch die Knoten mit einem Plus-/Minus-Zeichen überlagert.
<i>twwTreelinesPlusMinusPicture-Text</i>	Diese Einstellung stellt den Text, das Symbol und die Linienverbindung mit dem Plus-/Minuszeichen auf den Knoten dar.

Tabelle 3.7: Konstanten der Eigenschaft *Style*

Je nachdem, ob Sie das Plus-/Minus-Zeichen darstellen lassen oder nicht, beeinflusst dies die Navigationsmöglichkeit des Benutzers. Wenn Sie das Zeichen weglassen, dann muss der Benutzer einen Knoten über den Texteintrag öffnen. Die Benutzung des Zeichens ermöglicht es ihm, die Knoten zusätzlich über das Zeichen zu öffnen oder zu schließen. Außerdem wird durch das Zeichen angezeigt, ob der Knoten geöffnet (Minus wird angezeigt) oder geschlossen ist (Plus wird angezeigt).

LineStyle: Diese Eigenschaft wird nur verwendet, wenn Sie in der *Style*-Eigenschaft eine Darstellung der Verbindungen in Form einer Linie eingestellt haben. Sie können zwischen der Einstellung *twwTreeLines* und *twwRootLines* wählen. Bei *twwRootLines* wird zusätzlich zu den Linien zwischen den unter- und den jeweils übergeordneten Einträgen eine Verbindung zwischen den obersten Objekten (Wurzelebene) dargestellt.

Indentation: Legt die Weite der Einrückung von untergeordneten Einträgen fest. Dieser Wert darf keine negative Zahl sein. Wenn Sie diese Eigenschaft zur Laufzeit ändern, wird die Darstellung der Hierarchie entsprechend angepasst.

PathSeparator: In dieser Eigenschaft geben Sie das Trennungszeichen vor, das in Pfadangaben der *FullPath*-Eigenschaft verwendet wird. Das Standardzeichen dieser Eigenschaft ist „\“. Sie können diese Einstellung so belassen.

LabelEdit: In dieser Eigenschaft legen Sie fest, ob der Benutzer den Text des *Node*-Objekts bearbeiten darf. Diese Bearbeitungsfunktion wird durch einfaches Klicken mit der Maus auf den Text gestartet. Wenn Sie diese Eigenschaft auf den Wert *tvwManual* einstellen, ist die Bearbeitung gesperrt. Andernfalls müssten Sie auf das ausgelöste Ereignis reagieren.

ImageList: In dieser Eigenschaft wird die Abbildungsliste, die die Symbole der *Node*-Objekte enthält, gebunden. Dies können Sie wieder nur zur Laufzeit vornehmen und sollte somit die erste Aufgabe sein, die Sie im Quellcode erledigen.

AppearancelBorderStyle: Die *BorderStyle*-Eigenschaft wird nur verwendet, wenn Sie die Eigenschaft *Appearance* auf 2D einstellen. Dann können Sie mit *BorderStyle* festlegen, ob ein Rahmen um das Steuerelement gezogen werden soll oder nicht.

OleDragMode/OleDropMode: Diese beiden Eigenschaften legen das Verhalten für Ziehen und Ablegenvorgänge fest. *OleDragMode* bestimmt, ob das Steuerelement als Quelle solcher Vorgänge dienen kann und ob dies manuell oder automatisch erfolgen soll. *OleDropMode* bestimmt, ob auf dem Steuerelement ein Objekt abgelegt werden kann.

HideSelection: Mit dieser Eigenschaft steuern Sie, ob die Auswahl eines oder mehrerer Einträge auch angezeigt werden soll, wenn das Steuerelement nicht mehr den Fokus besitzt (*TRUE*), oder nicht (*FALSE*).

Sorted: Mit dieser Eigenschaft können Sie festlegen, ob *Node*-Objekte einer Hierarchiestufe alphabetisch sortiert werden sollen (*TRUE*) oder nicht (*FALSE*).

FullRowSelect: Durch die Einstellung dieser Eigenschaft auf den Wert *TRUE* können Sie festlegen, dass die gesamte Zeile des Eintrags selektiert wird, wenn der Benutzer sie auswählt. Dieser Effekt wird allerdings nur bei der Verwendung von Kontrollkästchen erzielt. In allen übrigen Einstellungen wird lediglich der Text des *Node*-Objekts ausgewählt.

Checkboxes: Mit dieser Eigenschaft legen Sie fest, ob vor dem *Node*-Objekt ein Kontrollkästchen angezeigt werden soll (*TRUE*) oder nicht (*FALSE*). Dieses Kontrollkästchen können Sie zur Mehrfachauswahl von Einträgen verwenden.

SingleSel: Wenn Sie diese Eigenschaft aktivieren (*TRUE*), dann wird nur das aktive *Node*-Objekt einschließlich der übergeordneten *Node*-Objekte geöffnet bzw. geschlossen. Ist bei der Auswahl eines Eintrags gleichzeitig ein anderes *Node*-Objekt geöffnet, das außerhalb der gewählten Hierarchie liegt, dann wird es geschlossen. Im *SingleSel*-Modus wird ein Eintrag bereits durch einfaches Anklicken geöffnet, während im anderen Modus ein Doppelklick auf den Eintrag notwendig ist.

Scroll: Diese Eigenschaft aktiviert (*TRUE*) oder deaktiviert (*FALSE*) die Verwendung der Bildlaufleisten. Wurden die Bildlaufleisten aktiviert, dann werden diese angezeigt, wenn die Liste der Einträge nicht mehr in den sichtbaren Bereich des Steuerelements passt.

HotTracking: Mit dieser Eigenschaft legen Sie fest, ob mit der Maus angezeigt werden soll, dass sich diese über einem Eintrag befindet (*TRUE*), oder nicht (*FALSE*). Der Mauszeiger wechselt dann auf das Handsymbol wie bei einem Hyperlink und der Eintrag wird unterstrichen.

Verwaltung der Node-Objekte

Alle Einträge werden in der *Nodes*-Auflistung des Steuerelements verwaltet und können über diese hinzugefügt, entfernt oder abgerufen werden. Ein einzelner Eintrag dieser Auflistung entspricht einem *Node*-Objekt, auch Knoten genannt. In diesem Knoten legen Sie im Prinzip die

Hierarchie der Elemente fest. Aber eins nach dem anderen. Zunächst müssen die einzelnen Elemente hinzugefügt werden. Dazu können Sie wieder die *Add*-Funktion der Auflistung verwenden. Die Parameter dieser Anweisung sind etwas komplizierter als die, die Sie von anderen *Add*-Anweisungen her kennen.

Bei dieser Anweisung sind vor allem die ersten beiden Parameter interessant. Im Parameter *Relative* können Sie die Indexnummer oder den Schlüssel eines vorhandenen *Node*-Objekts festlegen, mit dem das neue *Node*-Objekt eine Beziehung eingehen soll. Die Art dieser Beziehung wird im zweiten Parameter *Relationship* festgelegt. Für diesen Parameter existieren fünf verschiedene Einstellungsmöglichkeiten, die Sie der folgenden Tabelle entnehmen können.

Konstante	Beschreibung
<i>twwFirst</i>	Das neue <i>Node</i> -Objekt wird vor allen anderen <i>Node</i> -Objekten auf derselben Ebene angezeigt, die der Ebene des in <i>Relativ</i> angegebenen <i>Node</i> -Objekts entspricht.
<i>twwLast</i>	Das neue <i>Node</i> -Objekt wird hinter allen anderen <i>Node</i> -Objekten auf derselben Ebene angezeigt, die der Ebene des in <i>Relativ</i> angegebenen <i>Node</i> -Objekts entspricht.
<i>twwNext</i>	Das neue <i>Node</i> -Objekt wird hinter dem in <i>Relativ</i> angegebenen <i>Node</i> -Objekt platziert.
<i>twwPrevious</i>	Das neue <i>Node</i> -Objekt wird vor dem in <i>Relativ</i> angegebenen <i>Node</i> -Objekt platziert.
<i>twwChild</i>	Das neue <i>Node</i> -Objekt wird ein Unterelement des in <i>Relativ</i> angegebenen <i>Node</i> -Objekts

Tabelle 3.8: Konstanten des Parameters *Relationship*

Mit den restlichen Parametern bestimmen Sie wieder den Schlüssel (*Key*), den angezeigten Text (*Text*), die Nummer des angezeigten Symbols aus der gebundenen Abbildungsliste (*Image*) und die Nummer des Symbols, das angezeigt wird, wenn der Eintrag selektiert wurde (*SelectedImage*). Alle diese Parameter finden Sie auch als Eigenschaft im *Node*-Objekt wieder. Bei den ersten beiden Parametern ist das etwas anders. Sie finden den Verweis auf das *Node*-Objekt, das Sie im ersten Parameter festgelegt haben, in der Eigenschaft wieder, die durch den zweiten Parameter bestimmt wurde. Hört sich kompliziert an, ist es aber nicht.

Anhand des Beispielformulars zu diesem Thema lässt sich die Funktionalität der *Add*-Anweisung besser verdeutlichen. Das Ziel der Prozedur ist es, die Namen der geöffneten Zeichnungen sowie die darin enthaltenen Layer, Linientypen und Textstile hierarchisch im Steuerelement anzuzeigen. Schauen wir uns zunächst die gesamte Prozedur an, bevor ich die Details genau erkläre:

```
Private Sub UserForm_Initialize()  
Dim RootNode As MSComctlLib.Node  
Dim DrawNode As MSComctlLib.Node  
Dim xNode As MSComctlLib.Node  
Dim Drawing As AcadDocument  
TreeView1.ImageList = ImageList1  
For Each Drawing In Application.Documents  
Set RootNode = TreeView1.Nodes.Add(Text:=Drawing.Name, _  
Image:=1)
```

```

Set xNode = TreeView1.Nodes.Add(RootNode.Index, tvwChild, , _
    Text:="Layer", Image:=2)
For Each ActData In Drawing.Layers
    TreeView1.Nodes.Add xNode, tvwChild, , ActData.Name, _
        Image:=2
Next
Set xNode = TreeView1.Nodes.Add(RootNode.Index, tvwChild, , _
    Text:="Linientypen", Image:=3)
For Each ActData In Drawing.Linetypes
    TreeView1.Nodes.Add xNode, tvwChild, , ActData.Name, _
        Image:=3
Next
Set xNode = TreeView1.Nodes.Add(RootNode.Index, tvwChild, , _
    Text:="Textstile", Image:=4)
For Each ActData In Drawing.TextStyles
    TreeView1.Nodes.Add xNode, tvwChild, , ActData.Name, _
        Image:=4
Next
Next
End Sub

```

Wie Sie sehen können, ist der Code bereits hierarchisch aufgebaut. Die erste Schleife ermittelt die Zeichnungsamen der AutoCAD-Sitzung. Im zweiten Schritt wird jeweils ein Container für Layer, Linientypen und Textstile für jede Zeichnung angelegt. Der dritte Schritt umfasst die Ermittlung der jeweiligen Containerinhalte, die dann dem entsprechenden Eintrag hinzugefügt werden.

Das bedeutet im Einzelnen: Bei jedem Durchlauf der *Documents*-Auflistung wird ein Zeichnungsobjekt ausgelesen und der *Nodes*-Auflistung hinzugefügt. Der Verweis auf das so entstandene *Node*-Objekt wird in der Objektvariablen *RootNode* gespeichert (wird im nächsten Schritt benötigt). Da die Zeichnungen in der Wurzel angezeigt werden sollen, muss an dieser Stelle keine Beziehung angegeben werden. Diese Maßnahme ist nur erforderlich, wenn Sie eine Beziehung zu einem anderen *Node*-Objekt herstellen möchten.

```

For Each Drawing In Application.Documents
    Set RootNode = TreeView1.Nodes.Add(Text:=Drawing.Name, _
        Image:=1)

```

Der nächste Schritt ist die Erzeugung eines Sammelobjekts für alle Layer der Zeichnung, damit die Layer von den übrigen Einträgen separiert werden. Hier wird bereits die erste Beziehung festgelegt. Der Layercontainer soll ein Unterelement des *Node*-Objekts der aktuellen Zeichnung werden. Dazu wird als Beziehungstyp *twvChild* bestimmt und als übergeordnetes *Node*-Objekt *RootNode* (gibt den Wert der Standardeigenschaft *Index* zurück). Wenn Sie für das *Node*-Objekt der Zeichnung einen Schlüssel vergeben haben, dann können Sie auch die *Key*-Eigenschaft verwenden. Auch hier wird wieder der Verweis auf das neue Objekt in einer Objektvariablen (*xNode*) gespeichert.

```

Set xNode = TreeView1.Nodes.Add(RootNode, tvwChild, , _
    Text:="Layer", Image:=2)

```

Im dritten Schritt werden alle Layer der Zeichnung nacheinander ausgelesen und mit den Layernamen ein *Node*-Objekt der *Nodes*-Auflistung hinzugefügt. Da die Layernamen dem Layercontainer untergeordnet werden sollen, wird im Parameter *RelationShip twvChild* angegeben und im Parameter *Relativ* das dem Layer übergeordnete *Node*-Objekt (Layercontainer).

```

For Each ActData In Drawing.Layers
    TreeView1.Nodes.Add xNode, tvwChild, , ActData.Name, _
        Image:=2
Next

```

Dieser Vorgang wiederholt sich für die Linientypen:

```

Set xNode = TreeView1.Nodes.Add(RootNode.Index, tvwChild, , _
    Text:="Linientypen", Image:=3)
For Each ActData In Drawing.Linetypes
    TreeView1.Nodes.Add xNode, tvwChild, , ActData.Name, _
        Image:=3
Next

```

und für die Textstile der Zeichnung:

```

Set xNode = TreeView1.Nodes.Add(RootNode.Index, tvwChild, , _
    Text:="Textstile", Image:=4)
For Each ActData In Drawing.TextStyles
    TreeView1.Nodes.Add xNode, tvwChild, , ActData.Name, _
        Image:=4
Next

```

Dieser Vorgang wird für jede Zeichnung wiederholt, so dass eine Hierarchie wie in Abbildung 3.41 entsteht.

Ein *Node*-Objekt kann zwar über die *Item*-Eigenschaft abgerufen werden, in den meisten Fällen wird dies aber eher über die Beziehungen des jeweiligen *Node*-Objekts erfolgen. Stellen Sie sich vor, Sie wollen die untergeordneten *Node*-Objekte des aktuell gewählten Eintrags ermitteln. Ein Verweis auf das aktuell gewählte *Node*-Objekt wird in der Eigenschaft *SelectedItem* gespeichert. Dieses Objekt dient nun als Ausgangspunkt. Mit der *Node*-Eigenschaft *Children* lässt sich zunächst überprüfen, ob das *Node*-Objekt überhaupt untergeordnete Objekte besitzt. Diese Eigenschaft enthält nämlich die Anzahl dieser Objekte.

```

Set SelNode = TreeView1.SelectedItem
If SelNode.Children > 0 Then
End If

```

Die *Child*-Eigenschaft enthält einen Verweis auf das erste untergeordnete *Node*-Objekt. Also wird im nächsten Schritt dieser Verweis an eine Objektvariable übergeben.

```

Set ChildNode = SelNode.Child

```

Die nächste Eigenschaft, die nun zum Einsatz kommt, ist *Next*. Diese Eigenschaft enthält einen Verweis auf das nächstfolgende *Node*-Objekt derselben Hierarchiestufe. Da sich alle nun folgenden Objekte auf derselben Stufe befinden, kann die Abfrage in einer Schleife erfolgen. Ist das letzte *Node*-Objekt der Hierarchie erreicht, gibt die Eigenschaft *Next* den Wert *Nothing* zurück und die Schleife kann verlassen werden.

```

Do
    ' Auswertung des Node-Objekts kann hier erfolgen
    Set ChildNode = ChildNode.Next
    If TypeName(ChildNode) = "Nothing" Then Exit Do
Loop

```


Das *Node*-Objekt hält aber noch weitere Eigenschaften zur Navigation durch die Hierarchie bereit. Im Folgenden sind alle Eigenschaften aufgeführt, die Verweise auf *Node*-Objekte gespeichert haben.

Child: Diese Eigenschaft enthält einen Verweis auf das erste untergeordnete *Node*-Objekt.

Parent: Diese Eigenschaft enthält einen Verweis auf das übergeordnete *Node*-Objekt.

Root: Diese Eigenschaft enthält einen Verweis auf das oberste *Node*-Objekt der Hierarchie, der dieses *Node*-Objekt angehört.

Die folgenden Eigenschaften beziehen sich immer auf dieselbe Hierarchiestufe, der auch das *Node*-Objekt angehört, mit dem Sie die Eigenschaft abrufen.

FirstSibling: In dieser Eigenschaft ist ein Verweis auf das erste *Node*-Objekt gespeichert.

Next: In dieser Eigenschaft ist ein Verweis auf das nächste *Node*-Objekt gespeichert.

Previous: In dieser Eigenschaft ist ein Verweis auf das vorhergehende *Node*-Objekt gespeichert.

LastSibling: In dieser Eigenschaft ist ein Verweis auf das letzte *Node*-Objekt gespeichert.

Eigenschaft	Beschreibung
<i>BackColor</i>	Legt die Hintergrundfarbe des angezeigten Texts fest.
<i>Bold</i>	Bestimmt, ob der Text fett (<i>TRUE</i>) oder normal (<i>FALSE</i>) dargestellt werden soll.
<i>Checked</i>	Diese Eigenschaft speichert, ob der Eintrag aktiviert ist (<i>TRUE</i>) oder nicht (<i>FALSE</i>). Eine Aktivierung wird durch ein Häkchen vor dem Eintrag angezeigt. Es muss also die Eigenschaft <i>Checkboxes</i> aktiviert werden.
<i>Children</i>	Gibt die Anzahl der untergeordneten <i>Node</i> -Objekte zurück.
<i>Expanded</i>	Zeigt an, ob ein Knoten geöffnet ist (<i>TRUE</i>) oder geschlossen (<i>FALSE</i>). Eine Öffnung eines Knotens kann nur erfolgen, wenn das <i>Node</i> -Objekt untergeordnete Elemente besitzt. Sie können diese Eigenschaft auch zum programmgesteuerten Öffnen und Schließen von Knoten verwenden.
<i>ExpandedImage</i>	In dieser Eigenschaft können Sie den Index des Symbols der gebundenen Abbildungsliste speichern, das angezeigt werden soll, wenn der Knoten geöffnet ist.
<i>ForeColor</i>	Legt die Farbe des angezeigten Texts fest.
<i>FullPath</i>	Diese Eigenschaft gibt den vollständigen Pfad des <i>Node</i> -Objekts zurück. Dieser Pfad wird aus der Verkettung der Werte der <i>Text</i> -Eigenschaft der jeweils übergeordneten Knoten bis hin zur Wurzel gebildet. Die einzelnen Texte werden durch das Zeichen, das Sie in der <i>PathSeparator</i> -Eigenschaft festgelegt haben, getrennt.
<i>Image</i>	Legt den Index des Symbols aus der gebundenen Abbildungsliste fest, das vor dem Text des <i>Node</i> -Objekts angezeigt werden soll.
<i>Selected</i>	Diese Eigenschaft bestimmt, ob das <i>Node</i> -Objekt ausgewählt ist.

Tabelle 3.9: Weitere Eigenschaften des *Node*-Objekts

Eigenschaft	Beschreibung
<i>SelectedImage</i>	In dieser Eigenschaft können Sie den Index des Symbols aus der gebundenen Abbildungsliste speichern, das angezeigt werden soll, wenn der Knoten selektiert wurde.
<i>Text</i>	Legt den Text fest, der in der Strukturansicht angezeigt wird.
<i>Visible</i>	Bestimmt, ob das <i>Node</i> -Objekt sichtbar oder ausgeblendet ist.

Tabelle 3.9: Weitere Eigenschaften des *Node*-Objekts (Forts.)

Methoden des *Node*-Objekts

Mit dem *Node*-Objekt können Sie insgesamt zwei Anweisungen ausführen. Diese kennen Sie bereits vom Listenansicht-Steuerelement. Mit der ersten Anweisung *CreateDragImage* können Sie ein Bild, das sich aus dem Text und dem Symbol des *Node*-Objekts zusammensetzt, erstellen lassen. Dieses Bild kann dann z.B. als Symbol bei Drag&Drop-Operationen verwendet werden.

```
DragPic = TreeView1.SelectedItem.CreateDragImage
```

Bei der zweiten Anweisung handelt es sich um *EnsureVisible*. Mit dieser Anweisung können Sie sicherstellen, dass das *Node*-Objekt im sichtbaren Bereich des Steuerelements erscheint. Sollte sich der Eintrag in einem nicht sichtbaren Bereich befinden, wird ein Bildlauf durchgeführt.

```
TreeView1.SelectedItem.EnsureVisible
```

Steuerelementereignisse

Dieses Steuerelement verfügt über einige Ereignisse, die für die Entwicklung von VBA-Projekten recht wichtig erscheinen und darum nicht unerwähnt bleiben sollen.

AfterLabelEdit: Das *AfterLabelEdit*-Ereignis wird immer dann ausgelöst, wenn der Benutzer den Texteintrag des ausgewählten *Node*-Objekts geändert hat. Sie können dann entsprechend reagieren und gegebenenfalls diese Änderung rückgängig machen, indem Sie den Parameter *Cancel* auf den Wert *TRUE* setzen. Welchen Text der Benutzer eingegeben hat, das entnehmen Sie dem Parameter *NewString*. Der ursprüngliche Wert der *Text*-Eigenschaft des *Node*-Objekts bleibt so lange erhalten, bis diese Prozedur beendet wird. Haben Sie den Parameter *Cancel* nicht verändert, wird die neue Zeichenfolge endgültig an die *Text*-Eigenschaft übertragen.

```
Private Sub TreeView1_AfterLabelEdit(Cancel As Integer, _
NewString As String)
If NewString = "0" Then
MsgBox "Änderung des Layers 0 nicht gestattet!", vbCritical
Cancel = True
End If
End Sub
```

BeforeLabelEdit: Dieses Ereignis tritt auf, wenn der Benutzer versucht, den Text eines *Node*-Objekts zu ändern, aber noch bevor das Eingabefeld eingeblendet wird. Sie können so beispielsweise verhindern, dass der Benutzer bestimmte Einträge überhaupt ändern kann. Setzen Sie hierfür den Parameter *Cancel* auf den Wert *TRUE* und das Eingabefeld wird gar nicht erst eingeblendet.

```
Private Sub TreeView1_BeforeLabelEdit(Cancel As Integer)
If Node.Tag = "Locked" Then
    MsgBox "Dieser Eintrag kann nicht geändert werden!", vbCritical
    Cancel = True
End If
End Sub
```

Collapse: Wird immer dann ausgelöst, wenn ein Knoten geschlossen wird, unabhängig davon, ob der Benutzer das Schließen vorgenommen hat oder Sie die Eigenschaft *Expanded* auf *FALSE* gesetzt haben. Der Parameter *Node* enthält das Knoten-Objekt, das geschlossen wurde.

Expand: Dieses Ereignis ist das Gegenstück zu *Collapse*. Es wird immer dann ausgelöst, wenn ein Knoten geöffnet wird. Auch hier ist der Auslöser dieses Vorgangs ohne Bedeutung. Der Parameter *Node* enthält das Knoten-Objekt, das geöffnet wurde.

NodeCheck: Dieses Ereignis tritt auf, wenn der Benutzer das Kontrollkästchen eines *Node*-Objekts aktiviert oder deaktiviert. Um welches *Node*-Objekt es sich handelt, können Sie dem Parameter *Node* entnehmen.

NodeClick: Im Gegensatz zum Standard-Click-Ereignis wird dieses nur ausgelöst, wenn der Benutzer auf den Text oder das Symbol eines *Node*-Objekts klickt. Im Parameter *Node* ist der Verweis auf das gewählte *Node*-Objekt gespeichert.

3.3.9 Windows-Standarddialoge

(Beispielprojekt: 03\Standarddialoge.dvb)

Neben den Windows-Standardsteuerelementen bietet Microsoft ein weiteres Steuerelement, in dem Standarddialoge enthalten sind, die Sie für die tägliche Arbeit immer wieder gebrauchen werden. Zu diesen Dialogen gehören Formulare zum Öffnen und Speichern oder zur Auswahl eines Windows-Schriftstils. Alle diese Dialogfelder sind in einem Steuerelement (*CommonDialog*) enthalten. Das Steuerelement selbst ist in der Windows-Bibliothek *CommDlg.dll* gespeichert. Um die Dialoge zu verwenden, müssen Sie das Steuerelement Ihrem Formular zur Verfügung stellen, indem Sie es auf dem Formular platzieren oder im Quellcode eine neue Instanz des Steuerelements erstellen.

```
Dim ComDialog As MSComDlg.CommonDialog
Set ComDialog = New MSComDlg.CommonDialog
```

Die Platzierung des Steuerelements ist dahingehend von Vorteil, als dass Sie zur Entwurfszeit bereits einige Einstellungen an den Standarddialogen vornehmen können, die Sie verwenden möchten. Im folgenden Abschnitt werde ich auf die Verwendung der einzelnen Dialogfelder genauer eingehen.

Dateiauswahldialog

(Beispielformular: 03\Standarddialoge.dvb!OpenSaveFrm)

Der Dialog zur Auswahl von Dateien unterteilt sich in zwei Kategorien. Die erste dient zur Auswahl von Dateien, die geöffnet werden sollen. Die zweite Kategorie bietet einen **SPEICHERN UNTER**-Dialog an. Auf den ersten Blick sind beide Dialoge kaum zu unterscheiden.

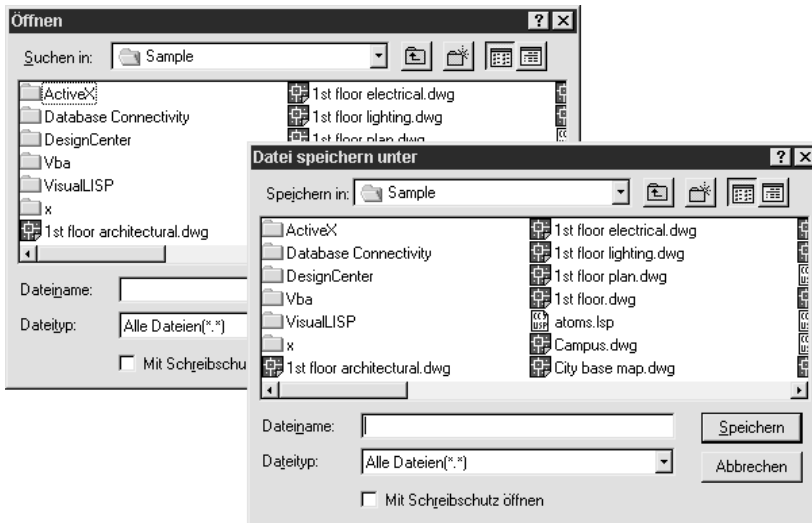


Abbildung 3.43: Standarddialoge zum Öffnen und Speichern von Dateien

Der Teufel steckt hier mal wieder im Detail. Erst durch Einstellung spezieller Eigenschaften erhalten beide Dialoge unterschiedliche Funktionalität. Das Ziel bleibt allerdings immer das selbe, die Auswahl eines Dateinamens. Dieser Dateiname wird in der Eigenschaft *Filename* gespeichert. Wenn Sie diese Eigenschaft bereits vor Aufruf eines Dialogs mit einem Dateinamen belegen, wechselt das Dialogfenster den Ordner (falls eine gültige Pfadangabe im Dateinamen enthalten ist) und zeigt den Namen der Datei im Textfeld „Dateiname“ an. Die zweite Eigenschaft, die bei beiden Dialogen dieselbe Wirkung erzielt, ist der Filter. Im Filter können Sie die Anzeige von Dateinamen nach bestimmten Dateitypen einschränken. Angenommen, Sie wollen nur Dateien des Typs *DWG* und *DXF* im Dialog anzeigen lassen. Dann definieren Sie die Filter-Eigenschaft wie folgt:

```
CommonDialog1.Filter = "Zeichnung (*.dwg)|*.dwg|DXF-Files|.dxf|"
```

Nach dem Start des Dialogs werden nur noch Dateien mit der Endung *DWG* und *DXF* im Auswahlfenster angezeigt. Die Definition ist dabei ganz einfach. Die Filtereigenschaft wird in eine Anzeigetext und den eigentlichen Filter unterteilt. Der jeweilige Filter wird dabei durch die Zeichen `||` eingegrenzt. Der Text, der übrig bleibt, wird in der Auswahlliste *Dateityp* angezeigt. Im vorliegenden Fall wären dies zwei Einträge: „Zeichnung (*.dwg)“ und „DXF-Files“. Sie können den Filter auch zusammenfassen, indem Sie nur einen Dateifilter angeben. Die einzelnen Dateitypen müssen dann durch ein Semikolon voneinander getrennt werden:

```
CommonDialog1.Filter = "Alle Zeichnungen|*.dwg;*.dxf|"
```

Jetzt werden alle Dateien im Fenster angezeigt, die den Dateitypen *DWG* und *DXF* entsprechen. Im Auswahlfeld „Dateityp“ ist aber nur noch der Eintrag „Alle Zeichnungen“ vorhanden. Mit der *FilterIndex*-Eigenschaft können Sie außerdem festlegen, welcher Auswahlfiler beim Start des Dialogs verwendet werden soll. Die Nummer, die Sie hier einstellen, entspricht der Position des Filters in der Liste „Dateityp“.

Eigenschaft	Beschreibung
<i>DialogTitle</i>	Mit dieser Eigenschaft können Sie die Überschrift des Dateiauswahldialogs festlegen. Standardmäßig wird der Titel ÖFFNEN bzw. SPEICHERN UNTER angezeigt, je nach Dialogtyp.
<i>DefaultExt</i>	In dieser Eigenschaft können Sie eine Dateierweiterung festlegen, die an den ausgewählten Dateinamen angehängt wird. Diese Eigenschaft ist besonders im Dateidialog SPEICHERN UNTER nützlich, wenn der Benutzer eine neue Datei erstellen möchte.
<i>InitDir</i>	Mit dieser Eigenschaft können Sie ein Verzeichnis festlegen, das eingestellt werden soll, bevor der Dialog gestartet wird. Dieses Verzeichnis gilt dann immer als Ausgangspunkt. Eventuelle Pfadangaben in der Eigenschaft <i>Filename</i> werden dann allerdings ignoriert.
<i>CancelError</i>	Setzen Sie diese Eigenschaft auf den Wert <i>TRUE</i> , dann wird der „Fehler“ <i>cdl-Cancel</i> ausgelöst, wenn der Benutzer im Auswahldialog die Schaltfläche ABBRECHEN betätigt. Das ist insofern nützlich, als dass Sie so erkennen können, dass der Benutzer den Vorgang abbrechen möchte.
<i>MaxFileSize</i>	Mit dieser Eigenschaft können Sie die maximale Länge der Zeichenfolge festlegen, die die <i>Filename</i> -Eigenschaft speichern kann. Wenn Sie eine Mehrfachauswahl festlegen, dann sollten Sie diesen Wert vergrößern, damit genügend Speicher für die Zeichenfolge reserviert wird.

Tabelle 3.10: Weitere Eigenschaften des Dateiauswahldialogs

Nun zu der wohl wichtigsten Eigenschaft des Steuerelements, der Eigenschaft *Flags*. Hierbei handelt es sich zwar lediglich um eine Eigenschaft, die einen Zahlenwert speichert, aber diese Zahl hat es in der Auswirkung auf den Auswahldialog in sich. Dass der Auswahldialog so viel Wert auf diese Eigenschaft legt, liegt daran, dass die Standarddialoge ursprünglich aus der Windows-API stammen und dort viel mit Konstanten definiert wird. Für die ÖFFNEN-/SCHLIESSEN-Dialoge existieren insgesamt 18 verschiedene Einstellungswerte für die *Flags*-Eigenschaft, die sich auch kombinieren lassen. Diese Werte und deren Bedeutung können Sie der folgenden Tabelle entnehmen.

Konstante	Wert	Beschreibung
<i>cdlOFNAllowMultiselect</i>	&H200	Durch Festlegung dieses Werts hat der Benutzer die Möglichkeit, mehrere Dateinamen gleichzeitig auszuwählen. Die Dateinamen werden alle in der Eigenschaft <i>Filename</i> gespeichert und durch ein Leerzeichen getrennt.
<i>cdlOFNCreatePrompt</i>	&H2000	Der Benutzer wird gefragt, ob er die Datei erstellen möchte, wenn die angegebene Datei nicht existiert.
<i>cdlOFNExplorer</i>	&H80000	Stellt den Dialog auf einen dem Explorer ähnlichen Stil um (unter Windows NT/95 ignoriert).
<i>cdlOFNExtensionDifferent</i>	&H400	Dieser Wert wird gesetzt, wenn sich die Erweiterung der Datei vom Wert der Eigenschaft <i>DefaultExt</i> unterscheidet.

Tabelle 3.11: Konstanten der *Flags*-Eigenschaft des Dateiauswahldialogs

Konstante	Wert	Beschreibung
<i>cdIOFNFileMustExist</i>	&H1000	Legt fest, dass der ausgewählte Dateiname existieren muss. Gibt der Benutzer im Textfeld des Dialogs einen Dateinamen an, der nicht existiert, wird eine Warnmeldung angezeigt.
<i>cdIOFNHelpButton</i>	&H10	Der Auswahldialog erhält eine HILFE-Schaltfläche.
<i>cdIOFNHideReadOnly</i>	&H4	Dieser Wert blendet das Kontrollkästchen SCHREIBGESCHÜTZT aus.
<i>cdIOFNLongNames</i>	&H200000	Lange Dateinamen werden unterstützt.
<i>cdIOFNNoChangeDir</i>	&H8	Legt fest, dass das Verzeichnis, das zuletzt gewählt wurde, im Auswahldialog eingestellt bleibt.
<i>cdIOFNNoDereferenceLinks</i>	&H100000	Shell-Verknüpfungen werden nicht aufgelöst. In der Voreinstellung wird eine Shell-Verknüpfung von der Shell automatisch aufgelöst, wenn sie ausgewählt wird.
<i>cdIOFNNoLongNames</i>	&H40000	Lange Dateinamen werden nicht unterstützt.
<i>cdIOFNNoReadOnlyReturn</i>	&H8000	Dieser Wert gibt an, dass die ausgewählte Datei weder schreibgeschützt ist noch sich in einem schreibgeschützten Verzeichnis befindet.
<i>cdIOFNNoValidate</i>	&H100	Legt fest, dass die Eingabe von ungültigen Zeichen zugelassen werden soll.
<i>cdIOFNOverwritePrompt</i>	&H2	Zeigt im SPEICHERN UNTER-Dialog eine Meldung an, wenn die angegebene Datei bereits existiert.
<i>cdIOFNPathMustExist</i>	&H800	Legt fest, dass der Benutzer nur gültige Pfade angeben darf. Wurde ein ungültiger Pfad festgelegt, wird eine Warnmeldung eingeblendet.
<i>cdIOFNReadOnly</i>	&H1	Das Kontrollkästchen SCHREIBGESCHÜTZT wird oder ist bereits aktiviert.
<i>cdIOFNShareAware</i>	&H4000	Legt fest, dass Fehler durch einen gemeinsamen Zugriff ignoriert werden.

Tabelle 3.11: Konstanten der Flags-Eigenschaft des Dateiauswahldialogs (Forts.)

Der Aufruf des jeweiligen Dialogs und die Auswertung des zurückgegebenen Dateinamens sind relativ einfach. Nachdem Sie alle Einstellungen vorgenommen haben, sei es zur Laufzeit oder in der Eigenschaftsseite des Steuerelements, können Sie den DATEI ÖFFNEN-Dialog mit der Anweisung `ShowOpen` einblenden. SPEICHERN UNTER wird mit der Anweisung `ShowSave` aktiviert.

Beispiel „Datei öffnen“:

```
On Error Resume Next
CommonDialog1.Flags = cdIOFNHelpButton + cdIOFNNoChangeDir
CommonDialog1.CancelError = True
CommonDialog1.FileName = TextBox1.Text
CommonDialog1.Filter = "Alle Dateien(*.*)|*.*|"
```

```

CommonDialog1.ShowOpen
If Err = cd1Cancel Then Exit Sub
TextBox1.Text = CommonDialog1.FileName

```

Beispiel „Datei speichern unter“:

```

On Error Resume Next
CommonDialog1.Flags = cd1OFNHelpButton + cd1OFNNoChangeDir + _
    cd1OFNOverwritePrompt
CommonDialog1.CancelError = True
CommonDialog1.FileName = TextBox1.Text
CommonDialog1.Filter = "Alle Dateien(*.*)|*.*|"
CommonDialog1.ShowSave
If Err = cd1Cancel Then Exit Sub
TextBox1.Text = CommonDialog1.FileName

```

Der Aufruf beider Anweisungen ist fast identisch. Zunächst werden die Werte für die *Flag*-Eigenschaft festgelegt, dann wird *CancelError* auf *TRUE* gesetzt, damit ein Hinweis darauf erfolgen kann, ob der Benutzer den Vorgang abgebrochen hat. Anschließend wird der Inhalt der TextBox übergeben, in der der Benutzer bereits einen Dateinamen festlegen kann. Bevor der jeweilige Dialog startet, wird noch der *Filter* festgelegt. Nachdem der Dialog beendet wurde, wird der Code mit der Zeile, die nach der *Show*-Anweisung folgt, fortgesetzt. Wurde der Vorgang durch den Benutzer abgebrochen, kann die Prozedur vorzeitig verlassen werden, andernfalls wird der gewählte Dateiname an die TextBox übergeben.

Hilfdateien starten

Dieser Teil des Standarddialog-Steuerelements dient dem Starten der WINHELP32.exe unter Angabe der Hilfdateri, die angezeigt werden soll. Wichtig hierbei ist die Angabe der Werte für die Eigenschaft *HelpFile*, die den Namen der Hilfdateri festlegt, und *HelpCommand* für die einige Konstanten existieren, die den Typ der Onlinehilfe festlegen. Erst wenn Sie diese beide Eigenschaften korrekt eingestellt haben, können Sie mit der Anweisung *ShowHelp* die Hilfdateri aufrufen.

Konstante	Wert	Beschreibung
<i>cdlHelpCommand</i>	&H102&	Ein Hilfemakro wird ausgeführt.
<i>cdlHelpContents</i>	&H3&	Zeigt das Thema mit dem Inhaltsverzeichnis der Hilfdateri an.
<i>cdlHelpContext</i>	&H1&	Zeigt die Hilfe für einen bestimmten Kontext an, den Sie in der Eigenschaft <i>HelpKontext</i> festgelegt haben.
<i>cdlHelpContextPopup</i>	&H8&	Zeigt ein Hilfethema in einem Popup-Fenster an, das durch die angegebene Kontextnummer identifiziert wird.
<i>cdlHelpForceFile</i>	&H9&	Mit diesem Wert können Sie sicherstellen, dass Winhelp die richtige Hilfdateri anzeigt.
<i>cdlHelpHelpOnHelp</i>	&H4&	Die Hilfe zur Verwendung der Hilfe wird angezeigt.

Tabelle 3.12: Konstanten der *HelpCommand*-Eigenschaft

Konstante	Wert	Beschreibung
<i>cdlHelpIndex</i>	&H3&	Der Index der angegebenen Hilfedatei wird angezeigt.
<i>cdlHelpKey</i>	&H101&	Wenn Sie diesen Wert verwenden, dann müssen Sie ein Stichwort in der Eigenschaft <i>HelpKey</i> festlegen.
<i>cdlHelpPartialKey</i>	&H105&	Dieser Wert legt fest, dass das Thema aus dem Stichwortverzeichnis angezeigt wird, das exakt mit dem angegebenen Stichwort übereinstimmt.
<i>cdlHelpQuit</i>	&H2&	Die <i>HelpCommand</i> -Eigenschaft enthält diesen Wert, wenn die angegebene Hilfedatei nicht mehr verfügbar ist.
<i>cdlHelpSetContents</i>	&H5&	Mit diesem Wert wird festgelegt, zu welchem Thema die Hilfe angezeigt wird, wenn der Benutzer die Taste F1 drückt.

Tabelle 3.12: Konstanten der *HelpCommand*-Eigenschaft (Forts.)

Zusätzlich zu den Eigenschaften *HelpFile* und *HelpCommand* existieren noch zwei weitere Eigenschaften, die in Zusammenhang mit der Anzeige von Hilfedateien verwendet werden. Mit *HelpContext* können Sie die Kennung eines Hilfethemas festlegen, das gestartet werden soll, wenn die Hilfedatei aufgerufen wird. Um diese Kennung zu verwenden, müssen Sie die Eigenschaft *HelpCommand* auf den Wert *cdlHelpContext* oder *cdlHelpContextPopup* einstellen. Mit der Eigenschaft *HelpKey* können Sie das Stichwort festlegen, nach dem in der Hilfedatei gesucht werden soll. Um diese Eigenschaft zu nutzen, müssen Sie für *HelpCommand* den Wert *cdlHelpKey* festlegen.

Wenn Sie alle entsprechenden Einstellungen und Vorgaben beachten, dann können Sie die Hilfe mit der Anweisung `ShowHelp` starten:

```
CommonDialog1.ShowHelp
```

Dialogfeld „Schriftart verwenden“

(Beispielformular: 03\Standarddialoge.dvb!SchriftartFrm)

Ein weiteres Dialogfenster, das das Steuerelement anbietet, ist der Dialog zur Auswahl der Schriftarten. Der Hauptverwendungszweck dieses Dialogs wird in AutoCAD sicherlich in der Festlegung von Schriftarten für Steuerelemente, z.B. Textfelder, liegen. In diesem Dialogfenster können Sie alle Eigenschaften eines *Font*-Objekts durch den Benutzer festlegen lassen. Diese Eigenschaften sind *FontBold*, *FontItalic*, *FontName*, *FontSize*, *FontStrikeThru* und *FontUnderLine*. Wenn Sie z.B. den Schriftstil eines Textfelds durch den Benutzer anpassen lassen möchten, dann müssen Sie die Werte der *Font*-Eigenschaft des Textfelds an das Steuerelement *CommonDialog* übergeben, den Auswahldialog für Schriftarten aufrufen und nach dem Ausblenden des Dialogs die Eigenschaften an das Textfeld zurückgeben.

Auch für die Verwendung dieses Dialogs können Sie wieder eine Reihe von Werten in der *Flags*-Eigenschaft setzen, die auf das Dialogfenster unterschiedliche Wirkungen haben.

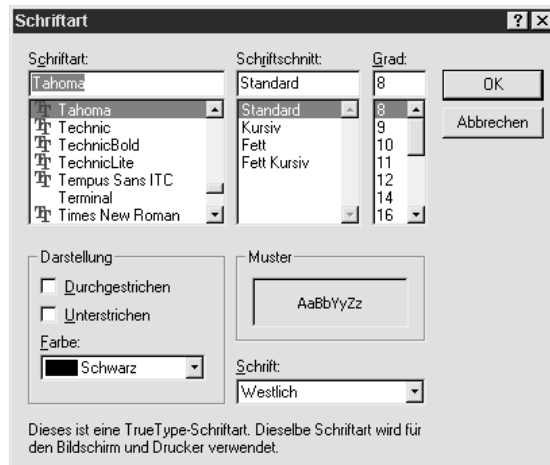


Abbildung 3.44: Auswahldialog für Schriftarten

Konstante	Wert	Beschreibung
<i>cdlCFANSIOnly</i>	&H400	Dieses Attribut unterdrückt die Auswahl von Schriftarten, die nur Symbole enthalten.
<i>cdlCFApply</i>	&H200	Aktiviert die Schaltfläche ÜBERNEHMEN im Dialog.
<i>cdlCFBoth</i>	&H3	Im Auswahldialog werden alle verfügbaren Bildschirm- und Druckerschriftarten angezeigt.
<i>cdlCFEffects</i>	&H100	Im Schriftdialog werden Effekte wie Durchstreichen, Unterstreichen und Farbeffekte aktiviert.
<i>cdlCFFixedPitchOnly</i>	&H4000	Im Dialogfeld können nur Schriftarten fester Breite ausgewählt werden.
<i>cdlCFForceFontExist</i>	&H10000	Aktiviert eine Fehlermeldung, wenn der Benutzer versucht, einen Schriftstil zu wählen, der nicht existiert.
<i>cdlCFHelpButton</i>	&H4	Aktiviert die Schaltfläche HILFE im Auswahldialog.
<i>cdlCFLimitSize</i>	&H2000	Legt fest, dass im Dialogfeld nur Schriftarten mit einem Schriftgrad ausgewählt werden können, der im durch die Werte der <i>Min</i> - und <i>Max</i> -Eigenschaft festgelegten Bereich liegt.
<i>cdlCFNoFaceSel</i>	&H80000	Legt fest, dass kein Schriftname nach dem Start des Dialogs ausgewählt wird.
<i>cdlCFNoSimulations</i>	&H1000	Legt fest, dass das Dialogfeld keine Simulation von Schriftarten über GDI zulässt.
<i>cdlCFNoSizeSel</i>	&H200000	Legt fest, dass keine Schrifthöhe nach dem Start des Dialogs ausgewählt wird.
<i>cdlCFNoStyleSel</i>	&H100000	Legt fest, dass kein Schriftstil nach dem Start des Dialogs ausgewählt wird.

Tabelle 3.13: Konstanten der Flags-Eigenschaft des Auswahldialogs für Schriftarten

Konstante	Wert	Beschreibung
<i>cdlCFNoVectorFonts</i>	&H800	Legt fest, dass keine Auswahl von Vektorschriftarten zugelassen werden soll.
<i>cdlCFPrinterFonts</i>	&H2	Legt fest, dass nur Schriftarten angezeigt werden, die durch die hDC-Eigenschaft festgelegten Drucker unterstützt werden.
<i>cdlCFScalableOnly</i>	&H20000	Legt fest, dass das Dialogfeld nur die Auswahl skalierbarer Schriftarten zulässt.
<i>cdlCFScreenFonts</i>	&H1	Legt fest, dass im Dialog nur die vom System unterstützten Bildschirmschriftarten angezeigt werden.
<i>cdlCFTTOnly</i>	&H40000	Legt fest, dass im Dialog nur die Auswahl von TrueType-Schriftarten zugelassen ist.
<i>cdlCFWYSIWYG</i>	&H8000	Gibt an, dass das Dialogfeld nur die Auswahl von Schriftarten zulässt, die sowohl auf dem Drucker als auch am Bildschirm verfügbar sind. Ist dieses Attribut gesetzt, dann sollten die Attribute <i>cdlCFBoth</i> und <i>cdlCFScalableOnly</i> ebenfalls gesetzt sein.

Tabelle 3.13: Konstanten der Flags-Eigenschaft des Auswahldialogs für Schriftarten (Forts.)

Das folgende Beispiel beschreibt die Verwendung des Schriftartendialogs. In diesem kann über den Schriftdialog einem Textfeld ein neuer Schriftstil zugewiesen werden.

```
On Error Resume Next
CommonDialog1.CancelError = True
CommonDialog1.Flags = cdlCFBoth + cdlCFEffects
CommonDialog1.Color = TextBox1.ForeColor
CommonDialog1.FONTBOLD = TextBox1.Font.Bold
CommonDialog1.FONTITALIC = TextBox1.Font.Italic
CommonDialog1.FONTNAME = TextBox1.Font.Name
CommonDialog1.FONTSIZE = TextBox1.Font.Size
CommonDialog1.FontStrikethru = TextBox1.Font.Strikethrough
CommonDialog1.FontUnderline = TextBox1.Font.Underline
CommonDialog1.ShowFont
If Err = cd1Cancel Then Exit Sub
TextBox1.ForeColor = CommonDialog1.Color
TextBox1.Font.Bold = CommonDialog1.FONTBOLD
TextBox1.Font.Italic = CommonDialog1.FONTITALIC
TextBox1.Font.Name = CommonDialog1.FONTNAME
TextBox1.Font.Size = CommonDialog1.FONTSIZE
TextBox1.Font.Strikethrough = CommonDialog1.FontStrikethru
TextBox1.Font.Underline = CommonDialog1.FontUnderline
```

Dialogfeld „Farbe verwenden“

(Beispielformular: 03\Standarddialoge.dvb!ColorFrm)

Dieser Farbauswahldialog ist nicht mit dem Farbdialog in AutoCAD vergleichbar. Im Gegensatz zu den AutoCAD-Farben sind die Farben in diesem Dialog nicht durch eine Nummer, sondern durch die Festlegung der Farbanteile oder der Helligkeit und Sättigung definiert. Somit kommt dieses Dialogfenster immer dann zur Anwendung, wenn Sie z.B. den Benutzer

auffordern, die Vorder- oder Hintergrundfarbe eines Steuerelements anzupassen. Die Verwendung des Farbdialogs ist ganz einfach. Sie müssen lediglich die Ausgangsfarbe der *Color*-Eigenschaft des Steuerelements zuweisen, dann den Dialog mit der Anweisung `ShowColor` starten und nach dessen Beendigung den neuen Farbwert, der wieder in der Eigenschaft *Color* gespeichert wird, dem Ausgangsobjekt übergeben.

Auch für dieses Dialogfenster kommen einige Einstellungen der *Flags*-Eigenschaft in Frage. Die Konstanten und deren Bedeutung können Sie der folgenden Tabelle entnehmen.

Konstante	Wert	Beschreibung
<i>cdlCCFullOpen</i>	&H2	Der gesamte Farbdialog inklusive des Abschnitts für benutzerdefinierte Farben wird angezeigt.
<i>cdlCCHelpButton</i>	&H8	Die Schaltfläche HILFE wird eingeblendet.
<i>cdlCCPreventFullOpen</i>	&H4	Der Abschnitt „Benutzerdefinierte Farben“ wird deaktiviert und der Benutzer kann auch nicht auf diesen Abschnitt umschalten.
<i>cdlCCRGBInit</i>	&H1	Legt fest, dass der Wert der <i>Color</i> -Eigenschaft als Anfangswert des Dialogs eingestellt wird.

Tabelle 3.14: Konstanten der *Flags*-Eigenschaft des Auswahldialogs für Farben

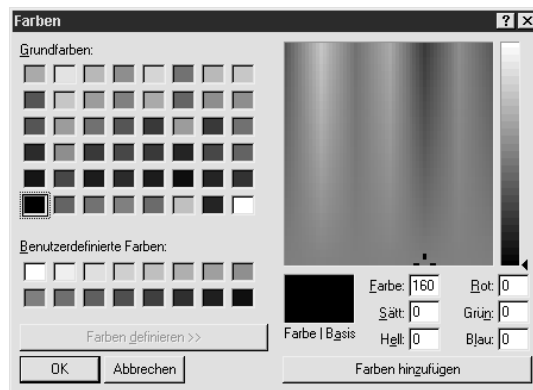


Abbildung 3.45: Standarddialog zur Farbauswahl

Das folgende Codebeispiel beschreibt die Initialisierung, den Aufruf und die Auswertung der Farbe mithilfe des Farbauswahldialogs.

```
On Error Resume Next
CommonDialog1.Color = TextBox1.ForeColor
CommonDialog1.CancelError = True
CommonDialog1.Flags = cdlCCFullOpen + cdlCCRGBInit
CommonDialog1.ShowColor
If Err = cdICancel Then Exit Sub
TextBox1.ForeColor = CommonDialog1.Color
```

Dialogfeld „Drucker verwenden“

Mit diesem Standarddialog bieten Sie dem Benutzer die Auswahl eines Druckers. Außerdem können Sie es dem Benutzer ermöglichen, gewisse Einstellungen am Drucker vorzunehmen. Sie können z.B. die Anzahl der Seiten einschränken, die gedruckt werden sollen, oder die Qualität des Drucks oder auch die Anzahl der zu druckenden Kopien festlegen.

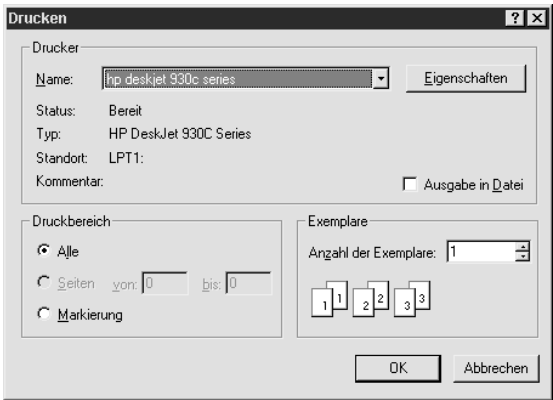


Abbildung 3.46: Standarddialog zur Druckerauswahl

Bevor Sie dieses Dialogfeld mit der Anweisung `ShowPrinter` aufrufen, können Sie noch einige Einstellungen über die Eigenschaft `Flags` vornehmen. Die Konstanten und deren Bedeutung sind in der Tabelle Tabelle 3.15 zusammengefasst. Daneben sind noch einige Eigenschaften des Steuerelements von Interesse. Mit der Eigenschaft `Copies` können Sie die Anzahl der zu druckenden Exemplare festlegen bzw. die Einstellung, die der Benutzer vorgenommen hat, abrufen. Mit den Eigenschaften `FromPage` und `ToPage` steuern Sie die Seiten, die gedruckt werden sollen. `FromPage` legt die erste, `ToPage` die letzte Seite fest. Die wichtigste Eigenschaft für dieses Dialogfenster ist `hDC`. In dieser Eigenschaft ist der Druckerkontext gespeichert, also die Zugriffsnummer zum verwendeten Drucker. Mit der Eigenschaft `PrinterDefault` können Sie außerdem festlegen, ob der Drucker, den der Benutzer im Dialog festgelegt hat, zum Standarddrucker werden soll.

Tipp

Da die VBA-Schnittstelle kein Printer-Objekt besitzt, bedarf es für die Implementierung dieser Schnittstelle einiger Klimmzüge. Mehr dazu erfahren Sie in Kapitel 9.3.

Konstante	Beschreibung	Wert
<code>cdlPDAllPages</code>	Dieser Wert legt den Zustand des Optionsfelds ALLES fest oder gibt dessen Zustand zurück.	&H0
<code>cdlPDCollate</code>	Dieser Wert legt den Zustand des Kontrollkästchens SORTIERT fest oder gibt dessen Zustand zurück.	&H10
<code>cdlPDDisablePrintToFile</code>	Deaktiviert das Kontrollkästchen AUSGABE IN DATEI.	&H80000
<code>cdlPDHelpButton</code>	Die Schaltfläche HILFE wird eingeblendet.	&H800

Tabelle 3.15: Konstanten der Flags-Eigenschaft des Druckerdialogs

Konstante	Beschreibung	Wert
<i>cdlPDHidePrintToFile</i>	Blendet das Kontrollkästchen AUSGABE IN DATEI aus.	&H100000
<i>cdlPDNoPageNums</i>	Deaktiviert das Optionsfeld SEITEN.	&H8
<i>cdlPDNoSelection</i>	Sperrt das Optionsfeld AUSWAHL.	&H4
<i>cdlPDNoWarning</i>	Deaktiviert die Warnmeldung für den Fall, dass kein Standarddrucker existiert.	&H80
<i>cdlPDPageNums</i>	Dieser Wert legt den Zustand des Optionsfelds SEITEN fest oder gibt dessen Zustand zurück.	&H2
<i>cdlPDPrintSetup</i>	Statt des Druckerdialogs wird der Dialog DRUCKER EINRICHTEN angezeigt.	&H40
<i>cdlPDPrintToFile</i>	Dieser Wert legt den Zustand des Kontrollkästchens AUSGABE IN DATEI fest oder gibt dessen Zustand zurück.	&H20
<i>cdlPDReturnDC</i>	Gibt den Gerätekontext in die Eigenschaft <i>hDC</i> zurück.	&H100
<i>cdlPDReturnDefault</i>	Gibt den Namen des Standarddruckers zurück.	&H400
<i>cdlPDReturnIC</i>	Gibt einen Informationskontext für die im Dialogfeld vorgenommene Druckerauswahl zurück.	&H200
<i>cdlPDSelection</i>	Dieser Wert legt den Zustand des Optionsfelds AUSWAHL fest oder gibt dessen Zustand zurück.	&H1
<i>cdlPDUseDevModeCopies</i>	Falls ein Drucker das Drucken mehrerer Exemplare nicht unterstützt, wird durch das Setzen dieses Attributs das entsprechende Bearbeitungs-Steuerelement deaktiviert.	&H40000

Tabelle 3.15: Konstanten der Flags-Eigenschaft des Druckerdialogs (Forts.)

Nun zum Aufruf des Druckerdialogs. Hier reicht eigentlich das Setzen der *Flags*-Eigenschaft und danach folgt nur noch der Aufruf des Dialogs.

```
CommonDialog1.Flags = cdlPDA11Pages
CommonDialog1.ShowPrinter
```

3.4 Das Benutzerformular

3.4.1 Benutzerformulare verwalten

Kommen wir zum Ende des Kapitels noch einmal auf das Benutzerformular und dessen Verwendung zurück. Das Benutzerformular dient als Container für die zuvor beschriebenen Steuerelemente und stellt somit die oberste Hierarchie dar. Die Grundlage jedes Formulars bildet die Klasse *UserForm*. Auch diese Klasse enthält eine Menge Eigenschaften, Methoden und Ereignisse, wobei Ihnen einige bereits von den Steuerelementen her bekannt sein dürften.

Formulare starten

Um Formulare aus dem Code heraus zu starten, müssen Sie lediglich die `Show`-Anweisung auf das entsprechende Formular anwenden. Ist das Formular noch nicht geladen, dann wird dies automatisch getan.

```
UserForm1.Show
```

Das so gestartete Formular wird dann gebunden (modal) angezeigt, was bedeutet, dass Sie das Formular erst schließen müssen, bevor der Quellcode fortgesetzt wird, der das Formular gestartet hat.

```
Sub ShowFormular  
UserForm1.Show
```

```
UserForm2.Show 'Dieses Formular wird erst angezeigt, nachdem das  
                'vorherige geschlossen wurde  
End Sub
```

Außerdem können Sie nicht in der Zeichnung arbeiten, solange das Formular angezeigt wird. Ab der AutoCAD-Version 2000i besteht auch die Möglichkeit, Formulare ungebunden (modeless) anzuzeigen. Dazu müssen Sie der `Show`-Anweisung nur den Parameter `vbModeless` anhängen.

```
UserForm1.Show vbModeless
```

Die Syntax des Aufrufs von gebundenen Formularen bleibt bestehen, da der Parameter `vbModal` die Standardeinstellung darstellt. Bei der Verwendung von ungebundenen Formularen müssen Sie allerdings einige Dinge beachten. Sie können ungebundene Formulare nicht starten, wenn noch ein gebundenes angezeigt wird. Der umgekehrte Weg funktioniert allerdings ohne Probleme. Außerdem sollten Sie immer ein ungebundenes Formular mit einer Prozedur ausführen, die einen Befehl in der AutoCAD-Befehlszeile ausführt, z.B. die Auswahl eines Punkts mit `GetPoint`. Das hat die Bewandnis, dass Sie z.B. Textfelder nicht bearbeiten können, wenn Sie das ungebundene Formular einfach nur anzeigen lassen.

Tipp

Sollten Sie auch Anwendungen für AutoCAD2000 entwickeln wollen, in denen ja die Form nicht ungebunden angezeigt werden kann, dann müssen Sie unbedingt den Parameter nach dem Formularaufruf weglassen. Selbst die Angabe von `vbModal` löst hier einen Fehler aus, da in AutoCAD2000 keine Parameter für den Formularaufruf definiert sind.

Wenn das Formular geladen wird, dann löst dies das Ereignis `Initialize` aus. Sie können diese Ereignisprozedur dazu verwenden, bestimmte Grundeinstellungen, wie z.B. die Wiederherstellung der letzten Formularposition, vorzunehmen.

Formulare schließen

Um ein Formular aus dem Code heraus zu schließen, benutzen Sie die `Hide`-Anweisung. Das Formular wird daraufhin ausgeblendet, aber nicht entladen. Das heißt, dass die aktuellen Einstellungen, wie Position oder der Inhalt von Listefeldern, erhalten bleiben.

```
UserForm1.Hide
```

Wenn Sie das Formular entladen möchten, so dass dieses einschließlich der dazugehörigen Steuerelemente und Prozeduren aus dem Speicher verschwindet, dann müssen Sie die `Unload`-

Anweisung verwenden. Bei dieser Anweisung handelt es sich um eine globale VBA-Prozedur, die auch auf andere Objekte, sprich Steuerelemente, angewendet werden kann. Als Parameter müssen Sie lediglich das Objekt angeben, das entladen werden soll.

```
Unload UserForm1
```

Beide Methoden erzielen rein optisch dieselbe Wirkung, nur dass eben das Formular mit *Hide* nur ausgeblendet wird, Sie aber dennoch auf die Daten des Formulars zugreifen können.

Tipp Ob ein Formular geladen ist oder nicht, können Sie mit der *UserForms*-Auflistung überprüfen. Wird das Formular in dieser Auflistung geführt, dann wurde es entweder mit der *Load*-Anweisung geladen oder bereits einmal gestartet und nicht wieder entladen.

Auch der Benutzer verfügt über die Möglichkeit, ein Formular zu schließen. Dazu muss er nur die SCHLIESSEN-Schaltfläche in der rechten oberen Ecke des Formulars betätigen. Da in VBA nicht die Möglichkeit besteht, dieses SCHLIESSEN-Feld auszublenden, müssen Sie gegebenenfalls auf die Ereignisprozedur *QueryClose* reagieren, wenn Sie z.B. verhindern möchten, dass das Formular geschlossen wird. Diese Ereignisprozedur enthält zwei Parameter, wobei der Parameter *CloseMode* die Methode liefert, die das Schließen des Formulars auslöst.

Konstante	Beschreibung	Wert
<i>vbFormControlMenu</i>	Der Benutzer hat das SCHLIESSEN-Feld des Formulars betätigt.	0
<i>vbFormCode</i>	Das Formular wird über die Unload-Anweisung geschlossen.	1
<i>vbAppWindows</i>	Windows wird beendet.	2
<i>vbAppTaskManager</i>	Die Anforderung zum Schließen des Dialogs wurde durch den Task-Manager gestellt.	3

Tabelle 3.16: Konstanten des Parameters *CloseMode*

Mit dem Parameter *Cancel* können Sie nun verhindern, dass das Formular tatsächlich geschlossen wird. Sie müssen hierfür nur den Parameter auf den Wert *TRUE* setzen und die Aktion wird abgebrochen. Wenn Sie z.B. verhindern wollen, dass der Benutzer das Formular über das SCHLIESSEN-Feld beendet, sieht das folgendermaßen aus:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As _
Integer)
If CloseMode = vbFormControlMenu Then Cancel = True
End Sub
```

Ein zweites Ereignis, das beim Entladen des Formulars ausgelöst wird, ist *Terminate*. Dieses Ereignis wird nach *QueryClose* aufgerufen, sofern *QueryClose* nicht mit *Cancel* abgebrochen wurde, aber noch bevor das Formular entladen ist. Sie könnten *Terminate* dazu benutzen, bestimmte Einstellungen zu speichern, bevor das Formular aus dem Speicher verschwindet.

Formularposition festlegen

Wie jedes andere sichtbare Element, besitzt auch ein Formular die Eigenschaften *Left*, *Top*, *Width* und *Height*. *Left* und *Top* legen die Position des Formulars vom linken und oberen Bildschirmrand fest. *Width* und *Height* bestimmen die Breite und Höhe des Formulars. Beach-

ten Sie, dass diese Eigenschaften sowohl die Höhe der Titelleiste als auch die Randbereiche des Formulars beinhalten. Wenn Sie also nur die Ausmaße des Client-Bereichs ermitteln wollen, dann müssen Sie die Eigenschaften *InsideHeight* und *InsideWidth* abfragen.

Nun zur Positionierung der Formulare. Normalerweise wird das Formular zunächst in der Mitte des AutoCAD-Fensters eingeblendet. Anschließend können Sie erst das Formular mit den Eigenschaften *Left* und *Top* neu positionieren. Das liegt an der Einstellung der Formular-Eigenschaft *StartupPosition*. Wenn Sie ein neues Formular anlegen, dann wird automatisch der Wert 1 (Fenstermitte) eingestellt, und solange Sie diesen Wert lassen, können Sie so viele Positionen festlegen, wie Sie wollen – das Formular wird immer erst in der Mitte des AutoCAD-Fensters angezeigt. Abhilfe schafft da nur die Änderung des Eigenschaftswerts und da haben Sie die Auswahl zwischen vier verschiedenen Positionierungsmöglichkeiten.

- 0 - Manuell: Das Formular soll manuell positioniert werden. Das bedeutet, dass das Formular keine konkrete Anfangsposition erhält, sondern irgendwo positioniert wird, wenn Sie nicht eingreifen. Diese Einstellung ist also dazu gedacht, die Eigenschaften *Left* und *Top* durch den Benutzer, also durch Sie, zu vergeben. Sie könnten beispielsweise gespeicherte Positionswerte abrufen und an die Eigenschaften vergeben.
- 1 - Fenstermitte (Standard): Das Formular wird in der Mitte des Formulars angezeigt, von dem aus es gestartet wird. Da das in AutoCAD immer das Sitzungsfenster ist, wird das Formular immer in dessen Mitte angezeigt, was bei Mehrplatzbildschirmen recht störend werden kann.
- 2 - Bildschirmmitte: Das Formular wird immer in der Mitte des Bildschirms positioniert, egal wo sich das AutoCAD-Fenster befindet.
- 3 - Windows-Standard: Das Formular wird immer in der linken oberen Ecke des Bildschirms positioniert.

Weitere Formular-Eigenschaften

Auch für das Benutzerformular können Sie, wie bei fast allen Steuerelementen, die Vorder- und Hintergrundfarbe mit den Eigenschaften *ForeColor* und *BackColor* einstellen. Die Farben, die Sie hier wählen, werden als Vorgabewerte für Steuerelemente verwendet, die Sie in dem Formular neu positionieren. Legen Sie z.B. für ein Formular die Vordergrundfarbe Rot fest, dann erhält jedes Steuerelement, das diese Eigenschaft unterstützt, ebenfalls die Vordergrundfarbe Rot, wenn es in das Formular eingefügt wird. Ebenfalls Auswirkung auf einzufügende Steuerelemente hat die *Font*-Eigenschaft. Hier legen Sie die Standardschriftart des Formulars fest.

Mit der Eigenschaft *Caption* können Sie den Titel, also den Text, der in der Titelleiste des Formulars erscheint, bestimmen.

Die *Picture*-Eigenschaft legt ein Bild fest, das an der Position angezeigt wird, die Sie in der *PictureAlignment*-Eigenschaft eingestellt haben. Mit der *PictureSizeMode*-Eigenschaft können Sie außerdem die Größe des Bilds bestimmen. Das Bild wird entweder in Originalgröße (*fmPictureSizeModeClip*) angezeigt, auf Formulargröße skaliert (*fmPictureSizeModeStretch*) oder so vergrößert (*fmPictureSizeModeZoom*), dass es in die minimale Ausdehnung des Formulars passt (das Originalseitenverhältnis bleibt erhalten). Eine weitere Möglichkeit wäre die Kachelung des Formularhintergrunds mit dem in der *Picture*-Eigenschaft angegebenen Bild.

Eine Besonderheit von VBA-Formularen ist die Möglichkeit, es mit verschiedenen Rahmen-effekten auszustatten. Dafür müssen Sie nur die Eigenschaft *SpecialEffect* mit einer der durch die Eigenschaft angegebenen Konstanten versehen.

Wie bei Rahmenfeldern besteht auch bei Formularen die Möglichkeit, den Inhalt zu scrollen. Dazu müssen Sie zunächst mit der Eigenschaft *ScrollBars* die Bildlaufleisten aktivieren, die Sie verwenden wollen. Mit den Eigenschaften *ScrollHeight* und *ScrollWidth* können Sie wieder die Ausdehnung des Scroll-Bereichs festlegen. *ScrollTop* und *ScrollLeft* bestimmen die Position des Client-Bereichs. Sie können auch die Bildlaufleisten mit der Eigenschaft *KeepScrollBarsVisible* ausblenden, solange sie nicht benötigt werden. Die Bildlaufleisten werden dann erst angezeigt, wenn die Werte *ScrollHeight* und *ScrollWidth* die Werte des Client-Bereichs (*InsideHeight*, *InsideWidth*) überschreiten.

Methoden

Das Benutzerformular kennt neben *Show* und *Hide* noch weitere Methoden, die Ihnen an dieser Stelle nicht vorenthalten werden sollen. Mit *SetDefaultTabOrder* können Sie eine Standardsortierreihenfolge (*TabIndex*) für die Steuerelemente des Formulars festlegen. Die Steuerelemente werden danach von oben nach unten und von links nach rechts durchnummeriert. Wenn Sie in dem Formular Bildlaufleisten verwenden, dann können Sie mit der Scroll-Anweisung einen Bildlauf per Code auslösen. Der Vorteil dabei ist, dass Sie sowohl die vertikale als auch die horizontale Bildlaufleiste in einem Abwasch ansprechen können. Die Parameter, die Sie angeben können, entsprechen aber keiner direkten Positionsangabe wie *ScrollLeft* oder *ScrollTop*, sondern vielmehr der Simulation von Tasten, die auch der Benutzer zum Bildlauf betätigen würde. Die Konstanten, die Sie verwenden können, gelten für beide Parameter. Folgende Zeile würde z.B. einen Bildlauf in horizontaler Richtung ignorieren und in vertikaler Richtung auf die oberste Position verschieben.

```
UserForm1.Scroll fmScrollActionNoChange, fmScrollActionBegin
```

Ereignisse

Neben den bereits beschriebenen Ereignissen kann ein Benutzerformular auch noch auf weitere Ereignisse reagieren. So treten die Ereignisse *Click*, *DbtClick*, *MouseDown*, *MouseUp* und *MouseMove* auf, wenn der Benutzer bestimmte Aktionen mit der Maus auf dem Formular durchführt. *KeyDown*, *KeyUp* und *KeyPress* reagieren auf Tastaturanschläge.

Die Ereignisse *Activate* und *Deactivate* sind für die Arbeit mit Formularen, die ungebunden angezeigt werden, interessant. *Activate* wird immer dann ausgelöst, wenn das Formular aktiviert wird. Das kann z.B. direkt durch das Starten des Formulars mit der *Show*-Anweisung geschehen oder wenn der Benutzer von einem ungebundenen Formular zu einem anderen wechselt, während beide sichtbar sind. *Deactivate* tritt immer dann auf, wenn das aktive Formular durch den Wechsel auf ein anderes Formular deaktiviert wird. Dies kann durch einfachen Wechsel zwischen zwei ungebundenen Formularen passieren oder durch den Aufruf eines weiteren Formulars mit der *Show*-Anweisung aus einem bereits aktiven Formular heraus.

3.4.2 Steuerelemente zur Laufzeit hinzufügen/entfernen

Angenommen, Sie wollen innerhalb eines Formulars die vorhandenen Attribute eines Blocks zur Eingabe anbieten. Da die Anzahl der Attribute im voraus nicht bekannt ist, müssen Sie die Anzahl der Textfelder ebenfalls flexibel halten. Deshalb ist es erforderlich, diese Textfelder erst zur Laufzeit dem Formular hinzuzufügen, wenn diese benötigt werden. Container, wie Formulare oder Rahmenfelder, besitzen hierfür die *Controls*-Auflistung. Über diese Auflistung kön-

nen Steuerelemente zur Laufzeit verwaltet und demnach auch hinzugefügt bzw. entfernt werden. Diese Auflistung funktioniert eigentlich wie jede andere auch. Mit der `Add`-Anweisung werden Elemente hinzugefügt, mit `Remove` entfernt und mit `Item` abgerufen.

Steuerelemente hinzufügen

Um ein Steuerelement hinzuzufügen, müssen Sie die ProgID des jeweiligen Elements kennen. Diese ProgID finden Sie beispielsweise in der Registrierung im Abschnitt `HKEY_CLASSES_ROOT\CLSID`. Jedes Steuerelement, das in Windows registriert wird, wird in diesem Abschnitt aufgeführt. Dort ist neben dem Klassennamen auch die ProgID des Steuerelements gespeichert. Den dort gespeicherten Eintrag können Sie mit der `Add`-Anweisung der *Controls*-Auflistung verwenden.

Ein konkretes Beispiel einer `TextBox`: Der Eintrag unter der ProgID lautet „Forms.TextBox.1“. Diesen Eintrag verwenden wir nun im ersten Parameter. Im zweiten Parameter können Sie den Namen des Steuerelements festlegen. Dieser Name sollte wieder eindeutig sein, sich also von anderen Steuerelementen unterscheiden. Wenn Sie diesen Parameter nicht vergeben, wird der Name genauso festgelegt, wie wenn Sie das Steuerelement zur Entwurfszeit hinzufügen (`TextBox1`, `TextBox2`, `TextBox3` usw.). Im dritten Parameter können Sie die Sichtbarkeit des Steuerelements bestimmen (*Visible*-Eigenschaft). Geben Sie den Wert `TRUE` an, dann wird es angezeigt, mit `FALSE` erfolgt keine Anzeige. Wenn Sie diesen Parameter nicht vergeben, wird der Wert `TRUE` verwendet. Also reicht im Prinzip folgender Aufruf vollkommen aus:

```
UserForm1.Controls.Add "Forms.TextBox.1"
```

Das so erstellte Steuerelement ist nun in der *Controls*-Auflistung verfügbar und kann darüber auch wieder abgerufen werden. Entweder Sie verwenden hierfür den Namen des Steuerelements oder die Position innerhalb der Auflistung.

```
UserForm1.Controls("TextBox1").Text = ""
```

Sie können auch direkt über den Container (Formular) auf das neue Steuerelement zugreifen, wobei es im Aufruf einen Unterschied zu den Steuerelementen gibt, die Sie zur Entwurfszeit erstellt haben. Hier müssen Sie ein Ausrufezeichen statt eines Punkts zwischen Formular und Steuerelement setzen.

```
UserForm1!TextBox1.Text = ""
```

Ansonsten ist die Syntax dieselbe wie bei Steuerelementen, die zur Entwurfszeit entstanden sind. Generell empfiehlt sich allerdings der Zugriff über die *Controls*-Auflistung, da diese Methode besser in Code umzusetzen ist.

Nachdem das Formular nun ein neues Steuerelement besitzt, müssen Sie noch verschiedene Einstellungen, z.B. Position, Breite und Textinhalt, vornehmen. Der folgende Beispielcode demonstriert das Hinzufügen eines Labels und einer `TextBox` zur Laufzeit und deren anschließende automatische Positionierung.

```
Dim NewLabel As MSForms.Label
Dim NewTBox As MSForms.TextBox
On Error Resume Next
Set NewLabel = Me.Controls.Add("Forms.Label.1")
Set NewTBox = Me.Controls.Add("Forms.TextBox.1")
NewLabel.top = LastY + 2
```

```
NewTBox.top = LastY
LastY = LastY + NewTBox.Height
NewTBox.Left = 50
NewTBox.Width = 150
NewLabel.Caption = NewTBox.Name
If NewTBox.top + NewTBox.Height > Me.InsideHeight Then
    Me.Height = NewTBox.top + NewTBox.Height + (Me.Height - _
        Me.InsideHeight)
End If
```

Tipp

Wenn Sie Steuerelemente Containern (z. B. Rahmen oder Multiseiten) hinzufügen wollen, die Formularen untergeordnet sind, dann können Sie auf dieselbe Weise vorgehen wie bei Formularen. Sie müssen dann lediglich die *Controls*-Auflistung des entsprechenden Steuerelements verwenden.

4

Vom Problem zur Lösung

In den letzten Kapiteln haben Sie die Grundzüge der Programmierung mit VBA kennen gelernt. Der folgende Teil des Buches enthält die AutoCAD-Objekte. Diese werden im einzelnen beschrieben und mit Beispielen erläutert. Dieses Kapitel vermittelt einen Eindruck, was mit AutoCAD-VBA alles möglich sein kann und wie Sie Mithilfe dieses Buchs von der Problemstellung zu Ihrer Lösung gelangen können.

Bei den einzelnen Schritten greife ich natürlich oft auf Wissen zurück, das ich im Buch bereits vermittelt habe – entweder in den ersten drei Kapiteln über VBA oder in den nachfolgenden Kapiteln über die Objekte. Damit Sie nachschlagen können, wenn Ihnen etwas nicht geläufig ist, habe ich auf die entsprechenden Kapitel verwiesen. Sie werden wahrscheinlich auch bei eigenen praktischen Problemen, für die Sie Programme schreiben möchten, so vorgehen, dass Sie sich dann genauer mit den Objekten beschäftigen, wenn Sie diese benötigen. Dieses Beispiel, das hier gelöst wird, diese können Sie wie ein Modell betrachten, wie Sie eigene Probleme lösen können. Ich verwende ein Beispiel, das für jeden AutoCAD-Anwender nachvollziehbar ist. Die Layer-Verwaltung wird von jedem AutoCAD-Anwender benutzt, egal welche Zusatzapplikation er sein eigen nennt.

Sie werden sich sicher auch schon gefragt haben, warum sich bei AutoCAD die Layer nicht in Gruppen unterteilen lassen, so dass eine vernünftige Verwaltung möglich wird. Da dies auch unter der Version AutoCAD 2004 noch nicht möglich ist, will ich Ihnen anhand des Projekts einmal zeigen, wie sich so etwas von der Problemstellung bis hin zum fertigen Projekt bewerkstelligen ließe.

Das komplette Beispielprojekt finden Sie auch auf der Buch-CD („04/LayerManager.dvb“). Sie können aber auch die einzelnen Schritte anhand der Teilprojekte nachvollziehen, die ich für die Abschnitte entworfen habe.

4.1 Konzeption des Managers

Da es sich um ein etwas umfangreicheres Projekt handelt, muss von vornherein klar sein, wie dieses Projekt in Angriff genommen werden soll. Es bedarf also einer gewissen Grundstruktur, an der wir uns orientieren können.

Dazu gehört beispielsweise, wie die Gruppen, in denen die Layer zusammengefasst werden sollen, organisiert werden. Dürfen die Layer nur einmal in der Gruppenstruktur auftauchen, also nur von Gruppe zu Gruppe verschoben werden (dies würde eine wesentlich einfachere Gruppenverwaltung ermöglichen, aber auch Einbußen an Flexibilität bedeuten) oder kann jeder Layer jeder Gruppe zugeordnet werden? Auf diese Weise wären mehrere Layer-Konfigurationen möglich.

Zunächst einmal benötigen wir natürlich ein Formular, in dem die gesamte Verwaltung der Layer und Layer-Gruppen dargestellt und dem Benutzer deren Bearbeitung ermöglicht wird. Hinzu kommt die Startprozedur, ein Makro, das der Benutzer in AutoCAD ausführen kann und das eigentlich nur dem Starten des Formulars dient.

Fügen Sie zu diesem Zweck ein neues Formular dem Projekt hinzu (*Kapitel 1.2*) und geben Sie ihm die Bezeichnung *LayerFrm*.

Die Startprozedur, die Sie dem Klassenmodul *ThisDrawing* hinzufügen, enthält zunächst nur den Aufruf des Formulars mittels der *Show*-Anweisung.

```
Public Sub showLayerManager()  
On Error Resume Next  
With LayerFrm  
    .Show  
End With  
Unload LayerFrm  
End Sub
```

4.2 Verwaltung der Gruppen

Problemstellung

Beschäftigen wir uns zunächst mit dem Aufbau der Layer-Gruppen. Diese müssten zum einen temporär verfügbar sein. Das heißt, dass die Daten zur Laufzeit aus der Zeichnung ausgelesen werden und durch den Benutzer verarbeitet werden können und bei Beendigung des Programms wieder aus dem Speicher entfernt werden. Neben der temporären Verfügbarkeit müssten die Gruppen auch so aufgebaut sein, dass sich bei der Entwicklung eine Gruppenstruktur ergibt (jede Gruppe sollte ein eigenes Objekt bilden) und die Anzahl der zu bildenden Gruppen nicht eingeschränkt wird.

Lösungsansatz

Als temporäre „Datenträger“ der Gruppeninformationen kommen zwei Wege in Frage. Zum einen könnte jede Gruppe in einem Feld eines Arrays (*Kapitel 2.1.5*) abgelegt werden. Als zweiter Ansatz bieten sich nach *Kapitel 2.5* aber auch Klassenmodule an, wobei ein Klassenmodul als Gruppenauflistung und ein zweites als Vorlage für die Instanzen der einzelnen Gruppen dienen würde.

Beide Ansätze führen zum Ziel, wobei sich aber die Klassenmodule als der programmtechnisch bessere Weg herausstellen würden. Dementsprechend wählen wir hier diesen Ansatz.

Problemlösung

Fügen Sie dem Projekt zwei Klassenmodule hinzu (EINFÜGEN→KLASSENMODUL). Das erste wird als Auflistungsobjekt dienen und die einzelnen Gruppen verwalten. Das zweite Klassenmodul wird dann als Vorlage für die einzelnen Instanzen der Gruppen verwendet. Entsprechend der Namenskonvention für solche Klassen (*Kapitel 2.4.4*) kommen für die Klassen beispielsweise die Bezeichnungen *LayerGroup* (Vorlage für einzelne Gruppen) und *LayerGroups* (Auflistung) in Frage. Benennen Sie dementsprechend die beiden Klassenmodule in *LayerGroup* und *LayerGroups* um.

Klassenmodul *LayerGroup*

Jetzt müssen wir uns entscheiden, welche Gruppeneigenschaften wir generell verwalten wollen. Dazu benutzen wir die *Property*-Prozeduren, wie bereits in *Kapitel 2.5.2* beschrieben wurde. Die erste Eigenschaft wird den jeweiligen Namen und die zweite die Position der Gruppen innerhalb der Gruppenhierarchie aufnehmen.

Zunächst erfolgt die Deklaration der lokalen Kopien der beiden Eigenschaften:

```
Private mvarGrpName As String
Private mvarGrpUrl As String
```

Anschließend folgt die *Property Let*-Prozedur für das Schreiben der Eigenschaft *GruppenName*:

```
Public Property Let GruppenName(GrpName As String)
mvarGrpName = GrpName
End Property
```

und die *Property Get*-Prozedur für das Lesen der Eigenschaft *GruppenName*:

```
Public Property Get GruppenName() As String
GruppenName = mvarGrpName
End Property
```

Danach dasselbe noch einmal für das Schreiben der Eigenschaft *GruppenUrl*:

```
Public Property Let GruppenUrl(GrpUrl As String)
mvarGrpUrl = GrpUrl
End Property
```

und das Lesen der Eigenschaft *GruppenUrl*:

```
Public Property Get GruppenUrl() As String
GruppenUrl = mvarGrpUrl
End Property
```

Normalerweise würde es natürlich auch ausreichen, statt der *Property*-Prozeduren einfach zwei Variablen (*GruppenName* und *GruppenUrl*) öffentlich zu deklarieren.

```
Public GruppenName As String
Public GruppenUrl As String
```

Egal, in welcher Form Sie die beiden Eigenschaften verwalten wollen, mit diesen beiden Eigenschaften ist das erste Klassenmodul bereits fertig gestellt.

Klassenmodul *LayerGroups*

Das zweite Klassenmodul verwendet dieses Klassenmodul als Grundlage für die einzelnen Instanzen der Gruppen. Da *LayerGroups* als Auflistung geplant ist, benötigen wir hier zunächst eine lokale Variable, die die Verweise auf die einzelnen Gruppen speichert. Da es sich hierbei um eine unbekannte Anzahl von Gruppen handeln kann, also 1 bis n Gruppen, eignet sich hierfür am besten das *Collection*-Objekt. Informationen zu diesem Objekt können Sie dem *Kapitel 2.5.1* entnehmen.

Die Deklaration der Variablen *mCol* in der ersten Zeile des Klassenmoduls lautet demnach:

```
Private mCol As Collection
```

Jetzt könnte man natürlich die Variable *mCol* auch global deklarieren, womit man einen direkten Zugriff auf die Auflistung erhält. Der folgende Weg ist allerdings eleganter und dazu auch noch flexibler. Um nun diese Auflistung nutzen zu können, muss eine neue Instanz der Auflistung erzeugt werden. Entsprechend den Erläuterungen in *Kapitel 2.5.1* kann dies im gleichen Zuge erfolgen, in dem auch eine neue Instanz der Klasse *LayerGroups* im Programm erzeugt wird.

```
Private Sub Class_Initialize()  
Set mCol = New Collection  
End Sub
```

Genauso müssen wir auch darauf reagieren, wenn der Verweis auf die Instanz der Klasse *LayerGroups* wieder aus dem Speicher entfernt wird.

```
Private Sub Class_Terminate()  
Set mCol = Nothing  
End Sub
```

Da das *Collection*-Objekt (*mCol*) bereits einige Funktionen zum Hinzufügen, Abrufen und Entfernen von Auflistungsobjekten enthält, müssen wir zunächst diese Funktionen auf die Klasse übertragen, damit wir die gleiche Funktionalität auch in unserem Klassenmodul nutzen können.

Beginnen wir mit der *Add*-Anweisung. Diese wird eine neue Instanz der Klasse *LayerGroup* erzeugen und den Verweis auf dieses Objekt in der Auflistung *mCol* speichern. Als Argumente der Anweisung werden die Variablen *GruppenName* und *url* festgelegt. Diese werden dann bereits beim Aufruf der *Add*-Anweisung übergeben und sofort mit der neuen Instanz der Klasse *LayerGroup* verarbeitet.

```
Public Function Add(GruppenName, url) As LayerGroup
```

Als Erstes wird in dieser Prozedur eine Variable vom Klassentyp *LayerGroup* deklariert und in dieser Variable der Verweis auf die neue Instanz eben dieser Klasse gespeichert.

```
Dim NewGroup As LayerGroup  
Set NewGroup = New LayerGroup
```

Anschließend werden die beim Aufruf der *Add*-Anweisung übergebenen Variablen den Eigenschaften des Objekts vom Typ *LayerGroup* übergeben. Damit sind die Daten der Gruppe festgelegt.

```
NewGroup.GruppenName = GruppenName  
NewGroup.GruppenUrl = url
```

Als Letztes muss die neue Instanz nur noch der Auflistung *mCol* hinzugefügt werden. Dazu müssen wir zunächst prüfen, ob bereits eine Gruppe mit demselben Namen *und* derselben Gruppenposition existiert. Die Funktion, die dies ermittelt, schauen wir uns im Anschluss an diese Prozedur an. Im Moment reicht es, die folgenden Zeilen erst einmal so zu übernehmen.

```
If GroupExist(GruppenName, url) = False Then  
    mCol.Add NewGroup  
    Set Add = NewGroup  
Else  
    Set Add = Nothing
```



```
End If
Set NewGroup = Nothing
End Function
```

Wie bereits erwähnt, muss durch die *Add*-Anweisung geprüft werden, ob die gewünschte Gruppe bereits existiert. Dabei ist es entscheidend, dass nicht nur der Name der Gruppe, sondern auch deren Position innerhalb der Hierarchie identisch ist, damit die Gruppe als bereits existierend gekennzeichnet wird. Um beide Parameter zu prüfen, ist es deshalb erforderlich, den Namen und die Position der gesuchten Gruppe in den Argumenten der Funktion zu übergeben.

```
Public Function GroupExist(GruppenName, url) As Boolean
Dim ActGroup As LayerGroup
On Error Resume Next
```

Jetzt werden alle Objekte der Auflistung *mCol* einzeln in einer *For Next*-Schleife (Kapitel 2.3.3) durchlaufen und mit den Argumenten der Funktion verglichen:

```
For i = 1 To mCol.Count
Set ActGroup = mCol.Item(i)
If UCase(ActGroup.GruppenName) = UCase(GruppenName) And _
UCase(ActGroup.GruppenUrl) = UCase(url) Then
```

Sind sowohl Name als auch Position bereits durch ein Gruppenobjekt belegt, gibt die Funktion den Wert *TRUE* zurück, woraufhin diese Gruppe nicht mehr erzeugt werden kann.

```
GroupExist = True
Exit Function
End If
Next
End Function
```

Kommen wir nun wieder zur Verwaltung der Objekte der Auflistung zurück. Zum Abrufen der einzelnen Gruppenobjekte verwenden wir, wie bei dem *Collection*-Objekt, die *Item*-Eigenschaft. Da diese nur schreibgeschützt sein sollte, reicht die Erzeugung einer *Property Get*-Prozedur, die das Gruppenobjekt an einer bestimmten Position aus der Auflistung abrufen und als Objekt der Klasse *LayerGroup* zurückgibt.

```
Public Property Get Item(vntIndex As Variant) As LayerGroup
Set Item = mCol(vntIndex)
End Property
```

Da diese *Item*-Eigenschaft allein nicht ausreichen wird, um eine Gruppe abzurufen, erstellen wir eine zweite *Property Get*-Prozedur, mit der es möglich ist, eine Gruppe anhand des Gruppennamens und der Position abzurufen. Die Vorgehensweise ist dabei dieselbe wie bei der *GroupExist*-Funktion. Allerdings wird hier nicht der Wert *TRUE* zurückgegeben, wenn die angegebene Gruppe gefunden wurde, sondern der Verweis auf das Gruppenobjekt, das dem Namen und der Position entspricht.

```
Public Property Get ItemByNameAndUrl(GruppenName, url) As _
LayerGroup
Dim ActGrp As LayerGroup
For i = 1 To mCol.Count
Set ActGrp = mCol(i)
```

```

    If UCase(ActGrp.GruppenName) = UCase(GruppenName) And _
        UCase(ActGrp.GruppenUrl) = UCase(url) Then
        Set ItemByNameAndUrl = ActGrp
        Exit For
    End If
Next
Set ActGrp = Nothing
End Property

```

Natürlich brauchen wir auch noch eine Prozedur, mit der ein einzelnes Gruppenobjekt aus der Auflistung entfernt werden kann. Auch hier teilen wir die Funktion wieder in verschiedene Prozeduren auf. Mit der ersten wird ein Objekt aus der Auflistung an einer bestimmten Position entfernt, so wie Sie es bereits vom *Collection*-Objekt (Kapitel 2.5.1) kennen.

```

Public Sub Remove(vntIndex As Variant)
mCol.Remove vntIndex
End Sub

```

Die zweite ermittelt das Gruppenobjekt wieder anhand seines Namens und der Position innerhalb der Gruppenauflistung. Diese beiden Informationen werden wiederum als Argument der Anweisung mitgegeben.

```

Public Sub RemoveByNameAndUrl(GruppenName, url)
Dim ActGrp As LayerGroup
For i = 1 To mCol.Count
    Set ActGrp = mCol(i)
    If UCase(ActGrp.GruppenName) = UCase(GruppenName) And _
        UCase(ActGrp.GruppenUrl) = UCase(url) Then
        mCol.Remove i
        Exit Sub
    End If
Next
End Sub

```

Um das Klassenmodul vorerst abzuschließen, fügen wir diesem noch zwei Prozeduren hinzu, die auch in anderen Auflistungen vorhanden sind. Zum einen die Eigenschaft *Count*, die die Anzahl der in der Auflistung gespeicherten Objekte zurückgibt:

```

Public Property Get Count() As Long
Count = mCol.Count
End Property

```

Und die Funktion *Clear* zum Leeren des Auswahlsatzes. Hierbei brauchen wir einfach nur eine neue Instanz des *Collection*-Objekts in der Variablen *mCol* zu erzeugen, womit die Auflistung insgesamt bereinigt ist.

```

Public Sub Clear()
Set mCol = New Collection
End Sub

```

Formulargestaltung

So gerüstet können wir mit der Gestaltung des Formulars fortfahren. Um in diesem die Layer-Gruppen darzustellen, benötigen wir natürlich ein entsprechendes Steuerelement. Wenn Sie das Kapitel 3.3 durchgearbeitet haben, werden Sie sich sicherlich für die Verwendung des

Steuerelements *TreeView* (Kapitel 3.3.8) entscheiden, da hiermit eine hierarchische Abbildung der Gruppen möglich wird. (Alternativ können Sie auch das *AtlTreeView*-Steuerelement der Buch-CD verwenden.) Um eine komplette Verwaltung der Gruppen zu ermöglichen, benötigen Sie außerdem eine Reihe von Schaltflächen, über die der Benutzer später Befehle zum Hinzufügen und Entfernen und einige weitere Funktionen ausführen kann. Entweder Sie verwenden hierfür einzelne Schaltflächen (Kapitel 3.2.11) oder ein *Toolbar*-Steuerelement (Kapitel 3.3.7), in dem einige Schaltflächen eingefügt werden. Wir verwenden hier das *Toolbar*-Steuerelement.

Fügen Sie also beide Steuerelemente (*TreeView* und *ToolBar*) in einem Rahmen zusammengefasst ein, so dass sich vorerst folgendes Bild für das Formular ergibt.

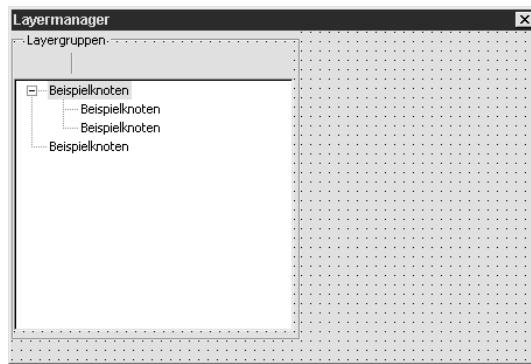


Abbildung 4.1: Steuerelemente zur Abbildung und Verwaltung der Layer-Gruppen

Das *Toolbar*-Steuerelement können Sie mit *TVToolbar* benennen. Für das *TreeView*-Steuerelement und den Rahmen können Sie die Namen so belassen, wie sie durch VBA vorgeschlagen wurden.

Dem Symbolleisten-Steuerelement fügen Sie nun zwei Schaltflächen hinzu, so wie es in Kapitel 3.3.7 beschrieben ist. Die erste Schaltfläche erhält den Schlüssel *New* und die zweite den Schlüssel *Delete*. Außerdem können Sie in den *Tooltip*-Eigenschaften der Schaltflächen einen geeigneten Text eintragen, der angezeigt wird, wenn der Mauszeiger über der Schaltfläche positioniert wird.

Als Nächstes muss natürlich das *TreeView*-Steuerelement mit den Gruppeneinträgen gefüllt werden. Dazu verwenden wir die Ereignisprozedur *Initialize* des Formulars, da diese immer beim Start des Programms automatisch ausgeführt wird. In diese Prozedur tragen wir zunächst nur die Codezeilen zur Erzeugung einer neuen Instanz der Klasse *LayerGroups* ein, die zuvor ja bereits besprochen wurde. Um diese Instanz in einer Objektvariablen zu speichern, müssen Sie natürlich eine globale Variable deklarieren, damit Sie aus jeder Prozedur heraus Zugriff auf die Gruppenauflistung haben. Sie deklarieren diese Variable – ich habe sie mit *Layergruppen* bezeichnet – entweder innerhalb des Formulars oder in einem Modul. Den Unterschied zwischen beiden Methoden erklärt das Kapitel 2.2.4. In unserem Fall reicht die Deklaration innerhalb des Formulars, da wir aus anderen Modulen heraus nicht auf diese Variablen zugreifen werden.

```
Dim Layergruppen As LayerGroups
Private Sub UserForm_Initialize()
On Error Resume Next
Set Layergruppen = New LayerGroups
```

Außerdem fügen wir dem *TreeView*-Steuerelement bereits einen ersten Eintrag hinzu. Dieser Eintrag bildet das oberste Element, dem alle weiteren Gruppen untergeordnet und bei dessen Auswahl durch den Benutzer alle Layer der Zeichnung angezeigt werden. Ich habe diesen Eintrag mit „Global“ bezeichnet, Sie können aber auch jeden anderen Text verwenden, das ist an dieser Stelle erst einmal egal.

```
Set xNode = TreeView1.Nodes.Add(, , "Root", "Global")
End Sub
```

Als Nächstes werden die beiden Schaltflächen, die dem *Toolbar*-Steuerelement hinzugefügt wurden, mit dem entsprechenden Code belegt. Diesen Code fügen Sie der Ereignisprozedur *ButtonClick* hinzu. Diese wird ausgeführt, wenn der Benutzer irgendeine Schaltfläche des *Toolbar*-Steuerelements betätigt hat. Innerhalb dieser Prozedur identifizieren wir die gedrückte Schaltfläche anhand einer *Select Case*-Anweisung. Um welche Schaltfläche es sich handelt, gibt das Argument *Button* zurück.

```
Private Sub TVToolbar_ButtonClick(ByVal Button As _
MSComctlLib.Button)
Select Case Button.Key
Case "New"
```

Da der Code zum Hinzufügen einer Gruppe etwas umfangreicher ist, fassen wir diesen in der Prozedur *AddNewGroup* zusammen und rufen dann einfach diese Prozedur auf, wenn der Benutzer die Schaltfläche mit dem Schlüssel *New* betätigt hat.

```
AddNewGroup
Case "Delete"
```

Das Entfernen einer Gruppe ist etwas einfacher, da hierfür bereits die Prozedur *RemoveByNameAndUrl* in der Klasse *LayerGroups* existiert. Ausschlaggebend dafür, welche Gruppe gelöscht wird, ist der Verweis auf das *Node*-Objekt, das in der *TreeView*-Eigenschaft *Selecte- dItem* (entspricht dem aktuell gewählten *Node*-Objekt) gespeichert ist.

Zunächst wird geprüft, ob es sich bei dem gewählten *Node*-Objekt um den Eintrag „Global“ handelt, den wir natürlich nicht löschen wollen. Ist dies der Fall, gibt die *Parent*-Eigenschaft den Wert *Nothing* zurück. Dieser weist darauf hin, dass es kein übergeordnetes Objekt mehr gibt. In diesem Fall wird die Prozedur vorzeitig verlassen.

```
If TypeName(TreeView1.SelectedItem.Parent) = "Nothing" _
Then Exit Sub
```

Andernfalls kann die gewählte Gruppe aus der Gruppenuflistung gelöscht werden. Um welche Gruppe genau es sich handelt, darüber informieren die Eigenschaften *Text* und *Parent.FullPath*. Mit diesen Angaben können wir nun die *RemoveByNameAndUrl*-Anweisung der Klasse *LayerGroups* aufrufen, damit die Gruppe aus der Auflistung gelöscht wird. Wenn Sie es ganz genau nehmen wollen, könnten Sie auch alle der Gruppe zugeordneten Untergruppen löschen, was aber nicht unbedingt notwendig ist, da diese Gruppen nicht mehr in der *TreeView*-Liste erscheinen werden. Sie hängen sozusagen in der Luft und haben keine Bedeutung mehr.

```
Layergruppen.RemoveByNameAndUrl _
TreeView1.SelectedItem.Text, _
TreeView1.SelectedItem.Parent.FullPath
```

Jetzt braucht nur noch der gewählte Eintrag aus der *Nodes*-Auflistung des *TreeView*-Steuerelements entfernt zu werden. Als Positionsangabe verwenden wir einfach den Index des gewählten *Node*-Objekts.

```
TreeView1.Nodes.Remove TreeView1.SelectedItem.Index
End Select
End Sub
```

Nun zu der Prozedur, die eine Gruppe der Auflistung und dem *TreeView*-Steuerelement hinzufügt. Als Erstes ermitteln wir den Namen der neuen Gruppe, der mittels der *InputBox*-Funktion abgefragt wird. Gibt diese eine Leerzeichenfolge zurück, hat der Benutzer entweder keine Zeichenfolge festgelegt oder die Schaltfläche ABBRECHEN betätigt. In jedem Fall kann mit dieser Eingabe keine Gruppe erzeugt werden, weshalb die Prozedur vorzeitig verlassen wird.

```
Sub AddNewGroup()
Dim NewGroup As LayerGroup
On Error Resume Next
sValue = InputBox("Geben Sie den Namen der Layergruppe ein:", _
    "Neue Layergruppe")
If sValue = "" Then Exit Sub
```

Die Position innerhalb der Auflistung steht ja bereits durch das selektierte *Node*-Objekt fest. Aus diesem Grund übernehmen wir die Pfadangabe aus der *FullPath*-Eigenschaft des *SelectedItem*-Objekts. Damit sind dann auch beide Parameter für die Erzeugung einer neuen Gruppe festgelegt und wir können diese mit der *Add*-Anweisung erzeugen.

```
url = TreeView1.SelectedItem.FullPath
Set NewGroup = Layergruppen.Add(sValue, url)
```

Sollte die neue Gruppe nicht erzeugt werden können (falls bereits eine Gruppe mit demselben Namen existiert), gibt die *Add*-Anweisung *Nothing* zurück. In diesem Fall geben wir eine Fehlermeldung aus.

```
If TypeName(NewGroup) = "Nothing" Then
    MsgBox "Layergruppe " & sValue & _
        " konnte nicht hinzugefügt werden!", vbCritical, _
        "Neue Gruppe"
Else
```

Andernfalls wird der Name der Gruppe als Unterelement des gewählten *Node*-Objekts hinzugefügt.

```
TreeView1.Nodes.Add TreeView1.SelectedItem, tvwChild, , sValue
End If
End Sub
```

Damit wäre die erste Problemstellung bereits gelöst und der Benutzer kann bereits neue Gruppen hierarchisch erzeugen.

Problemstellung

Nun ist es aber auch erforderlich, diese Gruppen in der Zeichnung zu speichern und bei erneutem Aufruf des Formulars wieder herzustellen. In diesem Zusammenhang stellt sich das Problem, wo in der Zeichnung die Gruppen gespeichert werden sollen?

Lösungsansatz

Da es sich bei den Gruppen um keine „regulären“ Zeichnungsdaten, sondern vielmehr um erweiterte Daten handelt, bieten sich zwei Möglichkeiten. Nach der Lektüre der *Kapitel 7.4* und *7.14* wissen Sie, dass die Gruppen entweder in den erweiterten Daten der Layer-Auflistung (*Kapitel 7.4*) oder in einem Erweiterungswörterbuch (*Kapitel 7.14*) gespeichert werden können.

Problemlösung

Für die Layer-Gruppen wählen wir die zweite Möglichkeit, da dieser Weg eleganter ist als der erste. Die Prozeduren, die das Lesen und Speichern der Daten in dem Erweiterungswörterbuch steuern werden, fügen Sie bitte dem Klassenmodul *LayerGroups* hinzu. Bei der Speicherung und dem Lesen der Daten des Erweiterungswörterbuchs gehen wir nach dem Grundprinzip aus *Kapitel 7.14.2* vor. Die erweiterten Daten werden also in einem Datensatz vom Typ *AcadXRecord* gespeichert.

Speichern der Layer-Gruppen

Fangen wir mit der Speicherung der Daten an. Stellen Sie sich vor, der Benutzer hätte bereits eine gewisse Anzahl von Gruppen erzeugt, wodurch nun das Auflistungsobjekt *LayerGruppen* diese Anzahl an Gruppenobjekten gespeichert hat. Um nun diese Informationen in der Zeichnung abzuspeichern, benötigen wir eine Prozedur, die zunächst auf ein Wörterbuch zugreift, um diesem dann die Daten hinzuzufügen. Da das Wörterbuch einen Namen benötigt, mit dem wir es eindeutig in der Wörterbuchsammlung identifizieren können, legen wir hierfür die Bezeichnung „LAYER_GROUP“ fest. Sie könnten natürlich auch einen anderen Namen wählen, müssen aber in jedem Fall darauf achten, dass dieser Name nicht bereits von einer anderen Anwendung verwendet wird.

```
Public Sub saveLayerGroups()  
Dim GrpDict As AcadDictionary  
Dim GrpObj As AcadXRecord  
Dim ActGrp As LayerGroup
```

Da wir nur jeweils zwei Datenzeilen im XRecord ablegen werden (Gruppenname und URL), müssen die Arrays für den Code und die Daten dementsprechend deklariert werden. Wenn Sie noch mehr Informationen im XRecord ablegen wollen, beispielsweise einen Beschreibungstext, müssen Sie die Dimension der Arrays entsprechend erhöhen.

```
Dim xRecordType(1) As Integer, xRecordData(1) As Variant  
On Error Resume Next
```

Als Erstes muss natürlich ein eventuell vorhandenes Wörterbuch mit dem Namen „LAYER_GROUP“ geleert werden. Am einfachsten ist es, das Wörterbuch einfach zu löschen und danach ein neues anzulegen. Sie könnten alternativ aber auch auf die einzelnen Datensätze des Wörterbuchs zugreifen und diese dann daraus entfernen.

```
ThisDrawing.Dictionaries("LAYER_GROUPS").Delete  
Set GrpDict = ThisDrawing.Dictionaries.Add("LAYER_GROUPS")
```

Anschließend wird die Auflistung *mCol* (enthält ja die aktuellen Layer-Gruppen) mithilfe einer *For Next*-Schleife durchlaufen. Bei jedem Durchlauf wird ein neuer XRecord dem Erweiterungswörterbuch hinzugefügt und der Verweis auf diesen XRecord in der Variablen *GrpObj* gespeichert.

```

For i = 1 To mCol.Count
    Set GrpObj = GrpDict.AddXRecord("Gruppe" & CStr(i))
    Set ActGrp = mCol.Item(i)

```

Danach müssen wir nur noch die Felder der beiden Arrays mit den entsprechenden Daten füllen. Als Gruppencode verwenden wir dabei 0 und 1, die ja eine Zeichenfolge als Datentyp definieren. Sie können auch einen anderen Gruppencode verwenden. Sie müssen dabei nur darauf achten, dass der Gruppencode eine Zeichenfolge definiert. Die Gruppencodes, die Sie hierfür verwenden können, entnehmen Sie ebenfalls dem *Kapitel 7.14.2*.

```

xRecordType(0) = 0: xRecordData(0) = ActGrp.GruppenName
xRecordType(1) = 1: xRecordData(1) = ActGrp.GruppenUrl

```

Zum Schluss werden dann nur noch diese Daten in das zuvor hinzugefügte XRecord geschrieben.

```

    GrpObj.SetXRecordData xRecordType, xRecordData
Next
End Sub

```

Lesen der Layer-Gruppen

Jedes Mal, wenn das Programm gestartet wird, müssen die Layer-Gruppen gelesen werden, damit immer der aktuelle Stand der aktuellen Zeichnung ermittelt werden kann.

```

Public Sub readLayerGroups()
    Dim GrpDict As AcadDictionary
    Dim GrpObj As AcadXRecord
    Dim xRecordType, xRecordData
    On Error Resume Next

```

Bevor das Auslesen der Daten beginnt, müssen erst einmal die vorhandenen Daten der Auflistung *mCol* gelöscht werden. Für diese Aufgabe haben wir ja bereits die Prozedur *Clear* angelegt.

```
Clear
```

Als Erstes wird ein Verweis auf das Wörterbuch, das unter dem Namen „LAYER_GROUPS“ abgespeichert wurde, ermittelt und in der Variablen *GrpDict* gespeichert. Sollte noch kein Wörterbuch dieses Namens existieren, wird ein solches angelegt und dieser Verweis in der Variablen gespeichert.

```

Set GrpDict = ThisDrawing.Dictionaries("LAYER_GROUPS")
If TypeName(GrpDict) = "Nothing" Then
    Set GrpDict = ThisDrawing.Dictionaries.Add("LAYER_GROUPS")
End If

```

Danach müssen Sie nur noch die einzelnen Eintragungen des Wörterbuchs nacheinander abrufen. Da dieses Wörterbuch nur XRecords (*Kapitel 7.14.2*) enthält, können wir bei jedem Durchlauf die erweiterten Daten des jeweiligen Eintrags abrufen. Jeder Eintrag enthält im ersten Feld den Namen und im zweiten Feld die Position der Gruppe. Mit diesen Informationen bestückt, rufen wir nun die *Add*-Anweisung auf, wobei jedes Mal ein neues Objekt der Gruppenauflistung hinzugefügt wird.

```

For Each GrpObj In GrpDict
    GrpObj.GetXRecordData xRecordType, xRecordData
    Add xRecordData(0), xRecordData(1)
Next
End Sub

```

Beide Funktionen, sowohl *saveLayerGroups* also auch *readLayerGroups*, sind nun in der Klasse *LayerGroups* integriert und können demnach über die Objektvariable *LayerGruppen*, die ja eine Instanz dieser Klasse darstellt, aufgerufen werden.

Funktionserweiterung des Formulars

Nachdem das Klassenmodul um die Lese- und Schreibfunktionen erweitert wurde, benötigen wir natürlich auch noch eine Prozedur, die die Daten in das *TreeView*-Steuerelement überträgt. (Diese Prozedur fügen Sie bitte dem Formularcode hinzu.)

```

Private Sub readLayerGroups()
    Dim xNode As MSComctlLib.Node
    On Error Resume Next

```

Der erste Schritt in dieser Prozedur ist das Lesen der Gruppen aus dem Erweiterungswörterbuch mit der Funktion *readLayerGroups*.

```

Layergruppen.readLayerGroups

```

Als Nächstes werden die Gruppen an das *TreeView*-Steuerelement übertragen. Dabei wird zunächst der Eintrag „Global“ erzeugt. Sie erinnern sich, dies war der Eintrag, dem alle anderen Gruppen untergeordnet werden und der zuvor in der *Initialize*-Ereignisprozedur des Formulars angelegt wurde.

```

Set xNode = TreeView1.Nodes.Add(, , "Root", "Global")

```

(Sie müssen diese Zeile in der *Initialize*-Ereignisprozedur des Formulars nun durch die Anweisung *readLayerGroups* ersetzen.)

Anschließend wird die Gruppenuflistung in einer *For Next*-Schleife durchlaufen und die Namen der einzelnen Gruppen werden dem *TreeView*-Steuerelement hinzugefügt. Da jede Gruppe einer anderen Gruppe oder dem Eintrag untergeordnet ist – welchem Element, entscheidet die Positionsangabe in der *GruppenUrl*-Eigenschaft – benötigen wir das jeweils übergeordnete *Node*-Objekt. Dieses liefert die *GetXNode*-Funktion, die als Nächstes besprochen wird. Gehen Sie einfach davon aus, dass diese Funktion das entsprechende *Node*-Objekt liefert.

```

For i = 1 To Layergruppen.Count
    Set xNode = GetXNode(Layergruppen.Item(i).GruppenUrl)
    TreeView1.Nodes.Add xNode, tvwChild, , _
        Layergruppen.Item(i).GruppenName
Next

```

Zum Schluss können wir noch den ersten Eintrag der *TreeView*-Liste aktivieren und den Knoten eventuell öffnen, falls er untergeordnete Elemente enthält.

```

TreeView1.SelectedItem = TreeView1.Nodes(1)
TreeView1.SelectedItem.Expanded = True
End Sub

```


Nun zur Ermittlung des *Node*-Objekts, das der angegebenen Position entspricht. Dafür müssen wir nur die *Nodes*-Auflistung des *TreeView*-Steuerelements durchlaufen und die *FullPath*-Eigenschaft des jeweiligen Eintrags mit dem Argument *Url* abgleichen. Stimmen beide überein, handelt es sich um das gesuchte *Node*-Objekt, woraufhin der Verweis auf dieses Objekt an den Funktionsnamen *GetXNode* übergeben wird.

```
Public Function GetXNode(url) As MSComctlLib.Node
On Error Resume Next
For i = 1 To TreeView1.Nodes.Count
    If TreeView1.Nodes(i).FullPath = url Then
        Set GetXNode = TreeView1.Nodes(i)
        Exit Function
    End If
Next
End Function
```

Jetzt müssen wir nur noch die Speicherung der Layer-Gruppen irgendwo unterbringen. Am besten ist es natürlich, dem Benutzer hierfür eine Schaltfläche anzubieten (wie in *Kapitel 3.2.11* beschrieben), mit der er vor Beendigung des Programms die Änderungen speichern kann. Alternativ dazu sollte er natürlich auch eine Schaltfläche zum Abbrechen der Bearbeitung erhalten. Fügen Sie also dem Formular zwei Schaltflächen hinzu, die erste beschriften Sie mit *SPEICHERN* und die zweite mit *ABBRECHEN*.

Der Code, den die jeweiligen *Click*-Ereignisprozeduren ausführen sollen, ist recht simpel. In beiden Fällen wird das Formular mit der *Hide*-Anweisung ausgeblendet. Nur die *SPEICHERN*-Schaltfläche erhält zusätzlich die *saveLayerGroups*-Anweisung, die auf die aktuelle Instanz der Klasse *LayerGroups* angewendet wird.

```
Private Sub SaveButton_Click()
Layergruppen.saveLayerGroups
Me.Hide
End Sub

Private Sub CancelButton_Click()
Me.Hide
End Sub
```

Bis hierhin funktioniert das Ganze schon recht reibungslos. Was noch fehlt, ist die Darstellung der Schaltflächenfunktionen (*GRUPPE HINZUFÜGEN* und *GRUPPE ENTFERNEN*) durch ein Symbol. Da diese Schaltflächen durch ein *Toolbar*-Steuerelement dargestellt werden, müssen wir zunächst die Symbole in einem *ImageList*-Steuerelement (*Kapitel 3.3.1*) zwischenlagern und dann zur Laufzeit den Schaltflächen des *Toolbar*-Steuerelements zuordnen. Fügen Sie also dem Formular ein *ImageList*-Steuerelement hinzu und laden Sie über dessen Eigenschaftsseite zwei Bilder, die die Funktionen optisch darstellen sollen. Anschließend fügen Sie die folgenden Codezeilen der *Initialize*-Ereignisprozedur des Formulars hinzu.

```
Set TVToolbar.ImageList = TVImageList
TVToolbar.Buttons("New").Image = 1
TVToolbar.Buttons("Delete").Image = 2
```

Damit wäre der erste Teil des Programms bereits abgeschlossen. Alles, was bis hierhin durchgeführt wurde, finden Sie auch im VBA-Projekt *Teil1.dvb* wieder.

4.3 Verwaltung der Layer

Problemstellung

Kommen wir nun zu den Layern der Zeichnung, die ja noch den einzelnen Gruppen zugeordnet werden müssen. Da wir bereits mit den Klassen *LayerGroups* und *LayerGroup* den Weg der temporären Verarbeitung eingeschlagen haben, also alle Informationen zunächst im Formular verarbeiten lassen wollen, um diese dann wahlweise speichern oder verwerfen zu können, sollte dieser Weg ebenfalls bei den Layer-Daten beschriftet werden. In diesem Zusammenhang stellt sich das Problem der temporären Verarbeitung der Layer-Informationen.

Lösungsansatz

Da eine direkte Verarbeitung der Layer nach *Kapitel 7.4.2* nicht in Frage kommt, da dadurch die Layer direkt geändert werden, stehen wieder nur zwei Möglichkeiten zur Verfügung. Der erste führt wieder über ein Array, in dem jedes Feld die Daten der einzelnen Layer speichern würde. Die zweite Möglichkeit bestünde wieder in der Verwaltung durch Klassenmodule, so wie dies bereits bei den Layer-Gruppen realisiert wurde.

Problemlösung

Auch hierfür eignen sich Klassenmodule am besten. Also fügen wir dem Projekt zwei weitere Klassenmodule hinzu. Das erste (*GroupLayers*) wird als Layer-Auflistung dienen, das zweite (*GroupLayer*) als Vorlage für die einzelnen Layer-Definitionen der Auflistung.

Beginnen wir mit der Klasse *GroupLayer*. Da diese die Daten der einzelnen Layer aufnehmen soll, muss sie zunächst auch über die Eigenschaften verfügen, die auch ein einzelner Layer hergibt (*Kapitel 7.4.2*). Die Namen, die wir für die einzelnen Eigenschaften verwenden, entsprechen im großen und ganzen denen der Layer-Eigenschaften. Eine Ausnahme bildet hier die *Lock*-Eigenschaft der Layer. Da diese Bezeichnung bereits als Schlüsselwort in VBA existiert, können wir diesen Namen laut *Kapitel 2.1.1* nicht verwenden. Aus diesem Grund erweitern wir den Namen und nennen diese Eigenschaft *Locked*.

```
Public Name As String
Public LayerOn As Boolean
Public Freeze As Boolean
Public Locked As Boolean
Public Plottable As Boolean
Public Color As Long
Public LineType As String
Public LineWeight As Long
Public PlotStyleName As String
Public ViewportDefault As Boolean
```

Zusätzlich zu diesen Variablen benötigen wir noch einige Eigenschaften, die Informationen aufnehmen werden, die ein normales Layer-Objekt nicht hergibt. Dazu gehören die Eigenschaften *Description* (verwaltet einen Beschreibungstext für den jeweiligen Layer) und *ViewportFreeze* (speichert den aktuellen Status für Frieren im aktuellen AF).

```
Public Description As String
Public ViewportFreeze As Boolean
```

Außerdem müssen wir den Original-Layer-Namen speichern, damit wir die Zuordnung zum Layer wiederfinden, falls der Benutzer dessen Namen während der Bearbeitung im Dialog geändert hat.

```
Public OriginalName As String
```

Jetzt fehlt nur noch eine Variable, die die Gruppenzuordnung des Layers aufnimmt. Da wir es dem Benutzer ermöglichen wollen, den Layer in mehreren Gruppen abzulegen, reicht hier eine einfache *String*-Variable nicht aus. In diesem Fall müssten wir mindestens eine einfache Auflistung (*Kapitel 2.5.1*) verwenden. Wenn Sie es ganz genau machen wollen, können Sie aber auch eine eigene Klasse definieren. Der Grund dafür wird sich noch im Verlauf des Programms ergeben. Zunächst sollte ein *Collection*-Objekt ausreichen.

```
Public Groups As Collection
```

Damit mit dieser Objektvariablen gearbeitet werden kann, müssen wir eine neue Instanz der Auflistung erstellen. Den Vorgang kennen Sie ja bereits von der Klasse *LayerGroups*.

```
Private Sub Class_Initialize()  
Set Groups = New Collection  
End Sub  
  
Private Sub Class_Terminate()  
Set Groups = Nothing  
End Sub
```

Der Aufbau der Layer-Auflistung entspricht weitgehend Gruppenu Auflistung. Auch hier werden die einzelnen Objekte in der Objektvariablen *mCol* verwaltet. Eine neue Instanz der Auflistung wird wieder bei der Initialisierung der Klasse erzeugt.

```
Private mCol As Collection  
  
Private Sub Class_Initialize()  
Set mCol = New Collection  
End Sub  
  
Private Sub Class_Terminate()  
Set mCol = Nothing  
End Sub
```

Auch diese Klasse erhält zunächst die Grundfunktionen einer Auflistung. Dazu gehört natürlich eine *Add*-Funktion, um der Auflistung ein neues Layer-Objekt hinzuzufügen:

```
Public Function Add(LayerName) As GroupLayer  
Dim NewLayer As GroupLayer  
Set NewLayer = New GroupLayer  
NewLayer.Name = LayerName  
mCol.Add NewLayer  
Set Add = NewLayer  
Set NewLayer = Nothing  
End Function
```

Außerdem eine *Item*-Eigenschaft zum Abrufen eines Layer-Objekts an einer bestimmten Position:

```
Public Property Get Item(vntIndex As Variant) As GroupLayer  
Set Item = mCol(vntIndex)  
End Property
```

Die *Count*-Eigenschaft, um die Anzahl der in der Auflistung vorhandenen Layer zu ermitteln:

```
Public Property Get Count() As Long
Count = mCol.Count
End Property
```

Und die *Remove*-Anweisung, um ein Layer-Objekt aus der Auflistung wieder zu entfernen:

```
Public Sub Remove(vntIndexKey As Variant)
mCol.Remove vntIndexKey
End Sub
```

Darüber hinaus benötigen wir noch eine Funktion, die das Layer-Objekt anhand des Layer-Namens ermittelt. Ich will der Entwicklung des Programms zwar nicht vorgreifen, aber wir können in diese Funktion gleich eine Option einbauen, mit der entschieden werden soll, auf welche Variable sich die Funktion bei der Auswahl beziehen soll (*Name* oder *OriginalName*).

```
Public Property Get ItemByName(Name, Optional UseOriginalName As _
Boolean = False) As GroupLayer
Dim ActGrp As GroupLayer
For i = 1 To mCol.Count
Set ActGrp = mCol(i)
If UseOriginalName = True Then
```

Handelt es sich bei dem gesuchten Layer-Namen um den Originalnamen des Layers, verwenden wir für den Vergleich die Eigenschaft *OriginalName*.

```
    If UCase(ActGrp.OriginalName) = UCase(Name) Then
        Set ItemByName = ActGrp
        Exit For
    End If
Else
```

Andernfalls die *Name*-Eigenschaft, um das Layer-Objekt zu identifizieren:

```
    If UCase(ActGrp.Name) = UCase(Name) Then
        Set ItemByName = ActGrp
        Exit For
    End If
End If
Next
End Property
```

Bei der *RemoveByName*-Prozedur brauchen wir diese Auswahl nicht zu treffen. Hier ist der temporäre Name des Layers ausschlaggebend, da der Benutzer immer nur diesen Namen angezeigt bekommt.

```
Public Sub RemoveByName(Name)
Dim ActGrp As GroupLayer
On Error Resume Next
```

Der Löschvorgang funktioniert genau wie bei den Layer-Gruppen. Es werden mithilfe einer *For Next*-Schleife die einzelnen Objekte der Auflistung abgerufen und mit dem gesuchten Layer-Namen verglichen. Stimmen beide überein, wird das Layer-Objekt aus der Auflistung entfernt.

```

For i = 1 To mCol.Count
    Set ActGrp = mCol(i)
    If UCase(ActGrp.Name) = UCase(Name) Then
        mCol.Remove I
        Exit For
    End If
Next
End Sub

Public Sub Clear()
Set mCol = New Collection
End Sub

```

Lesen der Layer

Jetzt benötigen Sie nur noch eine Prozedur, die die Informationen der Layer der Zeichnung in die Auflistung überträgt, damit diese dann dem Programm zur Verfügung stehen.

```

Public Sub ReadLayers()
Dim ActLayer As AcadLayer
Dim NewLayer As GroupLayer
On Local Error Resume Next

```

Für das Programmbeispiel wird es ausreichen, die Layer der aktuellen Zeichnung zu verwenden. Der Zugriff auf dieses Dokumentobjekt und dessen Layer ist in *Kapitel 7.4* beschrieben. Um die einzelnen Layer-Objekte der *Layers*-Auflistung abzurufen, verwenden wir eine *For Each Next*-Schleife (*Kapitel 2.3.3*). Bei jedem Durchlauf enthält die Objektvariable *ActLayer* einen Verweis auf den jeweils nächsten Layer.

```

For Each ActLayer In ThisDrawing.Layers

```

Über die *Add*-Anweisung können wir nun ein neues Objekt der Klasse *GroupLayer* der Auflistung hinzufügen. Als Argument der Anweisung dient der Name des jeweiligen Layers. Den durch *Add* zurückgegebenen Verweis speichern wir in der Variablen *NewLayer*.

```

    Set NewLayer = Add(ActLayer.Name)

```

Mit dieser Objektvariablen wird anschließend auch weitergearbeitet. Die verschiedenen Eigenschaftswerte des Layers werden an die entsprechenden Eigenschaften weitergegeben.

```

NewLayer.OriginalName = ActLayer.Name
NewLayer.Color = ActLayer.Color
NewLayer.Freeze = ActLayer.Freeze
NewLayer.LayerOn = ActLayer.LayerOn
NewLayer.LineType = ActLayer.LineType
NewLayer.LineWeight = ActLayer.LineWeight
NewLayer.Locked = ActLayer.Lock
NewLayer.PlotStyleName = ActLayer.PlotStyleName
NewLayer.Plottable = ActLayer.Plottable
NewLayer.ViewportDefault = ActLayer.ViewportDefault

```

Bleiben noch die Eigenschaften, die nicht direkt über das Layer-Objekt abrufbar sind. Als Erstes haben wir den Status des Layers im aktuellen Ansichtsfenster. Diesen Status (gefroren = *TRUE*/getaut = *FALSE*) müssen wir gesondert ermitteln, was die Funktion *FreezeInActiveViewport* für uns erledigt.

```

NewLayer.ViewportFreeze = FreezeInActiveViewport(ActLayer.Name)

```

Den Beschreibungstext des Layers speichern wir in seinen erweiterten Daten, die mit *GetXData* abgerufen werden können (*Kapitel 7.1.2*). Als Anwendungsname (ist im eigentlichen Sinne zwar keine Anwendung, macht aber nichts) habe ich „Description“ gewählt. Sie können aber auch „Beschreibung“ oder irgendeinen anderen Namen wählen, das spielt keine Rolle. Da dieser Datensatz lediglich den Namen der „Anwendung“ und den eigentlichen Beschreibungstext enthält, kann die Zuordnung direkt erfolgen. Der Beschreibungstext wird immer im Feld mit dem Index 1 zu finden sein.

```
ActLayer.GetXData "Description", xTypeOut, xDataOut
If IsEmpty(xDataOut) = False Then
    NewLayer.Description = xDataOut(1)
Else
    NewLayer.Description = ""
End If
```

Bei den Gruppen, denen der Layer zugeordnet wurde, ist das schon etwas anders. Hier handelt es sich um eine unbekannte Anzahl von Gruppennamen, die erst durch die *UBound*-Anweisung ermittelt werden kann, nachdem der Datensatz über die *GetXData*-Anweisung ausgelesen wurde.

```
ActLayer.GetXData "LAYERGRUPPEN", xTypeOut, xDataOut
```

Da im ersten Feld wieder der Anwendungsname, in diesem Fall „LAYERGRUPPEN“, zu finden sein wird, brauchen wir nach den Gruppennamen nur in den nachfolgenden Feldern, angefangen mit dem zweiten (Index 1) bis hin zur Obergrenze des Arrays, zu suchen. Die einzelnen Gruppennamen fügen wir der Auflistungsvariablen *Groups* mithilfe der *Add*-Anweisung hinzu.

```
For i = 1 To UBound(xDataOut)
    NewLayer.Groups.Add xDataOut(i)
Next
Next
End Sub
```

Kommen wir nun noch zu der Funktion, die uns den Status des Layers im aktuellen Ansichtsfenster liefert. Wie Sie ja vielleicht wissen, können Ansichtsfenster im Papierbereich jedes für sich einen eigenen Layer-Status erhalten, also Layer in AF1 gefroren und in AF2 getaut usw. Da diese Funktionalität nicht im *Layer*-Objekt enthalten ist, müssen wir uns auf eine andere Art weiterhelfen. Welche Layer im jeweiligen Ansichtsfenster gefroren sind, ist in den erweiterten Daten des AF gespeichert.

```
Function FreezeInActiveViewport(Name As String) As Boolean
Dim xRypeOut, xDataOut
On Error Resume Next
```

Da wir im Layer-Manager nur auf das aktuelle Ansichtsfenster im Papierbereich zurückgreifen wollen, können wir die erweiterten Daten der *ActivePViewport*-Eigenschaft (*Kapitel 6.2.2*) des aktuellen Dokuments abrufen.

```
ThisDrawing.ActivePViewport.GetXData "ACAD", xTypeOut, xDataOut
```

Neben einigen anderen Einstellungen, die ebenfalls in den erweiterten Daten des AF gespeichert wurden, finden wir die Namen der gefrorenen Layer unter dem Gruppencode 1003 wieder. Also durchlaufen wir das Array, das die *GetXData*-Anweisung zurückgegeben hat, Feld für Feld.

```

For i = 0 To UBound(xDataOut)
  If xTypeOut(i) = 1003 Then
    If UCase(xDataOut(i)) = UCase(Name) Then

```

Befindet sich der angegebene Layer-Name in einem Datenfeld, dessen Gruppencode 1003 lautet, ist der Layer in diesem AF gefroren.

```

      FreezeInActiveViewport = True
    Exit For
  End If
End If
Next
End Function

```

Das sollte für den Anfang erst einmal reichen. Wir wenden uns nun der Darstellung der Layer in dem Formular zu.

Problemstellung

Nachdem wir die Grundlagen für die Verwaltung der Layer entwickelt haben, stellt sich natürlich die Frage, wo und wie die Informationen der Layer im Formular angezeigt werden sollen.

Lösungsansatz

Um alle Informationen der Layer auf einen Blick darstellen zu können, benötigen wir ein entsprechendes Steuerelement. Für diese Aufgabe eignet sich wohl am besten das *List View*-Steuerelement (*Kapitel 3.3.2*), da hiermit nicht nur die Informationen spaltenweise angezeigt, sondern außerdem auch Symbole abgebildet werden können. Diese Symbole wiederum, die den Status Frieren, Ein, Sperren usw. optisch darstellen werden (ähnlich dem AutoCAD-Layermanager), müssen in einem *ImageList*-Steuerelement (*Kapitel 3.3.1*) zwischengespeichert werden.

Alternativ finden Sie auf der Buch-CD auch Steuerelemente, die die gleiche Funktion erfüllen wie die Standardsteuerelemente von Microsoft.

Problemlösung

Nachdem die grobe Vorgehensweise für die Darstellung und Steuerung der Layer festgelegt wurde, können wir zunächst das Formular um die angesprochenen Steuerelemente erweitern, so dass sich folgende Darstellung ergibt.

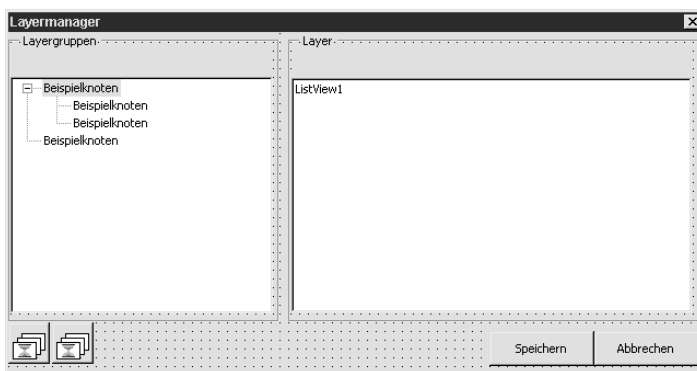


Abbildung 4.2: Erweitertes Layermanager-Formular

Da wir uns bei der Darstellung an dem AutoCAD-Layermanager orientieren wollen, bedarf es noch einiger Vorarbeiten beim Start des Programms. Einige Eigenschaften sollen durch Symbole dargestellt werden, also müssen wir diese Symbole zunächst in dem *ImageList*-Steuerelement zwischenspeichern. Das zuvor eingefügte *ImageList*-Steuerelement benennen Sie in *LVImageList* um. In dieses Steuerelement laden Sie nun die entsprechenden Symbole, in der Reihenfolge, wie es in Abbildung 4.3 zu sehen ist. Die Symbole finden Sie ebenfalls auf der Buch-CD im Verzeichnis *04/Images*.



Abbildung 4.3: Symbole des *LVImageList*-Steuerelements

Auch das *ListView*-Steuerelement bedarf noch einer kleinen Anpassung. So muss die *View*-Eigenschaft auf den Wert *lvwReport* eingestellt werden, damit eine Listenansicht während der Laufzeit entsteht. Außerdem müssen noch einige Spalten angelegt werden, von denen jede eine bestimmte Layer-Eigenschaft darstellt. Wie dies wiederum gemacht wird, erklärt das *Kapitel 3.3.2*.

Nr.	Spaltentext	Spaltenbreite	Nr.	Spaltentext	Spaltenbreite
1	Name	50	6	Plotten	15
2	Beschreibung	100	7	Farbe	50
3	Ein	15	8	Linientyp	50
4	Frieren	15	9	Linienstärke	50
5	Sperren	15			

Tabelle 4.1: Spalten des *ListView*-Steuerelements

Nun werden Sie in dieser Anordnung sicher die Spalten „PlotStil, in neuem AF frieren“ und „in aktuellem AF frieren“ vermissen. Nun diese Spalten sind abhängig von der aktuellen Einstellung der AutoCAD-Zeichnung und werden somit erst zur Laufzeit hinzugefügt. Also bewegen wir uns wieder zur Ereignisprozedur *Initialize* des Formulars und nehmen die notwendigen Ergänzungen am Code vor. Zunächst die Zuordnung des *LVImageList*-Steuerelements zur *SmallIcons*-Eigenschaft des *ListView*-Steuerelements, damit eine Verbindung zwischen *ImageList*- und *ListView*-Icons hergestellt wird. Zusätzlich können Sie gleich deren *Sorted*-Eigenschaft aktivieren, damit die Layer später nach Namen sortiert im *ListView*-Steuerelement dargestellt werden.

```
Set ListView1.SmallIcons = LVImageList
ListView1.Sorted = True
```

Anschließend erfolgt die Ergänzung der fehlenden Spalten, wenn gewisse Voraussetzungen erfüllt werden. Zunächst prüfen wir die Einstellung der Systemvariablen *PSTYLEMODE* (*Kapitel 6.2.3*). Gibt diese den Wert 1 zurück, befindet sich die Zeichnung im farbabhängigen Modus und wir benötigen keine Spalte, um den Plotstil anzuzeigen:

```
If ThisDrawing.GetVariable("PSTYLEMODE") = 0 Then
```


Andernfalls wird eine zusätzliche Spalte eingefügt, um den verwendeten Plotstilnamen des jeweiligen Layers anzuzeigen.

```
ListView1.ColumnHeaders.Add , , "Plotstil", 50
End If
```

Für die Eigenschaften „Frieren in neuem und im aktuellen Ansichtsfenster“ ist es erforderlich, dass sich die Zeichnung im Papierbereich befindet. Laut *Kapitel 6.2.2* gibt diesen Status der Inhalt der Eigenschaft *ActiveSpace* des aktuellen Dokumentobjekts (*ThisDrawing*) wieder.

```
If ThisDrawing.ActiveSpace = acPaperSpace Then
```

Befindet sich die Zeichnung im Papierbereich, werden folgende Spalten dem *ListView*-Steuerelement hinzugefügt.

```
ListView1.ColumnHeaders.Add , , "Frieren in neuem AF", 15
ListView1.ColumnHeaders.Add , , "Frieren im aktuellen AF", 15
End If
```

Nachdem das *ListView*-Steuerelement angepasst wurde und nun zur Aufnahme der Layer-Daten bereit ist, können die Layer-Informationen an das *ListView*-Steuerelement übertragen werden.

Damit dies funktioniert, müssen wir zunächst eine neue Instanz der Klasse *GroupLayers* erstellen. Deklarieren Sie hierfür die Variable *Layers* im Formular.

```
Dim Layers As GroupLayers
```

Da die Erzeugung der neuen Instanz dieser Klasse und der Aufruf der *ReadLayers*-Anweisung vor dem Start des Formulars erfolgen soll, können wir diese Codezeilen der *Initialize*-Ereignisprozedur des Formulars hinzufügen.

```
Set Layers = New GroupLayers
Layers.ReadLayers
```

Damit sind die Layer-Informationen temporär verfügbar und können über die Objektvariable *Layers* abgerufen und angepasst werden.

Auswahl der anzuzeigenden Layer

Um die Layer einer bestimmten Gruppe anzuzeigen, muss der Benutzer einen bestimmten Eintrag im *TreeView*-Steuerelement ausgewählt haben. Wie im *Kapitel 3.3.8* nachzulesen ist, wird durch diese Auswahl die Ereignisprozedur *NodeClick* ausgelöst. Also setzen wir an dieser Stelle mit dem Code zur Anzeige der zur jeweiligen Gruppe gehörigen Layer an. Den Code der hier ausgeführt werden soll, werden wir in zwei weitere Prozeduren aufteilen. Die erste dieser Prozeduren benennen Sie mit *ReadGroupLayers*. Als Argument dieser Prozedur legen wir eine Variable vom Typ *Node* fest. Die Prozedur wird dann vom *NodeClick*-Ereignis aufgerufen, aber dazu kommen wir später. Wenden wir uns zunächst dem Code zu.

```
Sub ReadGroupLayers(xNode As MSComctlLib.Node)
On Error Resume Next
```

Als Erstes muss natürlich das *ListView*-Steuerelement von eventuell vorhandenen Einträgen bereinigt werden.

```
ListView1.ListItems.Clear
```

Jetzt bedarf es einer Unterscheidung, welchen Eintrag der Benutzer im *TreeView*-Steuerelement gewählt hat.

```
If TypeName(xNode.Parent) = "Nothing" Then
```

War es der oberste Eintrag („Global“), was daran zu erkennen ist, dass das *Node*-Objekt kein übergeordnetes Element besitzt, dann lassen wir im *ListView*-Steuerelement alle Layer der aktuellen Zeichnung anzeigen. Hierfür durchlaufen wir die Auflistung, in der die Eigenschaften der Layer temporär gespeichert wurden. Bei jedem Durchlauf wird dann eine Zeile mit den Informationen des jeweiligen Layers dem *ListView*-Steuerelement hinzugefügt. Für diese Aufgabe legen wir uns eine gesonderte Prozedur (*AddToListView*) an, die dann nur noch aufgerufen werden muss.

```
For i = 1 To Layers.Count
    AddToListView Layers.Item(i)
Next
Else
```

Warum die Auslagerung des Hinzufügens einer Zeile in eine Prozedur der richtige Weg ist, zeigen die nächsten Codezeilen.

Hat der Benutzer einen untergeordneten Eintrag gewählt, muss zusätzlich kontrolliert werden, ob das Layer-Objekt der gewählten Gruppe zugeordnet wurde. Hierfür legen wir eine weitere Funktion im Klassenmodul *GroupLayer* an. *IsInGroup* wird dann entweder *TRUE* zurückgegeben, wenn der Layer der angegebenen Gruppe zugeordnet wurde, oder *FALSE*, wenn dies nicht der Fall ist.

```
For i = 1 To Layers.Count
    If Layers.Item(i).IsInGroup(xNode.FullPath) = True Then
```

Um den Layer darzustellen, rufen wir auch hier wieder die Anweisung *AddToListView* auf, wenn der Layer in der angegebenen Gruppe vorhanden ist.

```
        AddToListView Layers.Item(i)
    End If
Next
End If
End Sub
```

Werfen wir nun noch einen kurzen Blick auf die zuvor angesprochene Funktion, die kontrollieren soll, ob der Layer der angegebenen Gruppe zugeordnet wurde. Diese Funktion in der Klasse *GroupLayer* abzulegen, macht dahingehend Sinn, das zum einen die Übersichtlichkeit des Codes im Formular gefördert wird und zum anderen sofort auf den angesprochenen Layer Bezug genommen werden kann. Würden wir diese Prozedur beispielsweise im Klassenmodul *GroupLayers* ablegen, müsste zusätzlich immer erst der Layer ermittelt werden, auf den sich die Abfrage bezieht.

Der Code selbst ist nicht allzu kompliziert. Es muss lediglich die *Groups*-Auflistung durchlaufen werden, über die ja jedes Layer-Objekt verfügt, und bei jedem Durchlauf die Gruppe mit dem Argument *url* verglichen werden (in diesem Argument wurde ja die gesuchte Gruppe übergeben).

```
Public Function IsInGroup(url) As Boolean
On Error Resume Next
For i = 1 To Groups.Count
    If UCase(Groups(i)) = UCase(url) Then
```

Stimmen beide Angaben überein, wurde der Layer der Gruppe zugeordnet und dem Funktionsnamen kann vor dem Verlassen der Funktion der Wert *TRUE* zugewiesen werden.

```

        IsInGroup = True
    Exit For
End If
Next
End Function

```

Nachdem alle notwendigen Vorbereitungen für die Anzeige der Layer getroffen wurden, können wir uns nun der Aufgabe des Hinzufügens einer einzelnen Zeile zum *ListView*-Steuerelement zuwenden. Den Namen dieser Prozedur hatten wir ja bereits mit *AddToListView* und als Argument eine Variable vom Typ *GroupLayer* (definiert einen einzelnen Layer):

```

Public Function AddToListView(ActLayer As GroupLayer) As _
MSComctlLib.ListItem
Dim LV As MSComctlLib.ListItem
On Error Resume Next

```

Das Grundprinzip dieses Vorgangs wurde bereits in *Kapitel 3.3.2* erläutert. Als Erstes wird ein neuer Eintrag im *ListView*-Steuerelement erzeugt. Der Verweis auf diesen Eintrag wird in der Variablen *LV* gespeichert, die wir im weiteren Verlauf dieser Prozedur verwenden werden.

```

Set LV = ListView1.ListItems.Add(, , ActLayer.Name)
LV.ToolTipText = ActLayer.Name

```

Um den aktuellen Layer der Zeichnung gesondert darzustellen, wird dieser mit einem anderen Symbol dargestellt als alle anderen Layer.

```

LV.SmallIcon = IIf(ThisDrawing.ActiveLayer.Name = _
    ActLayer.Name, 2, 1)

```

Anschließend werden dann noch alle anderen Eigenschaften an die entsprechenden Spalten übertragen. Dazu wird für jede Spalte ein neues *ListSubItem*-Objekt für den gerade erzeugten Eintrag angelegt.

```

LV.ListSubItems.Add , , ActLayer.Description

```

Für die Spalten, die den Status Ein, Frieren, Sperren und Plotten darstellen, legen wir lediglich ein Symbol fest. Welches Symbol dargestellt wird, hängt dabei vom Wert der jeweiligen Eigenschaft ab. Ist der Layer beispielsweise eingeschaltet, wird das dritte Symbol des *LVImageList*-Steuerelements angezeigt. Ist er hingegen ausgeschaltet, das vierte Symbol.

```

LV.ListSubItems.Add , , IIf(ActLayer.LayerOn = True, 3, 4)
LV.ListSubItems.Add , , IIf(ActLayer.Freeze = True, 5, 6)
LV.ListSubItems.Add , , IIf(ActLayer.Locked = True, 7, 8)
LV.ListSubItems.Add , , IIf(ActLayer.Plottable = True, 9, 10)

```

Für die Spalte, in der die Farbe dargestellt werden soll, fügen wir zusätzlich zum Symbol noch den Namen der Farbe hinzu. Den Namen der Farbe ermittelt dabei die Funktion *GetColorName* der Klasse *MyUtility* (*Kapitel A.3*), die Sie zunächst dem Projekt hinzufügen müssen (DATEI→DATEI IMPORTIEREN). Anschließend deklarieren Sie eine globale Variable *Util*. Am besten fügen Sie hierfür dem Projekt ein Modul hinzu und fügen in diesem folgende Zeile hinzu.

```

Public Util As MyUtility

```

Anschließend müssen Sie in der Startprozedur nur noch eine neue Instanz der Klasse *MyUtility* erzeugen und in der Variablen *Util* speichern.

```
Set Util = New MyUtility
```

Nun aber weiter im Text. Wie gesagt, gibt die Funktion *GetColorName* den Namen der angegebenen Farbe zurück. Das Symbol zur Darstellung der Farbe ergibt sich ebenfalls aus der Farbnummer.

```
LV.ListSubItems.Add , , Util.GetColorName(ActLayer.Color), _  
    IIf(ActLayer.Color < 8, ActLayer.Color + 10, 18)
```

Der Linientyp und die Linienstärke werden lediglich als Text dargestellt. Den Linientyp können wir dabei direkt abrufen, der Text für die Linienstärke wird durch die Funktion *GetLineWeightText* zurückgegeben.

```
LV.ListSubItems.Add , , ActLayer.LineType  
LV.ListSubItems.Add , , Util.GetLineWeightText( _  
    ActLayer.LineWeight)
```

Der Plotstil-Name des Layers wird nur angezeigt, wenn die Systemvariable *PSTYLEMODE* auf den Wert 0 gesetzt ist.

```
If ThisDrawing.GetVariable("PSTYLEMODE") = 0 Then  
    LV.ListSubItems.Add , , ActLayer.PlotStyleName  
End If
```

Die Symbole zur Darstellung des Status der Ansichtsfenster im Papierbereich müssen nur dargestellt werden, wenn sich die Zeichnung im Papierbereich befindet.

```
If ThisDrawing.ActiveSpace = acPaperSpace Then  
    LV.ListSubItems.Add , "FVP", , IIf(ActLayer.ViewportFreeze _  
        = True, 19, 20)  
    LV.ListSubItems.Add , "FNVP", , IIf(ActLayer.ViewportDefault _  
        = True, 21, 22)  
End If
```

Abschließend müssen wir noch kontrollieren, ob der gerade verarbeitete Layer aus einer externen Referenz stammt. Ist dies der Fall, enthält der Layer-Name das Trennungszeichen | und wir stellen das Layer-Symbol und den Layer-Namen etwas abgeschwächt dar.

```
If InStr(1, ActLayer.Name, "|") > 0 Or ActLayer.Name = "0" Then  
    LV.Ghosted = True  
    If InStr(1, ActLayer.Name, "|") > 0 Then LV.ForeColor = _  
        &H808080  
End If  
Set AddToListView = LV  
End Function
```

Nach so viel Vorbereitung brauchen wir nun nur noch die Anweisung *ReadGroupLayers* unter Angabe des *Node*-Objekts aufzurufen, sobald das Ereignis *NodeClick* durch den Benutzer ausgelöst wurde.

```
Private Sub TreeView1_NodeClick(ByVal Node As MSComctlLib.Node)  
On Error Resume Next  
ReadGroupLayers Node  
End Sub
```

Layer zu einer Gruppe hinzufügen

Problemstellung

Nachdem die Anzeige der Layer-Informationen im *ListView*-Steuerelement funktionieren sollte, müssen wir dem Benutzer natürlich auch die Möglichkeit geben, die Layer einer beliebigen Gruppe zuzuordnen zu können, da sonst das Ganze ja keinen Sinn machen würde. Stellt sich uns das Problem: Wie kann die Zuordnung durch den Benutzer umgesetzt werden, so dass sich diese für den Benutzer so einfach wie möglich gestaltet?

Lösungsansatz

Als Ansatz für die Lösung dieses Problems könnte die *DragAndDrop*-Methode dienen, so wie Sie es vom Windows-Explorer kennen. Der Benutzer sollte also die Möglichkeit erhalten, verschiedene Layer im *ListView*-Steuerelement auszuwählen und diese dann mithilfe des Mauszeigers auf eine bestimmte Gruppe im *TreeView*-Steuerelement zu bewegen. Sobald der Benutzer die Maustaste über einem *Node*-Objekt loslässt, soll durch das Programm ein Bezug auf die gewählte Gruppe hergestellt werden.

Problemlösung

Bevor die *DragAndDrop*-Methode durchgeführt werden kann, müssen wir noch einige Einstellungen am *TreeView*- und am *ListView*-Steuerelement vornehmen. Wenn Sie sich die Beschreibung der Eigenschaften dieser Steuerelemente in Kapitel 3.3.2 und 3.3.8 einmal anschauen, werden Sie dort zwei Eigenschaften finden, die das Verhalten der Ziehen- und Ablegenvorgänge steuern. Da das *ListView*-Steuerelement den Ausgangspunkt dieses Vorgangs bildet, müssen wir die Eigenschaft *OleDragMode* auf *ccOLEDragAutomatic* einstellen, damit der Ziehvorgang automatisch durch das Steuerelement ausgelöst wird. Beim *TreeView*-Steuerelement ist das anders. Da dieses Steuerelement als Ziel des Ziehvorgangs dienen wird, muss die *OleDropMode* auf *ccOLEDropManual* umgeschaltet werden. Durch diese Einstellung wird dann das *OLEDragDrop*-Ereignis ausgelöst, wenn der Benutzer die gewählten Layer auf dem *Node*-Objekt ablegt.

Der erste Schritt, der zunächst erledigt werden muss, damit der Vorgang erfolgreich abgeschlossen werden kann, ist die Markierung des *Node*-Objekts, über dem sich der Mauszeiger während des Ziehvorgangs momentan befindet. Hierfür verwenden wir das *OLEDragOver*-Ereignis. Dieses wird bei jeder Mausbewegung, die während des Ziehvorgangs über dem Objekt ausgeführt wird, ausgelöst.

```
Private Sub TreeView1_OLEDragOver(Data As MSComctlLib.DataObject _
, Effect As Long, Button As Integer, Shift As Integer, X As _
Single, Y As Single, State As Integer)
On Error Resume Next
```

Mit der *HitTest*-Funktion können wir das *Node*-Objekt ermitteln, über dem sich der Mauszeiger momentan befindet. Den Verweis auf diesen Knoten übergeben wir der Eigenschaft *DropHighlight*. Diese zeigt dem Benutzer das entsprechende *Node*-Objekt auch rein optisch an.

```
Set TreeView1.DropHighlight = TreeView1.HitTest(X * 20, Y * 20)
```

Gleichzeitig muss mit der *Expanded*-Eigenschaft geprüft werden, ob der Knoten geöffnet ist. Ist dies nicht der Fall, wird dies nachgeholt.

```

If TreeView1.DropHighlight.Expanded = False Then
    TreeView1.DropHighlight.Expanded = True
End If
End Sub

```

Wie bereits erwähnt, ist *OLEDragDrop* die Ereignisprozedur, die durch das Ablegen auf dem *TreeView*-Steuerelement ausgelöst wird. In dieser Ereignisprozedur starten wir die Zuordnung der im *ListView*-Steuerelement ausgewählten Layer. Auch hier empfiehlt sich die Verwendung einer separaten Prozedur, die diese Funktion übernimmt. Nennen wir diese Prozedur einfach *CopyLayersToGroup*. Als Parameter werden wir das *Node*-Objekt übergeben, dem die gewählten Layer zugeordnet werden sollen.

```

Private Sub TreeView1_OLEDragDrop(Data As MSComctlLib.DataObject _
, Effect As Long, Button As Integer, Shift As Integer, X As _
Single, Y As Single)
On Error Resume Next
CopyLayersToGroup TreeView1.DropHighlight
Set TreeView1.DropHighlight = Nothing
End Sub

```

Schauen wir uns nun die Funktion, die die Layer zu den Gruppen hinzufügt, im Detail an. Eigentlich ist es ja eher umgekehrt, die Gruppen werden zu den Layer-Objekten hinzugefügt. Die Gruppe, um die es geht, wird mit dem Argument der Prozedur übergeben.

```

Public Sub CopyLayersToGroup(xNode As MSComctlLib.Node)
Dim ActLayer As GroupLayer, Found As Boolean
On Error Resume Next

```

Handelt es sich um den obersten Eintrag („Global“), kann die Prozedur vorzeitig verlassen werden, da hier ja sowieso alle Layer angezeigt werden und wir diesen Eintrag den Layern somit nicht explizit zuweisen müssen.

```

If TypeName(xNode.Parent) = "Nothing" Then Exit Sub

```

Jetzt durchlaufen wir die Einträge des *ListView*-Steuerelements und kontrollieren, ob dieser selektiert wurde.

```

For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then

```

Ist dies der Fall, wird zunächst der Verweis auf das Layer-Objekt ermittelt.

```

        Set ActLayer = Layers.ItemByName(ListView1. _
        ListItems(i).Text)
        If TypeName(ActLayer) <> "Nothing" Then
            Found = True

```

Anschließend wird in einer weiteren Schleife kontrolliert, ob die Gruppe bereits dem Layer-Objekt zugeordnet wurde.

```

        For X = 1 To ActLayer.Groups.Count
            If UCase(ActLayer.Groups(X)) = UCase(xNode.FullPath) _
            Then
                Found = True
                Exit For
            End If
        Next

```

Wurde die Gruppe nicht gefunden und somit auch noch nicht zugeordnet, kann dies im nächsten Schritt erfolgen. Dafür müssen wir nur die *Add*-Anweisung der *Groups*-Eigenschaft ausführen und dieser Anweisung die Gruppe mitgeben.

```

        If Found = False Then
            ActLayer.Groups.Add xNode.FullPath
        End If
    End If
End If
Next
End Sub

```

So, damit wäre der zweite Teil der Entwicklungsarbeit des Projekts abgeschlossen. Sie finden den kompletten Vorgang, der bis hierher abgearbeitet wurde, in der Projektdatei *Teil2.dvb*.

4.4 Layer-Eigenschaften anpassen

Problemstellung

An dieser Stelle fahren wir mit den Details zur Verarbeitung der einzelnen Layer fort. Dazu gehört natürlich, dem Benutzer die Möglichkeit zu geben, die Eigenschaften eines einzelnen oder mehrerer Layer im Formular anpassen zu können. Stellt sich die Frage: Wie soll dies bewerkstelligt werden?

Lösungsansatz

Wenn wir uns am *Kapitel 7.4.2* orientieren, das die Anpassung der Layer-Eigenschaften beschreibt, und dementsprechend das Formular erweitern, ergibt sich für die Darstellung der einzelnen Eigenschaften die Verwendung ganz bestimmter Steuerelemente. Für die Eigenschaften, die nur die Werte *TRUE* oder *FALSE* aufnehmen können, empfiehlt sich die Verwendung von Checkboxen (*Kapitel 3.2.8*). Für die Text Eigenschaften wie Layer-Name, Beschreibung und Plotstil sollten einfache Textfelder (*Kapitel 3.2.5*) ausreichen. Bleiben noch die Farbe, die Linienstärke und der Linientyp. Für den Linientyp könnten wir ein aufklappbares Listefeld aus *Kapitel 3.2.7* verwenden. Da es sich bei der Linienstärke um eine vorgegebene Anzahl von Einträgen und dafür vorgegebene Stärken handelt, was wiederum in *Kapitel 8.3.1* näher erläutert ist, lohnt sich hier die Verwendung eines Kombinationsfeld-Steuerelements aus *Kapitel 3.3.6*. Mit diesem können neben den Bezeichnungen auch die Linienstärken dargestellt werden. Das gleiche Prinzip ließe sich auch auf die Farbauswahlliste anwenden. Um Ihnen auch noch die Verwendung eines Zusatzsteuerelements und damit die schier unbegrenzten Möglichkeiten von VBA zu demonstrieren, werden wir für die Farbauswahl ein Steuerelement verwenden, das in diesem Buch nicht näher beschrieben ist. Auf der Buch-CD sind eine ganze Reihe dieser zusätzlichen Bibliotheken gespeichert und Sie würden im Internet weitere Bibliotheken finden, so dass für fast jede Aufgabe bereits eine Lösung verfügbar ist. Für die Farbauswahl verwenden wir das AutoCAD Color OLE Control von Cyrille Fauvel. Für Mitglieder im ADN (Autodesk Developer Network) handelt es sich um ein Freeware-Tool, für alle anderen wurde es als Shareware-Version konzipiert. Um dieses Steuerelement verwenden zu können, muss es zunächst in die VBA-Umgebung eingebunden werden (*Kapitel 3.1.1*).

Problemlösung

Nachdem wir nun alle Steuerelemente für die Layer-Eigenschaften zusammengestellt haben, müssen diese natürlich in das Formular eingefügt werden, so dass sich folgende Darstellung für das ergänzte Formular ergibt.

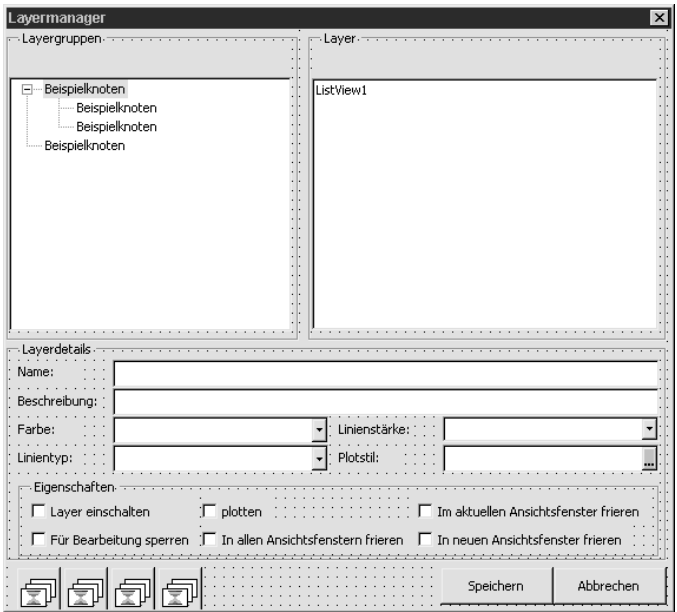


Abbildung 4.4: Erweitertes Layermanager-Formular

Um auch einen namentlichen Bezug zwischen Layer-Eigenschaft und Steuerelement herzustellen, werden die einzelnen Steuerelemente noch mit neuen Namen versehen.

Layer-Eigenschaft	Steuerelementname	Layer-Eigenschaft	Steuerelementname
Name	LNameText	LayerOn	LayerOn
Description	DescText	Lock	Locked
Color	ColorCombo	Plottable	Plottable
Linetype	LTypeCombo	Freeze	Freeze
Lineweight	LWCombo	ViewportFreeze	FreezeVP
PlotStilName	PlotStyleText	ViewportDefault	FreezeNewVP

Tabelle 4.2: Namen der Steuerelemente zur Darstellung der Layer-Eigenschaften

Bevor die Integration der Steuerelemente in den Programmcode erfolgen kann, müssen noch die Steuerelemente zur Auswahl des Linientyps und der Linienstärke mit Auswahlwerten gefüllt werden. Hierfür ergänzen wir die *Initialize*-Prozedur des Formulars – zunächst mit dem Hinzufügen der Linientypen der aktuellen Zeichnung. Auch hierfür können wir auf einen Abschnitt des Buchs zurückgreifen. Wenn Sie sich an die Beschreibung dieses Themas in *Kapitel 7.5* halten und zusätzlich den *Abschnitt 3.2.7* zu Rate ziehen, ergibt sich folgender Code.


```

LTypeCombo.Clear
For Each LType In ThisDrawing.Linetypes
    If UCase(LType.Name) <> "BYLAYER" And UCase(LType.Name) <> _
        "BYBLOCK" And InStr(1, LType.Name, "|") = 0 Then
        LTypeCombo.AddItem LType.Name
    End If
Next
LTypeCombo.Text = ThisDrawing.ActiveLinetype.Name

```

Bei den Linienstärken sieht dies schon etwas anders aus. Zunächst einmal benötigen wir die verschiedenen symbolhaften Darstellungen der Linienstärken, die in einem *ImageList*-Steuerelement (*Kapitel 3.3.1*) zwischengelagert werden. Die entsprechenden Symbole finden Sie wiederum im Verzeichnis *04/Images*.

In der *Initialize*-Prozedur des Formulars erfolgt dann die normale Zuordnung der einzelnen Linienstärken, so wie es das *Kapitel 3.3.6* beschreibt. Die jeweiligen Symbole werden dabei von der *GetLWImage*-Funktion ermittelt und der Text der jeweiligen Linienstärke durch die Funktion *GetLineWeightText* der Klasse *MyUtility* (*Kapitel A.3*), die ja bereits in das Projekt integriert wurde.

```

Set LWCombo.ImageList = LWeightImageList
LWCombo.ComboItems.Clear
LWCombo.ComboItems.Add , , "Vorgabe", _
    GetLWImage(ThisDrawing.GetVariable("LWDEFAULT")), _
    GetLWImage(ThisDrawing.GetVariable("LWDEFAULT"))
For i = 4 To 26
    LWCombo.ComboItems.Add , , _
        Util.GetLineWeightText(Util.GetLineweightValue(i)), _
        GetLWImage(Util.GetLineweightValue(i)), _
        GetLWImage(Util.GetLineweightValue(i))
Next
LWCombo.SelectedItem = LWCombo.ComboItems(1)

```

Das Farbauswahl-Steuerelement muss in diesem Beispiel zunächst nicht weiter initialisiert werden, bis auf die *AllowMetaColor*-Eigenschaft ($\rightarrow$ *FALSE*), was aber bereits zur Entwurfszeit erfolgen kann. Diese Eigenschaft schaltet die Auswahl der Farbe *vonLayer* und *vonBlock* ab, was durchaus notwendig ist, wenn Sie sich die Anmerkungen des *Kapitels 7.4.2* durchlesen.

So gerüstet kommen wir nun zur Übergabe der Layer-Eigenschaften an die einzelnen Steuerelemente. Da die einzelnen Layer durch den Benutzer im *ListView*-Steuerelement ausgewählt werden, können wir laut *Kapitel 3.3.2* auf das *ItemClick*-Ereignis zurückgreifen.

```

Private Sub ListView1_ItemClick(ByVal Item As _
MSComctlLib.ListItem)
Dim ActLayer As GroupLayer
On Local Error Resume Next
NoChange = True

```

Zunächst muss hier unterschieden werden, ob der Benutzer einen oder mehrere Layer ausgewählt hat. Da diese Information nicht direkt vom *ListView*-Steuerelement zu erhalten ist, müssen wir selbst die Initiative ergreifen und eine Funktion schreiben, die diese Aufgabe übernimmt.

```

Function LVSelectedCount() As Long
On Error Resume Next
For i = 1 To ListView1.ListItems.Count

```

```

    If ListView1.ListItems(i).Selected = True Then
        LVSelectedCount = LVSelectedCount + 1
    End If
Next
End Function

```

Jetzt können wir den Code in der *ItemClick*-Prozedur fortsetzen.

```

If LVSelectedCount > 1 Then

```

Wurden also mehrere Einträge gewählt, müssen wir das Textfeld zur Anzeige und Änderung des Layer-Namens unzugänglich für den Benutzer machen. Außerdem können die *Value*-Eigenschaften der Checkboxes auf den Wert *Null* gesetzt werden. Die Gründe dafür können Sie dem *Kapitel 3.2.8* entnehmen.

```

    LNameText.Enabled = False
    PlotStyleText = ActLayer.PlotStyleName
    LayerOn.Value = Null
    Freeze.Value = Null
    Locked.Value = Null
    Plottable.Value = Null
    FreezeVP.Value = Null
    FreezeNewVP.Value = Null
    GrpBox.Clear
Else

```

Wurde hingegen nur ein einzelner Layer ausgewählt, können wir die Werte der Layer-Eigenschaften direkt zuordnen. Außerdem lässt sich das Textfeld für den Layer-Namen für den Benutzer wieder aktivieren, damit er diesen ändern kann.

```

    LNameText.Enabled = True
    If ListView1.SelectedItem.Ghosted = True Then

```

Es sei denn, es handelt sich um einen Layer aus einer externen Referenz. In diesem Fall wird die *Locked*-Eigenschaft aktiviert, damit der Benutzer den Text nicht ändern kann.

```

        LNameText.Locked = True
    Else
        LNameText.Locked = False
    End If
    Set ActLayer = Layers.ItemByName(Item.Text)
    LNameText = ActLayer.Name
    DescText = ActLayer.Description
    ColorCombo.SetSelColorIndex ActLayer.Color
    If ColorCombo.GetSelColorIndex = -1 Then
        ColorCombo.AddColorIndex ActLayer.Color
        ColorCombo.SetSelColorIndex ActLayer.Color
    End If
    LTypeCombo.Text = ActLayer.LineType
    Set LWCombo.SelectedItem = LWCombo.ComboItems( _
        Util.GetLineweightIndex(ActLayer.LineWeight))
    PlotStyleText = ActLayer.PlotStyleName
    LayerOn.Value = ActLayer.LayerOn
    Freeze.Value = ActLayer.Freeze
    Locked.Value = ActLayer.Locked

```

```

    Plottable.Value = ActLayer.Plottable
    FreezeVP.Value = ActLayer.ViewportFreeze
    FreezeNewVP.Value = ActLayer.ViewportDefault
End If
NoChange = False
End Sub

```

Nachdem nun die Übergabe der Eigenschaften an die Steuerelemente abgeschlossen ist, bleibt nur noch das Zurückschreiben einer Änderung in die entsprechende Eigenschaft des oder der ausgewählten *ListView*-Elemente. Da dieser Vorgang bei allen Steuerelementen gleich bzw. ähnlich abläuft, sollte stellvertretend für alle Steuerelemente die Beschreibung zweier Eigenschaften ausreichend sein. Auf welches Ereignis Sie im Speziellen reagieren müssten, erklären die Kapitel der jeweiligen Steuerelemente. Im Prinzip wird aber bei jeder Eigenschaft der gleiche Vorgang ausgeführt. In einer Schleife werden alle Elemente des *ListView*-Steuerelements durchlaufen und geprüft, ob der jeweilige Eintrag durch den Benutzer ausgewählt wurde. Ist dies der Fall, wird das Layer-Objekt der Klasse *GroupLayers* aus dem Namen des *ListView*-Elements ermittelt und die entsprechende Eigenschaft angepasst. Anschließend wird dann zusätzlich die Darstellung der Eigenschaft im *ListView*-Steuerelement geändert, so dass der *ListView*-Eintrag wieder mit dem Wert der Eigenschaft übereinstimmt.

Änderung der Beschreibung

```

Private Sub DescText_AfterUpdate()
Dim ActLayer As GroupLayer
On Error Resume Next
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        Set ActLayer = Layers.ItemByName( _
            ListView1.ListItems(i).Text)
        ActLayer.Description = DescText
        ListView1.ListItems(i).ListSubItems(1).Text = DescText
    End If
Next
End Sub

```

Änderung der Freeze-Eigenschaft

```

Private Sub Freeze_Click()
Dim ActLayer As GroupLayer
On Error Resume Next
If NoChange = True Then Exit Sub
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        Set ActLayer = Layers.ItemByName( _
            ListView1.ListItems(i).Text)
        ActLayer.Freeze = Freeze.Value
        ListView1.ListItems(i).ListSubItems(3).ReportIcon = _
            IIf(ActLayer.Freeze = True, 5, 6)
    End If
Next
End Sub

```

So weit bis hierher. Die Entwicklung der Anwendung bis zu diesem Punkt ist in der Projektdatei *Teil3.dvb* gespeichert.

4.5 Layer zurückspeichern

Problemstellung

Bevor wir die Arbeit an dem Projekt mit dessen Verfeinerung fortsetzen, muss noch ein Punkt abgehandelt werden. Hierbei handelt es sich um die Speicherung oder auch Rückübertragung der Layer-Eigenschaften. Da diese Eigenschaften beim Start des Programms an eine Instanz des Klassenmoduls *GroupLayers* von der tatsächlichen *Layers*-Auflistung temporär übertragen wurden, würden die Änderungen, die der Benutzer an den Layer-Eigenschaften vorgenommen hat, mit Beendigung des Programms verworfen werden.

Lösungsansatz

Für die Lösung dieses Problems empfiehlt sich die Verwendung einer Prozedur, die dem Klassenmodul *GroupsLayers* hinzugefügt werden sollte. In dieser Prozedur werden alle Speichervorgänge, angefangen von der Übergabe der Layer-Eigenschaft, dem Anlegen und Entfernen von Layern bis hin zur Speicherung der Gruppenzuordnung, erfolgen.

Problemlösung

Nachdem der Weg festgelegt wurde, werden wir diesen auch beschreiten. Fügen Sie also dem Klassenmodul *GroupLayers* eine Prozedur mit dem Namen *SaveLayers* hinzu. In dieser Prozedur erstellen wir den Code zur Speicherung der Layer-Informationen.

```
Public Sub SaveLayers()  
    Dim ActLayer As AcadLayer  
    Dim GrpLayer As GroupLayer  
    Dim xData, xType, VPLayers  
    On Local Error Resume Next
```

Beginnen wir zunächst mit den Layern, die der Benutzer der Auflistung hinzugefügt hat. Da es sich hier um Layer handelt, die der Benutzer der aktuellen Zeichnung hinzufügen möchte, benötigen wir hierfür eine Funktion, die dies für uns erledigt. Über die Vorgehensweise gibt wieder das *Kapitel 7.4.1* Auskunft. Entscheidend dafür, welcher Layer neu ist und damit der *Layers*-Auflistung hinzugefügt werden muss, ist die Eigenschaft *OriginalName*. Enthält diese eine Leerzeichenfolge, handelt es sich um einen neuen Layer.

```
For i = 1 To mCol.Count  
    Set GrpLayer = mCol(i)  
    If GrpLayer.OriginalName = "" Then  
        ThisDrawing.Layers.Add GrpLayer.Name  
        GrpLayer.OriginalName = GrpLayer.Name  
    End If  
Next
```

Jetzt müssen noch die Eigenschaften der bereits vorhandenen Layer angepasst werden, zu denen ja jetzt auch die durch den Benutzer erzeugten Layer zählen. Außerdem müssen die Layer, die nicht mehr in Auflistung vorhanden sind auch aus der Zeichnung gelöscht werden.

```
For Each ActLayer In ThisDrawing.Layers  
    Set GrpLayer = ItemByLayer(ActLayer.Name)  
    If TypeName(GrpLayer) = "Nothing" Then  
        ActLayer.Delete  
    Else  
        ReDim xType(1) As Integer
```

```

ReDim xData(1) As Variant
xType(0) = 1001: xData(0) = "Description"
xType(1) = 1000: xData(1) = GrpLayer.Description
ActLayer.Name = GrpLayer.Name
ActLayer.SetXData xType, xData
ActLayer.Color = GrpLayer.Color
ActLayer.Freeze = GrpLayer.Freeze
ActLayer.LayerOn = GrpLayer.LayerOn
ActLayer.LineType = GrpLayer.LineType
ActLayer.LineWeight = GrpLayer.LineWeight
ActLayer.Lock = GrpLayer.Locked
ActLayer.PlotStyleName = GrpLayer.PlotStyleName
ActLayer.Plottable = GrpLayer.Plottable
ActLayer.ViewportDefault = GrpLayer.ViewportDefault

```

Für die Speicherung der Eigenschaft „Frieren“ im aktuellen Ansichtsfenster müssen wir uns etwas Besonderes einfallen lassen, da, wie bereits angemerkt, das *Layer*-Objekt keine eigene Eigenschaft besitzt. Sammeln wir zunächst die Layer-Namen in dem Array *VPLayers*. Wie das funktioniert, können Sie in *Kapitel 2.1.7* nachlesen.

```

If GrpLayer.ViewportFreeze = True Then
  If IsEmpty(VPLayers) = True Then
    ReDim VPLayers(0) As String
  Else
    ReDim Preserve VPLayers(UBound(VPLayers) + 1) _
      As String
  End If
  VPLayers(UBound(VPLayers)) = ActLayer.Name
End If

```

Abschließend werden dann auch noch die Layer-Gruppen, denen die einzelnen Layer zugeordnet wurden, in den erweiterten Daten des *Layer*-Objekts gespeichert (*Kapitel 7.1.2*)

```

ActLayer.GetXData "Groups", xTypeOut, xDataOut
ReDim xType(GrpLayer.Groups.Count) As Integer
ReDim xData(GrpLayer.Groups.Count) As Variant
xType(0) = 1001: xData(0) = "LAYERGRUPPEN"
For i = 1 To GrpLayer.Groups.Count
  xType(i) = 1000: xData(i) = GrpLayer.Groups(i)
Next
ActLayer.SetXData xType, xData
End If
Next

```

Nachdem alle Layer ausgewertet wurden, befassen wir uns nun noch mit dem Setzen der Eigenschaft „Frieren“ im aktuellen Ansichtsfenster. Da dies nur erforderlich ist, wenn die Zeichnung sich im Papierbereich befindet, wird dieser Status zunächst geprüft (*Kapitel 6.2.2*). Sollte sich die Zeichnung im Papierbereich befinden, wird das Array, in dem die Namen der zu frierenden Layer gesammelt wurden, an die Funktion *FreezeActiveVPLayers* übergeben. In dieser Prozedur wird dann das Frieren der Layer abgearbeitet.

```

If ThisDrawing.ActiveSpace = acPaperSpace Then
  FreezeActiveVPLayers VPLayers
End If
End Sub

```

Das Frieren der Layer im Ansichtsfenster ist nicht ganz so einfach umzusetzen, wie das Auslesen dieses Status. Prinzipiell werden die Layer wieder in einem Array zusammengefasst und in die erweiterten Daten des Ansichtsfensters zurückgeschrieben. Allerdings enthalten die erweiterten Daten des Ansichtsfensters außer den Layer-Namen noch weitere Informationen, so dass wir zunächst auf das ursprüngliche Array zurückgreifen müssen.

```
Function FreezeActiveVPLayers(LayerArray)
Dim xTypeOut, xDataOut
Dim objPViewport As AcadPViewport
On Error Resume Next
```

Wichtig beim Ansichtsfenster ist, alle Layer erst einmal zu tauen, bevor die Daten ausgelesen und angepasst werden. Da hierfür keine geeignete Prozedur oder Funktion existiert, bleibt nichts anderes übrig, als auf den AutoCAD-Befehl `aflayer` zurückzugreifen. Dieser wird mit der *SendCommand*-Anweisung (Kapitel 6.2.3) ausgeführt und die Eingaben zum Tauen aller Layer im aktuellen Ansichtsfenster, die dieser Befehl in der AutoCAD-Befehlszeile erwartet, werden an dieser Stelle gleich mitgeliefert.

```
ThisDrawing.SendCommand "aflayer t *" & vbCrLf & vbCrLf & vbCrLf
```

Anschließend werden die erweiterten Daten ausgelesen und die Stelle ermittelt, an der wir mit den Layer-Namen ansetzen müssen.

```
Set objPViewport = ThisDrawing.ActivePViewport
objPViewport.GetXData "ACAD", xTypeOut, xDataOut
For i = LBound(xTypeOut) To UBound(xTypeOut)
    If (xTypeOut(i) = 1002 And xTypeOut(i + 1) = 1003) Or _
        (xTypeOut(i) = 1002 And xTypeOut(i + 1) = 1002) Then
        Exit For
    End If
Next
```

Jetzt dimensionieren wir das Array neu und zwar in der ursprünglichen Größe zuzüglich der Anzahl der Layer, die gefroren werden sollen. Die Layer werden an die entsprechenden Felder des Arrays übergeben und mit der *SetXData*-Anweisung zurückgeschrieben.

```
If IsEmpty(LayerArray) = True Then
    ReDim Preserve xTypeOut(i + 2) As Integer
    ReDim Preserve xDataOut(i + 2) As Variant
Else
    ReDim Preserve xTypeOut(i + UBound(LayerArray) + 3) As Integer
    ReDim Preserve xDataOut(i + UBound(LayerArray) + 3) As Variant
    X = 0
    For i = i + 1 To UBound(xTypeOut) - 2
        xTypeOut(i) = 1003
        xDataOut(i) = LayerArray(X)
        X = X + 1
    Next
End If
xTypeOut(UBound(xTypeOut) - 1) = 1002
xDataOut(UBound(xTypeOut) - 1) = ""
xTypeOut(UBound(xTypeOut)) = 1002
xDataOut(UBound(xTypeOut)) = ""
objPViewport.SetXData xTypeOut, xDataOut
```

Jetzt folgen noch einige verwirrende Operationen, um die Änderungen auch am Bildschirm sichtbar zu machen. Das Grundprinzip ist, das aktuelle Ansichtsfenster zu deaktivieren, es dann zu aktualisieren und danach wieder sichtbar zu machen.

```
OldMSpace = ThisDrawing.MSpace
ThisDrawing.MSpace = False
objPViewport.Display False
ThisDrawing.ActivePViewport.Update
objPViewport.Display True
ThisDrawing.MSpace = OldMSpace
End Function
```

Alternativ zu dieser Vorgehensweise können Sie aber auch den AutoCAD-Befehl `aflayer` verwenden und das Frieren der Ansichtsfenster über die *SendCommand*-Anweisung ausführen. Allerdings ist diese Variante etwas langsamer und auch nicht so elegant, führt aber zum selben Ergebnis.

Damit wäre das nächste Teilstück des Projekts abgeschlossen. Sie finden alles, was bis zu diesem Punkt abgehandelt wurde, in der Projektdatei *Teil4.dvb*.

4.6 Optimierung des Projekts

Problemstellung

Nachdem das Projekt in den grundlegenden Strukturen funktioniert, lässt es sich noch in seiner Funktionalität und Darstellung optimieren und verfeinern. Die erste Aufgabe ist in diesem Zusammenhang das Ändern des Layer-Status einer gesamten Gruppe. Stellen Sie sich vor, der Benutzer wählt eine Gruppe und möchte nun alle Layer, die dieser Gruppe untergeordnet sind (einschließlich der Layer, die in untergeordneten Gruppen zusammengefasst wurden), ausschalten oder frieren usw.

Lösungsansatz

Am einfachsten geben Sie dem Benutzer Möglichkeiten für das Auswählen einer bestimmten Einstellung eines Status, indem Sie für jede Eigenschaft eine eigene Schaltfläche im *Toolbar*-Steuerelement anlegen. Beim Umschalten der Schaltfläche wird dann der jeweilige Status auf die Layer übertragen, die der ausgewählten Gruppe zugeordnet wurden.

Problemlösung

Am besten fügen Sie die besagten Schaltflächen dem *Toolbar*-Steuerelement hinzu, das Sie bereits im Rahmen der Layer-Gruppen positioniert haben. Zusätzlich zu den Schaltflächen, die den jeweiligen Status der Layer darstellen, sollte eine Trennlinie die neuen Schaltflächen von den bereits vorhandenen optisch abtrennen. Fügen Sie also die besagten Elemente dem *Toolbar*-Steuerelement hinzu, so dass sich bei der Ausführung eine Darstellung wie in Abbildung 4.5 ergibt. Die Symbole, die Sie dem *TVImageList*-Steuerelement hinzufügen müssen, finden Sie wie gehabt auf der Buch-CD im Verzeichnis *04/images*.

Damit die Schaltflächen die entsprechende Darstellung des jeweiligen Status anzeigen, müssen Sie als Schaltflächentyp *tbrCheck* festlegen. So wird garantiert, dass die Schaltflächen entweder eingedrückt oder nicht gedrückt dargestellt werden.

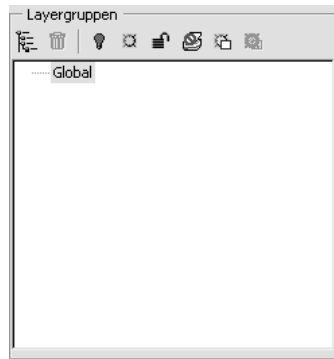


Abbildung 4.5: Erweitertes Toolbar-Steuerelement

Nachdem die Vorbereitungen der Schaltfläche abgeschlossen sind, wenden wir uns nun dem Code zu, der den Status der Layer-Gruppen ändert. Da der Vorgang an sich immer derselbe ist, also Gruppe wählen und Status ändern, kann dieser in einer Prozedur zusammengefasst werden. Welcher Status jeweils geändert werden soll, werden wir im ersten Argument der Prozedur übergeben.

Die Prozedur selbst fügen Sie dem Klassenmodul *GroupLayers* hinzu, da sie sich ja auf die Layer dieser Auflistung bezieht. Sie könnten die Prozedur natürlich auch im Formularcode oder in einem Modul entwerfen. Meiner Meinung nach ist sie aber aus programmtechnischer Sicht im Klassenmodul *GroupLayers* besser aufgehoben.

Um das Ganze etwas professioneller zu gestalten, fassen wir die möglichen Übergabewerte in einer Konstantenauflistung zusammen.

```
Enum LMLayerState
    LMLayerOn = 1
    LMFreeze = 2
    LMLocked = 3
    LMPlottable = 4
    LMViewportDefault = 5
    LMViewportFreeze = 6
End Enum
```

Diese Konstantenauflistung legen wir als Datentyp des ersten Arguments der Prozedur fest. Neben dem Statustyp benötigen wir allerdings noch weitere Informationen, um die Eigenschaften der Layer anpassen zu können. Also legen wir als zweites Argument die Variable *url* fest, die die Gruppe aufnehmen wird, deren Layer angepasst werden sollen, und die Variable *Value*. Diese Variable erhält den jeweiligen Statuswert. Da alle möglichen Statustypen dem Datentyp *Boolean* entsprechen, können wir diesen Datentyp auch für die Variable *Value* verwenden.

```
Public Sub SetLayerGroupState(LMState As LMLayerState, url, _
    Value As Boolean)
    Dim ActLayer As GroupLayer
    On Error Resume Next
```


Die Anpassung der Layer ist wieder eine recht einfache Angelegenheit. Über eine *For Next*-Schleife wird die *mCol*-Auflistung durchlaufen und bei jedem Durchlauf der Verweis auf das jeweilige Layer-Objekt in der Variablen *ActLayer* gespeichert.

```
For i = 1 To mCol.Count
    Set ActLayer = mCol(i)
```

Jetzt gilt es zu unterscheiden, welche Gruppe ausgewählt wurde. Entweder handelt es sich um die Gruppe, in der alle Layer angezeigt werden. Dann kann der jeweilige Status in jedem Fall angepasst werden.

```
    If url = "Global" Then
        Select Case LMState
            Case LMLayerOn
                ActLayer.LayerOn = Value
            Case LMFreeze
                ActLayer.Freeze = Value
            Case LMLocked
                ActLayer.Locked = Value
            Case LMPlottable
                ActLayer.Plottable = Value
            Case LMViewportDefault
                ActLayer.ViewportDefault = Value
            Case LMViewportFreeze
                ActLayer.ViewportFreeze = Value
        End Select
    Else
```

Oder es handelt sich um eine dem Eintrag „Global“ untergeordnete Gruppe. In diesem Fall muss zusätzlich die Auflistung der Gruppen durchlaufen werden. Jede zugeordnete Gruppe wird dann darauf überprüft, ob es sich um die im Parameter *url* angegebene Gruppe oder eine dieser untergeordneten Gruppe handelt. Trifft einer der beiden Fälle zu, kann die jeweilige Eigenschaft des Layers angepasst werden.

```
        For X = 1 To ActLayer.Groups.Count
            If UCase(ActLayer.Groups(X)) = UCase(url) Or url = _
                "Global" Or IsChild(ActLayer.Groups(X), url) = True _
            Then
                Select Case LMState
                    Case LMLayerOn
                        ActLayer.LayerOn = Value
                    Case LMFreeze
                        ActLayer.Freeze = Value
                    Case LMLocked
                        ActLayer.Locked = Value
                    Case LMPlottable
                        ActLayer.Plottable = Value
                    Case LMViewportDefault
                        ActLayer.ViewportDefault = Value
                    Case LMViewportFreeze
                        ActLayer.ViewportFreeze = Value
                End Select
            End If
        Next
    End If
```

```

Next
End Sub
Function IsChild(FirstUrl, SecondUrl) As Boolean
On Error Resume Next
If Len(FirstUrl) > Len(SecondUrl) Then
    If UCase(Left(FirstUrl, Len(SecondUrl) + 1)) = UCase( _
        SecondUrl & "\") Then
        IsChild = True
    End If
End If
End Function

```

Nachdem wir nun das Klassenmodul ergänzt haben, fällt die Änderung des jeweiligen Status nicht mehr schwer. Sie müssen lediglich die *ButtonClick*-Ereignisprozedur des *TVToolbar*-Steuerelements ergänzen. Als Beispiel habe ich einmal den Code für zwei Schaltflächen dargestellt. Daraus ergibt sich dann der Code für die restlichen Schaltflächen.

```

Case "LayerOn"
    If Button.Value = tbrPressed Then
        Button.Image = 3
        Layers.SetLayerGroupState LMLayerOn, _
            TreeView1.SelectedItem.FullPath, True
    Else
        Button.Image = 4
        Layers.SetLayerGroupState LMLayerOn, _
            TreeView1.SelectedItem.FullPath, False
    End If
Case "Freeze"
    If Button.Value = tbrPressed Then
        Button.Image = 6
        Layers.SetLayerGroupState LMFreeze, _
            TreeView1.SelectedItem.FullPath, True
    Else
        Button.Image = 5
        Layers.SetLayerGroupState LMFreeze, _
            TreeView1.SelectedItem.FullPath, False
    End If

```

Problemstellung

Als Letztes soll der Benutzer noch die Möglichkeit erhalten, Gruppen umzubenennen. Auch hier sollte der Augenmerk darauf gerichtet sein, dass der Benutzer nicht allzu viele Aktionen durchführen muss, um zu einem Ergebnis zu gelangen. In jedem Fall müssen wir unbedingt darauf achten, dass der Gruppenname, den der Benutzer neu vergibt, noch nicht in derselben Gruppe existiert.

Lösungsansatz

Hierfür bieten sich generell zwei Lösungsmöglichkeiten an. Zum einen könnte der Benutzer eine Gruppe markieren und anschließend eine Schaltfläche betätigen. Daraufhin würde ein Eingabefenster oder -feld aufgerufen werden, in dem der Benutzer dann den neuen Namen der Gruppe eingeben könnte. Die andere Möglichkeit bestünde darin, dass der Benutzer den Eintrag im *TreeView*-Steuerelement direkt bearbeitet und wir auf diese Änderung entsprechend reagieren.

Problemlösung

Bevor wir uns für eine Möglichkeit oder auch für beide entscheiden, wird eine Prozedur benötigt, die das Umbenennen letztendlich ausführt. Um der Konzeption des Projekts weiter zu folgen, sollte diese Prozedur dem Klassenmodul *LayerGroups* hinzugefügt werden.

```
Public Sub Rename(GroupName, url, NewName)
Dim ActGrp As LayerGroup
For i = 1 To mCol.Count
    Set ActGrp = mCol(i)
    If UCase(ActGrp.GruppenName) = UCase(GroupName) And _
        UCase(ActGrp.GruppenUrl) = UCase(url) Then
        ActGrp.GruppenName = NewName
    ElseIf IsChild(ActGrp.GruppenUrl, url & "\" & GroupName) = _
        True Then
        ActGrp.GruppenUrl = url & "\" & NewName & Mid( _
            ActGrp.GruppenUrl, Len(url & "\" & GroupName) + 1)
    End If
Next
End Sub
```

Natürlich müssen auch die Gruppeneinträge, die in der *Groups*-Auflistung der Layer gespeichert wurden, eine neue Bezeichnung erhalten, damit diese auch auf den aktuellen Stand gebracht und somit wieder in der Gruppe angezeigt werden. Also fügen wir der Klasse *GroupsLayer* ebenfalls eine Prozedur hinzu, die diese Aufgabe übernimmt. Anhand dieser Prozedur wird nun auch klar, warum die Verwendung eines weiteren Klassenmoduls statt der Auflistungsvariable *mCol* von Vorteil sein kann.

```
Public Sub RenameGroups(OldUrl, NewUrl)
Dim ActLayer As GroupLayer
On Error Resume Next
For i = 1 To mCol.Count
    Set ActLayer = mCol(i)
    For X = ActLayer.Groups.Count To 1 Step -1
        If UCase(ActLayer.Groups.Item(X)) = UCase(OldUrl) Then
            ActLayer.Groups.Add NewUrl, , , X
            ActLayer.Groups.Remove X
        ElseIf IsChild(ActLayer.Groups(X), OldUrl) = True Then
            ActLayer.Groups.Add CStr(NewUrl & Mid(ActLayer.Groups( _
                X), Len(OldUrl) + 1)), , , X
            ActLayer.Groups.Remove X
        End If
    Next
Next
End Sub
```

Nachdem die Vorbereitungen abgeschlossen sind, brauchen wir den Aufruf dieser Prozeduren nur noch in den Code einzubinden, der das Umbenennen der Gruppen steuert. Schauen wir uns dies einmal anhand des Falls an, dass der Benutzer den Namen der Gruppe direkt im *TreeView*-Steuerelement ändert.

```
Private Sub TreeView1_AfterLabelEdit(Cancel As Integer, _
NewString As String)
On Error Resume Next
```

```

If Layergruppen.GroupExist(NewString, TreeView1. _
    SelectedItem.Parent.FullPath) = False Then
    Layers.RenameGroups TreeView1.SelectedItem.FullPath, _
        TreeView1.SelectedItem.Parent.FullPath & "\" & NewString
    Layergruppen.Rename TreeView1.SelectedItem.Text, _
        TreeView1.SelectedItem.Parent.FullPath, NewString
    TreeView1_NodeClick TreeView1.SelectedItem
Else
    Cancel = 1
End If
End Sub

```

Jetzt könnten Sie auch noch eine weitere Schaltfläche dem *Toolbar*-Steuerelement hinzufügen und dort ebenfalls einige Codezeilen unterbringen, die zunächst den neuen Namen der gewählten Gruppe abfragen und dann gegebenenfalls die Gruppe umbenennen.

Problemstellung

Das, was für die Layer-Gruppen gilt, sollte auch bei den Layern funktionieren, also das Hinzufügen, Umbenennen und Löschen einzelner Layer.

Lösungsansatz

Auch hier könnten wir den gleichen Ansatz wählen wie bei den Layer-Gruppen. Das Hinzufügen und Löschen der Layer kann über eine Schaltfläche im *LVToolbar*-Steuerelement aktiviert werden, das Umbenennen lässt sich über die direkte Bearbeitung im *ListView*-Steuerelement abarbeiten.

Problemlösung

Lösen wir zunächst die Aufgaben des Hinzufügens und Entfernens von einzelnen bzw. mehreren Layern. Dazu benötigen wir zwei Schaltflächen im *LVToolbar*-Steuerelement. Die erste erhält den Schlüssel „New“, die zweiten „Delete“. Für die Symbole können Sie wiederum auf Bilder des Verzeichnisses *04\images* zurückgreifen. Wie diese den Schaltflächen zugeordnet werden, ist ja bereits mehrfach besprochen worden.

```

Private Sub LVToolbar_ButtonClick(ByVal Button As _
    MSComctlLib.Button)
    Dim LayerName As String
    Dim NewLayer As GroupLayer
    On Error Resume Next
    Select Case Button.Key

```

Hat der Benutzer die Schaltfläche mit dem Schlüssel „New“ betätigt, können wir den Code zum Anlegen eines neuen Layers ausführen. Hierbei werden wir uns wiederum an der Funktionsweise des AutoCAD-Layermanagers orientieren. Also wird zunächst ein Vorgabename ermittelt und mit diesem ein Layer-Objekt der Layer-Auflistung hinzugefügt.

```

    Case "New"
        X = 1
        Do
            LayerName = "Layer " & CStr(X)
            If TypeName(Layers.ItemByName(LayerName)) = "Nothing" _
                Then

```

```

Exit Do
End If
X = X + 1
Loop
Set NewLayer = Layers.Add(LayerName)

```

Als Eigenschaftswerte des neuen Layers können wir die aktuellen Einstellungen des Programms verwenden.

```

NewLayer.Freeze = Freeze.Value
NewLayer.Color = ColorCombo.GetSelColorIndex
NewLayer.LayerOn = LayerOn.Value
NewLayer.LineType = LTypeCombo.Text
NewLayer.LineWeight = Util.GetLineweightValue( _
    LWCombo.SelectedItem.Index)
NewLayer.Locked = Locked.Value
NewLayer.OriginalName = ""
NewLayer.PlotStyleName = PlotStyleText
NewLayer.Plottable = Plottable.Value
NewLayer.ViewportDefault = FreezeNewVP.Value
NewLayer.ViewportFreeze = FreezeVP.Value

```

Natürlich muss dem Layer auch die ausgewählte Gruppe zugeordnet werden, damit dieser auch in der Gruppe angezeigt wird – es sei denn, bei der gewählten Gruppe handelt es sich um den ersten Eintrag. In diesem Fall wird der Layer einfach nur erzeugt.

```

If TypeName(TreeView1.SelectedItem.Parent) <> "Nothing" Then
    NewLayer.Groups.Add TreeView1.SelectedItem.FullPath
End If

```

Jetzt brauchen wir den Layer-Namen nur noch dem *ListView*-Steuerelement hinzuzufügen und als aktuell gewähltes Element der Liste zu markieren.

```

Set LV = AddToListView(NewLayer)
Set ListView1.SelectedItem = LV
ListView1_ItemClick ListView1.SelectedItem
ListView1.SelectedItem.EnsureVisible
ListView1.SetFocus

```

Damit der Benutzer nun auch noch den Eintrag sofort bearbeiten kann, wird einfach die Anweisung *StartLabelEdit* ausgeführt.

```

ListView1.StartLabelEdit

```

Beim Löschen eines oder mehrerer Layer ist nur darauf zu achten, dass der Benutzer festlegen kann, ob der Layer generell aus der Zeichnung gelöscht oder nur aus der Gruppe entfernt werden soll. Diese Auswahl ist allerdings nur erforderlich, wenn eine Untergruppe eingestellt wurde. Falls der oberste Eintrag im *TreeView*-Steuerelement gewählt wurde, können wir uns die Abfrage sparen und den Layer aus der Auflistung entfernen.

```

Case "Delete"
    If TypeName(TreeView1.SelectedItem.Parent) = "Nothing" Then
        mValue = vbYes
    Else
        mValue = MsgBox("Ausgewählte Layer auch aus der " & _
            "Zeichnung löschen?", vbYesNo, "Layer löschen")
    End If

```

```

End If
For i = ListView1.ListItems.Count To 1 Step -1
  If ListView1.ListItems(i).Selected = True Then
    If mValue = vbYes Then
      Layers.RemoveByName ListView1.ListItems(i).Text
    Else
      Layers.ItemByName(ListView1.ListItems(i).Text). _
        RemoveGroup TreeView1.SelectedItem.FullPath
    End If
    ListView1.ListItems.Remove i
  End If
Next

```

Da der eigentliche Löschvorgang erst erfolgt, wenn der Benutzer die Schaltfläche **SPEICHERN** betätigt hat, kann auch erst dann kontrolliert werden, ob der Layer auch wirklich gelöscht werden konnte. Sie sollten für diesen Fall eventuell der Speicherungsprozedur ein Protokoll hinzufügen, damit der Benutzer am Ende einen Überblick erhält, welche Layer nicht umbenannt und/oder gelöscht werden konnten.

```

End Select
End Sub

```

Damit wäre der vorletzte Teil des Projekts abgeschlossen. Alles, was bis hierhin besprochen wurde, können Sie noch einmal in der Projektdatei *Teil5.dvb* nachvollziehen.

4.7 Layer-Filter

Problemstellung

Als kleines Highlight können wir dem Projekt noch eine Filterfunktion hinzufügen. Ausgangspunkt könnten hierbei die vorhandenen Filter der aktuellen Zeichnung sein. Damit stellen sich zunächst zwei Probleme: zum einen natürlich die Frage, wie an diese Filter heranzukommen ist und wie genau die Definitionen der Filter zu verarbeiten sind. Die andere Frage, mit der wir uns auseinander setzen müssen, lautet: Wie lassen sich diese Filter auf das *List View*-Steuerelement anwenden?

Lösungsansatz

Grundlegend sollten wir zunächst den Namen des aktuellen Filters global in einer Variablen speichern, damit wir jederzeit auf das Filterobjekt zugreifen können. Als Nächstes benötigen wir natürlich auch eine Art Liste, in der die vorhandenen Filter angezeigt und durch den Benutzer ausgewählt werden können.

Problemlösung

Deklarieren wir zunächst die Variable, in der der Name des aktuellen Filters gespeichert wird. Diese Deklarationszeile fügen Sie bitte dem Formularcode hinzu.

```
Dim ActiveFilter As String
```

Bevor wir uns der Anwendung des Filters zuwenden, beschäftigen wir uns zunächst mit dessen Ermittlung. Alle Filter der Zeichnung werden in den erweiterten Daten der Layer-Auflistung gespeichert. Zu diesem Zweck legt AutoCAD ein Erweiterungswörterbuch mit dem Namen „ACAD_LAYERFILTERS“ an. In diesem Wörterbuch werden die Filter dann als einzelne

XRecords abgelegt. Das wäre also geklärt. Bleibt noch die Frage, wo die Filternamen im Formular zur Auswahl angeboten werden sollen. Auch hierfür würde sich eine Schaltfläche im *LVToolbar*-Steuerelement anbieten. Für jeden Filternamen würden wir dann einen Schaltflächenmenüpunkt anlegen, den der Benutzer dann über den Drop-down-Button neben der Schaltfläche einblenden kann. In diesem Fall müssten Sie die *Style*-Eigenschaft der Schaltfläche auf *tbrDropdown* einstellen, damit die ButtonMenus angezeigt werden können. Die Schaltflächen werden dann durch folgende Prozedur, die beim Start des Formulars ausgeführt wird (*Initialize*-Prozedur), dem Schaltflächenmenü hinzugefügt.

```
Sub ReadFilters()
Dim FltDict As AcadDictionary
Dim FltXRecord As AcadXRecord
On Error Resume Next
Set FltDict = ThisDrawing.Layers.GetExtensionDictionary( _
    "ACAD_LAYERFILTERS")
LVToolbar.Buttons("Filter").ButtonMenus.Clear
LVToolbar.Buttons("Filter").ButtonMenus.Add , , _
    "Alle Layer anzeigen"
If TypeName(FltDict) <> "Nothing" Then
    For i = 0 To FltDict.Count - 1
        Set FltXRecord = FltDict.Item(i)
        LVToolbar.Buttons("Filter").ButtonMenus.Add , , _
            FltXRecord.Name
    Next
End If
End Sub
```

Damit kann der Benutzer die Filter auswählen und es muss nur noch eine Funktion eingebunden werden, die die tatsächliche Filterung der Layer nach der Filtervorlage ausführt. Für diesen Fall besitzt das *Toolbar*-Steuerelement die Ereignisprozedur *ButtonMenuClick* (Kapitel 3.3.7). Diese Prozedur wird nicht nur ausgeführt, wenn ein Menüeintrag ausgewählt wurde, sondern liefert den gewählten Menüpunkt als Argument der Prozedur gleich mit. An diesem Punkt setzen wir nun an und ergänzen diese Prozedur mit folgendem Code.

```
Private Sub LVToolbar_ButtonMenuClick(ByVal ButtonMenu As _
MSComctlLib.ButtonMenu)
ActiveFilter = ButtonMenu.Text
TreeView1_NodeClick TreeView1.SelectedItem
End Sub
```

Der Filter kann durch den Benutzer festgelegt werden. Fehlt natürlich noch dessen Einbindung in die Zusammenstellung des *ListView*-Steuerelements. Da dieser Code etwas umfangreicher ist, erstellen wir zunächst eine Funktion, die als Ergebnis den Wert *TRUE* zurückliefern soll, wenn das angegebene Layer-Objekt dem aktuell eingestellten Filter entspricht, oder eben *FALSE*, wenn dies nicht der Fall sein sollte. Diese Funktion braucht dann nur noch in die *AddToListView*-Prozedur eingebunden zu werden, damit die Auswertung des Filters in die Zusammenstellung der Elemente des *ListView*-Steuerelements implementiert wird.

```
If showLayer(ActLayer) = True Then
End If
```

Setzen wir uns nun noch mit der *showLayer()*-Funktion auseinander. Zunächst werden wir den aktuellen Inhalt der Variablen *ActiveLayer* überprüfen. Enthält diese die Zeichenfolge „Alle Layer anzeigen“, legen wir als Ergebnis der Funktion den Wert *TRUE* fest, da der Benutzer ja alle Layer angezeigt haben möchte.

```
Function showLayer(ActLayer As GroupLayer) As Boolean
Dim FltDict As AcadDictionary
Dim FltXRecord As AcadXRecord
Dim xTypeOut, xDataOut
On Error Resume Next
showLayer = True
If ActiveFilter = "Alle Layer anzeigen" Then
    showLayer = True
Else
```

Sollte die Variable allerdings einen anderen Inhalt zurückgeben, lesen wir das Filterobjekt aus dem Wörterbuch der *Layers*-Auflistung der aktuellen Zeichnung aus.

```
Set FltDict = ThisDrawing.Layers.GetExtensionDictionary( _
    "ACAD_LAYERFILTERS")
Set FltXRecord = FltDict.Item(ActiveFilter)
FltXRecord.GetXRecordData xTypeOut, xDataOut
```

Die Filterdaten sind wiederum in den einzelnen Feldern des Arrays *xDataOut* enthalten. Jedes Feld dieses Arrays enthält bestimmte Einstellungen des jeweiligen Filters. Welches Feld welchen Wert speichert, können Sie der Tabelle 4.3 entnehmen.

Datenfeld	Inhalt	Gruppencode
<i>XData(0)</i>	Filtername	1
<i>XData(1)</i>	Layer-Name	1
<i>XData(2)</i>	Farbe	1
<i>XData(3)</i>	Linientyp	1
<i>XData(4)</i>	Ein, Frieren, Sperren usw.	72
<i>XData(5)</i>	Linienstärke	1
<i>XData(6)</i>	Plotstil	1

Tabelle 4.3: Zusammensetzung eines Filterrecords

Mit diesen Informationen kann nun die Funktion *showLayer* ergänzt werden. Das Prinzip ist immer dasselbe. Zunächst wird geprüft, ob der jeweilige Eigenschaftswert (Beide, Ein, Aus) geprüft werden soll. Wann welcher Eigenschaftswert für die Filterung verwendet werden soll, ergibt sich aus den gesetzten Bitwerten des vierten Datenfeldes des Arrays. Eine Übersicht über die einzelnen Werte erhalten Sie in Tabelle 4.4.

Eigenschaft	Status					
<i>LayerOn</i>	Eingeschaltet	= 1	Ausgeschaltet	= 3	Beides	= 0
<i>Freeze</i>	Gefroren	= 4	Getaut	= 12	Beides	= 0
<i>ViewportFreeze</i>	Gefroren	= 16	Getaut	= 48	Beides	= 0
<i>ViewportDefault</i>	Gefroren	= 64	Getaut	= 192	Beides	= 0
<i>Locked</i>	Gesperrt	= 256	Nicht gesperrt	= 768	Beides	= 0
<i>Plottable</i>	Ein	= 1024	Aus	= 3072	Beides	= 0

Tabelle 4.4: Werte der einzelnen Einstellungsmöglichkeiten


```

If (xDataOut(4) And 3) = 3 Then 'Layer aus = 3:
    showLayer = Not ActLayer.LayerOn
ElseIf (xDataOut(4) And 1) = 1 Then 'Layer ein = 1
    showLayer = ActLayer.LayerOn
Else
    showLayer = True
End If

```

Genauso funktioniert das auch bei den anderen Eigenschaften, nur dass hier logischerweise ein anderes Bit gesetzt ist.

```

If showLayer = False Then Exit Function
If (xDataOut(4) And 12) = 12 Then 'Layer getaut =12
    showLayer = Not ActLayer.Freeze
ElseIf (xDataOut(4) And 4) = 4 Then 'Layer gefroren = 4
    showLayer = ActLayer.Freeze
Else
    showLayer = True
End If
If showLayer = False Then Exit Function
If (xDataOut(4) And 48) = 48 Then 'aktuelles AF getaut = 48
    showLayer = Not ActLayer.ViewportFreeze
ElseIf (xDataOut(4) And 16) = 16 Then 'aktuelles AF gefroren _
    = 16
    showLayer = ActLayer.ViewportFreeze
Else
    showLayer = True
End If
If showLayer = False Then Exit Function
If (xDataOut(4) And 192) = 192 Then 'neues AF getaut = 192
    showLayer = Not ActLayer.ViewportDefault
ElseIf (xDataOut(4) And 64) = 64 Then 'neues AF gefroren = 64
    showLayer = ActLayer.ViewportDefault
Else
    showLayer = True
End If
If showLayer = False Then Exit Function
If (xDataOut(4) And 768) = 768 Then 'Layer nicht gesperrt = 768
    showLayer = Not ActLayer.Locked
ElseIf (xDataOut(4) And 256) = 256 Then 'Layer gesperrt = 256
    showLayer = ActLayer.Locked
Else
    showLayer = True
End If
If showLayer = False Then Exit Function
If (xDataOut(4) And 3072) = 3072 Then 'Layer nicht plotten _
    = 3072
    showLayer = Not ActLayer.Plottable
ElseIf (xDataOut(4) And 1024) = 1024 Then 'Layer plotten = 1024
    showLayer = ActLayer.Plottable
Else
    showLayer = True
End If

```

Die Zeichenfolgen, wie Layername oder Linientyp, werden dann mit dem *Like*-Operator geprüft. Gibt die Überprüfung den Wert *FALSE* zurück, entspricht die Zeichenfolge nicht dem jeweiligen Muster und der Layer wird nicht dargestellt.

```
If showLayer = False Then Exit Function
showLayer = ActLayer.Name Like xDataOut(1)
If showLayer = False Then Exit Function
showLayer = Util.GetColorName(ActLayer.Color) Like xDataOut(2)
If showLayer = False Then Exit Function
showLayer = ActLayer.LineType Like xDataOut(3)
If showLayer = False Then Exit Function
showLayer = Util.GetLineWeightText(ActLayer.LineWeight) Like _
    xDataOut(5)
If showLayer = False Then Exit Function
showLayer = ActLayer.PlotStyleName Like xDataOut(6)
If showLayer = False Then Exit Function
End If
End Function
```

Damit wäre der bis dahin letzte Teil des Layermanager-Projekts abgeschlossen. Sie können natürlich wieder alle bisherigen Schritte anhand der Projektdatei *Teil6.dvb* nachvollziehen. Die endgültige Projektdatei finden Sie unter dem Namen *Layermanager.dvb* auf der Buch-CD. Diese enthält noch einige Ergänzungen, die in diesem Kapitel nicht besprochen wurden.

5

Das Anwendungsobjekt

Die Anwendung AutoCAD wird in der Objekthierarchie an oberster Stelle geführt und bildet somit den Container für alle anderen Objekte. Das bedeutet, dass alle in AutoCAD verwendeten Objekte – ob Zeichnungselement, Menüeintrag oder komplette Zeichnung – über dieses Objekt abgerufen werden können. Das vorliegende Kapitel befasst sich mit diesem Objekt und soll klären, wie die AutoCAD-Sitzung über die VBA-Schnittstelle grundlegend zu steuern ist.

Kapitelübersicht

5.1 Zugriff auf die AutoCAD-Sitzung

5.2 Verwaltung der AutoCAD-Menüstruktur

5.3 AutoCAD-Einstellungen steuern

5.1 Zugriff auf die AutoCAD-Sitzung

Wie können Sie nun aber auf dieses Anwendungsobjekt zugreifen? Nun, jedes Objekt besitzt eine Eigenschaft, die einen Verweis auf das Anwendungsobjekt zurückgeben kann. Dabei handelt es sich um die *Application*-Eigenschaft. Sie könnten also eine Objektvariable mit diesem Objekt füllen, indem Sie einen Verweis auf die *Application*-Eigenschaften jedes beliebigen AutoCAD-Objekts erstellen:

```
Dim AcadApp As AcadApplication  
Set AcadApp = Object.Application
```

Einfacher ist natürlich der direkte Weg. Sie können auf das *Application*-Objekt von jedem Ort aus zugreifen, indem Sie das Objekt direkt mit *Application* ansprechen. Die Zuweisung an die Objektvariable *AcadApp* würde demnach wie folgt aussehen.

```
Dim AcadApp As AcadApplication  
Set AcadApp = Application
```

Wenn Sie die Deklaration der Variablen *AcadApp* global vornehmen, können Sie mit einem einzigen Verweis eine Objektvariable erstellen, die Sie im gesamten Projekt verwenden können.

5.1.1 Ereignisprozeduren des Anwendungsobjekts

Genau wie die Dokumentklasse besitzt auch das Anwendungsobjekt eine ganze Reihe von Ereignisprozeduren, die bei den verschiedensten Aktionen ausgelöst werden. Da das Anwendungsobjekt nicht als Klassenmodul vorliegt, müssen wir erst eine Objektvariable so initialisieren, dass die Ereignisse auch ausgelöst werden können. Da wir diesen Vorgang nur in Klassenmodulen vornehmen können, nehmen wir die Deklaration des Anwendungsobjekts im Modul *ThisDrawing* vor, das ja bereits erstellt wurde und die Ereignisprozeduren des Dokuments beinhaltet. Folgende Codezeilen deklarieren das Objekt auf diese Weise:

```
Public WithEvents AcadApp As AcadApplication
```

Nachdem diese Zeile dem Klassenmodul *ThisDrawing* hinzugefügt wurde, sind zwar die Ereignisprozeduren abrufbar, sie werden allerdings erst dann ausgelöst, wenn Sie einen Verweis auf das Anwendungsobjekt an diese Variable übertragen haben.

AppActivate/AppDeactivate

Das *AppActivate*-Ereignis wird immer ausgelöst, bevor das AutoCAD-Anwendungsfenster aktiviert wird, *AppDeactivate* entsprechend, bevor das Anwendungsfenster wieder deaktiviert wird. Das bedeutet nichts anderes als einen Fokuswechsel von einem Windows-Anwendungsfenster zu AutoCAD bzw. von AutoCAD zurück zu einer anderen Windows-Anwendung.

```
Private Sub AcadApp_AppActivate()  
End Sub  
Private Sub AcadApp_AppDeactivate()  
End Sub
```

ARXLoaded/ARXUnloaded

Referenzargument: *AppName* As String

Diese Ereignisse werden ausgelöst, wenn eine ARX-Anwendung in AutoCAD geladen (*ARXLoaded*) bzw. aus AutoCAD entfernt (*ARXUnloaded*) wird. Der Name der Anwendung, die geladen bzw. entfernt wurde, inklusive Pfad, wird im Parameter *AppName* mitgeliefert.

```
Private Sub AcadApp_ARXLoaded(ByVal AppName As String)  
End Sub  
Private Sub AcadApp_ARXUnloaded(ByVal AppName As String)  
End Sub
```

BeginCommand/EndCommand

Referenzargument: *CommandName* As String

Die Prozeduren *BeginCommand* und *EndCommand* werden ausgeführt, nachdem der Benutzer einen AutoCAD-Befehl ausgeführt (*BeginCommand*) oder beendet (*EndCommand*) hat. In beiden Fällen wird der entsprechende AutoCAD-Befehl (z.B. LINE oder ERASE) in dem Parameter *CommandName* mitgeliefert. Diese Prozeduren eignen sich z.B. hervorragend dazu, die Aufrufe von AutoCAD-Befehlen zu protokollieren.

```
Private Sub AcadApp_BeginCommand(ByVal CommandName As String)
End Sub

Private Sub AcadApp_EndCommand(ByVal CommandName As String)
End Sub
```

BeginFileDrop

Referenzargument: `FileName As String`, `Cancel As Boolean`

Seit Windows 3.1 hat der Benutzer die Möglichkeit, Dateien auf ein Anwendungsfenster zu ziehen, um diese Dateien mit der entsprechenden Anwendung weiterzuverarbeiten. Auch AutoCAD unterstützt dieses Verhalten. Dieses Ereignis wird somit ausgelöst, sobald der Benutzer eine AutoCAD-Zeichnungsdatei im Dokumentfenster ablegt. AutoCAD reagiert zwar nur auf das Ablegen von Zeichnungsdateien, trotzdem ist diese Prozedur eine gute Möglichkeit, auf solch einen Vorgang zu reagieren. In den Argumenten dieser Prozedur wird zum einen der vollständige Name der Datei übergeben und zum anderen ein Parameter, mit dem Sie das weitere Vorgehen zur Verwendung der Datei festlegen können. Wird diese Prozedur ausgelöst und Sie vergeben für den Parameter `Cancel` den Wert `TRUE`, dann wird der Ladevorgang der Datei abgebrochen.

```
Private Sub AcadApp_BeginFileDrop(ByVal FileName As String, _
Cancel As Boolean)
If MsgBox("Ladevorgang fortsetzen?", vbYesNo) = vbNo Then
    Cancel = True
End If
End Sub
```

Noch eine kurze Anmerkung zum Ablegen von Zeichnungsdateien in AutoCAD: Wenn Sie eine Zeichnungsdatei auf einem Dokumentfenster ablegen, wird diese als Blockreferenz im Dokument eingefügt. Legen Sie die Datei hingegen im Anwendungsfenster ab, wird sie als separate Zeichnung in AutoCAD geöffnet.

BeginLisp/EndLisp/LispCancelled

Referenzargument: `FirstLine As String` (nur `BeginLisp`)

Für diejenigen, die gezielt mit Lisp arbeiten, könnten diese drei Ereignisprozeduren interessant sein. Diese reagieren direkt auf die Anwendung von Lisp-Routinen. Aus dem Namen der jeweiligen Prozedur ergibt sich bereits das Ereignis, bei dem sie ausgeführt werden. Die *BeginLisp*-Prozedur übergibt zusätzlich den Namen des Lisp-Ausdrucks in der Variable *FirstLine*. Dabei ist Folgendes zu beachten:

- Handelt es sich um eine benutzerdefinierte Lisp-Funktion, die direkt in der Befehlszeile aufgerufen werden soll (z.B. `C:TEST`), und wird dann in der Befehlszeile `TEST` eingegeben, so enthält die Variable *FirstLine* die Zeichenfolge (`C:TEST`).
- Wurde in der Befehlszeile eine benutzerdefinierte Lisp-Funktion eingegeben (z.B. `(test)`), die nicht als direkter AutoCAD-Befehl auszuführen ist, dann enthält die Variable *FirstLine* die Zeichenfolge `(test)` und zwar in der gleichen Schreibweise (Groß-/Kleinschreibung), wie sie in der Befehlszeile eingegeben wurde, nicht die der Definition.

Die beiden anderen Ereignisprozeduren enthalten hingegen keine weiteren Übergabevariablen. Das *EndLisp*-Ereignis wird immer dann ausgeführt, sobald eine Lisp-Funktion beendet wird, egal ob dies regulär geschieht oder die Funktion abgebrochen wurde.

Wurde eine Lisp-Funktion hingegen abgebrochen, so wird vor der *EndLisp*-Prozedur das *LispCancelled*-Ereignis ausgelöst.

```
Private Sub AcadApp_BeginLisp(ByVal FirstLine As String)
End Sub

Private Sub AcadApp_EndLisp()
End Sub

Private Sub AcadApp_LispCancelled()
End Sub
```

BeginModal/EndModal

Die *BeginModal*-Prozedur wird ausgeführt, bevor ein Dialogfenster modal (gebunden) in AutoCAD eingeblendet wird. Das *EndModal*-Ereignis weist dagegen darauf hin, dass ein modaler Dialog beendet wurde. Da während der modalen Anzeige eines Dialogfensters keine Ereignisse außerhalb des Dialogfensters ausgelöst werden, sind diese beiden Prozeduren dazu geeignet, solche Formulare zu überwachen, um z.B. Operationen durchzuführen oder fortzusetzen, wenn ein Dialogfenster abgeschlossen wurde.

```
Private Sub AcadApp_BeginModal()
End Sub

Private Sub AcadApp_EndModal()
End Sub
```

BeginOpen/EndOpen

Referenzargument: FileName As String

Wie der Name es bereits andeutet, werden diese Ereignisprozeduren ausgelöst, wenn ein Öffnungsvorgang in AutoCAD ausgeführt wird. Das erste Ereignis (*BeginOpen*) wird aktiviert, unmittelbar nachdem AutoCAD eine Anforderung zum Öffnen einer Zeichnung erhalten hat. Diese Anforderung könnte z.B. durch den *Open*-Befehl erfolgen, nachdem der Benutzer dort eine Datei im Auswahldialog selektiert und das Öffnen dieser Zeichnung bestätigt hat. Die zweite Ereignisprozedur (*EndOpen*) wird ausgeführt, nachdem der Ladevorgang abgeschlossen, die Zeichnung also in AutoCAD geladen wurde.

```
Private Sub AcadApp_BeginOpen(FileName As String)
MsgBox "Datei " & FileName & " wird geöffnet"
End Sub

Private Sub AcadApp_EndOpen(ByVal FileName As String)
MsgBox "Datei " & FileName & " wurde geöffnet"
End Sub
```

BeginPlot/EndPlot

Referenzargument: DrawingName As String

Mit diesen Ereignisprozeduren können Sie auf die Anforderung an AutoCAD reagieren, eine Zeichnung zu drucken, um z.B. noch einige temporäre Operationen an der Zeichnung durchzuführen und nach Beendigung des Druckvorgangs diese wieder rückgängig zu machen. Ich habe in diesen Prozeduren immer die Freistellung bestimmter Punktsymbole von Linien/Polylinien durchgeführt, die nach Beendigung des Plotvorgangs wieder zurückgesetzt wurde. Sowohl beim *BeginPlot*- als auch beim *EndPlot*-Ereignis wird der Name der zu druckenden Zeichnung mitgeliefert. Dabei handelt es sich allerdings nur um den reinen Zeichnungsnamen ohne Pfadangabe und Dateierweiterung, was die Identifizierung der Zeichnung innerhalb der Dokumentauflistung etwas schwieriger gestaltet.

```
Private Sub AcadApp_BeginPlot(ByVal DrawingName As String)
End Sub
Private Sub AcadApp_EndPlot(ByVal DrawingName As String)
End Sub
```

Siehe auch *Kapitel 9*

BeginQuit

Referenzargument: Cancel As Boolean

Mithilfe dieser Ereignisprozedur können Sie auf das Beenden der AutoCAD-Sitzung reagieren. Es wird immer dann ausgelöst, wenn eine Anforderung zum Schließen des Sitzungsfensters von AutoCAD gestellt, aber noch bevor der Beendigungsvorgang ausgeführt wird. Durch die mitgelieferte Variable werden Sie in die Lage versetzt, diesen Vorgang bei Bedarf abubrechen. Ändern Sie den Wert der Variablen auf *TRUE*, wird die Anforderung zum Schließen von AutoCAD zurückgesetzt und die AutoCAD-Sitzung bleibt erhalten.

```
Private Sub AcadApp_BeginQuit(Cancel As Boolean)
On Local Error Resume Next
If MsgBox("AutoCAD-Sitzung beenden?", vbYesNo) = vbNo Then
    Cancel = True
End If
End Sub
```

BeginSave/EndSave

Referenzargument: FileName As String

Nachdem eine Anforderung zum Speichern eines Dokuments an AutoCAD gestellt wurde, jedoch noch bevor der Sicherungsvorgang beginnt, wird dieses Ereignis ausgelöst. Die Speichieranforderung kann dabei durch den Benutzer selbst oder durch ein Programm gestellt worden sein. Dem *BeginSave*-Ereignis folgt dann immer *EndSave* nach Abschluss des Speichervorgangs. Beide Prozeduren übergeben den Namen (inklusive Verzeichnis), unter dem die Zeichnung gespeichert werden soll bzw. dann gespeichert wurde.

```
Private Sub AcadDocument_BeginSave(ByVal FileName As String)
End Sub
Private Sub AcadDocument_EndSave(ByVal FileName As String)
End Sub
```

NewDrawing

Bevor AutoCAD eine neue Zeichnung in der Sitzung erstellt, wird erst dieses Ereignis ausgelöst. Das weist Sie dann darauf hin, dass im nächsten Schritt ein neues Dokument zur Dokumentauflistung hinzugefügt wird. Wenn Sie das Erstellen einer neuen Zeichnung aus einer eigenen Anwendung heraus ausführen, müssen Sie darauf achten, dass keine modalen Dialogfenster aufgerufen wurden, da sonst das Ereignis nicht ausgelöst wird.

```
Private Sub AcadApp_NewDrawing()  
End Sub
```

SysVarChanged

Referenzargument: SysvarName As String, newVal As Variant

Nachdem der Inhalt einer Systemvariablen geändert wurde, wird diese Ereignisprozedur ausgelöst. In den Parametern dieser Prozedur erhalten Sie einen Hinweis, um welche Variable es sich handelt und welchen neuen Wert diese nun enthält. Je nach Typ der geänderten Systemvariablen erhalten Sie dann auch in der Variablen *newVal* Werte unterschiedlichen Datentyps.

Beispiele:

Änderung der aktuellen Texthöhe: Die Variable *newVal* enthält einen Wert des Datentyps *Double*.

Änderung des zuletzt eingegebenen Punkts: Die Variable *newVal* enthält jetzt ein 3D-Array mit den entsprechenden Koordinaten:

```
Private Sub AcadApp_SysVarChanged(ByVal SysvarName As String, _  
ByVal newVal As Variant)  
End Sub
```

Siehe auch *Kapitel 6.2.3*

WindowChanged

Referenzargument: WindowState As AcWindowState

Diese Ereignisprozedur ist hilfreich für Anwendungen, die auf die Änderung des Status des AutoCAD-Fensters reagieren müssen. Die Variable *WindowState* liefert dabei den aktuellen Status eben dieses Fensters. Sie kann eine der folgenden Konstanten enthalten:

acMin – Das Fenster wurde minimiert.

acMax – Das Fenster wurde maximiert.

acNorm – Das Fenster wird normal, also weder minimiert noch maximiert, dargestellt.

```
Private Sub AcadApp_WindowChanged(ByVal WindowState As _  
AcWindowState)  
End Sub
```

WindowMovedOrResized

Referenzargument: HwndFrame As Long, bMoved As Boolean

Dieses Ereignis wird ausgelöst, wenn das Anwendungsfenster im Status geändert oder verschoben wurde. Es fällt also in den gleichen Anwendungsbereich wie die vorherige Prozedur. Der Unterschied ist, dass diese nicht den Status, sondern die Referenz (Handle) des Fensters

und die Aktion, die mit ihm vorgenommen wurde, als Variablen mitliefert. Die erste Variable (*HWNDFrame*) liefert das Handle des Fensters, das es vom Windows-Betriebssystem bekommen hat und das es eindeutig im System identifiziert. Die zweite Variable (*Moved*) kann entweder den Wert *TRUE* enthalten, was bedeuten würde, dass das Fenster verschoben wurde, oder *FALSE*, was eine Änderung des Status mitteilt.

```
Private Sub AcadApp_WindowMovedOrResized(ByVal HWNDFrame As _  
Long, ByVal bMoved As Boolean)  
End Sub
```

5.1.2 Eigenschaften des Anwendungsobjekts

Im folgenden Abschnitt erhalten Sie eine Übersicht über alle Eigenschaften, die das Anwendungsobjekt zur Verfügung stellt. Mithilfe dieser Eigenschaften können Sie Einfluss auf die Darstellung des Anwendungsfensters nehmen oder einige Auflistungen, z.B. die der Dokumente oder der Menüstruktur, abrufen. Der Zugriff erfolgt wie gehabt über das Containerobjekt, also *Application*.

ActiveDocument

Datentyp: *AcadDocument*

ActiveDocument enthält das aktuelle Dokumentobjekt der AutoCAD-Sitzung und entspricht somit der aktuellen Zeichnung. Der Inhalt dieses Objekts ist identisch mit dem des Klassenmoduls *ThisDrawing*, was darauf schließen lässt, dass *ThisDrawing* aus dieser Eigenschaft gebildet wurde. Ändern Sie den Inhalt der Eigenschaft, durch Zuweisung eines Dokuments aus der Dokumentauflistung, ändert sich ebenfalls der Inhalt des Klassenmoduls *ThisDrawing*. Da dieses Eigenschaft auch global deklariert wurde, müssen Sie den Container der Eigenschaft nicht voranstellen.

```
ActiveDocument = Documents(N)
```

Wenn Sie der Eigenschaft ein Dokument zuweisen, bei dem es sich bereits um das aktuelle Dokument handelt, löst AutoCAD einen Fehler aus. Außerdem müssen Sie darauf achten, dass beim Schreiben der Eigenschaft das Schlüsselwort *Set* nicht verwendet wird, beim Auslesen dagegen schon.

```
Set ActDoc = ActiveDocument
```

Siehe auch *Kapitel 6*

Application

Datentyp: *AcadApplication*

Die Verwendung dieser Eigenschaft ist in Zusammenhang mit dem Anwendungsobjekt recht unsinnig, da sie auf das Anwendungsobjekt, also auf sich selbst, verweist.

Caption (schreibgeschützt)

Datentyp: *String*

Diese Eigenschaft enthält die Beschriftung der Titelzeile der AutoCAD-Sitzung, wie in Abbildung 5.1 zu sehen.



Abbildung 5.1: AutoCAD-Titelzeile

Diese Titelzeile kann nicht mit VBA beeinflusst werden. Wenn Sie die Titelzeile ändern möchten, um z.B. den Namen Ihrer Anwendung anzeigen zu lassen, müssen Sie wohl oder übel auf *ObjectARX* zurückgreifen.

Documents **(schreibgeschützt)**

Datentyp: *AcadDocuments*

Über dieses Objekt erhalten Sie Zugriff auf sämtliche Zeichnungen, die in der aktuellen AutoCAD-Sitzung geöffnet wurden.

```
Set AllDocs = Application.Documents
```

oder

```
Set AllDocs = Documents
```

Siehe auch *Kapitel 6*

FullName/Path/Name **(schreibgeschützt)**

Datentyp: *String*

Die Eigenschaft *Name* enthält den Anwendungsnamen von AutoCAD, so wie er im Windows-System benutzt wird. Der Name entspricht nicht dem Namen der ausführenden Datei, sondern vielmehr der Bezeichnung, die in den Versionsinformationen des Programms, im Abschnitt „interner Name“ angegeben ist. Im Normalfall wird dies die Bezeichnung „AutoCAD“ sein. Mit der *Path*-Eigenschaft können Sie die Verzeichnisposition der AutoCAD-Anwendung ermitteln. Da VBA selbst kein *App*-Objekt besitzt, könnten Sie diese Verzeichnisangabe benutzen, um beispielsweise Konfigurationsdateien, die für Ihre Anwendung benötigt werden, zentral zu speichern und jederzeit wiederzufinden. Die *FullName*-Eigenschaft besitzt als Ergänzung zur Verzeichnisangabe noch den Dateinamen, womit sich dann ein vollständiger Applikationsname ergibt.

Height/Width

Datentyp: *Long*

Diese beiden Eigenschaften bestimmen die Breite und Höhe des AutoCAD-Anwendungsfensters. Wenn Sie für das Fenster die Höhe und/oder die Breite festlegen und dieses sich gerade im maximiertem oder minimiertem Zustand befindet, so wird der Status des Fensters in jedem Fall auf normal gesetzt, was dann das Ereignis *WindowChanged* auslöst.

LocaleId **(schreibgeschützt)**

Datentyp: *Long*

Die *LocaleId* (auch als LCID bezeichnet) gibt eine ID der AutoCAD-Sitzung zurück. Diese ID enthält die Identifikation des Gebietsschemas (Sprache und Land) von AutoCAD. Diese

Informationen werden bei der Erstellung der Anwendung mitkompiliert. Bei deutschen Programmen ist dies der Code 1031. Sie können somit feststellen, für welchen Raum die Anwendung (deutsch, englisch, japanisch usw.) entwickelt worden ist.

MenuBar **(schreibgeschützt)**

Datentyp: *AcadMenuBar*

In dieser Eigenschaft ist das *MenuBar*-Objekt, also die Menüauflistung, die das aktuelle AutoCAD-Menü enthält, abgelegt.

Siehe auch *Kapitel 5.2*

MenuGroups **(schreibgeschützt)**

Datentyp: *AcadMenuGroups*

Im *MenuGroups*-Objekt werden alle Menügruppen, die wiederum jeweils Menüs und Werkzeugkästen enthalten, verwaltet.

Siehe auch *Kapitel 5.2.2*

Preferences **(schreibgeschützt)**

Datentyp: *AcadPreferences*

Diese Eigenschaft ruft das *Preferences*-Objekt ab, mit dem Sie dann die verschiedensten Programmeinstellungen vornehmen können, die auch im Optionen-Dialog verfügbar sind.

Siehe auch *Kapitel 5.3*

StatusId **(schreibgeschützt)**

Datentyp: *Boolean*

Diese Eigenschaft passt zwar nicht ganz in das *Application*-Objekt, aber sei es wie es sei, Sie können damit den aktuellen Status des Ansichtsfensters ermitteln, das Sie im Parameter dieser Eigenschaft festlegen. Gibt die Eigenschaft den Wert *TRUE* aus, handelt es sich um das aktive Ansichtsfenster, andernfalls nicht.

VBE **(schreibgeschützt)**

Datentyp: *Object*

Eine ausführliche Beschreibung dieses Objekts könnte sicherlich ein weiteres Buch füllen, weshalb ich an dieser Stelle darauf verzichte. Nur soviel am Rande: Es handelt sich um die VBA-Umgebung von AutoCAD. Über dieses Objekt erhalten Sie Informationen zu den geladenen VBA-Projekten, zu AddIns, die in VBA implementiert wurden, und vieles mehr. Einen Eindruck vom Umfang dieses Objekts erhalten Sie, indem Sie es in das Überwachungsfenster holen und den Inhalt durchsuchen. Fügen Sie dazu folgende Zeile dem Überwachungsfenster hinzu (natürlich zur Laufzeit):

Applications.VBE

Und Sie erhalten eine Darstellung, wie in Abbildung 5.2.

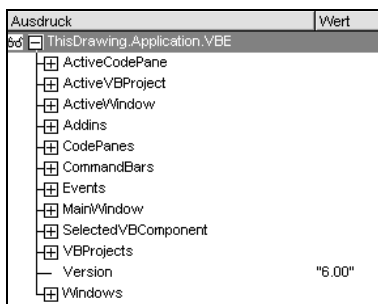


Abbildung 5.2: VBE-Objekt

Wenn Sie mehr Informationen zu diesem Objekt benötigen, können Sie in der VBA-Online-Hilfe nachschlagen.

Version (schreibgeschützt)

Datentyp: *String*

Wenn Sie die Versionsnummer der ausführenden AutoCAD-Anwendung ermitteln müssen, können Sie diese über diese Eigenschaft abrufen.

Visible

Datentyp: *Boolean*

Diese Eigenschaft ist besonders bei der Erstellung von AutoCAD-Applikationen mithilfe von VB zu beachten. Sie legt die Sichtbarkeit des Anwendungsfensters fest. Ist diese Eigenschaft auf *TRUE* gesetzt, ist das AutoCAD-Fenster sichtbar, andernfalls nicht.

WindowLeft/WindowTop

Datentyp: *Long*

Diese beiden Eigenschaften enthalten die genaue Position des AutoCAD-Anwendungsfensters auf dem Windows-Desktop. Die *WindowLeft*-Eigenschaft bestimmt dabei den Abstand des Fensters vom linken Bildschirmrand, *WindowTop* den vom oberen Rand. Beide Abstände sind in Pixel angegeben und können jederzeit geändert werden.

```
Application.WindowLeft = 100
Application.WindowTop = 50
```

WindowState

Datentyp: *AcWindowState (Long)*

Um den Status des AutoCAD-Stützungsfensters, also dessen aktuelle Darstellung in der Windows-Oberfläche zu ermitteln, können Sie den Wert der *WindowState*-Eigenschaft anrufen. Für diese Eigenschaft sind folgende Konstanten als Wert vorgesehen.

- *acMin* – Das AutoCAD-Sitzungsfenster wird minimiert dargestellt.
- *acMax* – Das AutoCAD-Sitzungsfenster wird maximiert dargestellt.
- *acNorm* – Das AutoCAD-Sitzungsfenster wird normal, also weder maximiert noch minimiert, dargestellt.

5.1.3 Prozeduren des Anwendungsobjekts

Neben den Eigenschaften bietet das Anwendungsobjekt natürlich auch noch einige Methoden zur Ausführung verschiedener Funktionen. Im Folgenden sollen diese Prozeduren genauer beschrieben werden.

Eval

Parameter: *ByVal Expression As String*

Sie können mit dieser Funktion eine VBA-Codezeile prüfen lassen, ohne die Codezeile in einem Modul einzufügen. Übergeben Sie den Code an diese Prozedur und diese führt dann den Code aus, mit all den Ergebnissen, die die Ausführung hervorbringen könnte. Haben Sie einen ungültigen Code verwendet, z.B. Syntaxfehler eingebaut oder einen Datentypenzwang nicht beachtet, löst AutoCAD einen Fehler aus.

GetAcadState

Rückgabedatentyp: *AcadState*

Eine wichtige Sache, die es bei der Entwicklung von Anwendungen zu beachten gilt, ist der Status der Anwendung. Damit nicht unbeabsichtigt aktive AutoCAD-Befehle abgebrochen werden oder Sie in eine laufende Bearbeitung von AutoCAD hineinplatzen und VBA dann den lästigen Fehler „Macro kann nicht ausgeführt werden“ ausgibt, müssen Sie prüfen, ob sich AutoCAD sozusagen im Leerlauf befindet. Diesem Zweck dient die Prozedur *GetAcadState*. Sie gibt ein Objekt zurück, das neben dem üblichen *Application*-Objekt auch die Information zum Zustand der AutoCAD-Sitzung enthält. Ist die *IsQuiescent*-Eigenschaft dieses Objekts auf den Wert *TRUE* gesetzt, ist AutoCAD im Ruhezustand, andernfalls wird gerade ein Befehl ausgeführt.

```
Dim State As AcadState
Set State = GetAcadState
If State.IsQuiescent Then
    MsgBox "AutoCAD ist aufnahmenbereit."
Else
    MsgBox "AutoCAD ist beschäftigt."
End If
```

GetInterfaceObject

Parameter: *ByVal ProgID As String*

Rückgabedatentyp: *Object*

Hierbei handelt es sich um eine sehr interessante Funktion, die das *Application*-Objekt zur Verfügung stellt. Sie können dieser Funktion eine Programm-ID übergeben, woraufhin AutoCAD versucht, das dazugehörige ActiveX-Programm in AutoCAD einzubinden. Wenn die

Prozedur erfolgreich ausgeführt wurde, erhalten Sie ein Objekt, das alle Eigenschaften und Methoden dieses ActiveX-Programms zur Verfügung stellt. Die Programm-ID kennen Sie bereits aus Kapitel 3. Dort wurde diese ID verwendet, um Steuerelemente zur Laufzeit zu erstellen (*Controls*-Auflistung).

```
Set AppObj = Application.GetInterfaceObject("Forms.Label.1")
```

Wenn Sie diese Funktion schrittweise in VBA ausführen und sich das Ergebnis (*AppObj*) im Überwachungsfenster anschauen, werden Sie erkennen, dass die komplette Typenbibliothek dieses Programms verfügbar ist, obwohl das ActiveX-Programm nicht in der Objektbibliothek von VBA eingebunden wurde.

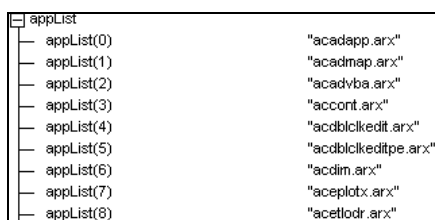
ListArx

Rückgabedatentyp: *Variant*

Diese Funktion gibt ein Array mit allen geladenen ARX-Anwendungen zurück. Wurden keine ARX-Anwendungen geladen, erhalten Sie eine leere Liste. Nachdem Sie folgende Zeilen ausgeführt haben,

```
Dim appList As Variant
appList = Application.ListArx
```

könnte die zurückgegebene Liste wie in Abbildung 5.3 aussehen.



Index	Value
appList(0)	"acadapp.arx"
appList(1)	"acadmap.arx"
appList(2)	"acadvba.arx"
appList(3)	"acctnt.arx"
appList(4)	"acdbtblckedit.arx"
appList(5)	"acdbtblckeditpe.arx"
appList(6)	"acdim.arx"
appList(7)	"aceplotx.arx"
appList(8)	"acetlodr.arx"

Abbildung 5.3: Geladene ARX-Anwendungen

LoadArx/UnloadARX

Parameter: *ByVal Name As String*

Müssen Sie eine ARX-Anwendung über die VBA-Schnittstelle laden oder aus AutoCAD wieder entfernen, so können Sie dies mithilfe einer der beiden Funktionen erledigen. *LoadARX* lädt die angegebene ARX-Datei, während *UnloadARX* diese entfernt, falls sie sich in der AutoCAD-Umgebung befindet. Wenn Sie im Parameter *Name* nicht den vollständigen Namen der Anwendung angeben, um die Applikation zu laden, müssen Sie darauf achten, dass sich die Anwendung in einem der Suchverzeichnisse von AutoCAD befindet.

LoadDVB/UnloadDVB

Parameter: *ByVal Name As String*

Ähnlich den vorherigen Anweisungen, können Sie auch mit diesen Funktionen Anwendungen in AutoCAD laden und entladen. Bei den Applikationen, die mit diesen Funktionen angesprochen werden, handelt es sich allerdings um VBA-Projekte. Ansonsten gelten dieselben Bedingungen wie zuvor.

Quit

Wie der Name der Prozedur bereits verrät, können Sie damit AutoCAD zum Beenden der Sitzung veranlassen. Sie kann allerdings nur ausgeführt werden, wenn sich AutoCAD im Ruhezustand befindet – lässt sich mit der Funktion *GetAcadState* prüfen –, da sonst ein Fehler ausgelöst wird. Sollten noch Zeichnungen in AutoCAD geöffnet sein, deren Änderungen noch nicht gespeichert wurden, wird eine Abfrage durch AutoCAD automatisch durchgeführt.

RunMacro

Parameter: *ByVal MacroPath As String*

AutoCAD bietet die Möglichkeit, über den VBA-Manager Makros ausführen zu lassen. Innerhalb der Entwicklungsumgebung werden die Prozeduren normalerweise intern aufgerufen, was aber nicht heißen soll, dass es in VBA keine Möglichkeit gibt, ein Makro im Stile des Managers auszuführen. Sie geben der Prozedur nur den Namen des Makros mit auf den Weg und AutoCAD führt den Befehl dann aus dem Code heraus aus. Bezüglich der Syntax für die Ausführung eines Makros gelten dieselben Regeln wie beim VBA-Manager.

```
[<Dateiname.dvb>].[ProjektName.][ModuleName.]MacroName
```

```
AcadApp.RunMacro Applications.dvb!ThisDrawing.Example_RunMacro
```

Update

Mit dieser Anweisung aktualisieren Sie das AutoCAD-Anwendungsfenster.

5.1.4 Zoomfunktionen des Anwendungsobjekts

Die nun folgenden Funktionen sind besonders wichtig für die Arbeit innerhalb einer Zeichnung. Es handelt sich dabei um eine ganze Reihe von Zoomfunktionen, die die gleiche Wirkung wie die AutoCAD-Zoombefehle haben. Sie werden diese Funktion erfahrungsgemäß häufiger verwenden, um den Zeichnungsinhalt ohne Eingreifen des Benutzers auf eine bestimmte Stelle zu positionieren oder die Zeichnungsgrenzen neu zu ermitteln. Normalerweise können Sie die Zoomfunktion anwenden, auch wenn ein AutoCAD-Befehl ausgeführt wird. Einige dieser Funktionen können allerdings zu ungewöhnlichen Ergebnissen führen, wenn Sie sie als transparente Befehle ausführen. Sie sollten aus diesem Grund immer ihre selbst definierten Zoomfunktion im transparenten Modus prüfen, um sicherzustellen, dass sie ordnungsgemäß funktionieren.

ZoomAll

Die erste Zoomfunktion entspricht in ihrer Wirkungsweise der des AutoCAD-Befehls `_ZOOM AL`.

```
Application.ZoomAll
```

ZoomCenter

Parameter: *ByVal Center, ByVal Magnify As Double*

Mit der zweiten Zoomfunktion können Sie das Ansichtsfenster auf einen vorgegebenen Zentrumspunkt positionieren. Über den zweiten Parameter legen Sie die Vergrößerung des Ausschnitts fest. Der hier festgelegte Wert entspricht der Höhe des sichtbaren Bereichs. Den

aktuellen Wert für die Höhe können Sie mit der Systemvariablen *VIEWSIZE* ermitteln. Verringern Sie den Wert für die Höhe, führt das zu einer höheren Vergrößerung. Durch die Eingabe eines größeren Werts wird die Vergrößerung verringert.

```
ActSize = ThisDrawing.GetVariable("VIEWSIZE")
Center = ThisDrawing.Utility.GetPoint(, "Zentrum wählen")
Application.ZoomCenter Center, ActSize
```

ZoomExtents

Auch für das Zoomen auf die Zeichnungsgrenzen stellt das Anwendungsobjekt eine Funktion zur Verfügung. Rufen Sie einfach *ZoomExtents* auf und die Prozedur wird diese Sache für Sie erledigen.

```
Application.ZoomExtents
```

ZoomPickWindow

Für das Zoomen auf einen Ausschnitt stellt das Anwendungsobjekt zwei Funktionen zur Verfügung. Die erste brauchen Sie nur zu aktivieren. Die Benutzerabfrage der linken unteren und der rechten oberen Ecke übernimmt dann das Programm. Nachdem der Benutzer das entsprechende Fenster gewählt hat, zoomt die Prozedur auf den entsprechenden Ausschnitt.

```
Application.ZoomPickWindow
```

ZoomPrevious

Um auf den jeweils vorherigen Zeichnungsausschnitt zu zoomen, müssen Sie diese Prozedur verwenden. Sie entspricht dem AutoCAD-Befehl *ZOOM V*.

```
Application.ZoomPrevious
```

ZoomScaled

Parameter: *ByVal Scale As Double, ByVal ScaleType As AcZoomScaleType*

Diese Prozedur dient dem Vergrößern bzw. Verkleinern des aktuellen Zeichnungsausschnitts. Je nachdem, welchen Wert Sie für den Parameter *Scale* verwenden, wird der Ausschnitt vergrößert oder verkleinert. Ausgangspunkt ist der Faktor 1. Bei diesem Wert bleibt der Ausschnitt in der aktuellen Größe bestehen. Wählen Sie einen größeren Wert, wird der Ausschnitt vergrößert. Ist der Wert kleiner, dann wird er verkleinert.

Im zweiten Parameter (*ScaleType*), können Sie noch die Art der Skalierung festlegen. Hierfür stehen Ihnen folgende Konstanten zur Verfügung:

- *acZoomScaledAbsolute* – relativ zu den Zeichnungsgrenzen zoomen
- *acZoomScaledRelative* – relativ zur aktuellen Ansicht zoomen
- *acZoomScaledRelativePSpace* – relativ zu den Einheiten des Papierbereichs zoomen

```
ScaleFactor = 0.5
ScaleType = acZoomScaledRelative
Application.ZoomScaled ScaleFactor, ScaleType
```


ZoomWindow

Parameter: *ByVal LowerLeft*, *ByVal UpperRight*

Dies ist die zweite Funktion, die das Zoomen eines Ausschnitts ermöglicht. Im Gegensatz zu *ZoomPickWindow* müssen Sie hier allerdings die Koordinaten des Fensters selbst definieren, entweder durch Benutzerabfrage oder durch eine andere Methode. Diese Eckpunkte geben Sie dann in den Variablen *LowerLeft* und *UpperRight* an die Funktion weiter und schon zoomt die Funktion auf die angegebenen Koordinaten.

```
LowerLeft = ThisDrawing.Utility.GetPoint(, "linke untere Ecke")
UpperRight = ThisDrawing.Utility.GetCorner(LowerLeft, _
    "rechte obere Ecke")
Application.ZoomWindow LowerLeft, UpperRight
```

5.2 Verwaltung der AutoCAD-Menüstruktur

Für die Verwaltung der Menüstruktur in AutoCAD stellt die Klasse *AcadApplication* das Objekt *MenuGroups* zur Verfügung, mit dem Sie die gesamte Struktur, seien es die Menüs oder die Werkzeugkästen, verwalten bzw. anpassen können. Dieses Objekt enthält alle geladenen Menüdateien, einschließlich der dargestellten Menüstruktur von AutoCAD. Die aktive Menüstruktur wird zusätzlich im Objekt *MenuBar* verwaltet. Um es Ihnen etwas verständlicher zu machen, werden wir das Ganze einmal optisch untermauern. Starten Sie zu diesem Zweck die Menüanpassungsfunktion von AutoCAD (Befehl `_MENULOAD`).

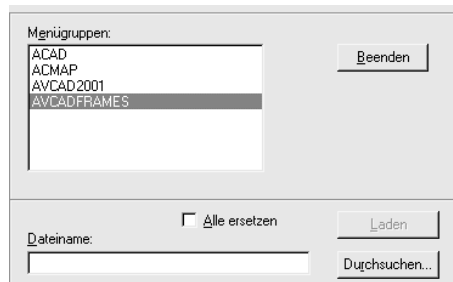


Abbildung 5.4: Menügruppenbeispiel

Nachdem das Dialogfenster der Menüverwaltung gestartet wurde, erhalten Sie einen Überblick über alle geladenen Menüs. Sämtliche Menüs sind auch im *MenuGroups*-Objekt zusammengefasst. Das bedeutet, dass Sie auf diese Menüs zurückgreifen können, indem Sie sich der Eigenschaften und Methoden des *MenuGroups*-Objekts bedienen. Wenn Sie nun auf den Reiter `MENÜLEISTE` dieses Dialogfensters umschalten, sehen Sie eine Zusammenstellung der Menüleiste, wie sie beispielsweise in Abbildung 5.5 zu sehen ist.



Abbildung 5.5: AutoCAD-Menüleiste

Da diese Menüleiste aus Menüs unterschiedlicher Gruppen zusammengestellt wird, übernimmt das Objekt *MenuBar* die Verwaltung dieser speziellen Menügruppe. Zugriff auf diese beiden Objekte erhalten Sie mittels folgender Codezeilen:

```
Dim MGroups As AcadMenuGroups  
Set MGroups = Application.MenuGroups
```

und

```
Dim MBar As AcadMenuBar  
Set MBar = Application.MenuBar
```

Sie müssen die Objekte nicht unbedingt an die Variablen übergeben, sondern können auch direkt mit den jeweiligen Objektvariablen arbeiten.

5.2.1 Menügruppen

Beschäftigen wir uns zunächst mit den Menügruppen. Dieses Objekt bildet die oberste Instanz, über die Sie auf alle Menüs und Werkzeugkästen zugreifen können. Dazu stehen Ihnen folgende Eigenschaften und Methoden dieser Klasse zur Verfügung.

Eigenschaften

Application

(schreibgeschützt)

Datentyp: *AcadApplication*

Gibt das Anwendungsobjekt (Klasse *AcadApplication*) zurück.

Count

(schreibgeschützt)

Datentyp: *Long*

Mithilfe dieser Eigenschaft können Sie die Anzahl der geladenen Menüdateien ermitteln, wobei jede Datei einen Gruppeneintrag darstellt.

Parent

(schreibgeschützt)

Datentyp: *AcadApplication*

Dieses Objekt gibt das der *MenuGroups*-Klasse übergeordnete Objekt (*Application*) zurück.

Prozeduren

Item

Parameter: *Index*

Rückgabedatentyp: *AcadMenuGroup*

Über dieses Objekt können Sie eine bestimmte Menügruppe abrufen.

Load

Parameter: *MenuFileName* As String, [*BaseMenu*]

Rückgabedatentyp: *AcadMenuGroup*

Dieser Befehl lädt eine Menügruppe aus einer Menüdatei, die als Parameter an diese Funktion übergeben wird.

Da die Klasse *MenuGroups* keine *Add*-Methode anbietet, müssen Sie auf die *Load*-Anweisung zurückgreifen, um eine Gruppe hinzuzufügen. Angenommen, Sie wollen die Menüdatei *Meinmenu.mnu*, welche sich im Verzeichnis *C:\VBA* befindet, zur Menügruppe hinzufügen. Sie müssten in diesem Fall folgenden Code verwenden:

```
Application.MenuGroups.Load "C:\VBA\MeinMenu.mnu"
```

Wie Sie bereits bei der Eingabe der Codezeile erkannt haben dürften, können Sie dieser Prozedur noch einen zweiten, optionalen Parameter mitgeben. Wenn Sie ihn verwenden, kann er folgende Auswirkungen haben.

- **TRUE:** Das angegebene Menü wird als Basismenü interpretiert und geladen. Das bedeutet, dass alle bereits geladenen Menügruppen entfernt und durch die im ersten Parameter angegebene Menüdatei ersetzt werden. Das Setzen des Parameters mit diesem Wert hat somit die gleiche Wirkung, wie wenn Sie den AutoCAD-Befehl *_Menu* zum Laden einer Menüdatei verwenden.
- **FALSE:** Setzen Sie den Parameter auf diesen Wert oder lassen Sie ihn weg, wird die zu ladende Menüdatei als Teilmenü aufgefasst. Das neue Menü wird somit zu den bereits vorhandenen Menügruppen hinzugefügt. In seiner Wirkungsweise entspricht diese Methode dem AutoCAD-Befehl *_MenuLoad*.

Um die neue Menügruppe sofort weiterzuverarbeiten, können Sie einer Objektvariable das Ergebnis der *Load*-Funktion zuweisen lassen.

```
Dim MGroups As AcadMenuGroup  
Dim MenuFile As String  
MenuFile = "C:\VBA\MeinMenu.mnu"  
Set MGroup = Application.MenuGroups.Load MenuFile
```

Wenn Sie eine vorhandene Menügruppe bearbeiten wollen, können Sie diese mithilfe der *Item*-Funktion separieren und anschließend die Eigenschaften und Prozeduren zur Anpassung der entsprechenden Gruppe verwenden.

```
Dim MGroups As AcadMenuGroup  
Set MGroup = Application.MenuGroups.Item(Index)
```

Für den Parameter *Index* verwenden Sie entweder eine Zahl, die die Position innerhalb der Gruppenuflistung festlegt, oder den Namen der jeweiligen Gruppe.

```
Set MGroup = Application.MenuGroups.Item(0)
```

oder

```
Set MGroup = Application.MenuGroups.Item("Acad")
```

Sollten Sie die Menügruppe über den Namen abrufen und hierfür eine Variable benutzen, müssen Sie darauf achten, dass diese Variable mit dem Datentyp *String* deklariert worden ist.

```
Dim MnuName As String  
MnuName = "Acad"  
Set MGroup = Application.MenuGroups.Item(MnuName)
```

5.2.2 Einzelne Menügruppe bearbeiten

Nachdem eine Menügruppe ausgewählt wurde, können Sie über die Eigenschaften und Methoden, die die Klasse *MenuGroup* zur Verfügung stellt, die Menüs und Werkzeugleisten der Menügruppe weiterverarbeiten. Bevor Sie sich anhand von einigen Beispielen mit der Funktionsweise dieser Prozeduren vertraut machen, folgt zunächst einmal eine Auflistung mit einer Kurzbeschreibung der wichtigsten Elemente der Klasse *MenuGroup*.

Eigenschaften

Application

(schreibgeschützt)

Datentyp: *AcadApplication*

Gibt das Anwendungsobjekt (Klasse *AcadApplication*) zurück

MenuFileName

(schreibgeschützt)

Datentyp: *String*

Diese Eigenschaft gibt den Namen der Menüdatei zurück, die mit der entsprechenden Gruppe verbunden ist. Da diese Eigenschaft schreibgeschützt ist, können Sie den Namen nur ermitteln und nicht ändern.

Menus

(schreibgeschützt)

Datentyp: *AcadPopupMenu*

Dieses Objekt enthält alle Menüs, die in der gewählten Gruppe enthalten sind. Verwenden Sie dieses Objekt, um Zugriff auf die einzelnen Menüeinträge zu erhalten.

Name

(schreibgeschützt)

Datentyp: *String*

Über diese Eigenschaft können Sie den Namen der Menügruppe ermitteln. Diesen Namen lassen sich nutzen, um z.B. die Menügruppe über die *Item*-Funktion der Klasse *MenuGroups* abzurufen.

Parent

(schreibgeschützt)

Datentyp: *AcadMenuGroups*

Dieses Objekt gibt die der Menügruppe übergeordnete Menügruppenauflistung zurück.

Toolbars*(schreibgeschützt)*Datentyp: *AcadToolbars*

Dieses Objekt enthält alle Werkzeugkästen, die in der gewählten Gruppe enthalten sind. Verwenden Sie dieses Objekt, um Zugriff auf die einzelnen Werkzeugschaltflächen zu erhalten.

Type*(schreibgeschützt)*Datentyp: *AcMenuGroupType*

Mithilfe dieser Eigenschaft ermitteln Sie, um welchen Typ es sich bei der Menügruppe handelt. Als Ergebnis der Abfrage kommen folgende Möglichkeiten in Betracht:

- *acBaseMenuGroup*: Bei der Menügruppe handelt es sich um ein Basismenü.
- *acPartialMenuGroup*: Bei der Menügruppe handelt es sich um ein Teilmenü, also um eines von mehreren Menügruppen.

Prozeduren**Save**Parameter: *MenuFileType As AcMenuFileType*

Wenn Sie diese Anweisung ausführen, werden alle Änderungen, die Sie an der ausgewählten Menügruppe bis dato vorgenommen haben, in der Menüdatei gespeichert, die unter der Eigenschaft *MenuFileName* angegeben ist. Mit dem Parameter der Funktion können Sie außerdem festlegen, mit welchem Menütyp die Datei erzeugt werden soll. Folgende Möglichkeiten stehen hierfür zur Auswahl.

- *AcMenuFileCompiled* – kompilierte Menüdatei erzeugen
- *AcMenuFileSource* – Menüdatei im ASCII-Format erzeugen

```
MGroup.Save AcMenuFileSource
```

SaveAsParameter: *MenuFileName As String, MenuFileType As AcMenuFileType*

Genau wie die *Save*-Anweisung speichert auch diese Funktion alle Änderungen der Menügruppe. Allerdings können Sie bei dieser Funktion festlegen, in welcher Datei die Daten gespeichert werden sollen. Für den Typ der Menüdatei gelten die gleichen Bedingungen wie bei der *Save*-Anweisung.

- *AcMenuFileCompiled* – kompilierte Menüdatei erzeugen
- *AcMenuFileSource* – Menüdatei im ASCII-Format erzeugen

```
MGroup.SaveAs "C:\MeinMenu", AcMenuFileSource
```

Unload

Diese Anweisung ermöglicht es Ihnen, die komplette Menügruppe aus AutoCAD zu entfernen.

```
MGroup.Unload
```

Nachdem Sie einen kurzen Überblick über einige wichtige Elemente der Klasse *MenuGroup* erhalten haben, werden nun einige Methoden detaillierter besprochen.

Die erste Anweisung, die anhand einiger Codezeilen beschrieben wird, ist die *SaveAs*-Funktion. Wie zuvor bereits angemerkt, können Sie mit dieser Anweisung die Änderungen, die Sie an der Menügruppe vorgenommen haben, in einer Menüdatei speichern. Im folgenden kleinen Beispiel sollen die Daten der *Acad*-Menügruppe in der Datei *MeinMenü* gespeichert werden.

```
Dim MnuName As String
MnuName = "Acad"
Set MGroup = Application.MenuGroups.Item(MnuName)
MGroup.SaveAs "C:\MeinMenu", acMenuFileCompiled
```

Der erste Parameter, den Sie dieser Funktion übergeben, legt den Namen der Menüdatei fest, in der Sie die Daten der entsprechenden Menügruppe speichern wollen. Der zweite Parameter, der ebenfalls zwingend erforderlich ist, legt das Format fest, unter dem die Menüdatei gespeichert werden soll. Sie haben dabei die Auswahl zwischen folgenden Möglichkeiten:

- *acMenuFileCompiled*: Setzen Sie für den Parameter diese Konstante, wenn Sie die Daten der Menügruppe in einer kompilierten Menüdatei speichern möchten. Die Datei erhält dann die Dateierweiterung *Mnc*.
- *acMenuFileSource*: Sollten Sie diese Konstante als Parameter verwenden, so werden die Daten in einer Quellcodedatei gespeichert. Die Datei erhält in diesem Fall die Dateierweiterung *Mns*.

Für den Fall, dass Sie lediglich die *Save*-Anweisung zum Speichern der Menüdatei verwenden, fällt der erste Parameter weg und Sie müssen nur noch das Format der Menüdatei, also kompiliert oder Quellcode, festlegen.

```
Dim MnuName As String
MnuName = "Acad"
Set MGroup = Application.MenuGroups.Item(MnuName)
MGroup.Save acMenuFileCompiled
```

5.2.3 Verwaltung der Pull-down-Menüs in AutoCAD

Im vorherigen Abschnitt erfuhren Sie, in welchen Objekten sich die Menüs und Werkzeugkästen der jeweiligen Menügruppe verbergen. Kommen wir also nun zur Verwaltung der einzelnen Menüstrukturen und deren Menüeinträge. Wie vorher bereits angesprochen, greifen wir dafür auf das Objekt *Menus* der ausgewählten Menügruppe zurück. Dieses Objekt enthält eine Auflistung aller Menüeinträge der Datei, wobei jeder Eintrag der Auflistung einem Hauptmenü entspricht. In der Tabelle 5.1 sehen Sie eine Übersicht über einige Einträge der Datei *Acad.mn**, die mit folgender Codezeile abgerufen werden kann:

```
Application.MenuGroups("acad").Menus
```

Auflistung	Eintrag
.Item(0)	Objektfang-Cursormenü
.Item(1)	Datei
.Item(2)	Bearbeiten
.Item(3)	Ansicht

Tabelle 5.1: Menüauflistung der Menüdatei *Acad*n

Auflistung	Eintrag
.Item(4)	Einfügen
.Item(5)	Format
.Item(6)	Extras
.Item(7)	Zeichnen
.Item(n)	

Tabelle 5.1: Menüauflistung der Menüdatei Acadn (Forts.)

Da jeder Eintrag dieser Auflistung einem Menü und nicht einem einzelnen Menüeintrag entspricht, werden alle Elemente dieser Auflistung als Objekte vom Datentyp *AcadPopupMenu* zurückgegeben. Jedes dieser Elemente enthält somit eine Auflistung weiterer Objekte, die dann dem Typ *AcadPopupMenuItem* entsprechen. Bevor wir uns nun den einzelnen Menüeinträgen zuwenden, werden wir die Klasse *AcadPopupMenu* näher erläutern.

5.2.4 Verwendung der Klasse AcadPopupMenu

Diese Klasse wird immer dann verwendet, wenn es sich bei dem zurückgegebenen Objekt um ein Pull-down- oder Shortcut-Menü handelt. Bevor wir anhand von Beispielen einige Funktionen dieser Klasse näher betrachten, sollten Sie sich die Eigenschaften und Methoden anschauen.

Eigenschaften

Application

(schreibgeschützt)

Datentyp: *AcadApplication*

Gibt das Anwendungsobjekt (Klasse *AcadApplication*) zurück.

Count

(schreibgeschützt)

Datentyp: *AcadApplication*

Mit dieser Eigenschaft können Sie die Anzahl von Einträgen des Menüs abrufen.

Name

Datentyp: *String*

Gibt den Namen des Menüs zurück, z.B. „&Datei“.

NameNoMnemonic

(schreibgeschützt)

Datentyp: *String*

Gibt den Namen des Menüs ohne Mnemonic (Zugriffstaste) zurück, z.B. „Datei“.

OnMenuBar
(schreibgeschützt)Datentyp: *Boolean*

Mithilfe dieser Eigenschaft können Sie prüfen, ob das Menü bereits in der AutoCAD-Menüleiste enthalten ist. Gibt *OnMenuBar* den Wert *TRUE* zurück, dann wurde das entsprechende Menü der AutoCAD-Menüleiste bereits zugeordnet.

Parent
(schreibgeschützt)Datentyp: *Object*

Dieses Objekt gibt das Objekt zurück, das dem aufrufenden Pull-down-Menü übergeordnet ist.

ShortcutMenu
(schreibgeschützt)Datentyp: *Boolean*

Diese Eigenschaft zeigt an, ob es sich bei dem Menü um ein Kontextmenü handelt oder nicht.

TagString
(schreibgeschützt)Datentyp: *String*

Die Eigenschaft enthält einen bestimmten ID-Text, der den Menüpunkt innerhalb einer Menüdatei eindeutig identifiziert. Dieser Text besteht aus alphanumerischen Zeichen und Unterstrichen (_), z.B. *ID_MnFile*.

Prozeduren***AddMenuItem***Parameter: *Index, Label As String, Macro As String*Rückgabotyp: *AcadPopupMenuItem*

Mit dieser Anweisung können Sie dem Menü einen Menüeintrag, der Befehle ausführen kann, hinzufügen. Im ersten Parameter geben Sie die Position des Menüpunkts an, vor dem der neue Eintrag eingefügt werden soll. Um den Menüpunkt am Ende des Menüs einzufügen, müssen Sie einen Wert angeben, der größer ist als die Positionsangabe des letzten vorhandenen Menüpunkts ist. Der zweite Parameter legt die Beschriftung des Menüpunkts fest und der Dritte die AutoCAD-Befehlsfolge.

```
ActMenu.AddMenuItem ActMenus.Count, "Linie zeichnen", "^C^C_Line"
```

AddSeparatorParameter: *Index*Rückgabotyp: *AcadPopupMenuItem*

Auch diese Funktion fügt einen neuen Eintrag dem aufrufenden Pull-down- oder Shortcut-Menü hinzu. Das Besondere ist allerdings, dass es sich bei diesem Eintrag um einen Menütren-

ner handelt. Aus diesem Grund müssen Sie auch lediglich die Position angeben, für die dieselben Vorgaben gelten wie für Menüpunkte, die Befehle ausführen können.

```
ActMenu.AddSeparator ActMenus.Count
```

AddSubMenu

Parameter: *Index, Label As String*

Rückgabetyt: *AcadPopupMenu*

Dieser Befehl dient dem Hinzufügen eines Untermenüs zum aufrufenden Menü. Das so entstandene Menü entspricht der Klasse *AcadPopupMenu*, dem wiederum Menüpunkte hinzugefügt werden können. In den Parametern der Funktion müssen Sie neben der Menüposition auch die Beschriftung festlegen.

```
ActMenu.AddSubMenu ActMenus.Count, "Mein Menü"
```

InsertInMenuBar

Parameter: *Index*

Diese Anweisung fügt das Menü zur AutoCAD-Menüleiste hinzu. Die Position, an der das Menü erscheinen soll, wird über den Parameter *Index* festgelegt. Nachdem diese Anweisung erfolgreich ausgeführt wurde, erscheint das Menü auch im *MenuBar*-Objekt und die Eigenschaft *OnMenuBar* wird auf *TRUE* gesetzt.

```
If ActMenu.OnMenuBar = False Then  
    ActMenu.InsertInMenuBar Application.MenuBar("Hilfe")  
End If
```

Item

Parameter: *Index*

Rückgabetyt: *AcadPopupMenuItem*

Gibt das Untermenü des Menüs an der angegebenen Position zurück.

RemoveFromMenuBar

Parameter: *Index*

Diese Anweisung entfernt das Menü wieder aus der AutoCAD-Menüleiste.

```
If ActMenu.OnMenuBar = True Then  
    ActMenu.RemoveFromMenuBar  
End If
```

5.2.5 Menüeinträge hinzufügen

Um Einträge zu einem Menü hinzuzufügen, verwenden Sie die Funktionen *AddMenuItem*, *AddSeparator* oder *AddSubMenu*. Jede dieser Funktionen erstellt zwar einen neuen Menüeintrag, allerdings erfüllt jeder Eintrag eine andere Funktion. Nehmen wir z.B. die Prozedur *AddMenuItem*, diese fügt einen Menüeintrag hinzu, der einen AutoCAD-Befehl aufnehmen kann, der nach Betätigung des Eintrags durch einen Benutzer ausgeführt wird. Schauen wir uns diese

Funktion einmal an einem kleinen Beispielcode an. Ziel dieses Codes soll es sein, im AutoCAD-Menü (Datei *acad.mn**) ein Hauptmenü (*MeinMenü*) und diesem Hauptmenü den Menüeintrag *LINIE ZEICHNEN* hinzuzufügen, der dann den AutoCAD-Befehl *LINIE* ausführt, wenn der Benutzer den Menüeintrag aufruft.

```
Dim AcGroup As AcadMenuGroup
Dim MMenu As AcadPopupMenu
Dim MItem As AcadPopupMenu.Item
On Local Error Resume Next
Set AcGroup = Application.MenuGroups("acad")
Set MMenu = AcGroup.Menus.Add("MeinMenu")
Set MItem = MMenu.AddMenuItem(0, "Linien zeichnen", "^C^C_line ")
```

Nachdem die benötigten Variablen deklariert wurden, wählen wir die Gruppe *Acad*, die ja der Menüdatei *acad.mn** entspricht und fügen dieser Gruppe das Hauptmenü *MeinMenü* hinzu. Mit der Codezeile

```
Set MItem = MMenu.AddMenuItem(0, "Linien zeichnen", "^C^C_line ")
```

wird in diesem Menü dann der Eintrag zum Zeichnen der Linien erstellt. Die Funktion *AddMenuItem* erfordert dabei die Übergabe von drei Parametern. Diese Parameter haben folgende Bedeutung:

- **Index:** Mit dem ersten Parameter können Sie die Position des neuen Menüs innerhalb der Struktur des aufrufenden Menüs bestimmen. Sie können hierfür entweder eine Zahl zwischen 0 und N-1 verwenden oder den Namen eines vorhandenen Menüeintrags. In jedem Fall wird der neue Eintrag an den Eintrag angehängt.
- **Label:** Der zweite Parameter legt den Namen des neuen Menüeintrags fest. Der Eintrag kann sowohl DIESEL-Ausdrücke als auch Tastaturkürzel (&) enthalten.
- **Macro:** Mit dem dritten und letzten Parameter legen Sie schließlich den Befehl fest, der ausgeführt werden soll, wenn der Menüeintrag durch den Benutzer ausgewählt wurde.

Die zweite Funktion (*AddSeparator*) fügt dem aufrufenden Menü eine Trennlinie an einer als Parameter mitgelieferten Position hinzu. Der Parameter entspricht in seiner Verwendung dem ersten Parameter der vorherigen Funktion.

Die dritte und letzte *Add*-Funktion ist die Anweisung *AddSubMenu*. Mit dieser Funktion können Sie ein Untermenü dem aufrufenden Menü hinzufügen. Der Eintrag, der dadurch erzeugt wird, gilt als Eingang zu weiteren Menüeinträgen.

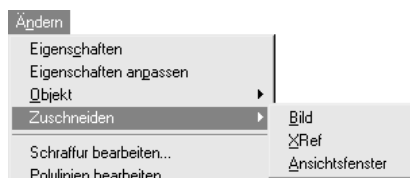


Abbildung 5.6: Beispiel eines Submenüs

Als Ergebnis dieser Funktion wird ein Objekt vom Typ *AcadPopupMenu* erzeugt. Dieses kann an eine Objektvariable zur weiteren Verarbeitung zurückgegeben werden. Im folgenden sehen Sie einen kleinen Beispielcode zur Erzeugung eines Untermenüs.

```

Dim AcGroup As AcadMenuGroup
Dim MMenu As AcadPopupMenu
Dim SMenu As AcadPopupMenu
On Local Error Resume Next
Set AcGroup = Application.MenuGroups("acad")
Set MMenu = AcGroup.Menus.Add("MeinMenu")
Set SMenu = MMenu.AddSubMenu(2, "Ändern")

```

In diesem Beispiel wurde dem Hauptmenü ein Untermenü an Position 3 hinzugefügt. Als Ergebnis des Codes wurde die Objektvariable *SMenu* erzeugt, die als Zugriff für das neue Untermenü dienen soll.

5.2.6 Menü zum AutoCAD-Menü hinzufügen/entfernen

Die AutoCAD-Menüleiste (siehe Abbildung 5.5) enthält ja nur ausgewählte Funktionen aller geladenen Menüs und ist somit rein theoretisch Bestandteil der Klasse *MenuGroups*, die ja den Container aller Menüs und Toolbars bildet. Sie brauchen also kein neues Menü zu erzeugen, sondern können einfach einen Verweis auf das entsprechende Menü in der Menüauflistung erstellen oder gegebenenfalls entfernen. Zu diesem Zweck stellt die Klasse *AcadPopupMenu* die Anweisungen *InsertInMenuBar* und *RemoveFromMenuBar* zur Verfügung.

- *InsertInMenuBar* – Mit dieser Funktion fügen Sie das Menü, mit dem Sie diese Funktion aufrufen, an einer bestimmten Position in der AutoCAD-Menüleiste hinzu. Als Parameter muss nur diese Position mitgeliefert werden. Diese Position muss im Bereich von 0 bis Anzahl der Menüs in der Menüleiste – 1 liegen. Generell wird das aufrufende Menü vor dem mit der Positionsnummer bezeichneten Menü eingefügt. Geben Sie also als Index die 0 an (*MMenu.InsertInMenuBar 0*), so wird das Menü an die erste Position in der AutoCAD-Menüleiste gesetzt.
- *RemoveFromMenuBar* – Um ein Menü wieder aus der AutoCAD-Menüleiste zu entfernen, müssen Sie die Anweisung auf das jeweilige Pop-up-Menü anwenden. Das Menü wird daraufhin aus der Leiste entfernt.

Zusätzlich können Sie auch noch prüfen, ob das jeweilige Menü bereits in der AutoCAD-Menüleiste enthalten ist oder nicht. Sie brauchen zu diesem Zweck nur den Inhalt der Eigenschaft *OnMenuBar* zu überprüfen. Ist das Menü in der Leiste positioniert worden, so wird *TRUE* zurückgegeben, im anderen Fall *FALSE*.

Um nun nicht die gesamten Gruppen zu durchsuchen, um an die Menüs zu gelangen, die in der Menüleiste enthalten sind, können Sie auf das *MenuBar*-Objekt zurückgreifen und sich dort die entsprechenden Pop-up-Menüs herausholen.

```

Dim ActPM As AcadPopupMenu
For i = 0 To MenuBar.Count - 1
    Set ActPM = MenuBar(i)
Next

```

Die so erhaltene Variable ist dann wieder vom Datentyp *AcadPopupMenu* und Sie können wie gehabt fortfahren.

5.2.7 Verwendung der Klasse *AcadPopupMenuItem*

Diese Klasse stellt einen einzelnen Menüeintrag dar, der entweder einen AutoCAD-Befehl als Ergebnis seiner Aktivierung ausführt oder Untermenüs enthält. Sie erhalten ein Objekt dieses Typs, als Ergebnis der Funktionen *AddMenuItem* und *AddSeparator*, die Sie auf ein Menü der Klasse *AcadPopupMenu* anwenden, oder wenn Sie ein Element des Pop-up-Menüs mit der *Item*-Funktion abrufen. Bevor wir einzelne Elemente dieser Klasse besprechen, verschaffen wir uns erst einmal einen Überblick über die Eigenschaften und Funktionen dieser Klasse.

Eigenschaften

Application

(schreibgeschützt)

Datentyp: *AcadApplication*

Gibt das Anwendungsobjekt (Klasse *AcadApplication*) zurück.

Caption

(schreibgeschützt)

Datentyp: *String*

Diese Eigenschaft enthält den Menütext. Sie können diesen Text mit dieser Eigenschaft nur abrufen. Wenn Sie den Text ändern möchten, müssen Sie dies mit der Eigenschaft *Label* vornehmen.

Check

Datentyp: *Boolean*

Über diese Eigenschaft können Sie ein Häkchen vor einem Menütext aktivieren bzw. deaktivieren oder prüfen, ob ein Menüpunkt mit Häkchen versehen wurde oder nicht.

Enable

Datentyp: *Boolean*

Über diese Eigenschaft können Sie ermitteln bzw. festlegen, ob der Menüeintrag aktiviert oder gesperrt ist.

EndSubMenuLevel

Datentyp: *Long*

Diese Eigenschaft gibt an, ob es sich bei diesem Menüeintrag um den letzten Eintrag eines Untermenüs handelt oder nicht. Sie kann folgende Werte annehmen:

- 0: Der Eintrag ist nicht der letzte Eintrag.
- 1: Es handelt sich um den letzten Eintrag.
- 2: Es handelt sich um den letzten Eintrag.

HelpString

Datentyp: *String*

Mit dieser Eigenschaft können Sie den Hilfetext, der in der Statuszeile von AutoCAD angezeigt wird, festlegen bzw. abrufen.

Index**(schreibgeschützt)**Datentyp: *Long*

Über diese Eigenschaft können Sie die Position ermitteln, an der sich der Menüeintrag innerhalb des Menüs befindet.

LabelDatentyp: *String*

Diese Eigenschaft enthält, genau wie die *Caption*-Eigenschaft, den Menütext, den Sie bei dieser Eigenschaft allerdings ändern können.

MacroDatentyp: *String*

Hier können Sie die AutoCAD-Befehlsfolge festlegen oder abrufen, die ausgeführt werden soll, nachdem der Menüpunkt aufgerufen worden ist.

Parent**(schreibgeschützt)**Datentyp: *AcadPopupMenu*

Dieses Objekt gibt das Objekt zurück, das dem aufrufenden Menüeintrag übergeordnet ist.

SubMenu**(schreibgeschützt)**Datentyp: *AcadPopupMenu*

Über dieses Objekt können Sie alle diesem Menü zugeordneten Untermenüeinträge abrufen. Enthält der jeweilige Menüpunkt keine Untermenüpunkte, wird als Ergebnis der Wert *Nothing* zurückgegeben.

TagStringDatentyp: *String*

Die Eigenschaft enthält einen bestimmten ID-Text, der den Menüpunkt innerhalb einer Menüdatei eindeutig identifiziert. Dieser Text besteht aus alphanumerischen Zeichen und Underlines (), z.B. `ID_MnFile`.

Type**(schreibgeschützt)**Datentyp: *AcMenuItemType*

Diese Eigenschaft gibt den Typ des aktuellen Menüeintrags zurück. Diese Eigenschaft enthält immer den Wert einer der folgenden Konstanten.

Konstante	Beschreibung	Wert
<i>acMenuItem</i>	Der aufrufende Menüeintrag ist ein Menüpunkt.	0
<i>acMenuSeparator</i>	Der aufrufende Menüeintrag ist ein Menütrenner.	1
<i>acMenuSubMenu</i>	Der aufrufende Menüeintrag ist ein Untermenü, das weitere Menüs enthält.	2

Tabelle 5.2: Konstanten der Type-Eigenschaft

Prozeduren

Delete

Mithilfe dieser Anweisung können Sie den Menüeintrag aus dem Menü entfernen.

5.2.8 Beispielprojekt

(Beispielprojekt: 05\ShortcutMenu.dvb)

Wenden wir uns nun einigen Beispielen zum Thema *PopupMenu* und *PopupMenuItem* zu. Es kommt schon mal vor, dass man Zeichnungen erhält oder selber erstellt, in denen eine Unmenge von Layouts enthalten sind. Bei mir sind das meist Pläne, für die ich dann Querprofil erstellen muss. Jedes Querprofil wird für sich in einem Layout erzeugt, wobei mal schnell 50 oder mehr zusammenkommen. Nun besteht die „Schwierigkeit“, zwischen den einzelnen Layouts schnell hin- und herzuschalten.

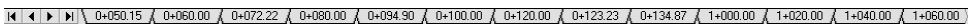


Abbildung 5.7: Beispiel für eine Vielzahl an Layouts

Ich bin dann auf die Idee gekommen, diese Layouts in ein Shortcut-Menü zu packen und dort zur Auswahl freizugeben, so dass nach Betätigung der rechten Maustaste im AutoCAD-Zeichnungsfenster, die ja das Befehlsmenü aufruft, alle vorhandenen Layouts an das Menü angehängt werden. Zusätzlich habe ich noch einen Schalter hinzugefügt, mit dem das Hinzufügen der Layouts aktiviert bzw. deaktiviert werden kann. Das Ergebnis dieser Prozeduren können Sie in Abbildung 5.8 sehen.

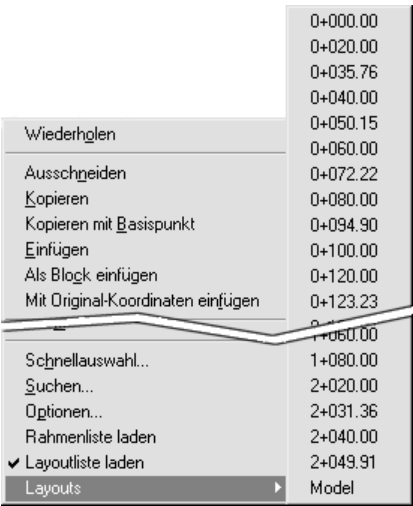


Abbildung 5.8: Layoutauflistung im Shortcut-Menü

Wie das Ganze nun im Quellcode zusammengestellt wird, werden wir uns im Folgenden genauer anschauen. Alle drei Prozeduren werden im Modul *ThisDrawing* positioniert, was bei der ersten Prozedur (*AcadDocument_BeginShortcutMenuDefault*) zwingend erforderlich ist, da es sich dabei um eine Ereignisprozedur des AutoCAD-Dokuments handelt, die ausgelöst wird, wenn der Benutzer die rechte Maustaste im Zeichnungsfenster betätigt. Sie können diese Prozedur erstellen, indem Sie die Prozedur aus der Prozedurliste im Codefenster auswählen.

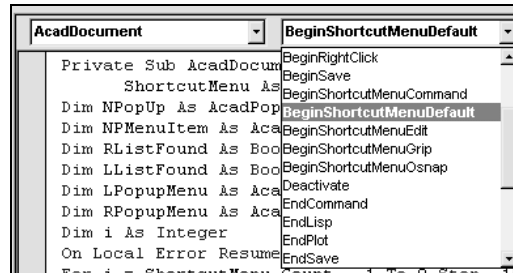


Abbildung 5.9: Ereignisprozedur auswählen

In dieser Prozedur erfolgt der eigentliche Ladevorgang der Layoutliste. Schauen wir uns zunächst den gesamten Code dieser Prozedur an und lassen ihn auf uns wirken.

```
Private Sub AcadDocument_BeginShortcutMenuDefault( _
    ShortcutMenu As IAcadPopupMenu)
    Dim NPopUp As AcadPopupMenu
    Dim NMenuItem As AcadPopupMenuItem
    Dim LListFound As Boolean
    Dim LPopupMenu As AcadPopupMenuItem
    Dim i As Integer
    On Local Error Resume Next
    For i = ShortcutMenu.Count - 1 To 0 Step -1
        If ShortcutMenu(i).Caption = "Layoutliste laden" Then
            LListFound = True
            Set LPopupMenu = ShortcutMenu(i)
        End If
        If ShortcutMenu(i).Caption = "Layouts" Then
            ShortcutMenu(i).Delete
        End If
    Next
    If LListFound = False Then
        Set LPopupMenu = ShortcutMenu.AddMenuItem(ShortcutMenu.Count, _
            "Layoutliste laden", "-vbarun LayoutItemClick" & vbCr)
    End If
    LPopupMenu.Check = CBool(GetSetting("Frames", "Setup", _
        "LoadLayouts", 1))
    If LPopupMenu.Check = True Then
        Set NPopUp = ShortcutMenu.AddSubMenu("Layouts", "Layouts")
        For i = 0 To ThisDrawing.Layouts.Count - 1
            NPopUp.AddMenuItem i, ThisDrawing.Layouts(i).Name, _
                "-vbarun setLayout " & ThisDrawing.Layouts(i).Name & vbCr
        Next
    End If
End Sub
```

Was passiert nun im Einzelnen in dieser Prozedur? Nun, zunächst werden, wie gehabt, die einzelnen Variablen deklariert. Wofür diese Variablen benötigt werden, wird sich im Verlauf der Erläuterung ergeben. Im ersten Abschnitt dieser Prozedur werden die einzelnen Einträge des Menüs durchlaufen, um zum einen den aktuellen Wert (LAYOUTS LADEN aktiviert oder deaktiviert) an den Menüeintrag LAYOUTLISTE LADEN zu übergeben. Zum anderen soll auch der Menüeintrag LAYOUTS, falls vorhanden, aus dem Menü entfernt werden, damit es mit den aktuellen Layouts neu erstellt werden kann. Dies ist zwar nicht die schnellste, aber dafür die einfachste Methode, alle Layouts in das Menü einzufügen. Sie können das Programm auch dahingehend anpassen, dass die einzelnen Layouts nicht jedes Mal geladen werden. Vielmehr könnte das Menü jedes Mal bei Änderungen der Layouts (Hinzufügen, Entfernen, Umbenennen) angepasst werden. Dazu müssten Sie die Ereignisprozeduren verwenden, die bei einer Änderung am Layout ausgelöst werden.

```
For i = ShortcutMenu.Count - 1 To 0 Step -1
    If ShortcutMenu(i).Caption = "Layoutliste laden" Then
        LListFound = True
        Set LPopupMenu = ShortcutMenu(i)
    End If
```

Nachdem der Eintrag LAYOUTLISTE LADEN gefunden wurde (wird mithilfe der Eigenschaft *Caption* geprüft), wird der entsprechende Menüeintrag an die Variable *LPopupMenu* übergeben. Außerdem wird für die Variable *LListFound* der Wert *TRUE* gesetzt.

```
    If ShortcutMenu(i).Caption = "Layouts" Then
        ShortcutMenu(i).Delete
    End If
Next
```

Im zweiten Teil der *For...Next*-Schleife wird das Menü, das die Layouts enthält, aus dem Shortcut-Menü entfernt, falls es dort vorhanden ist.

Als Nächstes werden wir nun die Variable *LListFound* aus. Besitzt diese den Wert *FALSE*, so wurde der Menüeintrag LAYOUTLISTE LADEN noch nicht zum Shortcut-Menü hinzugefügt. Also muss das an dieser Stelle nachgeholt werden:

```
If LListFound = False Then
    Set LPopupMenu = ShortcutMenu.AddMenuItem(ShortcutMenu.Count, _
        "Layoutliste laden", "-vbarun LayoutItemClick" & vbCr)
End If
```

Mit dieser Anweisung wird der Menüeintrag LAYOUTLISTE LADEN, welcher bei Aufruf des Makro *LayoutItemClick* ausführen soll, zum Shortcut-Menü hinzugefügt. Im nächsten Schritt wird der Menüpunkt auf den aktuellen Status gebracht. Dazu nutzen wir die Funktion *GetSetting*, die den aktuellen Wert aus der Windows-Registrierung herausliest.

```
LPopupMenu.Check = CBool(GetSetting("Frames", "Setup", _
    "LoadLayouts", 1))
```

Gespeichert ist dieser Wert unter dem Schlüssel *Frames/Setup*, in der Zeichenfolge „LoadLayouts“. Wurde der Schlüssel noch nicht in der Registrierung erzeugt, so gibt die Funktion *GetSetting* den Wert 1 zurück, was gleichbedeutend mit dem booleschen Wert *TRUE* ist. Andernfalls würde die Funktion die Zeichenfolgen „TRUE“ oder „FALSE“ bzw. „WAHR“

oder „FALSCH“ zurückgeben. In jedem Fall muss der Wert so umgewandelt werden, dass er vom Datentyp her zur Eigenschaft *Check* passt. Hierbei hilft die Funktion *CBool()*. Diese Funktion wandelt eine Zeichenfolge oder Zahl in einen booleschen Wert um.

Nachdem dieser Vorgang ebenfalls abgehandelt wurde, brauchen wir nur noch den Menüpunkt *LAYOUTS* zu erstellen, der als Untermenüs alle Layouts enthalten wird. Folgende Codezeilen erledigen dies für uns.

```
If LPopupMenu.Check = True Then
    Set NPopUp = ShortcutMenu.AddSubMenu("Layouts", "Layouts")
    For i = 0 To ThisDrawing.Layouts.Count - 1
        NPopUp.AddMenuItem i, ThisDrawing.Layouts(i).Name, _
            "-vbarun setLayout " & ThisDrawing.Layouts(i).Name & vbCr
    Next
End If
```

Zunächst wird geprüft, ob der Menüpunkt *LAYOUTLISTE LADEN* angehakt wurde. Ist dies der Fall, so wird der Code innerhalb des *If..End If*-Blocks fortgeführt. Die erste Anweisung innerhalb dieses Blocks erstellt einen neuen Menüpunkt. Dieses Menü trägt die Bezeichnung *LAYOUTS* und wird alle Layouts als Untermenüs in sich aufnehmen. Aus diesem Grund nutzen wir auch die Anweisung *AddSubMenu*. Als Ergebnis der Funktion wird ein Verweis auf das neue Menü an die Variable *NPopUp* übergeben. Jetzt brauchen wir nur noch die Auflistung der Layouts (*ThisDrawing.Layouts*) zu durchlaufen und für jeden Layoutnamen einen neuen Menüpunkt zu erzeugen. Dazu kann bei jedem Durchlauf folgende Anweisung verwendet werden:

```
NPopUp.AddMenuItem i, ThisDrawing.Layouts(i).Name, _
    "-vbarun setLayout " & ThisDrawing.Layouts(i).Name & vbCr
```

Im ersten Parameter wird jedes Mal der Wert der Variablen *i* benutzt, damit jeder Eintrag an das Ende der Menüliste gesetzt wird. Als zweiter Parameter wird der Name des jeweiligen Layouts verwendet. Dieser erscheint damit als Bezeichnung des entsprechenden Menüpunkts. Als dritten und damit letzten Parameter übergeben wir der Funktion eine AutoCAD-Befehlszeile, die ausgeführt werden soll, wenn der Menüpunkt durch den Benutzer gewählt wurde. Dieser Befehl soll das Makro *setLayout* ausführen und diesem den Namen des Layouts übermitteln, damit in das entsprechende Layout gewechselt werden kann. Damit wäre die erste Prozedur abgeschlossen und wir können uns dem Code der nächsten Prozedur zuwenden.

In der nun folgenden Prozedur wird der Code abgehandelt, der aufgerufen wird, wenn der Benutzer auf einen Layoutnamen im Shortcut-Menü klickt.

```
Public Sub setLayout()
    Dim InVal As String
    On Local Error Resume Next
    InVal = ThisDrawing.Utility.GetString(True, "Layoutname: ")
    ThisDrawing.ActiveLayout = ThisDrawing.Layouts(InVal)
End Sub
```

Als Erstes wird der Name des Layouts abgefragt, in das gewechselt werden soll.

```
InVal = ThisDrawing.Utility.GetString(True, "Layoutname: ")
```

Da der Name des Layouts bereits durch den aufgerufenen Befehl eingegeben wird, muss der Benutzer die Eingabeaufforderung nur beantworten, wenn er die Prozedur direkt aufruft.

Nachdem sich der Layoutname, als Ergebnis der Funktion *GetString*, in der Variablen *InVal* befindet, muss nur noch dieser Name als Index der *Layouts*-Auflistung übergeben werden.

```
ThisDrawing.ActiveLayout = ThisDrawing.Layouts(InVal)
```

Wurde ein korrekter Name angegeben, wird das entsprechende Layout zum aktuellen Layout und die Prozedur kann mithilfe der *Exit Sub*-Anweisung verlassen werden.

Die letzte Prozedur wird ausgeführt, wenn der Benutzer den Menüpunkt *LAYOUTLISTE LADEN* ausgewählt hat. Diese Prozedur soll den Status dieses Menüpunkts umschalten und in die Registrierung schreiben.

```
Public Sub LayoutItemClick()  
On Local Error Resume Next  
If CBool(GetSetting("Frames","Setup","LoadLayouts",1)) = True Then  
    SaveSetting "Frames", "Setup", "LoadLayouts", False  
Else  
    SaveSetting "Frames", "Setup", "LoadLayouts", True  
End If  
End Sub
```

Der erste Schritt wäre also die Prüfung des aktuellen Status. Dazu wird der Inhalt des Registrierungseintrags „Frames/Setup/LoadLayouts“ ausgelesen und verglichen. Gibt diese Überprüfung *TRUE* zurück, so muss auf *FALSE* umgeschaltet werden und umgekehrt. Der jeweils neue Wert wird nun mittels der Funktion *SaveSetting* in die Registrierung eingetragen.

```
SaveSetting "Frames", "Setup", "LoadLayouts", True  
  
bzw.  
  
SaveSetting "Frames", "Setup", "LoadLayouts", False
```

5.2.9 Verwaltung der AutoCAD-Werkzeugkästen

Ähnlich wie die Pull-down-Menüs werden auch die Werkzeugkästen über VBA verwaltet. Dazu greifen wir ebenfalls auf das Objekt *Menus* der ausgewählten Menügruppe zurück. Dieses Objekt enthält neben den Menüeinträgen auch alle Werkzeugkästen der Datei, wobei jeder Eintrag der Auflistung einem einzelnen Werkzeugkasten entspricht. In der Tabelle 5.3 sehen Sie eine Übersicht über einige Werkzeugkästen der Datei *Acad.mn**, die mit folgender Codezeile abgerufen werden kann:

```
Application.MenuGroups("acad").Toolbars
```

Auflistung	Eintrag
.Item(0)	Bemaßung
.Item(1)	Zeichnen
.Item(2)	Abfrage
.Item(3)	Einfügen
.Item(4)	Layouts

Tabelle 5.3: Toolbar-Auflistung der Menüdatei *Acad*

Auflistung	Eintrag
.Item(5)	Ändern
.Item(6)	Ändern II
.Item(7)	Eigenschaften
.Item(n)	

Tabelle 5.3: Toolbar-Auflistung der Menüdatei Acad (Forts.)

Jeder einzelne Eintrag dieser Auflistung entspricht einem kompletten Werkzeugkasten und somit dem Datentyp *AcadToolbar*. Jeder weitere Eintrag im Werkzeugkasten selbst, also jede einzelne Schaltfläche, wird als Variable vom Datentyp *AcadToolbarItem* zurückgeben. Bevor wir uns nun einem einzelnen Werkzeugkasten zuwenden, noch eine Information zur Toolbar-Auflistung. Diese enthält eine Eigenschaft, mit der Sie auf große oder kleine Schaltflächen umschalten können.

```
Application.MenuGroups("acad").Toolbars.LargeButtons
```

Weisen Sie dieser Eigenschaft den Wert *TRUE* zu, so werden die Symbole der Werkzeugkästen in der Größe 24 mal 22 Pixel angezeigt, andernfalls in der Größe 16 mal 15 Pixel.

5.2.10 Verwendung der Klasse *AcadToolbar*

Wie bereits erwähnt, wird diese Klasse immer dann verwendet, wenn es sich bei dem Objekt, das zurückgegeben wird, um einen Werkzeugkasten handelt. Bevor wir anhand von Beispielen einige Funktionen dieser Klasse näher betrachten, sollten Sie sich eine Übersicht über alle Eigenschaften und Methoden dieses Objekts verschaffen.

Eigenschaften

Application

(schreibgeschützt)

Datentyp: *AcadApplication*

Gibt das Anwendungsobjekt (Klasse *AcadApplication*) zurück.

Count

(schreibgeschützt)

Datentyp: *Long*

Mit dieser Eigenschaft können Sie die Anzahl von Einträgen des Werkzeugkastens abrufen.

DockStatus

(schreibgeschützt)

Datentyp: *AcToolbarDockStatus*

Diese Eigenschaft zeigt an, ob der Werkzeugkasten fixiert, also am Rand verankert, oder verschiebbar ist. Folgende Konstanten können als Ergebnis zurückgegeben werden.

Konstante	Beschreibung	Wert
<i>acToolbarDockTop</i>	Am oberen Rand verankert	0
<i>acToolbarDockBottom</i>	Am unteren Rand verankert	1
<i>acToolbarDockLeft</i>	Am linken Rand verankert	2
<i>acToolbarDockRight</i>	Am rechten Rand verankert	3
<i>acToolbarFloating</i>	Nicht verankert	4

Tabelle 5.4: Konstanten der DockStatus-Eigenschaft

FloatingRowsDatentyp: *Long*

Hiermit können Sie die Anzahl der Zeilen für einen nicht verankerten Werkzeugkasten festlegen oder den aktuellen Wert ermitteln.

Height/Width
(schreibgeschützt)Datentyp: *Long*

Diese Eigenschaft gibt die aktuelle Höhe (*Height*) und Breite (*Width*) des Werkzeugkastens zurück.

HelpStringDatentyp: *String*

Mit dieser Eigenschaft können Sie den Hilfetext, der in der Statuszeile von AutoCAD angezeigt wird, festlegen bzw. abrufen.

LargeButtonsDatentyp: *Boolean*

Zeigt an, ob für die Werkzeugkästen große (24 x 22 Pixel) oder kleine (16 x 15 Pixel) Buttons verwendet werden.

Left / TopDatentyp: *Long*

Legt den Abstand des Werkzeugkastens vom linken (*Left*) und oberen (*Top*) Bildschirmrand fest oder ruft diesen ab.

NameDatentyp: *String*

Diese Eigenschaft legt den Namen (z.B. *Abfrage*) des Werkzeugkastens fest oder gibt den aktuellen Namen zurück.

Parent***(schreibgeschützt)***Datentyp: *Object*

Dieses Objekt gibt das dem aufrufenden Werkzeugkasten übergeordnete Objekt zurück.

TagString***(schreibgeschützt)***Datentyp: *String*

Die Eigenschaft enthält einen bestimmten ID-Text, der den Werkzeugkasten innerhalb einer Menüdatei eindeutig identifiziert. Dieser Text besteht aus alphanumerischen Zeichen und Underlines (_), z.B. ID_TbInquiry.

VisibleDatentyp: *Boolean*

Mit dieser Eigenschaft können Sie die Sichtbarkeit des Werkzeugkastens steuern. Folgende Werte können Sie hierfür verwenden:

- *TRUE* – Der Werkzeugkasten ist sichtbar.
- *FALSE* – Der Werkzeugkasten ist ausgeblendet.

Prozeduren***AddSeparator***Parameter: *Index*Rückgabety: *AcadToolBarItem*

Diese Funktion fügt dem aufrufenden Werkzeugkasten einen neuen Eintrag hinzu. Allerdings stellt dieser Eintrag keine Schaltfläche, sondern einen Trenner zwischen zwei Schaltflächen dar.

```
Dim ActSep As AcadToolBarItem
Set ActSep = ActTB.AddSeparator(2)
```

AddToolBarButtonParameter: *Index, Name As String, HelpString As String, Macro As String, [FlyoutButton]*Rückgabety: *AcadToolBarItem*

Über diese Anweisung können Sie dem Werkzeugkasten eine Schaltfläche hinzufügen.

Delete

Löscht den Werkzeugkasten aus der Toolbar-Auflistung

Dock

Parameter: *Side As AcToolbarDockStatus*

Da Sie mit der Eigenschaft *DockStatus* nur ermitteln können, ob ein Werkzeugkasten verankert ist oder nicht, müssen Sie diese Anweisung verwenden, um den Status zu verändern. Als Parameter nehmen Sie dann die gleichen Konstanten, wie sie bei der Eigenschaft *DockStatus* zurückgegeben werden (Tabelle 5.4).

ActTB.Dock acToolbarDockTop

oder

ActTB.Dock 0

Float

Parameter: top As Long, left As Long, NumberFloatRows As Long

Mithilfe von *Float* können Sie den Werkzeugkasten an eine bestimmte Position verschieben. Als Parameter für diese Anweisung müssen Sie natürlich die neue Position angeben – zum einen den Abstand vom linken Bildschirmrand und zum anderen den Abstand vom oberen Bildschirmrand. Zusätzlich legen Sie dann noch fest, in wie viele Zeilen der Werkzeugkasten nach der Verschiebung unterteilt werden soll.

ActTB.Float 120, 200, 2

Item

Parameter: *Index*

Rückgabety: *AcadToolbarItem*

Gibt die Schaltfläche des Werkzeugkastens an der angegebenen Position zurück.

5.2.11 Schaltflächen zum Werkzeugfenster hinzufügen

Bei einem Werkzeugkasten wird zwar zwischen drei verschiedenen Arten von Schaltflächen unterschieden, dem normalen Button, einem untergeordneten Werkzeugkasten und dem Trenner. Es stehen aber nur zwei Funktionen zur Verfügung, mit denen Schaltflächen hinzugefügt werden können. Die Anweisung *AddSeparator* ist klar. Diese fügt einen Trenner in den Werkzeugkasten ein. Bleibt also noch die Funktion *AddToolbarButton*. Diese kann sowohl Schaltflächen als auch untergeordnete Werkzeugkästen hinzufügen. Ausschlaggebend dafür, welche Form der Schaltfläche erzeugt werden soll, ist der Parameter *FlyoutButton*. Lassen Sie diesen Parameter leer oder weisen Sie ihm den Wert *FALSE* zu, so wird die neue Schaltfläche des Werkzeugkastens ein ganz normaler Button. Nur wenn der Parameter den Wert *TRUE* erhält, wird eine Schaltfläche erzeugt, die einen weiteren Werkzeugkasten zu einer Schaltfläche zusammenfasst.

Tipp

Sie können nur Schaltflächen zu einem Werkzeugkasten hinzufügen oder entfernen, wenn dieser sichtbar ist (Eigenschaft *Visible = TRUE*).

In jedem Fall wird als Ergebnis beider Funktionen ein Objekt geliefert, das dem Datentyp *AcadToolBarItem* entspricht. Mit diesem Objekt können Sie die neue Schaltfläche weiter bearbeiten.

5.2.12 Verwendung des Schaltflächen-Objekts

Wie bereits erwähnt, wird die Klasse *AcadToolBarItem* immer dann verwendet, wenn es sich bei dem zurückgegebenen Objekt um eine Schaltfläche eines Werkzeugkastens handelt. Bevor wir anhand von Beispielen einige Funktionen dieser Klasse näher betrachten, sollten Sie sich wieder eine Übersicht über alle Eigenschaften und Methoden dieser Klasse verschaffen

Eigenschaften

Application

(schreibgeschützt)

Datentyp: *AcadApplication*

Gibt das Anwendungsobjekt (Klasse *AcadApplication*) zurück.

Flyout

(schreibgeschützt)

Datentyp: *AcadToolBar*

Falls es sich bei der Schaltfläche um ein Flyout handelt, also um eine Schaltfläche, die einen weiteren Werkzeugkasten beherbergt, können Sie über diese Eigenschaft den entsprechenden Werkzeugkasten abrufen. Andernfalls löst der Abruf dieser Eigenschaft einen Fehler aus.

HelpString

Datentyp: *String*

Mit dieser Eigenschaft können Sie den Hilfetext, der in der Statuszeile von AutoCAD angezeigt wird, festlegen bzw. abrufen.

Index

(schreibgeschützt)

Datentyp: *Long*

Über diese Eigenschaft ermitteln Sie die Position, an der sich die Schaltfläche innerhalb des Werkzeugkastens befindet.

Macro

Datentyp: *String*

Hier können Sie die AutoCAD-Befehlsfolge festlegen oder abrufen, die ausgeführt werden soll, nachdem die Schaltfläche ausgeführt wurde.

Name

Datentyp: *String*

Diese Eigenschaft legt den Namen (z.B. „Abstand“) der Schaltfläche fest oder gibt den aktuellen Namen zurück.

Parent
(schreibgeschützt)

Datentyp: *AcadToolbar*

Dieses Objekt gibt das der aufrufenden Schaltfläche übergeordnete Objekt zurück.

TagString
(schreibgeschützt)

Datentyp: *String*

Die Eigenschaft enthält einen bestimmten ID-Text, der die Schaltfläche innerhalb einer Menüdatei eindeutig identifiziert. Dieser Text besteht aus alphanumerischen Zeichen und Underlines (), z.B. `ID_Dist`.

Type
(schreibgeschützt)

Datentyp: *AcToolbarItemType*

Zur Ermittlung der Art der Schaltfläche, um z.B. anschließend einen untergeordneten Werkzeugkasten abzurufen, können Sie diese Eigenschaft verwenden. Sie gibt als Ergebnis eine der folgenden Konstanten zurück.

Konstante	Beschreibung	Wert
<i>acToolbarButton</i>	Normale Schaltfläche	0
<i>acToolbarSeparator</i>	Trenner	1
<i>acToolbarControl</i>	Bei dem Element handelt es sich nicht um eine Schaltfläche, sondern um ein Steuerelement, wie z.B. der eingebetteten Layer-Auswahlliste oder der Farbauswahl.	2
<i>acToolbarFlyout</i>	Schaltfläche mit untergeordnetem Werkzeugkasten	3

Tabelle 5.5: Konstanten der Type-Eigenschaft

Prozeduren

AttachToolbarToFlyout

Parameter: *MenuGroupName As String, ToolbarName As String*

Mit dieser Anweisung ordnen Sie einem Flyout einen Werkzeugkasten zu. Um ein Flyout zu erzeugen, verwenden Sie zunächst die Anweisung *AddToolbarButton*. Der so erzeugten Schaltfläche können Sie dann unter Angabe der Menügruppe und des Werkzeugkastennamens den entsprechenden Werkzeugkasten hinzufügen.

```
Dim ActTB As AcadToolbar
Dim NewBtn As AcadToolbarItem
On Local Error Resume Next
Set ActTB = Application.MenuGroups("acad").Toolbars(11)
Set NewBtn = ActTB.AddToolbarButton(0, "Referenz", "Referenzen", _
    " ", True)
NewBtn.AttachToolbarToFlyout "acad", "Referenz"
```


Delete

Entfernt die Schaltfläche aus dem übergeordneten Werkzeugkasten.

GetBitmaps

Parameter: *SmallIconName As String, LargeIconName As String*

Ruft die Namen der Bitmaps ab, die als Symbole für das Werkzeugkastenelement (Schaltfläche/Flyout) verwendet werden. Die einzelnen Namen werden in den beiden Übergabeparametern zurückgegeben.

```
Dim smIconName As String
Dim lgIconName As String
ActBtn.GetBitmaps smIconName, lgIconName
```

SetBitmaps

Parameter: *SmallIconName As String, LargeIconName As String*

Hierbei handelt es sich um das Gegenstück zur vorab beschriebenen Funktion. Diese Anweisung setzt den Dateinamen der beiden Symbole des Werkzeugkastenelements (Schaltfläche/Flyout). Der erste Parameter enthält den Dateinamen (inklusive Pfad) für das kleine Symbol, der zweite den für das große Symbol.

```
Dim smIconName As String
Dim lgIconName As String
smIconName = "C:\Icons\smIcon01.bmp"
lgIconName = "C:\Icons\lgIcon01.bmp"
ActBtn.SetBitmaps smIconName, lgIconName
```

Kommen wir nun zu einem kleinen Beispiel, das uns den Vorgang des Hinzufügens von Schaltflächen etwas besser veranschaulicht. Das Ergebnis unseres kleinen Programms wird ein Werkzeugkasten sein, welcher einige Schaltflächen aus dem Werkzeugkasten mit den Standardfunktionen in einer eigenen Toolbar zusammenstellt.

```
Public Sub ToolbarExpmp()
Dim NewTB As AcadToolbar
Dim NewBtn As AcadToolbarItem
Dim smIcons As String, lgIcons As String
Dim HelpStr As String
On Local Error Resume Next
Set NewTB= Application.MenuGroups("acad").Toolbars.Add("Beispiel")
HelpStr = "Steuert die interaktive Ansicht von Objekten in 3D"
Set NewBtn = NewTB.AddToolbarButton(0, "3D-Orbit", HelpStr, _
    "'_3dorbit")
NewBtn.SetBitmaps "ICON_16_3DORBIT", "ICON_16_3DORBIT"
HelpStr = "Verschiebt die Ansicht im aktuellen Ansichtsfenster"
Set NewBtn = NewTB.AddToolbarButton(1, "Echtzeit-Pan", HelpStr, _
    "'_pan")
NewBtn.SetBitmaps "ICON_16_RTPAN", "ICON_16_RTPAN"
HelpStr = "Zoomt, um die sichtbare Größe von Objekten in " _
    & "der aktuellen Ansicht zu steigern oder zu verringern"
Set NewBtn = NewTB.AddToolbarButton(2, "Echtzeit-Zoom", HelpStr, _
    "'_zoom ;")
NewBtn.SetBitmaps "ICON_16_RTZOOM", "ICON_16_RTZOOM"
```

```

Set NewBtn = NewTB.AddToolBarButton(3, "Zoom", "", " ", True)
NewBtn.AttachToolBarToFlyout "acad", "zoom"
NewBtn.SetBitmaps "ICON_16_ZOOM", "ICON_16_ZOOM"
HelpStr = Zoomt auf die vorherige Ansicht"
Set NewBtn = NewTB.AddToolBarButton(4, "Zoom Vorher", HelpStr, _
    "'_zoom_p")
NewBtn.SetBitmaps "ICON_16_ZOOPRE", "ICON_16_ZOOPRE"
End Sub

```

Zu Beginn der Prozedur wird ein neuer Werkzeugkasten zur Menügruppe „Acad“ hinzugefügt. Zu beachten ist hier allerdings, dass dies nur einmal funktioniert, da der Werkzeugkasten beim nächsten Aufruf bereits in die Menügruppe integriert wurde. Sie müssten zur Prüfung, ob der Werkzeugkasten bereits vorhanden ist, eine kurze Abfrage durchführen. Eine Möglichkeit wäre dabei die Verwendung der *TypeName*-Funktion.

```

If TypeName(Application.MenuGroups("acad"). _
    Toolbars.Add("Beispiel")) = "Nothing" Then
    Application.MenuGroups("acad").Toolbars.Add("Beispiel")
End If
Set NewTB= Application.MenuGroups("acad").Toolbars.Add("Beispiel")

```

Nachdem das Werkzeugfenster erstellt und an die Objektvariable *NewTB* übergeben wurde, können die fünf Schaltflächen hinzugefügt werden. Bis auf die vierte sind alle anderen Schaltflächen ganz normale Buttons.

```

HelpStr = "Steuert die interaktive Ansicht von Objekten in 3D"
Set NewBtn = NewTB.AddToolBarButton(0, "3D-Orbit", HelpStr, _
    "'_3dorbit")
NewBtn.SetBitmaps "ICON_16_3DORBIT", "ICON_16_3DORBIT"

```

Zuerst wird der Text, der in der Statuszeile von AutoCAD angezeigt werden soll, an die Variable *HelpStr* übergeben. Anschließend wird die Schaltfläche mithilfe der Anweisung *AddToolBarButton* zum Werkzeugkasten hinzugefügt. Die erste Schaltfläche kommt an Position 0, die zweite an 1 usw. Abschließend ordnen wir nur noch die beiden Symbole mittels *SetBitmaps* zu und schon ist die jeweilige Schaltfläche dem Werkzeugkasten hinzugefügt.

Eine zusätzliche Anweisung muss nur noch bei der Erstellung der Flyout-Schaltfläche durchgeführt werden. Da diese Schaltfläche einen weiteren Werkzeugkasten beherbergt, müssen wir diesen Werkzeugkasten der Schaltfläche über folgende Anweisung zuordnen:

```

NewBtn.AttachToolBarToFlyout "acad", "zoom"

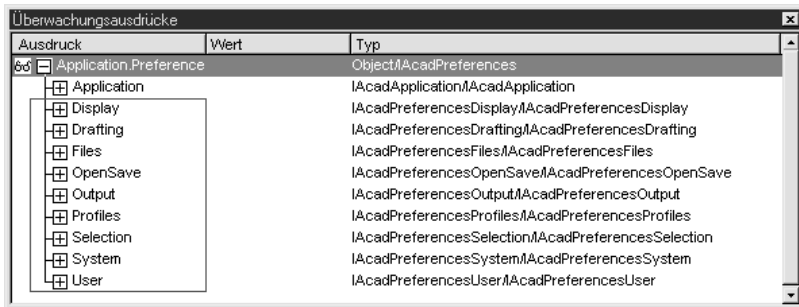
```

Der Rest funktioniert genau wie bei einer einfachen Schaltfläche.

5.3 AutoCAD-Einstellungen steuern

Alle Einstellungen von AutoCAD, die nicht in der Zeichnung, sondern in der Registrierung gespeichert werden, sind über das *Preferences*-Objekt abrufbar. Wenn Sie sich einmal dieses Objekt in den Überwachungsbereich laden, indem Sie einfach eine Prozedur im Einzelschrittmodus starten und die Zeile *Application.Preferences* als Überwachungsausdruck hinzufügen,

dann werden Sie feststellen, dass die in *Preferences* enthaltenen Objekte mit den Reitern des Optionen-Dialogs übereinstimmen. Daraus lässt sich also schließen, dass das *Preferences*-Objekt genauso aufgeteilt wurde, was die Arbeit wesentlich vereinfacht.



Ausdruck	Wert	Typ
Application Preference		Object/IAcadPreferences
Application		IAcadApplication/IAcadApplication
Display		IAcadPreferencesDisplay/IAcadPreferencesDisplay
Drafting		IAcadPreferencesDrafting/IAcadPreferencesDrafting
Files		IAcadPreferencesFiles/IAcadPreferencesFiles
OpenSave		IAcadPreferencesOpenSave/IAcadPreferencesOpenSave
Output		IAcadPreferencesOutput/IAcadPreferencesOutput
Profiles		IAcadPreferencesProfiles/IAcadPreferencesProfiles
Selection		IAcadPreferencesSelection/IAcadPreferencesSelection
System		IAcadPreferencesSystem/IAcadPreferencesSystem
User		IAcadPreferencesUser/IAcadPreferencesUser

Abbildung 5.10: Inhalt des Preferences-Objekts

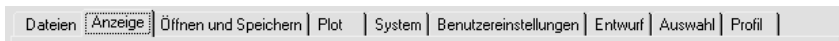


Abbildung 5.11: Reiter des Optionen-Dialogs

Für die meisten Eigenschaften der einzelnen *Preferences*-Objekte existieren auch Systemvariablen, so dass Sie auch diese zur Einstellung der entsprechenden Eigenschaft verwenden könnten, wobei der Zugriff über das *Preferences*-Objekt jedoch wesentlich bequemer ist.

5.3.1 Anzeigeeinstellungen (Display-Objekt)

Dieses Objekt beinhaltet alle Einstellungen, die auch im Optionen-Dialog unter dem Reiter ANZEIGE zu finden sind. Zugriff auf die Eigenschaften dieses Objekts erhalten Sie, indem Sie *Display* direkt über das *Preferences*-Objekt ansprechen:

```
Application.Preferences.Display
```

oder in einer Objektvariablen einen Verweis auf das *Display*-Objekt erstellen:

```
Dim PrefDisplay As AcadPreferencesDisplay
Set PrefDisplay = Application.Preferences.Display
```

Sie können nun auf die einzelnen Eigenschaften des *Display*-Objekts zugreifen und damit die entsprechenden Anzeigeeinstellungen vornehmen.

Farben anpassen

Für die Anpassung der verschiedensten Bildschirmfarben, wie z.B. die Farbe des Zeichnungshintergrunds oder der Textfarbe des Befehlszeilenfensters, existieren bestimmte Eigenschaften. Der Wert, den Sie in dieser Eigenschaft für eine Farbe festlegen können, entspricht nicht den AutoCAD-Farben, sondern den Windows-Farben, die Sie z.B. mit der *RGB()*-Funktion aus den Rot/Grün/Blau-Anteilen bestimmen können. Sie können aber auch die in der Systemsteuerung eingestellten Farben verwenden, indem Sie den entsprechenden Farbwert einstellen (siehe auch Kapitel 8.3.1).

Diese Eigenschaft kann durch den Benutzer auch im Farbdialog festgelegt werden, der über die Schaltfläche FARBEN... aktiviert wird.

AutoTrackingVecColor

Datentyp: *OLE_COLOR*

Diese Eigenschaft bestimmt den Farbwert für den AutoTrack-Vektor.

Die Farbe Weiß als Vektorfarbe festlegen:

```
PrefDisplay.AutoTrackingVecColor = RGB(255, 255, 255)
```

GraphicsWinLayoutBackgrndColor

Datentyp: *OLE_COLOR*

Diese Eigenschaft bestimmt die Hintergrundfarbe des Zeichnungsfensters, wenn sich dieses in einem Papierbereichslayout befindet.

Die Farbe Schwarz als Hintergrundfarbe festlegen:

```
PrefDisplay.GraphicsWinLayoutBackgrndColor = RGB(0, 0, 0)
```

GraphicsWinModelBackgrndColor

Datentyp: *OLE_COLOR*

Diese Eigenschaft bestimmt die Hintergrundfarbe des Zeichnungsfensters, wenn sich dieses im Modellbereichslayout befindet.

Die Farbe Schwarz als Hintergrundfarbe festlegen:

```
PrefDisplay.GraphicsWinModelBackgrndColor = RGB(0, 0, 0)
```

LayoutCrosshairColor

Datentyp: *OLE_COLOR*

Diese Eigenschaft legt die Farbe des Fadenkreuzes im Papierbereichslayout fest. Der Vorgabewert dieser Eigenschaft entspricht der Textfarbe aus der Systemeinstellung von Windows.

Die Farbe Rot als Fadenkreuzfarbe festlegen:

```
PrefDisplay.LayoutCrosshairColor = RGB(255, 0, 0)
```

ModelCrosshairColor

Datentyp: *OLE_COLOR*

Diese Eigenschaft legt die Farbe des Fadenkreuzes im Modellbereich fest. Der Vorgabewert dieser Eigenschaft entspricht der Textfarbe aus der Systemeinstellung von Windows.

Die Farbe Weiß als Fadenkreuzfarbe festlegen:

```
PrefDisplay.ModelCrosshairColor = RGB(255, 255, 255)
```

TextWinBackgrndColorDatentyp: *OLE_COLOR*

In dieser Eigenschaft wird die Farbe des Hintergrunds des Befehlszeilenfensters gespeichert. Der Vorgabewert dieser Eigenschaft entspricht der Fensterfarbe aus der Systemsteuerung von Windows.

Die Farbe Weiß als Hintergrundfarbe festlegen:

```
PrefDisplay.TextWinBackgrndColor = RGB(255, 255, 255)
```

TextWinTextColorDatentyp: *OLE_COLOR*

Die letzte Farbeigenschaft des *Display*-Objekts legt die Farbe für den Text des Befehlszeilenfensters fest. Als Vorgabe dieser Eigenschaft wird die Textfarbe verwendet, die in der Systemsteuerung von Windows eingestellt wurde.

Die Farbe Schwarz als Hintergrundfarbe festlegen:

```
PrefDisplay.TextWinTextColor = RGB(0, 0, 0)
```

Schrift des Textes des Befehlszeilenfensters festlegen

Mit den folgenden Eigenschaften legen Sie die Schriftart des Textes fest, der im Befehlszeilenfenster angezeigt wird. Diese Eigenschaft kann durch den Benutzer auch im Schriftartendialog bestimmt werden, der sich über die Schaltfläche SCHRIFTEN... aktivieren lässt.

TextFontDatentyp: *String*

Mit dieser Eigenschaft legen Sie die Schriftart der Texte des Befehlszeilenfensters fest.

```
PrefDisplay.TextFont = "Courier"
```

TextFontSizeDatentyp: *Long*

Diese Eigenschaft bestimmt die Schriftgröße der Texte in der Befehlszeile.

```
PrefDisplay.TextFontSize = 12
```

TextFontStyleDatentyp: *AcTextFontStyle*

Diese Eigenschaft bestimmt den Schriftstil der Texte in der Befehlszeile. Sie können hierfür auf folgende Konstanten zurückgreifen:

Konstante	Beschreibung	Wert
<i>acFontRegular</i>	Der Befehlszeilentext wird normal dargestellt.	0
<i>acFontItalic</i>	Der Befehlszeilentext wird kursiv dargestellt.	1

Konstante	Beschreibung	Wert
<i>acFontBold</i>	Der Befehlszeilentext wird in Fettschrift dargestellt.	2
<i>acFontBoldItalic</i>	Der Befehlszeilentext wird sowohl kursiv als auch in Fettschrift dargestellt.	3

```
PrefDisplay.TextFontStyle = acFontItalic
```

Weitere Anzeigeeigenschaften

Die folgenden Eigenschaften lassen sich größtenteils direkt im Optionen-Dialog im Abschnitt ANZEIGE einstellen. Sie steuern die verschiedensten Darstellungen, angefangen von der Größe der Fadenkreuzöffnung bis hin zum Verhalten von Pixelbildern beim Verschieben im Zeichnungsfenster.

CursorSize

Datentyp: *Long*

Systemvariable: *CURSORSIZE*

Mit dieser Eigenschaft bestimmen Sie die Größe der Fadenkreuzöffnung. Diese Größe wird nicht in absoluten Werten, sondern als Prozentsatz im Bereich von 1–100% angegeben.

Cursorgröße auf 5% einstellen:

```
PrefDisplay.CursorSize = 20
```

Im Optionen-Dialog finden Sie diese Eigenschaft im Abschnitt FADENKREUZGRÖSSE.

DisplayLayoutTabs

Datentyp: *Boolean*

Mit dieser Eigenschaft können Sie festlegen, ob die Registrierkarten für die Anzeige und die Auswahl der Layouts für den Modell- und Papierbereich an der unteren Kante des Zeichenbereichs dargestellt werden (*TRUE*) oder nicht (*FALSE*).

Registeranzeige aktivieren:

```
PrefDisplay.DisplayLayoutTabs = True
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt LAYOUT-ELEMENTE mit dem Kontrollkästchen REGISTRIERKARTEN LAYOUT UND MODELL ANZEIGEN einstellen.

DisplayScreenMenu

Datentyp: *Boolean*

Diese Eigenschaft bestimmt, ob das Bildschirmmenü am rechten Rand des Zeichnungsfensters angezeigt werden soll (*TRUE*) oder nicht (*FALSE*).

Bildschirmmenü ausschalten:

```
PrefDisplay.DisplayScreenMenu = False
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt FENSTERELEMENTE mit dem Kontrollkästchen BILDSCHIRMENÜ ANZEIGEN einstellen.

DisplayScrollBars

Datentyp: *Boolean*

Um die Anzeige der Bildlaufleisten des Zeichnungsfensters zu steuern, können Sie die Eigenschaft *DisplayScrollBars* verwenden. Wenn Sie dieser Eigenschaft den Wert *TRUE* zuweisen, dann werden die Bildlaufleisten angezeigt, andernfalls nicht.

Bildlaufleisten deaktivieren:

```
PrefDisplay.DisplayScrollBars = False
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt FENSTERELEMENTE im Kontrollkästchen BILDLAUFLEISTEN IN ZEICHNUNGSFENSTER ANZEIGEN einstellen.

DockedVisibleLines

Datentyp: *Long*

In dieser Eigenschaft ist die Anzahl der Textzeilen festgelegt, die im Befehlsfenster angezeigt werden, wenn dieses andockt worden ist. Der gültige Bereich für diese Eigenschaft liegt zwischen 1 und 100 .

Die Höhe des gedockten Befehlszeilenfensters auf fünf Zeilen festlegen:

```
PrefDisplay.DockedVisibleLines = 5
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt FENSTERELEMENTE im Textfeld TEXTZEILEN IM BEFEHLSZEILENFENSTER festlegen.

HistoryLines

Datentyp: *Long*

Mit *HistoryLines* können Sie die Anzahl der Zeilen des Befehlsfensters festlegen, die im Speicher verbleiben sollen. Je höher Sie den Wert für diese Eigenschaft festlegen, umso mehr Zeilen können Sie im Textfenster zurückverfolgen. Der Startwert von *HistoryLines* beträgt 400 Zeilen.

Die Speicherung der letzten 600 eingegebenen Zeilen festlegen:

```
PrefDisplay.HistoryLines = 600
```

ImageFrameHighlight

Datentyp: *Boolean*

Systemvariable: *IMAGEHLT*

Der Wert, den Sie für diese Eigenschaft festlegen, steuert die Anzeige von Pixelbildern während des Auswahlvorgangs. Wenn Sie den Wert dieser Eigenschaft auf *TRUE* setzen, wird lediglich der Rahmen des Pixelbilds, statt des gesamten Inhalts, hervorgehoben.

Nur die Anzeige des Pixelbildrahmens aktivieren:

```
PrefDisplay.ImageFrameHighlight = True
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt BILDSCHIRMLEISTUNG mit dem Kontrollkästchen NUR PIXELBILDRAHMEN MARKIEREN festlegen.

LayoutCreateViewport

Datentyp: *Boolean*

Mit dieser Eigenschaft geben Sie vor, ob bei der Erstellung eines neuen Layouts automatisch ein Ansichtsfenster erzeugt werden soll (*TRUE*) oder nicht (*FALSE*). Die Deaktivierung dieser Eigenschaft ist besonders dann hilfreich, wenn Sie per Programmcode ein Layout erzeugen und diesem sofort ein eigenes Ansichtsfenster hinzufügen wollen. (Beispiel hierfür ist die Übertragung eines Bereichs in den Papierbereich, *Kapitel 7.12.2*)

Automatische Erzeugung des Ansichtsfensters deaktivieren:

```
PrefDisplay.LayoutCreateViewport = False
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt LAYOUT-ELEMENTE mit dem Kontrollkästchen ANSICHTSFENSTER IN NEUEN LAYOUTS ERZEUGEN einstellen.

LayoutDisplayMargins

Datentyp: *Boolean*

Mit *LayoutDisplayMargins* können Sie festlegen bzw. ermitteln, ob die Ränder des Papierbereichslayouts angezeigt werden sollen (*TRUE*) oder nicht (*FALSE*). Diese Ränder sind als gestrichelte Linien zu erkennen und markieren den Bereich des Layouts, ab dem Objekte zugeschnitten bzw. ganz ignoriert werden.

Ränder anzeigen:

```
PrefDisplay.LayoutDisplayMargins = True
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt LAYOUT-ELEMENTE mit dem Kontrollkästchen RÄNDER ANZEIGEN einstellen.

LayoutDisplayPaper

Datentyp: *Boolean*

Diese Eigenschaft bestimmt, ob der Hintergrund des Papierbereichslayouts angezeigt werden soll (*TRUE*) oder nicht (*FALSE*). Wenn Sie diese Eigenschaft aktivieren, wird der gesamte Papierbereich in der Hintergrundfarbe angezeigt, die Sie in der Eigenschaft *GraphicsWinLayoutBackgrndColor* festgelegt haben. Andernfalls erfolgt die Anzeige des Layouts in Grau und nur das Blatt, das der eingestellten Papiergröße entspricht, wird in der entsprechenden Hintergrundfarbe dargestellt.

Den grauen Hintergrund des Papierbereichs anzeigen:

```
PrefDisplay.LayoutDisplayPaper = True
```

Im Optionen-Dialog finden Sie diese Eigenschaft im Abschnitt LAYOUT-ELEMENTE mit dem Kontrollkästchen PAPIERHINTERGRUND ANZEIGEN.

LayoutDisplayPaperShadow

Datentyp: *Boolean*

Diese Eigenschaft hängt mit der Eigenschaft *LayoutDisplayPaper* eng zusammen und wird nur berücksichtigt, wenn der Papierbereichshintergrund angezeigt wird. Hiermit steuern Sie die Anzeige eines Schattens für den Papierhintergrund.

Schatten aktivieren:

```
PrefDisplay.LayoutDisplayPaperShadow = True
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt LAYOUT-ELEMENTE mit dem Kontrollkästchen PAPIERSCHATTIERUNG ANZEIGEN einstellen.

LayoutShowPlotSetup

Datentyp: *Boolean*

Diese Eigenschaft bestimmt, ob bei der Erstellung eines neuen Layouts das Dialogfenster SEITE EINRICHTEN angezeigt werden soll (*TRUE*) oder nicht (*FALSE*).

Dialogfenster SEITE EINRICHTEN aktivieren:

```
PrefDisplay.LayoutCreateViewport = True
```

Das Gegenstück zu dieser Eigenschaft finden Sie gleich in zwei Dialogfenstern: zunächst im Reiter ANSICHT des Optionen-Dialogs im Abschnitt LAYOUT-ELEMENTE. Dort können Sie mit dem Kontrollkästchen DIALOGFELD SEITE EINRICHTEN FÜR NEUE LAYOUTS ANZEIGEN den Wert dieser Eigenschaft anpassen. Außerdem könnte der Benutzer diese Eigenschaft auch über das Dialogfenster SEITE EINRICHTEN festlegen.

MaxAutoCADWindow

Datentyp: *Boolean*

Mit dieser Eigenschaft legen Sie fest, ob das AutoCAD-Sitzungsfenster beim Start maximiert werden soll. Wenn Sie diese Eigenschaft auf den Wert *TRUE* einstellen, wird AutoCAD nach dem Start immer im Vollbild angezeigt.

AutoCAD immer im Vollbild anzeigen:

```
PrefDisplay.MaxAutoCADWindow = True
```

ShowRasterImage

Datentyp: *Boolean*

ShowRasterImage steuert die Anzeige von Pixelbildern während eines Echtzeit-PAN bzw. -Zooms. Wenn Sie den Wert dieser Eigenschaft auf *TRUE* festlegen, werden alle Details des Pixelbilds beim Zoomen angezeigt. Bei Deaktivierung dieser Eigenschaft wird nur der Rahmen des Bilds beim Zoomen dargestellt. Sie können diese Eigenschaft deaktivieren, um die Systemleistung von AutoCAD zu erhöhen.

Anzeige der Pixelbilder deaktivieren:

```
PrefDisplay.ShowRasterImage = False
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt BILDSCHIRMLEISTUNG mit dem Kontrollkästchen PAN UND ZOOM MIT PIXELBILD einstellen.

TrueColorImages

Datentyp: *Boolean*

Mit dieser Eigenschaft beeinflussen Sie die Anzeigequalität von Pixelbildern und Renderings. Wenn Sie für diese Eigenschaft den Wert *TRUE* festlegen, dann stellt AutoCAD die Bilder mit der höchstmöglichen Qualität dar, die mit den vorliegenden Systemeinstellungen möglich sind.

```
PrefDisplay.ShowRasterImage = False
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt BILDSCHIRMLEISTUNG mit dem Kontrollkästchen TRUE COLOR-PIXELBILDER UND -RENDERN einstellen.

XRefFadeIntensity

Datentyp: *Long*

Systemvariable: *XFADECTL*

Der Wert dieser Eigenschaft bestimmt die prozentuale Abblendintensität für die Objekte bei der direkten Bearbeitung von Referenzen. Wenn der Benutzer Blockreferenzen oder externe Referenzen direkt bearbeitet, werden die nicht benötigten Objekte während der Bearbeitung mit einer geringeren Intensität dargestellt. Gültige Werte liegen im Bereich von 0 bis 90%. Der Vorgabewert beträgt 50%.

```
PrefDisplay.ShowRasterImage = 50
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt FADINGINTENSITÄT BEI REFERENZBEARBEITUNG mit dem Schieberegler einstellen.

5.3.2 Entwurfseinstellungen (Drafting-Objekt)

Dieses Objekt beinhaltet alle Einstellungen, die auch im Optionen-Dialog unter dem Reiter ENTWURF zu finden sind. Zugriff auf die Eigenschaften dieses Objekts erhalten Sie, indem Sie *Drafting* direkt über das *Preferences*-Objekt ansprechen

```
Application.Preferences.Drafting
```

oder in einer Objektvariablen einen Verweis auf das *Drafting*-Objekt erstellen.

```
Dim PrefDrafting As AcadPreferencesDrafting  
Set PrefDrafting = Application.Preferences.Drafting
```

Sie können nun auf die einzelnen Eigenschaften des *Drafting*-Objekts zugreifen und damit die entsprechenden Einstellungen zur AutoCAD-Bearbeitung vornehmen.

AutoSnap-Einstellungen

AutoSnapAperture

Datentyp: *Boolean*

Systemvariable: *APBOX*

Mit dieser Eigenschaft können Sie die Anzeige der AutoSnap-Öffnung steuern. Wenn Sie diese Eigenschaft auf den Wert *TRUE* einstellen, wird eine rechteckige Öffnung im Fadenkreuz angezeigt, die den Fangbereich für die AutoCAD-Objekte kennzeichnet.

```
PrefDrafting.AutoSnapAperture = False
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUTOSNAP-EINSTELLUNGEN mit dem Kontrollkästchen AUTOSNAP-ÖFFNUNG ANZEIGEN einstellen.

AutoSnapApertureSize

Datentyp: *Long*

Systemvariable: *APERTURE*

Wenn Sie die AutoSnap-Öffnung aktivieren, können Sie zusätzlich die Größe dieser Öffnung anpassen. Die Größe dieser Öffnung bestimmt, wie nahe sich das Fadenkreuz an einem Objekt befinden muss, bevor der Objektfang aktiviert wird. Die Größe der Öffnung wirkt sich auch auf den Objektfang aus, wenn die Öffnung nicht angezeigt wird. Der Wertebereich dieser Eigenschaft liegt zwischen 1 und 50.

```
PrefDrafting.AutoSnapApertureSize = 10
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt GRÖSSE DER ÖFFNUNG mit dem Schieberegler einstellen.

AutoSnapMagnet

Datentyp: *Boolean*

Systemvariable: *AUTOSNAP*

Mit dieser Eigenschaft können Sie den AutoSnap-Magneten aktivieren (*TRUE*) oder deaktivieren (*FALSE*). Wenn Sie den Magneten aktivieren, wird das Fadenkreuz automatisch in der Objektfangmarkierung positioniert, wenn es sich in dessen unmittelbarer Nähe befindet.

```
PrefDrafting.AutoSnapMagnet = False
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUTOSNAP-EINSTELLUNGEN mit dem Kontrollkästchen MAGNET einstellen.

AutoSnapMarker

Datentyp: *Boolean*

Systemvariable: *AUTOSNAP*

Diese Eigenschaft bestimmt, ob die Objektfangmarkierung angezeigt werden soll (*TRUE*) oder nicht (*FALSE*). Diese Markierung kennzeichnet den gefangenen Punkt und dient der rein optischen Anzeige dieser Position. Der Objektfang selbst bleibt auch wirksam, wenn Sie diese Eigenschaft deaktivieren.

```
PrefDrafting.AutoSnapMarker = True
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUTOSNAP-EINSTELLUNGEN mit dem Kontrollkästchen MARKIERUNG einstellen.

AutoSnapMarkerColor

Datentyp: *AcColor*

Die Farbe des Markierungsquadrats für den Objektfang legen Sie mit dieser Eigenschaft fest. Dabei können Sie eine der folgenden Farbkonstanten verwenden:

Konstante	Farbe	Wert
<i>acRed</i>	Rot	1
<i>acYellow</i>	Gelb	2
<i>acGreen</i>	Grün	3
<i>acCyan</i>	Cyan	4
<i>acBlue</i>	Blau	5
<i>acMagenta</i>	Magenta	6
<i>acWhite</i>	Schwarz/Weiß je nach Hintergrundfarbe	7

Tabelle 5.6: AutoCAD-Grundfarben

`PrefDrafting.AutoSnapMarkerColor = acBlue`

Im Optionen-Dialog können Sie diese Eigenschaft in der Auswahlliste AUTOSNAP-MARKIERUNGSFARBE einstellen.

AutoSnapMarkerSize

Datentyp: *Long*

Die Größe der Markierungsöffnung bestimmen Sie mit dieser Eigenschaft. Die Größe dieser Öffnung muss im Bereich von 1 bis 20 Pixel liegen.

`PrefDrafting.AutoSnapMarkerSize = 5`

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUTOSNAP-MARKIERUNGSGRÖSSE mit dem Schieberegler einstellen.

AutoSnapTooltip

Datentyp: *Boolean*

Systemvariable: *AUTOSNAP*

Mit dieser Eigenschaft können Sie die Anzeige eines gelben Infotexts steuern, der eine Kurzbeschreibung des Objektfangs anzeigt. Diese Kurzbeschreibung könnte z. B. ENDPUNKT lauten, wenn der Endpunkt eines Objekts gefangen wurde, oder BASISPUNKT, um den Benutzer darüber zu informieren, dass der Basispunkt einer Blockreferenz gefangen wurde.

`PrefDrafting.AutoSnapTooltip = True`

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUTOSNAP-EINSTELLUNGEN mit dem Kontrollkästchen AUTOSNAP-QUIKINFO ANZEIGEN einstellen.

AutoTrack-Einstellungen

AutoTrackTooltip

Datentyp: *Boolean*

Systemvariable: *AUTOSNAP*

Mit dieser Eigenschaft steuern Sie die Anzeige des gelben Infotexts, der die Spurkoordinaten anzeigt.

```
PrefDrafting.AutoTrackTooltip = True
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUTOTRACK-EINSTELLUNGEN mit dem Kontrollkästchen QUIKINFO FÜR AUTOTRACK ANZEIGEN einstellen.

FullScreenTrackingVector

Datentyp: *Boolean*

Systemvariable: *TRACKPATH*

Diese Eigenschaft aktiviert (*TRUE*) oder deaktiviert (*FALSE*) die Anzeige der Vollbild-Spurvektoren.

```
PrefDrafting.FullScreenTrackingVector = True
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUTOTRACK-EINSTELLUNGEN mit dem Kontrollkästchen VOLLBILD-SPURVEKTOR ANZEIGEN einstellen.

PolarTrackingVector

Datentyp: *Boolean*

Systemvariable: *TRACKPATH*

Diese Eigenschaft aktiviert (*TRUE*) oder deaktiviert (*FALSE*) die Anzeige der polaren Spurvektoren.

```
PrefDrafting.PolarTrackingVector = True
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUTOTRACK-EINSTELLUNGEN mit dem Kontrollkästchen POLAREN SPURVEKTOR ANZEIGEN einstellen.

Weitere Entwurfseigenschaften

AlignmentPointAcquisition

Datentyp: *AcAlignmentPointAcquisition*

In dieser Eigenschaft wird die Art der Auswahl des Ausrichtungspunkts festgelegt. Sie können eine der folgenden Konstanten verwenden:

Konstante	Beschreibung	Wert
<i>acAlignPntAcquisitionAutomatic</i>	Automatische Auswahl	0
<i>acAlignPntAcquisitionShiftToAcquire</i>	Auswahl mit der  -Taste	1

```
PrefDrafting.AlignmentPointAcquisition = 0
```

Im Optionen-Dialog können Sie diese Eigenschaft im Abschnitt AUSWAHL VON AUSRICHTPUNKT durch Auswahl einer der beiden Optionsfelder einstellen.

5.3.3 Dateieinstellungen (Files-Objekt)

In diesem Objekt werden die Verzeichnisse verwaltet, in denen AutoCAD nach Support-, Treiber- oder Menüdateien suchen soll und die bestimmte Dateinamen, die AutoCAD-Definitionsdaten (z.B. Konfigurationsdateien) enthalten. Zugriff auf die Eigenschaften dieses Objekts erhalten Sie, indem Sie *Files* direkt über das *Preferences*-Objekt ansprechen:

```
Application.Preferences.Files
```

oder in einer Objektvariablen einen Verweis auf das *Files*-Objekt erstellen:

```
Dim PrefFiles As AcadPreferencesFiles
Set PrefFiles = Application.Preferences.Files
```

Sie können nun auf die einzelnen Eigenschaften des *Files*-Objekts zugreifen und damit die entsprechenden Einstellungen von Verzeichnissen und Dateien vornehmen.

Suchpfade festlegen

In den folgenden Eigenschaften sind die Verzeichnisnamen gespeichert, in denen AutoCAD nach den verschiedensten Dateien suchen soll. Dabei kann es vorkommen, dass in einer Eigenschaft mehrere Verzeichnisse auf einmal angegeben werden. Die einzelnen Namen sind dann durch ein Semikolon voneinander getrennt. Bestes Beispiel hierfür ist die Eigenschaft *SupportPath*, die die möglichen Positionen von Support-Dateien angibt. Um diese Verzeichnisnamen wieder zu einer Liste zusammenzustellen, könnten Sie folgende Funktion verwenden.

```
Function GetPathList(PathNames As String)
Dim PathList
ReDim PathList(0) As String
For i = 1 To Len(PathNames)
    If Mid(PathNames, i, 1) = ";" Then
        x = x + 1
        ReDim Preserve PathList(x) As String
    Else
        PathList(x) = PathList(x) + Mid(PathNames, i, 1)
    End If
Next
GetPathList = PathList
End Function
```

Diese Funktion separiert sämtliche Verzeichniseinträge und stellt ein Array zusammen, in dem jedes Feld einen Verzeichnisnamen enthält.

DriversPath

Datentyp: *String*

In dieser Eigenschaft werden die Namen der Verzeichnisse gespeichert, in denen AutoCAD nach den Treiberdateien suchen soll.

SupportPath

Datentyp: *String*

In dieser Eigenschaft werden die Verzeichnisnamen gespeichert, in denen AutoCAD nach den Support-Dateien suchen soll.

ObjectARXPath

Datentyp: *String*

In dieser Eigenschaft werden die Verzeichnisnamen gespeichert, in denen AutoCAD nach den ObjectARX-Anwendungen suchen soll, für die beim Laden der Zeichnung zugehörige benutzerdefinierte Objekte gefunden wurden. Im Gegensatz zu den übrigen Pfadangaben werden hier nur URL-Adressen angegeben (z. B. *file:///D:/ARX-Files*).

TextureMapPath

Datentyp: *String*

Diese Eigenschaft enthält die Namen der Verzeichnisse, in denen AutoCAD nach Texture-Maps zum Rendern suchen soll.

AutoSavePath

Datentyp: *String*

Systemvariable: *SAVEFILEPATH*

In dieser Eigenschaft wird der Name des Verzeichnisses gespeichert, in dem die automatische Sicherungsdatei abgelegt werden soll.

LogFilePath

Datentyp: *String*

Systemvariable: *LOGFILEPATH*

Das Verzeichnis, in dem die Protokolldateien gespeichert werden sollen, wird in dieser Eigenschaft abgelegt.

PrinterConfigPath

Datentyp: *String*

In dieser Eigenschaft ist der Name des Verzeichnisses gespeichert, in dem die Konfigurationsdateien (PC3-Dateien) des Druckers abgelegt wurden.

PrinterDescPath

Datentyp: *String*

Diese Eigenschaft speichert das Verzeichnis, in dem die Beschreibungsdateien (PMP-Dateien) des Druckers abgelegt wurden.

PrinterStyleSheetPath

Datentyp: *String*

Diese Eigenschaft gibt den Speicherort der benannten und farbabhängigen Plotstiltabellendateien (STB und CTB) an.

PrintSpoolerPath

Datentyp: *String*

In das Verzeichnis, das in dieser Eigenschaft angegeben wurde, schreibt AutoCAD die Druckspool-Dateien.

TempFilePath

Datentyp: *String*

Systemvariable: *TEMPPREFIX*

In dieser Eigenschaft wird das Verzeichnis für temporäre Dateien angegeben. In den meisten Fällen dürfte das „c:\tmp“ oder „c:\temp“ sein. Wenn Sie diese Eigenschaft leer lassen, dann verwendet AutoCAD das temporäre Windows-Systemverzeichnis zur Ablage der Daten.

TemplateDwgPath

Datentyp: *String*

Das Verzeichnis der Vorlagdateien wird in dieser Eigenschaft gespeichert. AutoCAD wechselt im Öffnen-Dialog automatisch in dieses Verzeichnis, wenn der Benutzer das DWT-Format auswählt. Außerdem wird die Vorlagendatei, die Sie beim Anlegen einer neuen Zeichnung im entsprechenden Parameter angegeben haben, in diesem Verzeichnis gesucht.

TempXrefPath

Datentyp: *String*

Systemvariable: *XLOADPATH*

Diese Eigenschaft gibt den Speicherort für temporäre externe Referenzen an. Dieses Verzeichnis wird für die Kopie der externen Referenzen benötigt, wenn die Option MIT KOPIE beim Laden der externen Referenz aktiviert wurde.

WorkspacePath

Datentyp: *String*

In dieser Eigenschaft ist die Position der Datenbank-Quelldatei gespeichert. Wenn Sie Änderungen an dieser Eigenschaft vornehmen, muss AutoCAD erst neu gestartet werden, bevor diese Änderung wirksam wird.

Suchpfad für Projektdaten

Neben den Eigenschaften zur Festlegung von Verzeichnissen enthält das *Files*-Objekt noch zwei Methoden, die ebenfalls Verzeichnisse abrufen bzw. auslesen. In diesen Verzeichnisse sucht AutoCAD dann nach externen Referenzen, wenn diesen kein eindeutiger Pfad zugeordnet wurde. Der Unterschied zu den anderen Verzeichnissen besteht darin, dass diese an Projekt-namen gebunden sind. Dieser Projektname bestimmt dann auch, welche der Verzeichnislisten, von denen es mehrere geben kann, verwendet werden soll. Der Name dieses Projekts wird in der Systemvariablen *PROJECTNAME* gespeichert.

GetProjectFilePath

Parameter: *ProjectName As String*

Rückgabotyp: *String*

Diese Prozedur liefert die Verzeichnisnamen, in denen AutoCAD nach externen Referenzen suchen soll. Im Parameter müssen Sie den Namen des Projekts angeben, dessen Verzeichnisse Sie ermitteln möchten. Unter Verwendung der Systemvariablen *PROJECTNAME* könnte der Aufruf folgendermaßen aussehen:


```
PrjPath = PrefFiles.GetProjectFilePath(ThisDrawing.GetVariable( _  
    "PROJECTNAME"))
```

Die Variable *PrjPath* enthält nun die Verzeichnisse des aktuell eingestellten Projekts.

SetProjectFilePath

Parameter: *ProjectName As String, ProjectFilePath As String*

Diese Prozedur ist das Gegenstück zu *GetProjectFilePath*. Sie können hiermit die Verzeichnisse des angegebenen Projekts in die Registrierung schreiben. Für das aktuelle Projekt sieht dann der Aufruf folgendermaßen aus:

```
PrefFiles.GetProjectFilePath ThisDrawing.GetVariable( _  
    "PROJECTNAME"), PrjPath
```

Eine Liste aller Projektname können Sie in der Registrierung unter dem Schlüssel

HKEY_CURRENT_USER\Software\Autodesk\AutoCAD\ und dann die entsprechende Versionsnummer z.B. R.15\ACAD-2:407\

finden. In diesem Schlüssel sind die einzelnen Profile gespeichert, die wiederum die entsprechenden Projektnamen enthalten.

Dateinamen festlegen

In den nun folgenden Eigenschaften können Sie die Namen verschiedener Dateien festlegen. Dies sind beispielsweise Menü-, Schrift- oder Hilfedateien.

AltFontFile

Datentyp: *String*

Systemvariable: *FONTALT*

Diese Eigenschaft bestimmt die alternative Schriftdatei, die verwendet werden soll, wenn AutoCAD die ursprüngliche Schriftdatei nicht finden kann und in der Schriftzuordnungsdatei keine andere Schrift angegeben wurde.

```
PrefFiles.AltFontFile = "Arial"
```

FontFileMap

Datentyp: *String*

Systemvariable: *FONTMAP*

In dieser Eigenschaft ist die Datei gespeichert, in der definiert ist, wie nicht gefundene Schriftdateien konvertiert werden sollen.

AltTabletMenuFile

Datentyp: *String*

Das alternative Tablettmenü wird in dieser Eigenschaft angelegt. Dieses Menü wird dann statt des vorgegebenen AutoCAD-Tablettmenüs verwendet. Diese Eigenschaft ist nur verfügbar, wenn Sie ein Digitalisiertablett in AutoCAD angemeldet haben.

ConfigFile
(schreibgeschützt)Datentyp: *String*

In dieser Eigenschaft wird der Name der Konfigurationsdatei gespeichert, in der die Informationen zu Hardware-Gerätetreibern enthalten sind.

MainDictionaryDatentyp: *String*Systemvariable: *DCTMAIN*

Diese Eigenschaft speichert den Namen des Hauptwörterbuchs, das zur Rechtschreibprüfung verwendet werden soll. Da *MainDictionary* keinen Dateinamen, sondern ein Kürzel des entsprechenden Wörterbuchs, z.B. „de“, verwendet, sollten Sie vor Veränderung dieser Eigenschaft prüfen, ob der angegebene Wert wirklich einem Hauptwörterbuch entspricht.

CustomDictionaryDatentyp: *String*Systemvariable: *DCTCUST*

Diese Eigenschaft kann den Namen eines benutzerdefinierten Wörterbuchs enthalten, das zusätzlich zum Hauptwörterbuch verwendet werden soll.

DefaultInternetURLDatentyp: *String*Systemvariable: *INETLOCATION*

Diese Eigenschaft legt die vorgegebene Internetadresse fest. Diese wird für die Option VERBINDUNG ZUM INTERNET im Hilfemenü und beim Start eines Browsers über die Schaltfläche BROWSER verwendet. Sie können hier eine URL oder den Namen einer lokalen Internetseite (z.B. *D:\Programme\home.html*) angeben.

MenuFileDatentyp: *String*

In dieser Eigenschaft ist der Dateiname die aktuell verwendete Hauptmenüdatei gespeichert.

PostScriptPrologFileDatentyp: *String*Systemvariable: *PSPPROLOG*

Hier ist der Prologabschnitt festgelegt, der aus der Datei *acad.psf* gelesen werden soll, wenn der Befehl *psout* verwendet wird.

PrintFileDatentyp: *String*

In dieser Eigenschaft können Sie einen alternativen Namen für die temporäre Plotdatei festlegen, die bei Plotskripten aus früheren Versionen von AutoCAD eingesetzt werden soll. Um

den Vorgabenamen zu verwenden (Zeichnungsname + Erweiterung "plt"), müssen Sie als Wert für diese Eigenschaft einen Punkt "." festlegen.

```
PrefFiles.PrintFile = "."
```

PrintSpoolExecutable

Datentyp: *String*

In dieser Eigenschaft kann der Name einer Anwendung festgelegt werden, die zum Plot-Spoolen verwendet werden soll. Sie können hier den Namen der Anwendung plus eventuell dazugehörige Befehlszeilenargumente angeben.

```
PrefFiles.PrintSpoolExecutable = "Spool.bat %s"
```

TextEditor

Datentyp: *String*

Systemvariable: *MTEXTED*

Diese Eigenschaft bestimmt den Texteditor, der für die Bearbeitung von *MText*-Objekten verwendet werden soll. Sie können entweder den Namen des Editors angeben oder den Eintrag „intern“ festlegen, um den internen AutoCAD-Editor zu verwenden.

```
PrefFiles.Texteditor = "intern"
```

LicenseServer ***(schreibgeschützt)***

Datentyp: *String*

Systemvariable: *ACADSERVER*

Mit dieser Eigenschaft können Sie eine Liste der für das Netzwerklicenz-Managerprogramm verfügbaren Client-Lizenzserver abrufen.

5.3.4 Einstellungen zum Öffnen und Speichern von Dateien (OpenSave-Objekt)

Dieses Objekt umfasst einige Eigenschaften, die zur Steuerung von Öffnen- bzw. Speichervorgängen von Dateien verwendet werden. Dies kann z.B. die Angabe sein, ob eine Miniaturansicht des DWG beim Speichern erzeugt werden soll, oder die Erweiterung der automatischen Sicherungsdatei. Zugriff auf die Eigenschaften dieses Objekts erhalten Sie, indem Sie *OpenSave* direkt über das *Preferences*-Objekt ansprechen

```
Application.Preferences.OpenSave
```

oder in einer Objektvariablen einen Verweis auf das *OpenSave*-Objekt erstellen:

```
Dim PrefOpenSave As AcadPreferencesOpenSave  
Set PrefOpenSave = Application.Preferences.OpenSave
```

Sie können nun auf die einzelnen Eigenschaften des *OpenSave*-Objekts zugreifen und damit die entsprechenden Einstellungen zu Öffnen- und Speichervorgängen in AutoCAD vornehmen.

Eigenschaften zum Speichern von Dateien

SaveAsType

Datentyp: *AcSaveAsType*

Diese Eigenschaft legt den Vorgabezeichnungstyp fest, der vorgeschlagen werden soll, wenn die Zeichnung das erste Mal gespeichert wird. In dieser Eigenschaft können Sie eine der folgenden Konstanten festlegen, als deren Ergebnis der entsprechende Dateityp im Speichern unter-Dialog automatisch eingestellt wird.

Konstante	Dateityp	Wert
<i>acR12_DXF</i>	AutoCAD Release12/LT2 DXF (*.dxf)	1
<i>acR13_DWG</i>	AutoCAD Release13/LT95 DWG (*.dwg)	4
<i>acR13_DXF</i>	AutoCAD Release13/LT95 DXF (*.dxf)	5
<i>acR14_DWG</i>	AutoCAD Release14/LT97 DWG (*.dwg)	8
<i>acR14_DXF</i>	AutoCAD Release14/LT97 DXF (*.dxf)	9
<i>acR15_DWG</i>	AutoCAD 2000 DWG (*.dwg)	12
<i>acR15_DXF</i>	AutoCAD 2000 DXF (*.dxf)	13
<i>acR15_Template</i>	AutoCAD 2000 Vorlagendatei (*.dwt)	14
<i>acNative</i>	Wenn Sie diese Konstante wählen, wird der Zeichnungstyp verwendet, der der ausführenden AutoCAD-Version entspricht.	12

Tabelle 5.7: Konstanten der *SaveAsType*-Eigenschaft

`PrefOpenSave.SaveAsType = acR15_DWG`

SavePreviewThumbnail

Datentyp: *Boolean*

Systemvariable: *RASTERPREVIEW*

In dieser Eigenschaft ist festgelegt, ob eine Miniaturansicht der Zeichnung beim Speichern miterzeugt werden soll. Diese Miniaturansicht wird dann für die Zeichnungsvorschau im Öffnen-Dialog angezeigt. Wenn diese Ansicht bei jedem Speichervorgang aktualisiert werden soll, müssen Sie den Wert *TRUE* festlegen, andernfalls *FALSE*.

IncrementalSavePercent

Datentyp: *Long*

Systemvariable: *ISAVEPERCENT*

Mit dieser Eigenschaft können Sie den maximalen Prozentsatz an ungenutztem Speicherplatz einer Zeichnung festlegen. Ungenutzter Speicherplatz entsteht bei der Teilsicherung einer Zeichnung und wird erst durch eine vollständige Speicherung der Zeichnung wieder freigegeben, die AutoCAD durchführt, wenn der angegebene Prozentsatz erreicht wurde. Wenn Sie den Wert dieser Eigenschaft auf 0 setzen, führt AutoCAD keine Teil-, sondern nur noch vollständige Sicherungen durch.

Tipp

Sie sollten den Prozentsatz der Teilsicherung aber nicht zu niedrig einstellen, da eine vollständige Speicherung wesentlich zeitaufwendiger ist als die Sicherung der veränderten Daten. Nach Angaben von AutoCAD verlangsamt ein Wert von 20% oder weniger die Sicherung der Daten erheblich.

Eigenschaften zum Öffnen von Dateien***MRUNumber***

Datentyp: *Long*

Diese Eigenschaft legt die Anzahl der Dateinamen fest, die im AutoCAD-Dateimenü im Abschnitt der zuletzt geöffneten Dateien angezeigt werden sollen.

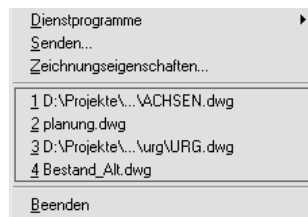


Abbildung 5.12: Zuletzt geöffnete Dateien

Gültige Werte für diese Eigenschaft liegen im Bereich von 0 bis 9. Als Standardwert ist 4 eingestellt.

Eigenschaften zur Steuerung der Datensicherheit***AutoSaveInterval***

Datentyp: *Long*

Systemvariable: *SAVETIME*

Mit dieser Eigenschaft können Sie das Intervall in Minuten einstellen, in dem eine automatische Sicherung der Zeichnung durchgeführt werden soll. Die Sicherung der Zeichnung erfolgt in das Verzeichnis, das Sie in der Eigenschaft *AutoSavePath* im *Files*-Objekt festgelegt haben.

`PrefOpenSave.AutoSaveInterval = 30`

CreateBackup

Datentyp: *Boolean*

Systemvariable: *ISAVEBAK*

Diese Eigenschaft bestimmt, ob eine Sicherungskopie von der Zeichnung bei deren Speicherung angelegt werden soll (*TRUE*) oder nicht (*FALSE*). Diese Sicherungskopie erhält dann die Dateierweiterung *bak* und wird in demselben Verzeichnis erstellt, in dem sich auch die Zeichnungsdatei befindet.

FullCRCValidation

Datentyp: *Boolean*

Mithilfe dieser Eigenschaft können Sie festlegen, ob bei jedem Einlesen eines Objekts in die Zeichnung eine zyklische Redundanzprüfung (CRC) durchgeführt werden soll. Die Aktivierung (*TRUE*) dieser Eigenschaft dient der Fehlerkontrolle und sollte vor allem dann eingesetzt werden, wenn die Dateien beschädigt sein könnten oder eventuell ein Hardware-Problem oder ein AutoCAD-Fehler vermutet wird.

LogFileOn

Datentyp: *Boolean*

Systemvariable: *LOGFILEMODE*

Diese Eigenschaft bestimmt, ob der Inhalt des Befehlszeilenfensters gleichzeitig in eine Protokolldatei geschrieben werden soll. Die Position der Protokolldatei ist in der Eigenschaft *LogFilePath* des *Files*-Objekts abgelegt.

TempFileExtension

Datentyp: *String*

In dieser Eigenschaft wird die Dateierweiterung für temporäre Dateien festgelegt. Die Vorgabe für diese Eigenschaft ist „ac\$“. Sie können aber auch eine andere Dateierweiterung festlegen, mit der beispielsweise die entsprechenden Dateien im Netzwerk sofort identifiziert werden können.

```
PrefOpenSave.TempFileExtension = "us1"
```

AutoAudit

Datentyp: *Boolean*

Die Aktivierung (*TRUE*) dieser Eigenschaft veranlasst AutoCAD zu einer automatischen Prüfung von Dateien, die mit DXFIN oder DXBIN eingelesen werden.

Eigenschaften zur Steuerung von externen Referenzen**XrefDemandLoad**

Datentyp: *AcXRefDemandLoad*

Systemvariable: *XLOADCTL*

Diese Eigenschaft bestimmt das Laden von externen Referenzen nach Bedarf. Dieses Laden nach Bedarf erhöht die Bearbeitungsgeschwindigkeit, da immer nur bestimmte Teile der externen Referenz geladen werden. Sie können für diese Eigenschaft folgende Konstanten verwenden:

- *acDemandLoadDisabled*: Deaktiviert das Laden bei Bedarf.
- *acDemandLoadEnabled*: Aktiviert das Laden bei Bedarf und erhöht damit die Verarbeitungsgeschwindigkeit von AutoCAD. Wenn Sie diese Einstellung wählen, können andere Benutzer nicht auf die Zeichnungsdatei zugreifen, solange diese in der Zeichnung referenziert wird.

- *acDemandLoadEnabledWithCopy*: Aktiviert ebenfalls das Laden bei Bedarf. AutoCAD verwendet allerdings eine Kopie der externen Referenz, so dass andere Benutzer mit der Originalzeichnung weiterarbeiten können.

Eigenschaften zur Steuerung von Drittanbieterobjekten

AutoCAD verfügt über die Möglichkeit, Objekte von Drittanbietern, die durch zusätzliche ObjectARX-Applikationen erzeugt wurden, in den Zeichnungen zu verwalten. Diese Objekte können aber nur angezeigt werden, wenn die entsprechende Anwendung vorhanden ist und geladen werden kann. Die folgenden Eigenschaften steuern das Zeichnungsverhalten bei der Verwendung dieser benutzerdefinierten Objekte.

DemandLoadARXApp

Datentyp: *AcARXDemandLoad*

Systemvariable: *DEMANDLOAD*

Mit dieser Eigenschaft können Sie steuern, ob und wann ObjectARX-Anwendungen geladen werden sollen, wenn eine Zeichnung benutzerdefinierte Objekte enthält, die mit eben dieser Anwendung erzeugt wurden. Sie können für diese Eigenschaft eine der folgenden Konstanten verwenden.

- *acDemandLoadDisable*: Deaktiviert das Laden bei Bedarf.
- *acDemandLoadOnObjectDetect*: Diese Einstellung bewirkt, dass die entsprechende Anwendung geladen wird, wenn AutoCAD ein Zeichnungsobjekt findet, das mit dieser Anwendung erstellt wurde. Die Anwendung wird aber nicht geladen, wenn der Benutzer einen Befehl dieser Anwendung in der Befehlszeile ausführt.
- *acDemandLoadCmdInvoke*: Diese Einstellung bewirkt, dass AutoCAD eine Anwendung automatisch lädt, wenn der Benutzer einen Befehl aus dieser Anwendung in der Befehlszeile ausführt.

Sie können auch die Konstanten zum Laden der Anwendung kombinieren, so dass die Anwendung sowohl beim Erkennen eines benutzerdefinierten Objekts als auch bei der Eingabe eines Befehls der entsprechenden Anwendung geladen wird.

```
PrefOpenSave.DemandLoadARXApp = acDemandLoadOnObjectDetect + _  
AcDemandLoadCmdInvoke
```

ProxyImage

Datentyp: *AcProxyImage*

Diese Eigenschaft steuert die Anzeige von benutzerdefinierten Objekten. Sie können mit einer der folgenden Konstanten festlegen, ob und wie diese Objekte in der Zeichnung dargestellt werden sollen.

- *acProxyNotShow*: Deaktiviert die Anzeige von benutzerdefinierten Objekten.
- *acProxyShow*: Die benutzerdefinierten Objekte werden in der Zeichnung komplett angezeigt.
- *acProxyBoundingBox*: Anstelle der benutzerdefinierten Objekte werden in der Zeichnung deren Umgrenzungsrahmen dargestellt.

ShowProxyDialogBoxDatentyp: *Boolean*Systemvariable: *PROXYNOTICE*

Durch Aktivierung dieser Eigenschaft (*TRUE*) legen Sie fest, dass AutoCAD eine Warnmeldung anzeigt, wenn der Benutzer eine Zeichnung öffnet, in der benutzerdefinierte Objekte vorhanden sind.

5.3.5 Ploteinstellungen (Output-Objekt)

Mit den Eigenschaften dieses Objekts können Sie einige Einstellungen zum Plotverhalten von AutoCAD steuern. Diese Einstellungen finden Sie auch im Abschnitt *PLOT* des Optionen-Dialogs. Zugriff auf die Eigenschaften dieses Objekts erhalten Sie, indem Sie *Output* direkt über das *Preferences*-Objekt ansprechen

```
Application.Preferences.Output
```

oder in einer Objektvariablen einen Verweis auf das *Output*-Objekt erstellen:

```
Dim PrefOutput As AcadPreferencesOutput  
Set PrefOutput = Application.Preferences.Output
```

Sie können nun auf die einzelnen Eigenschaften des *Output*-Objekts zugreifen und damit die entsprechenden Einstellungen zur AutoCAD-Plot-Ausgabe vornehmen.

Eigenschaften zur Steuerung des Verhaltens von Plotstilen**DefaultPlotStyleForLayer**Datentyp: *String*Systemvariable: *DEFLPLSTYLE*

Diese Eigenschaft gibt den Vorgabe-Plotstil des Layers „0“ für neue Zeichnungen an bzw. für Zeichnungen, die in früheren Versionen von AutoCAD erstellt und nicht im AutoCAD 2000-Format gespeichert wurden.

```
PrefOutput.DefaultPlotStyleForLayer = "Normal"
```

DefaultPlotStyleForObjectsDatentyp: *String*Systemvariable: *DEFLPLSTYLE*

Diese Eigenschaft richtet den Vorgabe-Plotstil für neue Zeichnungsobjekte ein.

```
PrefOutput.DefaultPlotStyleForObjects = "ByLayer"
```

DefaultPlotStyleTableDatentyp: *String*

Mit dieser Eigenschaft können Sie die Vorgabe-Plotstil-Tabelle festlegen, die einer Zeichnung zugeordnet werden soll. Je nachdem, welche Farbtabelle Sie in dieser Eigenschaft einstellen,

können auch die Werte der vorherigen beiden Eigenschaften angepasst werden, da die entsprechenden Plotstile dieser Tabelle entnommen werden.

```
PrefOutput.DefaultPlotStyleTable = "acad.stb"
```

PlotPolicy

Datentyp: *AcPlotPolicy*

Systemvariable: *PSTYLEPOLICY*

Der Wert dieser Eigenschaft legt fest, ob benannte oder farbabhängige Plotstile verwendet werden sollen. Folgende Konstanten können verwendet werden.

- *AcPolicyNamed*: AutoCAD verwendet benannte Plotstile in neuen Zeichnungen und in Zeichnungen, die in früheren AutoCAD-Versionen erzeugt wurden. Benannte Plotstile werden in Dateien mit der Erweiterung „stb“ gespeichert.
- *AcPolicyLegacy*: AutoCAD verwendet farbabhängige Plotstile in neuen Zeichnungen und in Zeichnungen, die in früheren AutoCAD-Versionen erzeugt wurden. Diese Plotstile sind in Dateien mit der Erweiterung „ctb“ gespeichert.

Eigenschaften zur Steuerung von Vorgaben für neue Zeichnungen und Layouts

DefaultOutputDevice

Datentyp: *String*

Diese Eigenschaft bestimmt das Ausgabegerät, das als Vorgabe für neue Layouts festgelegt werden soll. Das Ausgabegerät ergibt sich aus der zugehörigen PC3-Datei, die in dem entsprechenden Suchverzeichnis für Plotkonfigurationsdateien abgelegt werden.

Um eine Liste der verfügbaren Ausgabegeräte zu ermitteln, können Sie die Layoutanweisung *GetPlotDeviceNames* verwenden, die in *Kapitel 7.6.2* genauer beschrieben ist.

UseLastPlotSettings

Datentyp: *Boolean*

Mit dieser Eigenschaft können Sie festlegen, ob die Ploteinstellungen des letzten erfolgreichen Plots für neue Zeichnungen und/oder Layouts verwendet werden sollen.

Allgemeine Ausgabeeigenschaften

OLEQuality

Datentyp: *AcOleQuality*

Systemvariable: *OLEQUALITY*

Diese Eigenschaft steuert die Ausgabequalität von OLE-Objekten. Sie können für diese Eigenschaft eine der folgenden Konstanten verwenden, um die entsprechende Ausgabequalität einzustellen.

Konstante	Ausgabequalität	Wert
<i>acOQLineArt</i>	Linien, z.B. Exceltabellen	0
<i>acOQText</i>	Textausgabe, z.B. Textdokumente	1
<i>acOQGraphics</i>	Einfache Grafiken, z.B. Diagramme	2
<i>acOQPhoto</i>	Fotoqualität	3
<i>acOQHighPhoto</i>	Qualitativ hochwertige Fotos	4

`PrefOutput.OLEQuality = acOQPhoto`

PlotLegacy

Datentyp: *Boolean*

Systemvariable: *PLOTLEGACY*

Mit dieser Eigenschaft können Sie die Ausführung von Legacy-Plotskripten aktivieren (*TRUE*) bzw. deaktivieren (*FALSE*).

PrinterPaperSizeAlert

Datentyp: *Boolean*

Systemvariable: *PAPERALERT*

Wenn Sie diese Eigenschaft aktivieren (*TRUE*), dann wird der Benutzer benachrichtigt, sobald er ein Papierformat in einem Layout einstellt, das von den Vorgaben der Plotkonfigurationsdatei (*pc3*) abweicht.

PrinterSpoolAlert

Datentyp: *AcPrinterSpoolAlert*

Systemvariable: *PSPOOLALERT*

Mit dieser Eigenschaft bestimmen Sie, ob eine Warnung ausgegeben werden soll, wenn die geplottete Zeichnung aufgrund eines Konflikts mit dem Ein- oder Ausgabeanschluss durch einen Systemdrucker gespoolt werden muss. Sie können für diese Eigenschaft vier mögliche Einstellungen festlegen, die durch folgende Konstanten aktiviert werden.

- *AcPrinterAlwaysAlert*: Die Festlegung dieses Werts bewirkt, dass der Benutzer immer gewarnt und der entsprechende Fehler in der Protokolldatei geschrieben wird, wenn der Plot durch den Systemdrucker gespoolt wird.
- *AcPrinterAlertOnce*: Der Benutzer wird nur beim ersten Auftreten eines Fehlers gewarnt, es werden jedoch alle Fehler in die Protokolldatei geschrieben, wenn der Plot durch den Systemdrucker gespoolt wird.
- *AcPrinterNeverAlertLogOnce*: Es wird keine Meldung bei Auftreten eines Fehlers ausgegeben, die Fehler werden jedoch in die Protokolldatei geschrieben, wenn der Plot durch den Systemdrucker gespoolt wird.
- *AcPrinterNeverAlert*: Bei Auftreten eines Fehlers wird weder der Benutzer benachrichtigt noch der entsprechende Fehler protokolliert.

5.3.6 Auswahleinstellungen (Selection-Objekt)

Mit den Eigenschaften dieses Objekts nehmen Sie einige Einstellungen zum Verhalten von AutoCAD bei der Auswahl von Objekten vor. Diese Einstellungen finden Sie auch im Abschnitt AUSWAHL des Optionen-Dialogs. Zugriff auf die Eigenschaften dieses Objekts erhalten Sie, indem Sie *Selection* direkt über das *Preferences*-Objekt ansprechen

```
Application.Preferences.Selection
```

oder in einer Objektvariablen einen Verweis auf das *Selection*-Objekt erstellen:

```
Dim PrefSelection As AcadPreferencesSelection  
Set PrefSelection = Application.Preferences.Selection
```

Sie können nun auf die einzelnen Eigenschaften des *Selection*-Objekts zugreifen und damit die entsprechenden Einstellungen zur AutoCAD-Objektauswahl vornehmen.

Einstellung der Auswahlmodi

Die folgenden Eigenschaften finden Sie auch im Optionen-Dialog im Abschnitt AUSWAHL-MODI. Diese legen fest, wie der Benutzer einen Auswahl Satz von Objekten in der Zeichnung wählen kann, ohne einen AutoCAD-Befehl auszuführen.

PickFirst

Datentyp: *Boolean*

Systemvariable: *PICKFIRST*

Diese Eigenschaft bestimmt, ob Objekte, die vor der Ausführung eines AutoCAD-Befehls gewählt wurden, an diesen Befehl weitergegeben werden. Wenn der Benutzer beispielsweise Objekte wählt und anschließend den Befehl `_copy` ausführt, erzielt die Einstellung dieser Eigenschaft unterschiedliche Ergebnisse. Haben Sie den Wert *TRUE* eingestellt, werden die ausgewählten Objekte mit dem Befehl weiterverwendet. Wurde *FALSE* festgelegt, wird die Objektauswahl verworfen und der Benutzer muss mit dem Befehl neue Objekte auswählen.

```
PrefSelection.PickFirst = True
```

PickAdd

Datentyp: *Boolean*

Systemvariable: *PICKADD*

Mithilfe dieser Eigenschaft legen Sie fest, wie der Benutzer Objekte zu einem Auswahl Satz hinzufügen bzw. entfernen kann. Wenn Sie diese Eigenschaft auf den Wert *TRUE* setzen, dann muss der Benutzer die -Taste gedrückt halten, um Objekte zum Auswahl Satz hinzufügen zu können. Ist diese Eigenschaft auf *FALSE* gesetzt, dann bewirkt die Auswahl mit gedrückter Taste das Entfernen der selektierten Objekte aus dem Auswahl Satz.

PickAuto

Datentyp: *Boolean*

Systemvariable: *PICKAUTO*

Diese Eigenschaft bestimmt, ob der Benutzer die Objekte einzeln (*FALSE*) oder auch zusätzlich über ein Fenster (*TRUE*) auswählen kann, wenn der Benutzer einen Punkt außerhalb eines Objekts gewählt hat.

PickDrag

Datentyp: *Boolean*

Systemvariable: *PICKDRAG*

Wenn Sie diese Eigenschaft aktivieren (*TRUE*), muss der Benutzer das Fenster zur Auswahl der Objekte in einem Vorgang, ohne die Maustaste loszulassen, durchziehen. Ist diese Eigenschaft deaktiviert (*FALSE*), dann kann der Benutzer zwei einzelne Punkte bestimmen, die das Auswahlfenster definieren.

PickGroup

Datentyp: *Boolean*

Systemvariable: *PICKSTYLE*

Diese Eigenschaft legt fest, ob bei der Auswahl eines Objekts, das zu einer Gruppendefinition gehört, die gesamte Gruppe gewählt werden soll (*TRUE*) oder nur das einzelne Objekt (*FALSE*).

PickBoxSize

Datentyp: *Long*

Systemvariable: *PICKBOX*

Diese Eigenschaft bestimmt die Größe der Auswahlbox, die im Fadenkreuz angezeigt wird, wenn kein AutoCAD-Befehl ausgeführt wird. Objekte, die sich in Reichweite dieser Box bei der Auswahl befinden, werden selektiert. Je kleiner Sie den Wert für diese Eigenschaft wählen, umso dichter muss sich das Fadenkreuz bei der direkten Auswahl an dem Objekt befinden, um es auszuwählen. Der Wertebereich dieser Eigenschaft liegt zwischen 0 und 50.

`PrefSelection.PickBoxSize = 6`

Einstellung der Griffe steuern

Die folgenden Eigenschaften finden Sie auch im Optionen-Dialog im Abschnitt GRIFFE. Griffe werden in Form von kleinen Quadraten bei ausgewählten Objekten dargestellt.

DisplayGrips

Datentyp: *Boolean*

Systemvariable: *GRIPS*

Diese Eigenschaft steuert die Anzeige der Griffe. Wenn diese Eigenschaft auf *TRUE* gesetzt wurde, werden für alle ausgewählten Objekte Griffe angezeigt, über die der Benutzer einfach die Position des Griffs verändern oder auf ein Kontextmenü zugreifen kann. Durch Deaktivierung (*FALSE*) dieser Eigenschaft kann die Systemleistung wesentlich gesteigert werden.

DisplayGripsWithinBlocks

Datentyp: *Boolean*

Systemvariable: *GRIPBLOCK*

Wenn Sie diese Eigenschaft deaktivieren (*FALSE*), dann wird bei der Auswahl von Blockreferenzen lediglich der Griff, der den Einfügapunkt des Blocks darstellt, angezeigt. Setzen Sie den

Wert der Eigenschaft auf *TRUE*, werden auch die Griffe der Objekte angezeigt, die den Block definieren.



Abbildung 5.13: Blockreferenz ohne und mit aktivierten Griffen

GripColorUnselected

Datentyp: *AcColor*

Systemvariable: *GRIPCOLOR*

Diese Eigenschaft bestimmt die Farbe der Griffe, wenn dieser angezeigt, aber nicht durch den Benutzer ausgewählt wurde. Sie können eine der AutoCAD-Grundfarben für diese Eigenschaft verwenden, wie sie in Tabelle 5.6 beschrieben sind.

GripColorSelected

Datentyp: *AcColor*

Systemvariable: *GRIPHOT*

Mit *GripColorSelected* legen Sie die Farbe fest, in der der Griff dargestellt wird, wenn der Benutzer diesen mit dem Fadenkreuz ausgewählt hat. Sie können eine der AutoCAD-Grundfarben für diese Eigenschaft verwenden, wie sie in Tabelle 5.6 beschrieben sind.

GripSize

Datentyp: *Long*

Systemvariable: *GRIPSIZE*

GripSize bestimmt die Anzeigegröße der Griffe. Sie können für diese Eigenschaft einen Wert zwischen 0 und 255 festlegen. Allerdings sollte der Wert von maximal 20 Einheiten, wie er im Optionen-Dialog eingestellt werden kann, nicht überschritten werden, da die Griffe sonst zu groß werden.

5.3.7 Systemeinstellungen (System-Objekt)

Mit den Eigenschaften dieses Objekts nehmen Sie einige Einstellungen zum allgemeinen Verhalten von AutoCAD vor. Diese Einstellungen finden Sie auch im Abschnitt *SYSTEM* des Optionen-Dialogs. Zugriff auf die Eigenschaften dieses Objekts erhalten Sie, indem Sie *System* direkt über das *Preferences*-Objekt ansprechen

```
Application.Preferences.System
```

oder in einer Objektvariablen einen Verweis auf das *System*-Objekt erstellen:

```
Dim PrefSystem As AcadPreferencesSystem
Set PrefSystem = Application.Preferences.System
```

Sie können nun auf die einzelnen Eigenschaften des *System*-Objekts zugreifen und damit die entsprechenden Einstellungen zur AutoCAD-Sitzung vornehmen.

Allgemeine Systemeinstellung

BeepOnError

Datentyp: *Boolean*

Mit dieser Eigenschaft können Sie festlegen, ob AutoCAD ein akustisches Signal ausgeben soll, wenn der Benutzer eine ungültige Eingabe tätigt.

`PrefSystem.BeepOnError = False`

ShowWarningMessages

Datentyp: *Boolean*

Wenn Sie diese Eigenschaft aktivieren (*TRUE*), dann werden alle Warnmeldungen zurückgesetzt, die die Option DIESES DIALOGFELD NICHT MEHR ANZEIGEN enthalten. Die Meldungen werden dann zukünftig wieder angezeigt.

LoadAcadLspInAllDocuments

Datentyp: *Boolean*

Systemvariable: *ACADLSPASDOC*

Diese Eigenschaft bestimmt, ob die Datei *acad.lsp* in jede Zeichnung geladen wird. Wenn Sie diese Eigenschaft deaktivieren (*FALSE*), wird lediglich die Datei *acaddoc.lsp* in allen Zeichnungsdateien geladen.

SingleDocumentMode

Datentyp: *Boolean*

Systemvariable: *SDI*

Mit dieser Eigenschaft können Sie festlegen, ob die AutoCAD-Sitzung im SDI-Modus (Single Document Interface) oder im MDI-Modus (Multiple Document Interface) ausgeführt werden soll.

EnableStartupDialog

Datentyp: *Boolean*

Mit *EnableStartupDialog* können Sie festlegen, ob der Startdialog beim Start einer AutoCAD-Sitzung angezeigt werden soll (*TRUE*) oder nicht (*FALSE*).

DisplayOLEScale

Datentyp: *Boolean*

Diese Eigenschaft bestimmt, ob das Dialogfeld zur OLE-Skalierung angezeigt wird, wenn OLE-Objekte in eine Zeichnung eingefügt werden. Wenn Sie diese Eigenschaft deaktivieren (*FALSE*), wird das Dialogfenster erst nach Anforderung durch den Benutzer über ein Kontextmenü eingeblendet.



Abbildung 5.14: Eigenschaftsdialog für OLE-Objekte

Optionen für Datenbankanbindungen

StoreSQLIndex

Datentyp: *Boolean*

StoreSQLIndex bestimmt, ob der Datenbankindex in der Zeichnung gespeichert werden soll. Wenn Sie diese Eigenschaft aktivieren (*TRUE*), wird der Index direkt in der Zeichnung gespeichert, um die Leistung während der Verknüpfungsauswahl zu verbessern.

TablesReadOnly

Datentyp: *Boolean*

TablesReadOnly legt fest, ob Datenbanktabellen in der AutoCAD-Zeichnung schreibgeschützt geöffnet werden sollen (*TRUE*) oder nicht (*FALSE*).

5.3.8 Benutzereinstellungen (User-Objekt)

Mit den Eigenschaften dieses Objekts können Sie einige Einstellungen zur Optimierung der Arbeit des Benutzers mit AutoCAD steuern. Diese Einstellungen finden Sie auch im Abschnitt BENUTZEREINSTELLUNGEN des Optionen-Dialogs. Zugriff auf die Eigenschaften dieses Objekts erhalten Sie, indem Sie *User* direkt über das *Preferences*-Objekt ansprechen

```
Application.Preferences.User
```

oder in einer Objektvariablen einen Verweis auf das *User*-Objekt erstellen:

```
Dim PrefUser As AcadPreferencesUser  
Set PrefUser = Application.Preferences.User
```

Sie können nun auf die einzelnen Eigenschaften des *User*-Objekts zugreifen und damit die entsprechenden Einstellungen zur AutoCAD-Sitzung vornehmen.

AutoCAD-Verhalten bei Koordinateneingabe

KeyboardPriority

Datentyp: *AcKeyboardPriority*

Systemvariable: *OSNAPCOORD*

Diese Eigenschaft bestimmt, wie AutoCAD auf die Eingabe von Koordinaten reagieren soll. Für diese Eigenschaft können Sie folgende Konstanten verwenden.

- *AcKeyboardRunningObjSnap*: Die Auswahl von Koordinaten über den laufenden Objektfang hat Vorrang vor den eingegebenen Koordinaten.
- *AcKeyboardEntry*: Eingegebene Koordinaten haben immer Vorrang vor den Koordinaten, die mit dem laufenden Objektfang ermittelt wurden.
- *acKeyboardEntryExceptScripts*: Eingegebene Koordinaten haben Vorrang vor den Koordinaten, die mit dem laufenden Objektfang ermittelt wurden. Ausgenommen hiervon sind Koordinaten, die über Skripte bestimmt werden.

`PrefUser.KeyboardPriority = AcKeyboardRunningObjSnap`

Tastaturverhalten

KeyboardAccelerator

Datentyp: *AcKeyboardAccelerator*

Mit dieser Eigenschaft können Sie festlegen, welche Standardzugriffstasten AutoCAD verwenden soll. Wenn Sie diese Eigenschaft auf den AutoCAD-Standard umstellen (Konstante *acPreferenceClassic*), dann werden die Zugriffstasten nach dem AutoCAD-Standard interpretiert. Die Tastenkombination `[Strg] + [C]` würde dann den Abbruch-Befehl ausführen. Wenn Sie die Eigenschaft auf die Konstante *acPreferenceCustom* umstellen, würde `[Strg] + [C]` den CopyC-`lip`-Befehl ausführen.

Funktionalität des rechten Mausklicks anpassen

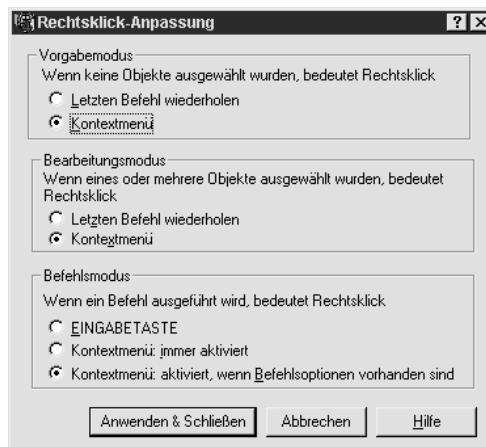


Abbildung 5.15: Rechtsklick-Anpassung über den Optionen-Dialog

Mit den folgenden Eigenschaften können Sie die Funktionalität von AutoCAD anpassen, wenn der Benutzer die rechte Maustaste drückt. Dies ist davon abhängig, wann dies passiert. Das Betätigen der rechten Maustaste hat beispielsweise unterschiedliche Wirkungen, wenn gerade ein AutoCAD-Befehl ausgeführt wird oder nicht. Die Einstellungen zu den folgenden Eigenschaften kann der Benutzer auch über den Optionen-Dialog aufrufen, indem er die Schaltfläche RECHTSKlick-ANPASSUNG betätigt.

Siehe auch *Kapitel 5.1.1*.

ShortCutMenuDisplay

Datentyp: *Boolean*

Systemvariable: *SHORTCUTMENU*

ShortCutMenuDisplay bestimmt, ob ein Kontextmenü im Zeichenbereich angezeigt wird, wenn der Benutzer die rechte Maustaste betätigt. Legen Sie für diese Eigenschaft den Wert *TRUE* fest, um das Kontextmenü einzublenden. Der Wert *FALSE* legt fest, dass die rechte Maustaste die Eingabetaste ausführt.

SCMCommandMode

Datentyp: *AcDrawingAreaSCMCommand*

Mit dieser Eigenschaft können Sie die Funktionalität von AutoCAD einstellen, wenn der Benutzer die rechte Maustaste betätigt, während ein AutoCAD-Befehl ausgeführt wird. Folgende Konstanten können Sie für die Einstellung dieser Eigenschaft verwenden:

Konstante	Beschreibung	Wert
<i>acEnter</i>	Der Rechtsklick mit der Maus führt die Eingabetaste aus.	0
<i>acEnableSCMOptions</i>	Der Rechtsklick mit der Maus aktiviert immer ein Kontextmenü.	1
<i>acEnableSCM</i>	Der Rechtsklick mit der Maus aktiviert dann ein Kontextmenü, wenn der ausgeführte Befehl über Optionen verfügt.	2

SCMDefaultMode

Datentyp: *AcDrawingAreaSCMDefault*

Mit dieser Eigenschaft können Sie die Funktionalität von AutoCAD einstellen, wenn der Benutzer die rechte Maustaste betätigt, während sich AutoCAD im Ruhezustand befindet. Folgende Konstanten können Sie für die Einstellung dieser Eigenschaft verwenden:

Konstante	Beschreibung	Wert
<i>acRepeatLastCommand</i>	Der Rechtsklick mit der Maus führt den zuletzt verwendeten AutoCAD-Befehl aus.	0
<i>acSCM</i>	Der Rechtsklick mit der Maus aktiviert das entsprechende Kontextmenü.	1

SCMEditMode

Datentyp: *AcDrawingAreaSCMEdit*

Mit dieser Eigenschaft bestimmen Sie die Funktionalität von AutoCAD, wenn der Benutzer Objekte ausgewählt hat und dann die rechte Maustaste betätigt. Folgende Konstanten können Sie für die Einstellung dieser Eigenschaft verwenden:

Konstante	Beschreibung	Wert
<i>acRepeatLastCommand</i>	Der Rechtsklick mit der Maus führt den zuletzt verwendeten AutoCAD-Befehl aus.	0
<i>acSCM</i>	Der Rechtsklick mit der Maus aktiviert das entsprechende Kontextmenü.	1

Hyperlink-Einstellungen

(siehe auch *Kapitel 7.3*)

HyperlinkDisplayCursor

Datentyp: *Boolean*

HyperlinkDisplayCursor bestimmt die Anzeige des Cursors und des Kontextmenüs für Hyperlinks. Der Hyperlink-Cursor wird neben dem Fadenkreuz angezeigt, wenn der Benutzer mit dem Mauszeiger auf ein Objekt zeigt, das eine Objektverbindung enthält.

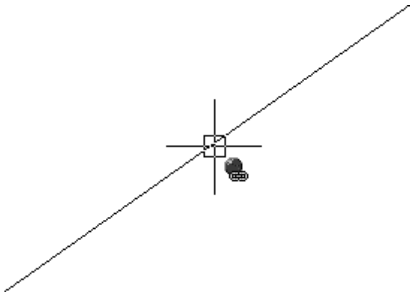


Abbildung 5.16: *Hyperlink-Symbol*

Im Hyperlink-Kontextmenü stehen dem Benutzer zusätzliche Optionen zur Verfügung, wenn er ein Objekt mit einem Hyperlink auswählt und mit der rechten Maustaste in den Zeichenbereich klickt. Wenn Sie diese Eigenschaft deaktivieren (*FALSE*), wird der Hyperlink-Cursor nicht angezeigt und die Option *HYPERLINK* ist im Kontextmenü nicht verfügbar (sofern Kontextmenüs aktiviert wurden).

HyperlinkDisplayTooltip

Datentyp: *Boolean*

Zusätzlich zum Symbol könnten Sie einen gelben Infotext aktivieren (*TRUE*), der das Verbindungsziel des Hyperlinks anzeigt.

Einheiteneinstellungen für das AutoCAD-DesignCenter

ADCInsertUnitsDefaultSource

Datentyp: *AcInsertUnits*

Systemvariable: *INSUNITS* / *INSUNITSDEFSOURCE*

Diese Eigenschaft legt fest, welche Einheit automatisch für Objekte verwendet werden soll, wenn diese in die AutoCAD-Zeichnung eingefügt werden und keine Einfügeeinheiten festgelegt wurden. Die Konstanten, die Sie für diese Eigenschaft verwenden können, bestimmen diese Einheit.

Konstante	Einheit	Wert
<i>acInsertUnitsUnitless</i>	Ohne Einheit	0
<i>acInsertUnitsInches</i>	Zoll	1
<i>acInsertUnitsFeet</i>	Fuß	2
<i>acInsertUnitsMiles</i>	Meilen	3
<i>acInsertUnitsMillimeters</i>	Millimeter	4
<i>acInsertUnitsCentimeters</i>	Zentimeter	5
<i>acInsertUnitsMeters</i>	Meter	6
<i>acInsertUnitsKilometers</i>	Kilometer	7
<i>acInsertUnitsMicroinches</i>	Mikro-Zoll	8
<i>acInsertUnitsMils</i>	Milli-Zoll	9
<i>acInsertUnitsYards</i>	Yards	10
<i>acInsertUnitsAngstroms</i>	Angström	11
<i>acInsertUnitsNanometers</i>	Nanometer	12
<i>acInsertUnitsMicrons</i>	Mikrometer	13
<i>acInsertUnitsDecimeters</i>	Dezimeter	14
<i>acInsertUnitsDecameters</i>	Dekameter	15
<i>acInsertUnitsHectometers</i>	Hektometer	16
<i>acInsertUnitsGigameters</i>	Gigameter	17
<i>acInsertUnitsAstronomicalUnits</i>	Astronomische Einheiten	18
<i>acInsertUnitsLightYears</i>	Lichtjahre	19
<i>acInsertUnitsParsecs</i>	Parsecs	20

Tabelle 5.8: Einheitenkonstanten

ADCInsertUnitsDefaultTarget

Datentyp: *AcInsertUnits*

Systemvariable: *INSUNITS* / *INSUNITSDEFTARGET*

Diese Eigenschaft legt die in der AutoCAD-Zeichnung verwendeten Einheiten fest, wenn keine Einfügeeinheiten definiert wurden. Die Konstanten, die in Tabelle 5.8 aufgelistet sind, bestimmen diese Einheit.

5.3.9 Benutzerprofile (Profiles-Objekt)

(Beispielprojekt: 05\Profiles.dvb)

Um die Einstellungen, die Sie im *Preferences*-Objekt vorgenommen haben, nicht zu verlieren oder dem Benutzer bei bestimmten Gelegenheiten wieder zur Verfügung zu stellen, bietet AutoCAD bekanntlich die Möglichkeit, diese unter einem Konfigurationsnamen (Profil) zu verwalten. In VBA werden diese Profile mit dem *Profiles*-Objekt angesprochen. Dieses Objekt enthält die verschiedensten Methoden, um Profile zu speichern, auszutauschen oder wieder zu löschen.

Zugriff auf die Eigenschaften und Methoden dieses Objekts erhalten Sie, indem Sie *Profiles* direkt über das *Preferences*-Objekt ansprechen

```
Application.Preferences.Profiles
```

oder in einer Objektvariablen einen Verweis auf das *Profiles*-Objekt erstellen:

```
Dim PrefProfiles As AcadPreferencesProfiles  
Set PrefProfiles = Application.Preferences.Profiles
```

Sie können nun auf die einzelnen Eigenschaften des *Profiles*-Objekts zugreifen und damit die entsprechenden Einstellungen zur AutoCAD-Sitzung vornehmen.

Aktuelles Profil ermitteln/zuweisen

ActiveProfile

Datentyp: *String*

Die Eigenschaft *ActiveProfile* bestimmt die aktuellen Benutzereinstellungen. Dabei ist es lediglich erforderlich, den Namen des Profils anzugeben, dessen Einstellungen übernommen werden sollen.

```
PrefProfiles.ActiveProfile = "MeinProfil"
```

Natürlich muss ein Profil mit dem angegebenen Namen existieren, da sonst ein Fehler ausgelöst wird. Eine Liste mit allen Profilnamen können Sie mit der Prozedur *GetAllProfileNames* ermitteln.

GetAllProfileNames

Parameter: *pNames*

Mit dieser Prozedur erstellen Sie, wie zuvor bereits erwähnt, eine Liste mit allen Profilnamen, die in AutoCAD angelegt wurden. Diese Liste wird im Parameter *pNames* als Array zurückgegeben, wobei in jedem Feld ein Profilname gespeichert wird. Da die Anzahl der existenten Profile im voraus nicht zu ermitteln ist (*Profiles* besitzt keine *Count*-Eigenschaft), müssen Sie die Obergrenze des Arrays mit der *UBound()*-Funktion bestimmen. Somit könnten Sie das zurückgegebene Array mit einer *For...Next*-Schleife durchlaufen und so auf jeden Profilnamen zugreifen und beispielsweise in einem ListBox-Steuererelement darstellen.

```
Dim pList As Variant  
PrefProfiles.GetAllProfileNames pList  
For I = 0 To UBound(pList)  
    ListBox1.AddItem pList(i)  
Next
```

Der Zugriff auf die Beschreibung der einzelnen Profile ist schon etwas schwieriger. Sie müssen dazu auf die Windows-Registrierung zugreifen und die Daten aus dem Schlüssel, der gleichlautend mit dem Profilnamen ist, auslesen. Dort würden Sie dann auch den Namen des Benutzers erfahren, der das entsprechende Profil angelegt hat.

Neues Profil erstellen

Die Vorgehensweise zur Erstellung eines neuen Profils ist vielleicht etwas unlogisch, muss aber über die Funktion *CopyProfile* vorgenommen werden. Da wir davon ausgehen, dass die aktuellen Einstellungen durch das aktive Profil dargestellt werden, können wir dieses auch dazu verwenden, ein neues Profil zu erstellen.

CopyProfile

Parameter: *oldProfileName As String, newProfileName As String*

Die *CopyFile*-Anweisung kopiert das im Parameter *oldProfileName* angegebene Profil in das im Parameter *newProfileName* angegebene Profil. Alle Einstellungen des Ausgangsprofils werden dabei auf das Zielprofil übertragen. Wird im Parameter *newProfileName* der Name eines Profils angegeben, das noch nicht existiert, erstellt AutoCAD automatisch ein neues Profil. Diesen Umstand können wir nutzen, um so ein neues Profil mit den aktuellen Einstellungen zu erzeugen.

```
Dim sValue As String
On Error Resume Next
sValue = InputBox("Geben Sie den Namen des neuen Profils ein:", _
    "Neues Profil")
If sValue = "" Then Exit Sub
PrefProfiles.CopyProfile PrefProfiles.ActiveProfile, sValue
```

Wenn Sie den Namen eines bereits vorhandenen Profils als Ziel des Kopiervorgangs angeben, werden die Einstellungen des Ausgangsprofils auf das Zielprofil übertragen.

Profile verwalten

RenameProfile

Parameter: *origProfileName As String, newProfileName As String*

Mit der Anweisung *RenameProfile* können Sie den Namen eines vorhandenen Profils ändern. Im ersten Parameter legen Sie hierfür den Namen des Profils fest, das umbenannt werden soll. Der zweite Parameter bestimmt schließlich den neuen Namen für das angegebene Profil. Wenn Sie als neuen Profilnamen den Namen eines bereits vorhandenen Profils verwenden, dann wird dieses mit den Daten des Ausgangsprofils überschrieben. Im folgenden kleinen Beispiel wird das aktive Profil in den Namen umbenannt, den der Benutzer über einen Eingabedialog festlegt.

```
Dim sValue As String
On Error Resume Next
sValue = InputBox("Geben Sie den neuen Namen für das aktive" & _
    " Profil ein:", "Neues Profil")
If sValue = "" Then Exit Sub
PrefProfiles.RenameProfile PrefProfiles.ActiveProfile, sValue
```

DeleteProfile

Parameter: *ProfileName As String*

Mit dieser Anweisung können Sie ein vorhandenes Profil aus AutoCAD entfernen. Sie müssen dazu lediglich den Namen des entsprechenden Profils im Parameter *ProfileName* festlegen. Um beispielsweise das Profil zu löschen, dessen Namen der Benutzer in einer ListBox gewählt hat (siehe Beispielprojekt), könnten Sie folgendermaßen vorgehen:

```
On Error Resume Next
If MsgBox("Soll das Profil " & ListBox1.Text & _
    " wirklich gelöscht werden?", vbYesNo, "Löschen") = vbYes Then
    Err.Clear
    PrefProfiles.DeleteProfile ListBox1.Text
    If Err = 0 Then
        ListBox1.RemoveItem ListBox1.ListIndex
    Else
        MsgBox Err.Description, vbCritical, "Fehler"
    End If
End If
```

Wichtig hierbei ist der Fakt, das das aktuelle Profile (*ActiveProfile*-Eigenschaft) nicht gelöscht werden kann. Wenn Sie dies versuchen, löst AutoCAD einen Fehler aus. Diese Eigenschaft können wir aber auch in den Code einbauen und müssen nicht erst kontrollieren, ob das Gewählte dem aktuellen Profil entspricht. In der Fehlermeldung, die AutoCAD ausgibt, ist nämlich die Fehlerbeschreibung „Kann kein Profil löschen, das benutzt wird“ enthalten. Versucht der Benutzer, das aktuelle Profil zu löschen, löst dies den Fehler aus und wir lassen einfach die Beschreibung des Fehlers (*Err.Description*) in einem Meldungsfeld anzeigen.

ResetProfile

Parameter: *ProfileName As String*

Um die Ausgangseinstellungen eines Profils wieder herzustellen, müssen Sie lediglich die Anweisung *ResetProfile* unter Angabe des Profilnamens ausführen. AutoCAD setzt daraufhin die Werte im angegebenen Profil auf die Vorgabewerte zurück.

```
If MsgBox("Soll das Profil " & ListBox1.Text & _
    " zurückgesetzt werden?", vbYesNo, "Zurücksetzen") = vbYes Then
    PrefProfiles.ResetProfile ListBox1.Text
End If
```

Profile exportieren/importieren

Um die Daten eines Profils auch anderen Benutzern zur Verfügung zu stellen, enthält das *Profiles*-Objekt zwei Funktionen, mit denen Sie die Profile exportieren und importieren können. Die Datei, in der die Profildaten gespeichert werden, ist nichts weiter als eine Registrierungsdatei, die auch über den Registrierungseditor (regedit) erstellt werden könnte. Um die Daten in die Registrierung zu übertragen, bräuchten Sie der Datei lediglich die Erweiterung „reg“ zu verpassen und diese Datei dann in Windows mit einem Doppelklick auszuführen.

ExportProfile

Parameter: *ProfileName As String, RegFile As String*

Mit dieser Anweisung können Sie das angegebene Profil in eine Registrierungsdatei exportieren. Im ersten Parameter legen Sie den Namen des Profils fest, das Sie exportieren wollen und im zweiten den Namen der Datei, in die die Profildaten geschrieben werden sollen.

Der Export des aktiven Profils könnte, unter Verwendung des Standarddialogs zur Auswahl einer Datei, folgendermaßen aussehen:

```
On Error Resume Next
ComDialog.FileName = "*.arg"
ComDialog.DialogTitle = "Profil exportieren"
ComDialog.Flags = cd10FNOoverwritePrompt
Err.Clear
ComDialog.ShowSave
If Err = 0 Then
    PrefProfiles.ExportProfile PrefProfiles.ActiveProfile, _
        ComDialog.FileName
End If
```

ImportProfile

Parameter: *ProfileName As String, RegFile As String, IncludePathInfo As Boolean*

Der Import von Profildaten läuft ähnlich ab wie der Export. Nur müssen Sie hier neben den Namen für Profil und Registrierungsdatei noch einen weiteren Parameter angeben. Mit *IncludePathInfo* können Sie festlegen, ob die Pfadinformationen, die in der Registrierungsdatei gespeichert sind, erhalten bleiben sollen (*TRUE*) oder nicht (*FALSE*).

6 Zeichnungsverwaltung in AutoCAD

Das nächste Objekt in der Hierarchie der AutoCAD-Typenbibliothek ist das Zeichnungsdokument. In diesem Dokument sind alle weiteren AutoCAD-Objekte abgelegt, egal ob sichtbar oder nicht. Im folgenden Abschnitt erfahren Sie alles über den Zugriff und den Umgang mit Zeichnungsdokumenten, die durch die Klasse *AcadDocument* dargestellt werden.

Kapitelübersicht

- 6.1 Zeichnungsauflistung (*Documents*)
- 6.2 Zugriff auf einzelne Zeichnungen
- 6.3 Mit der Zeichnung kommunizieren
- 6.4 Der Modell-/Papierbereich

6.1 Zeichnungsauflistung

Alle Zeichnungen in AutoCAD werden in einem Container, der Dokumentauflistung, zusammengestellt. Sie können also jede Zeichnung über diese Auflistung abrufen, wobei Ihnen die Klasse *AcadDocuments*, die diesem Container entspricht, einige Funktionen zur Verfügung stellt. Diese Auflistung bildet somit die oberste Instanz in der Dokumenthierarchie. Um auf die Auflistung zuzugreifen, müssen Sie das *Documents*-Objekt der *Application*-Klasse abrufen. Da das *Application*-Object in jeder Klasse abrufbar ist, ist es also kein Problem, die *Documents*-Auflistung zu extrahieren.

```
Set AllDocs = Object.Application.Documents
```

Im Normalfall werden Sie allerdings direkt auf die Dokumentauflistung zugreifen, da *Documents* global deklariert wurde und somit in jedem Modul, Formular oder Klassenmodul verfügbar ist..

```
Set AllDocs = Documents
```

Sie können jetzt mit der Variablen *AllDocs* weiterarbeiten, da diese stets den gleichen Inhalt hat wie die Dokumentauflistung selbst. Meine Empfehlung ist allerdings die direkte Verwendung von *Documents*.

Schauen wir uns nun die einzelnen Eigenschaften und Prozeduren der Dokumentauflistung an.

6.1.1 Eigenschaften der Dokumentauflistung

Application (schreibgeschützt)

Datentyp: *AcadApplication*

Mit dieser Eigenschaft können Sie das Anwendungsobjekt zurückgeben lassen. Dieses Anwendungsobjekt entspricht der AutoCAD-Sitzung, in der die Zeichnung geöffnet wurde.

Count (schreibgeschützt)

Datentyp: *Long*

Hier können Sie die Anzahl der in der AutoCAD-Sitzung geöffneten Zeichnungen ermitteln. Sie könnten diese Eigenschaft verwenden, um mithilfe einer *For...Next*-Schleife die Dokumentauflistung vom ersten bis zum letzten Dokument einzeln abzurufen. Achten müssen Sie dabei lediglich darauf, dass die Anzahl der Zeichnungen nicht dem letzten Index entspricht, da die erste Zeichnung mit dem Index 0 abgerufen wird.

```
For I = 0 To Documents.Count - 1
    Set ActDoc = Documents(i)
Next
```

Alternativ könnten Sie hierbei auch eine *For Each*-Anweisung verwenden:

```
For Each ActDoc In Documents
Next
```

Die *Count*-Eigenschaft können Sie nicht direkt beeinflussen, sie ändert sich nur, wenn Sie neue Zeichnungen erstellen, vorhandene öffnen oder Dokumente aus der Auflistung entfernen, indem Sie sie schließen.

6.1.2 Prozeduren der Dokumentauflistung

Add

Parameter: *[ByVal TemplateName As String]*

Rückgabedatentyp: *AcadDocument*

Diese Anweisung erstellt, ähnlich der *New*-Anweisung eines Dokuments, eine neue Zeichnung in AutoCAD. Die neue Zeichnung wird automatisch der Auflistung hinzugefügt. Als Parameter könnten Sie eine Zeichnungsvorlage einsetzen. Sie können diese Anweisung nicht verwenden, wenn sich AutoCAD im SDI-Modus (Single Document Interface) befindet. Nutzen Sie dann statt dieser Funktion die *New*-Anweisung eines einzelnen Dokuments.

```
Dim TempName As String
TempName = Application.Preferences.Files.TemplateDwgPath & _
    "\ansi-a.dwt"
Documents.Add TempName
```

oder

```
Set ActDoc = Documents.Add(TempName)
```

Close

Diese Anweisung schließt, im Gegensatz zur *Close*-Prozedur eines einzelnen Dokuments, alle Dokumente der AutoCAD-Sitzung. Das Schließen der Zeichnungen entfernt diese auch aus der Dokumentauflistung, wodurch Sie keinen Zugriff mehr darauf haben. Bevor die Zeichnungen geschlossen werden, speichert AutoCAD alle eventuell vorgenommenen Änderungen ohne vorherige Abfrage.

```
Documents.Close
```

Item

Parameter: *[ByVal Index]*

Rückgabedatentyp: *AcadDocument*

Wenn Sie auf ein bestimmtes Dokument, das sich in der Dokumentauflistung befindet, zugreifen möchten, dann können Sie dies mithilfe dieser Anweisung tun. Als Parameter können Sie den Index verwenden, also die genaue Position der Zeichnung innerhalb der Auflistung

```
Set ActDoc = Documents.Item(2)
```

oder den Inhalt der *Name*-Eigenschaft des abzurufenden Dokuments, wenn er Ihnen bekannt ist

```
Set ActDoc = Documents.Item("Zeichng1.dwg")
```

Open

Parameter: *[ByVal Name As String, ByVal ReadOnly]*

Rückgabedatentyp: *AcadDocument*

Mit dieser Funktion können Sie vorhandene Dokumente in AutoCAD öffnen. Dazu geben Sie im Parameter den Namen der Zeichnungsdatei an, die geöffnet werden soll. Diese Funktion kann nur angewendet werden, wenn sich AutoCAD im MDI-Modus (Multiple Document Interface) befindet. Wenden Sie diese Funktion im SDI-Modus an, wird ein Fehler ausgelöst. Verwenden Sie dann vielmehr die *Open*-Funktion einzelner Dokumente. Als Ergebnis liefert die Funktion ein Objekt vom Datentyp *AcadDocument*. Die Zeichnung wird aber gleichzeitig zum aktuellen Objekt der AutoCAD-Sitzung und ist somit sofort über das Klassenmodul *ThisDrawing* abrufbar.

```
Dim FileName As String  
FileName = Application.Path & "\Sample\Opera.dwg"  
Documents.Open FileName
```

oder

```
Set ActDoc = Documents.Open(FileName)
```

Da sich AutoCAD im MDI-Modus befindet, bietet diese Funktion zusätzlich die Möglichkeit, die Zeichnung schreibgeschützt zu öffnen, was im SDI-Modus nicht vorgesehen ist.

Zeichnung schreibgeschützt öffnen:

```
Documents.Open FileName, True
```

Zeichnung ohne Schreibschutz öffnen:

```
Documents.Open FileName, False
```

Wenn Sie den zweiten Parameter weglassen, wird als Vorgabewert *FALSE* verwendet und die Zeichnung somit immer zum Schreiben geöffnet.

Tipp

Die *Open*-Anweisung akzeptiert nur Dokumente im DWG-Format. Wollen Sie Zeichnungen öffnen, die in einem anderen Format gespeichert wurden (DXF, WMF, SAT, EPS oder BMP), müssen Sie die *Import*-Funktion der Dokumentklasse verwenden.

6.2 Auf einzelne Zeichnungen zugreifen

Einzelne Zeichnungen oder auch Dokumente, ganz wie Sie wollen, können eine Vielzahl von grafischen und nicht grafischen Objekten enthalten. Das Dokument stellt demnach die oberste Instanz der Zeichnungselemente, womit Sie folglich Zugriff auf alle Objekte erhalten, indem Sie sie über die entsprechende Zeichnung abrufen. Ein einzelnes Dokument wird in VBA durch die Klasse *AcadDocument* dargestellt. Sie können Zugriff auf eine Zeichnung erhalten, indem Sie über die Dokumentauflistungen der AutoCAD-Sitzung gehen oder die aktive Zeichnung abrufen, die in der Eigenschaft *ActiveDocument* des Anwendungsobjekts abgelegt wird.

Zugriff über die Dokumentauflistung:

```
Set ActDocument = Documents(0)
```

oder:

```
Set ActDocument = Documents("zeichng1.dwg")
```

Zugriff auf das aktuelle Dokument:

```
Set ActDocument = ActiveDocument
```

Nachdem Sie ein neues VBA-Projekt erstellt haben, ist Ihnen bestimmt aufgefallen, dass in dem Projekt bereits ein Modul deklariert wurde. Wie der Name dieses Klassenmoduls *ThisDrawing* bereits verrät, handelt es sich dabei um einen Verweis auf das aktuelle Dokument der AutoCAD-Sitzung.

6.2.1 Ereignisprozeduren der Zeichnung

Die Klasse *AcadDocument* kann auf eine ganze Reihe von Ereignissen reagieren, woraufhin die dazugehörige Prozedur ausgelöst wird.

Activate/Deactivate

Beide Ereignisse stehen in engem Zusammenhang. Das *Activate*-Ereignis wird ausgelöst, sobald ein Dokumentfenster aktiviert wird, *Deactivate*, wenn das Zeichnungsfenster als aktuelles Fenster abgelöst, also deaktiviert wird. *Activate* wird dabei vor dem *Deactivate*-Ereignis ausgeführt.

```
Private Sub AcadDocument_Activate()  
End Sub  
Private Sub AcadDocument_Deactivate()  
End Sub
```

BeginClose

Möchten Sie vor dem Schließen einer Zeichnung noch bestimmte Prozeduren ausführen, wie z.B. die Sicherungen von Programmeinstellungen, die in Zusammenhang mit der Zeichnung stehen? Für diesen Fall müssen Sie diese Ereignisprozedur verwenden. Sie wird immer dann ausgeführt, wenn eine Anforderung zum Schließen des Dokuments gestellt wurde, aber bevor die Zeichnung tatsächlich geschlossen wurde.

```
Private Sub AcadDocument_BeginClose()  
End Sub
```

BeginCommand/EndCommand

Referenzargument: *CommandName As String*

Die Prozeduren *BeginCommand* und *EndCommand* werden ausgeführt, nachdem der Benutzer einen AutoCAD-Befehl ausgeführt (*BeginCommand*) oder beendet (*EndCommand*) hat. In beiden Fällen wird der entsprechende AutoCAD-Befehl (z.B. *Line* oder *Erase*) in dem Parameter *CommandName* mitgeliefert. Diese Prozeduren würden sich also hervorragend dazu eignen, die Aufrufe von AutoCAD-Befehlen zu protokollieren.

```
Private Sub AcadDocument_BeginCommand(ByVal CommandName As String)  
End Sub  
Private Sub AcadDocument_EndCommand(ByVal CommandName As String)  
End Sub
```

BeginDoubleClick

Referenzargument: *PickPoint*

Dieses Ereignis wird ausgelöst, nachdem der Benutzer einen Doppelklick mit der Maus im Zeichnungsfenster ausgelöst hat. Da diese Prozedur ein 3D-Array mit den gewählten Koordinaten in der Variable *PickPoint* mitliefert, könnten Sie auch das Objekt identifizieren, das sich gegebenenfalls unter dem Cursor befindet.

```
Private Sub AcadDocument_BeginDoubleClick(ByVal PickPoint As _  
Variant)  
End Sub
```

BeginLisp/EndLisp/LispCancelled

Referenzargument: *FirstLine As String* (nur *BeginLisp*)

Für diejenigen, die gezielt mit Lisp arbeiten, könnten diese drei Ereignisprozeduren interessant sein. Diese reagieren nämlich direkt auf die Anwendung von Lisp-Routinen. Aus dem Namen der jeweiligen Prozedur ergibt sich ja bereits das Ereignis, bei dem sie ausgeführt werden. Die *BeginLisp*-Prozedur übergibt zusätzlich den Namen des Lisp-Ausdrucks in der Variablen *FirstLine*. Dabei gilt es Folgendes zu beachten:

- Handelt es sich um eine benutzerdefinierte Lisp-Funktion, die direkt in der Befehlszeile aufgerufen werden soll (z.B. C:TEST), und wird dann in der Befehlszeile TEST eingegeben, so enthält die Variable *FirstLine* die Zeichenfolge „(C:TEST)“.
- Wurde in der Befehlszeile eine benutzerdefinierte Lisp-Funktion eingegeben (z.B. (test)), die nicht als direkter AutoCAD-Befehl auszuführen ist, dann enthält die Variable *FirstLine* die Zeichenfolge „(test)“ und zwar in der gleichen Schreibweise (Groß- und Kleinbuchstabe), in der sie in der Befehlszeile eingegeben wurde, nicht die der Definition.

Die beiden anderen Ereignisprozeduren enthalten hingegen keine weiteren Übergabevariablen. Das *EndLisp*-Ereignis wird immer dann ausgeführt, sobald eine Lisp-Funktion beendet wird, egal ob dies regulär geschieht oder die Funktion abgebrochen wurde. Wurde eine Lisp-Funktion abgebrochen, so wird vor der *EndLisp*-Prozedur zusätzlich das *LispCancelled*-Ereignis ausgelöst.

```
Private Sub AcadDocument_BeginLisp(ByVal FirstLine As String)
End Sub

Private Sub AcadDocument_EndLisp()
End Sub

Private Sub AcadDocument_LispCancelled()
End Sub
```

BeginPlot/EndPlot

Referenzargument: *DrawingName As String*

Sobald AutoCAD eine Anfrage zum Plotten einer Zeichnung erhält, wird das *BeginPlot*-Ereignis aufgerufen. Geben Sie z.B. den Plot-Befehl in der Eingabeaufforderung ein, so startet die Prozedur, noch bevor das Plot-Dialogfenster gestartet wird. Nachdem der Code von *BeginPlot* abgearbeitet wurde, wird der eigentliche Plot-Befehl fortgesetzt.

```
Private Sub AcadDocument_BeginPlot(ByVal DrawingName As String)
End Sub
```

Nachdem der Plot-Befehl beendet wurde, egal ob geplottet wurde oder nicht, löst AutoCAD das Ereignis *EndPlot* aus. Beide Ereignisprozeduren erhalten als zusätzliche Informationen den Namen der Zeichnung, die die Plot-Anforderung gestellt hat.

```
Private Sub AcadDocument_EndPlot(ByVal DrawingName As String)
End Sub
```

BeginRightClick

Referenzargument: *PickPoint*

Dieses Ereignis wird ausgelöst, wenn der Benutzer die rechte Maustaste im Zeichnungsfenster gedrückt hat, noch bevor ein Kontextmenü aufgerufen wird. Dabei ist es egal, ob ein Befehl aktiv ist oder nicht. Sie könnten so z.B. feststellen, wann der Benutzer einen möglichen Abbruch durch Benutzung der rechten Maustaste vornehmen oder ein Shortcut-Menü aufrufen möchte. Auch diese Prozedur liefert, genau wie das *BeginDoubleClick*-Ereignis, die Koordinaten des Cursors während des Klick-Vorgangs. Diese Koordinaten werden als 3D-Array an die Variable *PickPoint* übergeben, die Sie dann bei Bedarf auswerten können.

```
Private Sub AcadDocument_BeginRightClick(ByVal PickPoint As _
Variant)
End Sub
```

BeginSave/EndSave

Referenzargument: *FileName As String*

Nachdem eine Anforderung zum Speichern eines Dokuments an AutoCAD gestellt wurde, jedoch noch bevor der Sicherungsvorgang beginnt, wird dieses Ereignis ausgelöst. Die Speichieranforderung kann dabei durch den Benutzer selbst oder durch ein Programm gestellt worden sein. Dem *BeginSave*-Ereignis folgt dann immer *EndSave* nach Abschluss des Speichervorgangs. Beide Prozeduren übergeben den Namen (inklusive Verzeichnis), unter dem die Zeichnung gespeichert werden soll bzw. dann gespeichert wurde.

```
Private Sub AcadDocument_BeginSave(ByVal FileName As String)
End Sub

Private Sub AcadDocument_EndSave(ByVal FileName As String)
End Sub
```

BeginShortcutMenu... /EndShortcutMenu

Referenzargument: *ShortcutMenu As AcadPopupMenu (EndShortcutMenu)*

Die folgenden Ereignisprozeduren werden immer dann ausgelöst, nachdem der Benutzer mit der rechten Maustaste auf das Zeichnungsfenster geklickt hat und bevor das jeweilige Kontextmenü angezeigt wird. Das *EndShortcutMenu*-Ereignis kommt dann nach Beendigung des jeweiligen Menüs zum Einsatz, egal um welches es sich handelt. Jede Prozedur liefert in der Variablen *ShortcutMenu* das jeweilige *Menu*-Objekt (Klasse *AcadPopupMenu*) das aufgerufen wird bzw. aufgerufen wurde.

BeginShortcutMenuCommand – wird ausgelöst, bevor das Kontextmenü im Befehlsmodus angezeigt wird.

Parameter: *ShortcutMenu As AcadPopupMenu, CommandString As String*

BeginShortcutMenuDefault – wird ausgelöst, bevor das Kontextmenü im Vorgabemodus angezeigt wird.

Parameter: *ShortcutMenu As AcadPopupMenu*

BeginShortcutMenuEdit – wird ausgelöst, bevor das Kontextmenü im Bearbeitungsmodus angezeigt wird.

Parameter: *ShortcutMenu As AcadPopupMenu, SelectionSet As AcadSelectionSet*

BeginShortcutMenuGrip – wird ausgelöst, bevor das Griffe-Kontextmenü angezeigt wird.

Parameter: *ShortcutMenu As AcadPopupMenu*

BeginShortcutMenuOsnap – wird ausgelöst, bevor das Objektfang-Kontextmenü angezeigt wird.

Parameter: *ShortcutMenu As AcadPopupMenu*

Die Prozeduren *BeginShortcutMenuCommand* und *BeginShortcutMenuGrip* liefern außer dem *Menu*-Objekt noch zusätzliche Informationen in Form von Variablen. Im Befehlsmodus wird eine Zeichenfolge in der Variablen *Command* gespeichert, die den AutoCAD-Befehl enthält, zu dem das spezielle Kontextmenü gehört. Im Griffe-Modus hingegen liefert die Variable *SelectionSet* den Auswahlset, der gerade bearbeitet werden soll.

```
Private Sub AcadDocument_BeginShortcutMenuCommand(ShortcutMenu _
As IAcadPopupMenu, ByVal Command As String)
End Sub

Private Sub AcadDocument_BeginShortcutMenuDefault(ShortcutMenu _
As IAcadPopupMenu)
End Sub

Private Sub AcadDocument_BeginShortcutMenuEdit(ShortcutMenu _
As IAcadPopupMenu, SelectionSet As IAcadSelectionSet)
End Sub

Private Sub AcadDocument_BeginShortcutMenuGrip(ShortcutMenu _
As IAcadPopupMenu)
End Sub

Private Sub AcadDocument_BeginShortcutMenuOsnap(ShortcutMenu _
As IAcadPopupMenu)
End Sub
```

LayoutSwitched

Referenzargument: *LayoutName As String*

Mit diesem Ereignis können Sie auf den Wechsel eines Layouts reagieren. Es wird ausgelöst, egal ob der Benutzer den Wechsel vollzogen oder eine Anwendung die Anweisung dazu gegeben hat. Die Information, in welches Layout innerhalb der Zeichnung gewechselt wurde, liefert dabei die Variable *Layout*, die durch die Prozedur mitgeliefert wird.

```
Private Sub AcadDocument_LayoutSwitched(ByVal LayoutName As _
String)
End Sub
```

Siehe auch *Kapitel 7.6*

ObjectAdded/ObjectErased/ObjectModified

Diese drei Ereignisse reagieren auf Änderungen von Zeichnungsobjekten, egal ob es sich dabei um ein sichtbares, wie z.B. Linien oder Polylinien, oder um ein nicht grafisches Objekt, wie Layer oder Linientypen, handelt. Jedes Objekt das Inhalt der Zeichnung ist, kann diese Ereignisse auslösen. Der Name der Prozeduren selbst verrät uns dabei schon das Ereignis, bei dem sie ausgelöst werden.

ObjectAdded – wird ausgelöst, nachdem ein Objekt der Zeichnung hinzugefügt worden ist. Das Objekt, um das es sich handelt, finden Sie in der Variablen *Object*, die an die Prozedur übergeben wird, wieder.

Referenzargument: *Object As Object*


```
Private Sub AcadDocument_ObjectAdded(ByVal Object As Object)
End Sub
```

ObjectErased – wird ausgelöst, nachdem ein Objekt aus der Zeichnung, die das Ereignis ausgelöst hat, gelöscht oder wiederhergestellt (*Zurück*-Funktion) wurde. Da nach dem Löschvorgang das Objekt ja nicht mehr vorhanden ist, liefert diese Prozedur in der Variablen *ObjectID* nicht das Objekt selbst, sondern nur die ID des Objekts, also eine Ganzzahl, die das Element eindeutig identifiziert hat. Ist das Objekt jedoch gelöscht worden, so können Sie nicht die *ObjectID* benutzen, um es abzurufen. Sie müssen dann einen anderen Weg finden, um die ID zur Identifizierung zu benutzen.

Referenzargument: *ObjectID As Long*

```
Private Sub AcadDocument_ObjectErased(ByVal ObjectID As Long)
End Sub
```

ObjectModified – Wie der Name der Prozedur schon verrät, wird dieses Ereignis ausgelöst, nachdem ein Objekt modifiziert, also geändert wurde. Solch eine Modifizierung kann z.B. die Änderung des Layers oder die Verschiebung des Objekts sein. Auch diese Prozedur liefert das geänderte Objekt in einer Variablen.

Referenzargument: *Object As Object*

```
Private Sub AcadDocument_ObjectModified(ByVal Object As Object)
End Sub
```

Diese Ereignisprozeduren stehen auch in Zusammenhang mit dem Event *Modified*, das jede grafische Objektklasse enthält (siehe *Kapitel 7.1.3*).

SelectionChanged

Das Auslösen dieses Ereignisses zeigt uns an, dass eine Änderung am *Pickfirst*-Auswahlsatz, also der Auswahl, die erstellt werden kann, ohne dass ein Befehl ausgeführt wurde, vorgenommen wurde. Das bedeutet, es wurde ein Objekt hinzugefügt oder aus dem Auswahlsatz entfernt. Es wird allerdings nicht ausgelöst, wenn Sie die Eigenschaften eines im Auswahlsatz befindlichen Objekts ändern. Dann greift das Ereignis *ObjectModified*.

```
Private Sub AcadDocument_SelectionChanged()
End Sub
```

Siehe auch *Kapitel 8.2*

WindowChanged

Referenzargument: *WindowState As AcWindowState*

Diese Ereignisprozedur ist hilfreich für Anwendungen, die auf die Änderung des Status des Zeichnungsfensters reagieren müssen. Die Variable *WindowState* liefert dabei den aktuellen Status eben dieses Fensters. Sie kann eine der folgenden Konstanten enthalten:

acMin – Das Fenster wurde minimiert.

acMax – Das Fenster wurde maximiert.

acNorm – Das Fenster wird normal dargestellt, also weder minimiert noch maximiert.

```
Private Sub AcadDocument_WindowChanged(ByVal WindowState As _
AcWindowState)
End Sub
```

WindowMovedOrResized

Referenzargument: *HWNDFrame As Long, bMoved As Boolean*

Dieses Ereignis wird ausgelöst, wenn das Zeichnungsfenster im Status geändert oder verschoben wurde. Es fällt also in den gleichen Anwendungsbereich wie die vorherige Prozedur. Der Unterschied ist, dass diese nicht den Status, sondern die Referenz (Handle) des Fensters und die damit verbundene Aktion als Variablen mitliefert. Die erste Variable (*HWNDFrame*) liefert den Handle des Fensters, den es vom Windows-Betriebssystem bekommen hat und der es eindeutig im System identifiziert. Die zweite Variable (*Moved*) kann entweder den Wert *TRUE* enthalten, was bedeuten würde, dass das Fenster verschoben wurde, oder *FALSE*, was Ihnen mitteilt, dass der Status geändert wurde.

```
Private Sub AcadDocument_WindowMovedOrResized(ByVal HWNDFrame As _
Long, ByVal bMoved As Boolean)
End Sub
```

6.2.2 Eigenschaften der Zeichnung

Mit den nun folgenden Eigenschaften nehmen Sie Einfluss auf die verschiedensten Objekte, die in einer Zeichnung enthalten sind. Sie erhalten z.B. Zugriff auf den Modellbereich und somit auf die einzelnen darin enthaltenen Elemente. Die Dokumentklasse dient ebenfalls als Container für alle Layer, Linientypen, Textstile usw.

Active

(*schreibgeschützt*)

Datentyp: *Boolean*

Mithilfe dieser Eigenschaft können Sie kontrollieren, ob es sich bei dem ausgewählten Dokument um das aktive Dokument der AutoCAD-Sitzung handelt oder nicht. Sie macht also wenig Sinn in Zusammenhang mit der Objektvariablen *ThisDrawing*, die ja sowieso dem aktiven Dokument entspricht. Vielmehr könnte die Eigenschaft bei der Arbeit mit der Dokumentauflistung zum Einsatz kommen.

Folgende Werte kann diese Eigenschaft zurückgeben:

TRUE – Das Dokument ist die aktive Zeichnung.

FALSE – Das Dokument ist nicht die aktive Zeichnung.

Aktuelle Zeichnungsobjekte abrufen

Die nun folgenden Eigenschaften enthalten alle einen Verweis auf bestimmte Zeichnungsobjekte. Diese Objekte stellen jeweils das aktive Objekt einer Auflistung dar. Wenn Sie z.B. das aktuelle Layer-Objekt ermitteln wollen, dann brauchen Sie nur auf die Dokumenteigenschaft *ActiveLayer* zugreifen. Diese Eigenschaft liefert nicht nur den Namen des Layers, sondern einen Verweis auf das komplette Layer-Objekt, wie es in der Layer-Auflistung enthalten ist. Sie könnten somit direkt auf die Layer-Eigenschaften zugreifen und beispielsweise die Farbe anpassen.

ActiveDimStyle

Datentyp: *AcadDimStyle*

Hiermit können Sie den aktuellen Bemaßungsstil der Zeichnung abrufen oder einen neuen setzen. Achten Sie darauf, dass beim Auslesen des aktiven Stils das Schlüsselwort *Set* verwendet werden muss, beim Setzen allerdings nicht.

Auslesen in eine Variable:

```
Set ActDimStyle = ThisDrawing.ActiveDimStyle
```

Setzen eines neuen Bemaßungsstils

```
ThisDrawing.ActiveDimStyle = ActDimStyle
```

Siehe auch *Kapitel 7.13*

ActiveLayer

Datentyp: *AcadLayer*

Diese Eigenschaft enthält das aktuelle Layer-Objekt der entsprechenden Zeichnung. Sie können dieses Objekt abrufen oder einen neuen Layer festlegen, indem Sie einen Layer aus der Layer-Auflistung an diese Eigenschaft übergeben. Auch hier gilt: Layer auslesen, Schlüsselwort *Set* nicht vergessen.

Auslesen des Layers in eine Variable:

```
Set ActLayer = ThisDrawing.ActiveLayer
```

Setzen eines neuen Layers:

```
ThisDrawing.ActiveLayer = ActLayer
```

Siehe auch *Kapitel 7.4*

ActiveLayout

Datentyp: *AcadLayout*

Über diese Eigenschaft erhalten Sie Zugriff auf das aktuelle Layout der Zeichnung oder Sie können ein neues einstellen, indem Sie ein Layout an diese Eigenschaft übergeben.

Übergabe des aktuellen Layouts an eine Variable:

```
Set ActLayout = ThisDrawing.ActiveLayout
```

Setzen eines neuen Layouts:

```
ThisDrawing.ActiveLayout = ActLayout
```

Siehe auch *Kapitel 7.6*

ActiveLinetype

Datentyp: *AcadLinetype*

Hier ist der aktuelle Linientyp als Objekt gespeichert. Sie können den aktuellen Linientyp der Zeichnung ändern, indem Sie dieser Eigenschaft ein Linientypobjekt aus der Linientypauflistung übergeben.

Übergabe des aktuellen Linientyps an eine Variable:

```
Set ActLineType = ThisDrawing.ActiveLinetype
```

Setzen eines neuen Linientyps

```
ThisDrawing.ActiveLinetype = ActLineType
```

Siehe auch *Kapitel 7.5*

ActivePViewport

Datentyp: *AcadPViewport*

Um zu ermitteln, welches Papierbereichsansichtsfenster das aktuelle ist, dient diese Eigenschaft. Voraussetzung dafür ist allerdings, dass sich die Zeichnung im Papierbereich befindet.

Übergabe des aktuellen Ansichtsfensters an eine Variable:

```
Set ActPAF = ThisDrawing.ActivePViewport
```

Setzen eines neuen Ansichtsfensters

```
ThisDrawing.ActivePViewport = ActPAF
```

Siehe auch *Kapitel 7.12.2*

ActiveSelectionSet

Datentyp: *AcadSelectionSet*

Da AutoCAD mehr als einen Auswahlsatz in einer Zeichnung verwalten kann, gibt es auch hier einen aktuellen Satz. Mit dieser Eigenschaft können Sie diesen abrufen oder gegebenenfalls einen neuen setzen. Hier kommen allerdings nur die definierten, nicht aber der *PickFirst*-Auswahlsatz zum Einsatz. Für diesen gibt es eine gesonderte Möglichkeit des Abrufens.

Übergabe des aktuellen Auswahlsatzes an eine Variable:

```
Set ActSSet = ThisDrawing.ActiveSelectionSet
```

Setzen eines neuen Auswahlsatzes:

```
ThisDrawing.ActiveSelectionSet = ActSSet
```

Siehe auch *Kapitel 8.2*

ActiveTextStyle

Datentyp: *AcadTextStyle*

Diese Eigenschaft speichert den aktuellen Textstil der Zeichnung. Sie können den aktuellen Textstil der Zeichnung ändern, indem Sie dieser Eigenschaft ein Textstil-Objekt aus der Auflistung aller Textstile übergeben.

Übergabe des aktuellen Textstils an eine Variable:

```
Set ActTStyle = ThisDrawing.ActiveTextStyle
```

Setzen eines neuen Textstils:

```
ThisDrawing.ActiveTextStyle = ActTStyle
```

Siehe auch *Kapitel 7.7*

ActiveUCS

Datentyp: *AcadUCS*

Mit dieser Eigenschaft können Sie das aktuelle Benutzerkoordinatensystem abrufen oder neu festlegen. Wenn Sie Änderungen am aktuellen BKS vorgenommen haben, so müssen Sie diese Eigenschaft aktualisieren, damit die Änderungen wirksam werden. Weisen Sie dazu einfach das geänderte BKS der Eigenschaft zu, auch wenn es sich bereits um das aktuelle BKS handelt.

Aktuelles BKS auslesen:

```
Set ActUCS = ThisDrawing.ActiveUCS
```

Änderungen am Benutzerkoordinatensystem vornehmen und das aktive BKS aktualisieren:

```
ThisDrawing.ActiveUCS = ActUCS
```

Siehe auch *Kapitel 7.8*

ActiveViewport

Datentyp: *AcadViewport*

Im Gegensatz zu den Ansichtsfenstern innerhalb des Papierbereichs, handelt es sich hier um die Ansichtsfenster, die im Modellbereich verwendet werden. Diese Eigenschaft speichert dabei das gerade aktuelle Ansichtsfenster.

Übergabe des aktuellen Ansichtsfensters an eine Variable:

```
Set ActAF = ThisDrawing.ActiveViewport
```

Setzen eines neuen Ansichtsfensters:

```
ThisDrawing.ActiveViewport = ActAF
```

Siehe auch *Kapitel 7.12*

Auflistungsobjekte

Über die folgenden Eigenschaften erhalten Sie Zugriff auf die Definitionsdaten

Blocks

(schreibgeschützt)

Datentyp: *AcadBlocks*

Diese Eigenschaft gibt die Auflistung aller in der entsprechenden Zeichnung vorhandenen Blockdefinitionen zurück. Da die Blockauflistung auch den Modell- und den Papierbereich enthält, ist diese Eigenschaft immer initialisiert, auch wenn die Zeichnung sonst keine Blockdefinitionen enthält.

```
Set AllBlocks = ThisDrawing.Blocks
```

Siehe auch *Kapitel 7.10*

Dictionaries
(schreibgeschützt)

Datentyp: *AcadDictionaries*

Ruft die Gruppierung der Erweiterungswörterbücher der Zeichnung ab.

Set `AllDict = ThisDrawing.Dictionaries`

Siehe auch *Kapitel 7.14*

DimStyles
(schreibgeschützt)

Datentyp: *AcadDimStyles*

Diese Eigenschaft enthält die Auflistung aller Bemaßungsstile der jeweiligen Zeichnung.

Set `AllDStyles = ThisDrawing.DimStyles`

Siehe auch *Kapitel 7.13*

Groups
(schreibgeschützt)

Datentyp: *AcadGroups*

Über diese Eigenschaft erhalten Sie Zugriff auf die Auflistung aller Gruppendefinitionen der Zeichnung. Dieses Objekt ist immer initialisiert, auch wenn keine Gruppen definiert wurden. In diesem Fall gibt die Eigenschaft *Count* der Auflistung den Wert 0 zurück.

Set `AllGroups = ThisDrawing.Groups`

Siehe auch *Kapitel 7.2*

Layers
(schreibgeschützt)

Datentyp: *AcadLayers*

Diese Eigenschaft gibt die Auflistung aller in der entsprechenden Zeichnung vorhandenen Layer zurück.

Set `AllLayers = ThisDrawing.Layers`

Siehe auch *Kapitel 7.4*

Layouts
(schreibgeschützt)

Datentyp: *AcadLayouts*

Diese Eigenschaft gibt die Auflistung aller in der entsprechenden Zeichnung vorhandenen Layouts zurück. Die Auflistung enthält sowohl das Modelllayout als auch alle Layouts des Papierbereichs.

Set `AllLayouts = ThisDrawing.Layouts`

Siehe auch *Kapitel 7.6*

Linetypes***(schreibgeschützt)***Datentyp: *AcadLinetypes*

Diese Eigenschaft gibt die Auflistung aller in der entsprechenden Zeichnung vorhandenen Linientypen zurück.

```
Set AllLinetypes = ThisDrawing.Linetypes
```

Siehe auch *Kapitel 7.5*

PlotConfigurations***(schreibgeschützt)***Datentyp: *AcadPlotConfigurations*

Dieses Objekt enthält die Auflistung aller Plot-Einstellungen des Dokuments.

```
Set ActPlotCfgs = ThisDrawing.PlotConfigurations
```

Siehe auch *Kapitel 9.1*

RegisteredApplications***(schreibgeschützt)***Datentyp: *AcadRegisteredApplications*

Wenn Sie Zugriff auf die in der Zeichnung registrierten Anwendungsnamen haben wollen, dann müssen Sie das Objekt dieser Eigenschaft abrufen. Es gibt eine Auflistung dieser Anwendungsnamen zurück.

```
Set ActRegApps = ThisDrawing.RegisteredApplications
```

Siehe auch *Kapitel 7.9*

SelectionSets***(schreibgeschützt)***Datentyp: *AcadSelectionSets*

Mithilfe dieser Eigenschaft können Sie auf die Auflistung der Auswahlsätze zugreifen, die der Zeichnung hinzugefügt wurden. Außer des *PickfirstSelectionSet* sind in dieser Auflistung alle Auswahlsätze enthalten.

```
Set AllSSet = ThisDrawing.SelectionSets
```

Siehe auch *Kapitel 8.2*

TextStyles***(schreibgeschützt)***Datentyp: *AcadTextStyles*

Diese Eigenschaft gibt die Auflistung aller in der entsprechenden Zeichnung definierten Textstile zurück.

```
Set AllTStyles = ThisDrawing.TextStyles
```

Siehe auch *Kapitel 7.7*

UserCoordinateSystems**(schreibgeschützt)**Datentyp: *AcadUCSs*

Die Auflistung, die alle Benutzerkoordinatensysteme enthält, kann über diese Eigenschaft abgerufen werden.

```
Set AllUCSs = ThisDrawing.UserCoordinateSystems
```

Siehe auch *Kapitel 7.8*

Viewports**(schreibgeschützt)**Datentyp: *AcadViewports*

Diese Eigenschaft enthält die Auflistung sämtlicher Ansichtsfenster der aufrufenden AutoCAD-Zeichnung.

```
Set AllVPs = ThisDrawing.Viewports
```

Siehe auch *Kapitel 7.12*

Views**(schreibgeschützt)**Datentyp: *AcadViews*

Das letzte Eigenschaftsobjekt der Dokumentklasse enthält alle benannten Ausschnitte, die in der AutoCAD-Zeichnung definiert wurden.

```
Set AllViews = ThisDrawing.Views
```

Siehe auch *Kapitel 7.11*

Weitere Dokumenteigenschaften**Application****(schreibgeschützt)**Datentyp: *AcadApplication*

Mit dieser Eigenschaft können Sie das Anwendungsobjekt zurückgeben lassen. Dieses Anwendungsobjekt entspricht der AutoCAD-Sitzung, in dem die Zeichnung geöffnet wurde.

```
Set AcadApp = ThisDrawing.Application
```

Siehe auch *Kapitel 5*

ActiveSpaceDatentyp: *acActiveSpace*

Diese Eigenschaft lässt nur zwei mögliche Einstellungen zu. Entweder Sie befinden sich im Papier- oder im Modellbereich. Wenn Sie dieser Eigenschaft eine der beide Konstanten zuweisen, wechselt die Zeichnung in den entsprechenden Bereich oder Sie lesen den aktuellen Wert aus, um festzustellen, in welchem Bereich sich die Zeichnung gerade befindet.

acPaperSpace – Papierbereich

acModelSpace – Modellbereich

`ThisDrawing.ActiveSpace = acPaperSpace`

Database

(schreibgeschützt)

Datentyp: *AcadDatabase*

Über diese Eigenschaft rufen Sie die Datenbank der Zeichnung ab. Das Datenbankobjekt ist sozusagen eine Zusammenfassung der Zeichnungseigenschaften. Sie enthält sämtliche Auflistungen (*Blocks*, *Layers*, *Linetypes* ...), die in der Zeichnung enthalten sind. Dieses Objekt wird in Zusammenhang mit der Klasse *AcadDictionary* verwendet. Ansonsten können Sie auch über die Dokumentklasse auf die einzelnen Objekte zugreifen.

`Set ActDatabase = ThisDrawing.Database`

ElevationModelSpace/ElevationPaperSpace

Datentyp: *Double*

Mit diesen beiden Eigenschaften können Sie die Erhebungen im Modell- und im Papierbereich festlegen oder die aktuellen Einstellungen abrufen. Die Erhebung entspricht der Z-Koordinate und wird immer dann verwendet, wenn der Benutzer bei der Auswahl eines 3D-Punkts nur die X- und Y-Koordinate angegeben hat.

FullName/Name/Path

(schreibgeschützt)

Datentyp: *String*

Die Eigenschaft *FullName* gibt den kompletten Zeichnungsamen inklusive Pfad, *Name* den Zeichnungsamen und *Path* das Verzeichnis, in dem die Zeichnung gespeichert ist, als Zeichenfolge zurück. Wurde die Zeichnung noch nicht gespeichert, so enthält die Eigenschaft *FullName* eine leere Zeichenfolge und Sie müssen die Eigenschaften *Path* und *Name* kombinieren, um einen kompletten Zeichnungsamen zu erhalten.

Folgende Funktion gibt den vollständigen Zeichnungsamen zurück, egal ob die Zeichnung gespeichert wurde oder nicht. Außerdem wird die Dateiendung (*dwg*, *dxf*, *dwt*) entfernt.

```
Public Function GetDocName() As String
On Local Error Resume Next
GetDocName = Application.ActiveDocument.FullName
If GetDocName = "" Then
    GetDocName = Application.Path + "\" + _
        Application.ActiveDocument.Name
End If
GetDocName = Left$(GetDocName, Len(GetDocName) - 4)
End Function
```

Height/Width

Datentyp: *Long*

Diese beiden Eigenschaften bestimmen die Breite und Höhe des jeweiligen Zeichnungsfensters. Wenn Sie für das Fenster die Höhe und/oder die Breite festlegen und dieses sich gerade im maximierten oder minimierten Zustand befindet, so wird der Status des Fensters in jedem Fall auf normal gesetzt, was dann das Ereignis *WindowChanged* auslöst.

HWND

Datentyp: *Long*

Bei dieser Eigenschaft handelt es sich um das Handle oder besser gesagt die Referenz des Zeichnungsfensters, in dem das entsprechende Dokument angezeigt wird. Mithilfe dieser Referenz wird das Fenster eindeutig im Windows-Betriebssystem identifiziert.

Limits

Datentyp: *Variant* (Array → *Double*)

In dieser Eigenschaft sind die Zeichnungsgrenzen des Dokuments gespeichert. Es handelt sich dabei um die linke untere und rechte obere Ecke. Beide Koordinaten werden in einem Array zusammengefasst, das folgendermaßen aufgebaut ist.

- LimPoints(0) = X-Koordinate linke untere Begrenzung
- LimPoints(1) = Y-Koordinate linke untere Begrenzung
- LimPoints(2) = X-Koordinate rechte obere Begrenzung
- LimPoints(3) = Y-Koordinate rechte obere Begrenzung

Die Koordinaten sind ebenfalls in den Systemvariablen *LIMMIN* und *LIMMAX* zu finden.

Zeichnungsgrenzen abrufen:

```
Dim LimPoints As Variant  
LimPoints = ThisDrawing.Limits
```

Zeichnungsgrenzen setzen:

```
Dim LimPoints(3) As Double  
LimPoints(0) = 20: LimPoints(1) = 20  
LimPoints(2) = 40: LimPoints(3) = 40  
ThisDrawing.Limits = LimPoints
```

ModelSpace

(schreibgeschützt)

Datentyp: *AcadModelSpace*

Über diese Eigenschaft erhalten Sie Zugriff auf die Daten des Modellbereichs der Zeichnung. Der Modellbereich entspricht vom Prinzip her einem definierten Block, weshalb er auch in der Blockauflistung geführt wird.

```
Set MoSpace = ThisDrawing.ModelSpace
```

Siehe auch *Kapitel 6.4*

MSpace

Datentyp: *Boolean*

Diese Eigenschaft bestimmt, ob in der Zeichnung im Modell- oder Papierbereich gearbeitet werden soll. Es ist im Prinzip so, als würden Sie den Schalter *Modell/Papier* in der Statuszeile von AutoCAD benutzen. Folglich ist diese Eigenschaft besonders interessant für die Arbeit mit Ansichtsfenstern im Papierbereich. Ist diese Eigenschaft auf den Wert *TRUE* gesetzt, so wird im Modellbereich gearbeitet, andernfalls im Papierbereich.

ObjectSnapModeDatentyp: *Boolean*

Hiermit können Sie festlegen, ob der Objektfangmodus aktiviert oder deaktiviert werden soll. Ist die Eigenschaft auf *TRUE* gesetzt, ist der Objektfang aktiviert, andernfalls nicht.

PaperSpace***(schreibgeschützt)***Datentyp: *AcadPaperSpace*

Über diese Eigenschaft erhalten Sie Zugriff auf die Daten des Papierbereichs der Zeichnung. Der Papierbereich entspricht, genau wie der Modellbereich, einer Blockdefinition, weshalb er auch in der Blockauflistung geführt wird.

```
Set PaSpace = ThisDrawing.PaperSpace
```

Siehe auch *Kapitel 6.4****PickfirstSelectionSet******(schreibgeschützt)***Datentyp: *AcadSelectionSet*

Bei dem *PickfirstSelectionSet* handelt es sich um den Auswahlsatz, der die Objekte enthält, die gewählt wurden, ohne dass ein AutoCAD-Befehl ausgeführt worden ist. Bei diesem Auswahlsatz sind dann die Griffe der einzelnen gewählten Objekte sichtbar. Mithilfe dieser Eigenschaft erhalten Sie Zugriff auf diesen speziellen Auswahlsatz.

```
Set ActSSet = ThisDrawing.PickfirstSelectionSet
```

Siehe auch *Kapitel 8.2****Plot******(schreibgeschützt)***Datentyp: *AcadPlot*

Diese Eigenschaft gibt das Plot-Objekt der Zeichnung zurück, das Eigenschaften und Methoden zum Ausplotten von Layouts enthält.

```
Set ActPlot = ThisDrawing.Plot
```

Siehe auch *Kapitel 9****Preferences******(schreibgeschützt)***Datentyp: *AcadDatabasePreferences*

Über dieses Objekt können Sie eine Reihe von Einstellungen für die Zeichnung abrufen bzw. vornehmen. Alle diese Eigenschaften finden Sie auch im Optionen-Dialog von AutoCAD. Der Unterschied zum *Preferences*-Objekt besteht lediglich darin, dass diese Einstellungen in der Zeichnung und nicht in der Registrierungsdatenbank gespeichert werden, also nur für die entsprechende Zeichnung gelten.

```
Set ActDrawPrefs = ThisDrawing.Preferences
```

Eigenschaft	Beschreibung
<i>AllowLongSymbolNames</i> (Boolean)	Legt fest, ob Symbolnamen erweiterte Zeichensätze oder mehr als 31 Zeichen enthalten dürfen. Wenn für diese Eigenschaft der Wert <i>TRUE</i> festgelegt wurde, können für Symbolnamen bis zu 255 Zeichen verwendet werden, andernfalls nur 31. Der Vorgabewert der Eigenschaft ist <i>TRUE</i> <code>ActDrawPrefs.AllowLongSymbolNames = True</code>
<i>ContourLinesPerSurface</i> (Long)	Legt die Anzahl der Konturlinien (Isolinien) pro Oberfläche auf Objekten fest. Sie können für diese Eigenschaft einen Wertebereich zwischen 0 und 2047 verwenden. Je höher Sie den Wert festlegen, umso feiner wird die Kontur. <code>ActDrawPrefs.ContourLinesPerSurface = 30</code> Systemvariable <i>ISOLINES</i>
<i>DisplaySilhouette</i> (Boolean)	Legt fest, ob Umrisskurven von Volumenkörperobjekten im Drahtkörpermodus angezeigt werden (<i>TRUE</i>) oder nicht (<i>FALSE</i>). <code>ActDrawPrefs.DisplaySilhouette = True</code> Systemvariable <i>DISPSILH</i>
<i>Lineweight</i> (AcLineWeight)	Über diese Eigenschaft können Sie die Standardlinienstärke für die Zeichnung abrufen bzw. festlegen. <code>ActDrawPrefs.Lineweight = acLnWt005</code> Systemvariable <i>LWDEFAULT</i>
<i>LineWeightDisplay</i> (Boolean)	Hiermit können Sie festlegen, ob die Linienstärken im Modellbereich für die Zeichnung angezeigt werden (<i>TRUE</i>) oder nicht (<i>FALSE</i>). <code>ActDrawPrefs.LineWeightDisplay = True</code>
<i>MaxActiveViewports</i> (Long)	Diese Eigenschaft legt die maximale Anzahl aktiver Ansichtsfenster fest. Sie können für diese Eigenschaft eine Zahl zwischen 2 und 48 verwenden. <code>ActDrawPrefs.MaxActiveViewports = 40</code> Systemvariable <i>MAXACTVP</i>
<i>ObjectSortByPlotting</i> (Boolean)	Aktiviert (<i>TRUE</i>) oder deaktiviert (<i>FALSE</i>) den Sortiervorgang von Zeichnungsobjekten nach einer Plot-Reihenfolge. Wenn diese Option auf <i>TRUE</i> gesetzt wird, sortiert AutoCAD die ausgewählten Objekte nach ihrer Erstellung (das zuerst erstellte Objekt an erster, das zuletzt erstellte an letzter Stelle). <code>ActDrawPrefs.ObjectSortByPlotting = True</code> Systemvariable <i>SORTENTS</i>

Tabelle 6.1: Eigenschaften des DatabasePreferences-Objekts

Eigenschaft	Beschreibung
<i>ObjectSortByPSOutput</i> (Boolean)	Aktiviert (<i>TRUE</i>) oder deaktiviert (<i>FALSE</i>) den Sortiervorgang von Zeichnungsobjekten nach einer PostScript-Reihenfolge. Wenn diese Option auf <i>TRUE</i> gesetzt wird, sortiert AutoCAD die ausgewählten Objekte nach ihrer Erstellung (das zuerst erstellte Objekt an erster, das zuletzt erstellte an letzter Stelle). <code>ActDrawPrefs.ObjectSortByPSOutput = True</code> Systemvariable <i>SORTENTS</i>
<i>ObjectSortByRedraws</i> (Boolean)	Aktiviert (<i>TRUE</i>) oder deaktiviert (<i>FALSE</i>) den Sortiervorgang von Zeichnungsobjekten nach einer Reihenfolge zum Neuzeichnen. Wenn diese Option auf <i>TRUE</i> gesetzt wird, sortiert AutoCAD die ausgewählten Objekte nach ihrer Erstellung (das zuerst erstellte Objekt an erster, das zuletzt erstellte an letzter Stelle). <code>ActDrawPrefs.ObjectSortByRedraws = True</code> Systemvariable <i>SORTENTS</i>
<i>ObjectSortByRegens</i> (Boolean)	Aktiviert (<i>TRUE</i>) oder deaktiviert (<i>FALSE</i>) den Sortiervorgang von Zeichnungsobjekten nach einer Reihenfolge zum Regenerieren. Wenn diese Option auf <i>TRUE</i> gesetzt wird, sortiert AutoCAD die ausgewählten Objekte nach ihrer Erstellung (das zuerst erstellte Objekt an erster, das zuletzt erstellte an letzter Stelle). <code>ActDrawPrefs.ObjectSortByRegens = True</code> Systemvariable <i>SORTENTS</i>
<i>ObjectSortBySelection</i> (Boolean)	Aktiviert (<i>TRUE</i>) oder deaktiviert (<i>FALSE</i>) den Sortiervorgang von Zeichnungsobjekten nach einer Objektauswahl. Wenn diese Option auf <i>TRUE</i> gesetzt wird, sortiert AutoCAD die ausgewählten Objekte nach ihrer Erstellung (das zuerst erstellte Objekt an erster, das zuletzt erstellte an letzter Stelle). <code>ActDrawPrefs.ObjectSortBySelection = True</code> Systemvariable <i>SORTENTS</i>
<i>ObjectSortBySnap</i> (Boolean)	Aktiviert (<i>TRUE</i>) oder deaktiviert (<i>FALSE</i>) den Sortiervorgang von Zeichnungsobjekten nach einer Sortierreihenfolge. Wenn diese Option auf <i>TRUE</i> gesetzt wird, sortiert AutoCAD die ausgewählten Objekte nach ihrer Erstellung (das zuerst erstellte Objekt an erster, das zuletzt erstellte an letzter Stelle). <code>ActDrawPrefs.ObjectSortBySelection = True</code> Systemvariable <i>SORTENTS</i>
<i>OLELaunch</i> (Boolean)	Legt fest, ob die Anwendung gestartet werden soll, wenn dazugehörige OLE-Objekte geplottet werden. <code>ActDrawPrefs.OLELaunch = False</code> Systemvariable <i>OLESTARTUP</i>

Tabelle 6.1: Eigenschaften des DatabasePreferences-Objekts (Forts.)

Eigenschaft	Beschreibung
<i>RenderSmoothness</i> (Double)	In dieser Eigenschaft ist der Wert für die Glättung von schattierten, gerenderten und verdeckten Objekten mit entfernten Linien festgelegt. Sie können für diese Eigenschaft einen Wert im Bereich 0.1 bis 10.0 festlegen <code>ActDrawPrefs.RenderSmoothness = 5.0</code> Systemvariable <i>FACETRES</i>
<i>SegmentPerPolyline</i> (Long)	Legt die Anzahl der Liniensegmente fest, die für jede Polylinienkurve generiert werden sollen. Je höher der Wert, desto runder wird die Kurve angezeigt. Mögliche Werte liegen im Bereich von 0 bis 42950. <code>ActDrawPrefs.SegmentPerPolyline = 8</code> Systemvariable <i>SPLINESEGS</i>
<i>SolidFill</i> (Boolean)	Legt fest, ob Multilinien, Bänder, Volumenkörper, alle Schraffuren (einschließlich Flächenfüllungen) und breite Polylinien gefüllt sind. <code>ActDrawPrefs.SolidFill = True</code> Systemvariable <i>FILLMODE</i>
<i>TextFrameDisplay</i> (Boolean)	Legt fest, ob die Anzeige von Rahmen für Textobjekte angezeigt werden soll (<i>TRUE</i>) oder der Text selbst (<i>FALSE</i>). <code>ActDrawPrefs.TextFrameDisplay = True</code> Systemvariable <i>QTEXTMODE</i>
<i>XRefEdit</i> (Boolean)	Legt fest, ob die aktuelle Zeichnung bearbeitet werden kann, während sie von einem anderen Benutzer referenziert wird. <code>ActDrawPrefs.XRefEdit = True</code> Systemvariable <i>XEDIT</i>
<i>XRefLayerVisibility</i> (Boolean)	Legt die Sichtbarkeit von XRef-abhängigen Layern fest und gibt an, ob Änderungen an verschachtelten XRef-Pfaden gespeichert werden. <code>ActDrawPrefs.XRefLayerVisibility = True</code> Systemvariable <i>VISRETAIN</i>

Tabelle 6.1: Eigenschaften des DatabasePreferences-Objekts (Forts.)

ReadOnly
(schreibgeschützt)
Datentyp: Boolean

Diese Eigenschaft zeigt lediglich an, ob die jeweilige Zeichnung schreibgeschützt geöffnet wurde (*TRUE*) oder nicht (*FALSE*).

Saved**(schreibgeschützt)**Datentyp: *Boolean*

Gibt diese Eigenschaft den Wert *TRUE* zurück, so enthält die Zeichnung noch nicht gespeicherte Änderungen. Das kann besonders dann hilfreich sein, wenn Sie Anwendungen schreiben, die mit mehreren Zeichnungen gleichzeitig arbeiten. Sie können so kontrollieren, ob die Zeichnungsänderungen gespeichert wurden, und es gegebenenfalls über entsprechende Codezeilen nachholen.

Utility**(schreibgeschützt)**Datentyp: *AcadUtility*

Bei dieser Eigenschaft handelt es sich um ein ganz besonderes Objekt. Es enthält die verschiedensten Funktionen zur Bearbeitung in AutoCAD. Mithilfe dieser Funktionen können Sie z.B. Koordinaten, Objekte oder Winkel über die AutoCAD-Befehlszeile in der Zeichnung wählen. Außerdem enthält es Funktionen zur Umrechnung von Einheiten.

```
Set AcadUtil = ThisDrawing.Utility
```

Siehe auch *Kapitel 6.3*.

WindowStateDatentyp: *AcWindowState (Long)*

Diese Eigenschaft gibt den Status, also die Darstellung des Zeichnungsfensters, wieder. Für diese Eigenschaft sind folgende Konstanten als Wert vorgesehen.

acMin – Das Zeichnungsfenster wird minimiert dargestellt.

acMax – Das Zeichnungsfenster wird maximiert dargestellt.

acNorm – Das Zeichnungsfenster wird normal dargestellt, also weder maximiert noch minimiert.

WindowTitle**(schreibgeschützt)**Datentyp: *String*

Als Letztes haben wir noch die *WindowTitle*-Eigenschaft. Diese enthält die aktuelle Eintragung des Dokumentfensters, was normalerweise der Zeichnungsname sein dürfte.

6.2.3 Prozeduren der Klasse *AcadDocument*

Außer den Eigenschaften bieten die Dokumentklassen auch noch einige Prozeduren, mit deren Hilfe Sie die verschiedensten Aufgabenstellungen abarbeiten können. All diese Prozeduren werden im folgenden Abschnitt genauer unter die Lupe genommen.

Activate

Mit dieser Anweisung können Sie das ausführende Dokument zur aktuellen Zeichnung der AutoCAD-Sitzung machen. Mit Ausführung dieser Prozedur ändert sich natürlich auch der Inhalt des Klassenmoduls *ThisDrawing*.

```
Documents(n).Activate
```

AuditInfo

Parameter: *ByVal FixErr As Boolean*

Wenn Sie diese Prozedur aufrufen, können Sie die Integrität der Zeichnung prüfen lassen. Der Aufruf dieser Prozedur hat die gleiche Wirkung, wie wenn Sie den Prüfen-Befehl in der AutoCAD-Befehlszeile aufrufen würden. Als zusätzlichen Parameter müssen Sie nur festlegen, ob die gefundenen Fehler behoben werden sollen oder nicht. Setzen Sie den Parameter auf *TRUE*, werden die Fehler behoben, andernfalls nicht. Da der Parameter nur geschrieben werden kann und keinen Wert zurückgibt, können Sie den Wert direkt verwenden, ohne erst eine Variable zu deklarieren.

```
ThisDrawing.AuditInfo True
```

Close

Parameter: *[ByVal SaveChanges], [ByVal FileName]*

Diese Anweisung schließt das entsprechende Dokument. Als Parameter können Sie zum einen festlegen, ob die Änderungen gespeichert werden sollen oder nicht und unter welchem Namen die Zeichnung abgespeichert werden soll. Wurden an der Zeichnung keine Änderungen vorgenommen, dann werden die beiden Parameter ignoriert, egal was Sie eventuell festgelegt haben.

Wenn Sie den Parameter *SaveChanges* ignorieren, so wird als Vorgabewert *TRUE* verwendet.

Legen Sie *TRUE* für den Parameter *SaveChanges* fest, werden die Änderungen automatisch gespeichert. Wurde die Zeichnung zuvor noch nicht unter einem Namen gespeichert, so kommt der Parameter *FileName* zum Einsatz. Die Zeichnung wird dann unter diesem Namen gespeichert. Wird in diesem Fall der Parameter *FileName* weggelassen, löst AutoCAD einen Fehler aus.

Wird der Parameter *SaveChanges* auf *FALSE* gesetzt, so wird die Zeichnung nicht gespeichert und alle Änderungen gehen verloren.

Nachdem die Zeichnung geschlossen wurde, ist sie auch aus der Dokumentauflistung verschwunden. Sie haben dann keinerlei Möglichkeit mehr, auf die einzelnen Zeichnungselemente zuzugreifen, es sei denn, Sie öffnen das Dokument wieder.

```
ThisDrawing.Close True
```

CopyObjects

Parameter: *ByVal Objects, [ByVal Owner], [IdPairs]*

Rückgabedatentyp: *Variant*

Diese Funktion erstellt eine Kopie der im Parameter *Objects* angegebenen Objekte und gibt die Kopien der Objekte in einem Datenfeld zurück, so dass diese dann weiterverarbeitet werden können. Sie können zusätzlich zum Array der zu kopierenden Objekte noch zwei weitere

Parameter festlegen. Der Parameter *Owner* legt das neue Eigentümerobjekt für die Kopien fest. Geben Sie hier keine Eigentümerobjekt an, werden die Objektkopien im Ursprungscontainer erzeugt. Wenn Sie eine Variable in dem *IdPairs*-Parameter festlegen, wird diese mit *IdPairs*-Objekten zu einem Array zusammengestellt, die dann verschiedene Informationen zu jedem der kopierten Objekte liefert.

Eigenschaft	Beschreibung
<i>Application</i>	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>IsCloned</i>	Zeigt an, ob das Quellobjekt kopiert wurde (<i>TRUE</i>) oder nicht (<i>FALSE</i>).
<i>IsOwnerXlated</i>	Zeigt an, ob das Eigentümerobjekt mit übersetzt wurde.
<i>IsPrimary</i>	Zeigt an, ob das Quellobjekt ein einzelnes Objekten (<i>TRUE</i>) oder Eigentum eines kopierten Elementes im primären Satz war (<i>FALSE</i>).
<i>Key</i>	Liefert die <i>ObjectID</i> des Ursprungsobjektes
<i>Value</i>	Liefert die <i>ObjectID</i> der erzeugten Kopie

Tabelle 6.2: Eigenschaften des *IdPairs*-Objektes

```
Dim ObjCol(1) As Object
Dim ObjInfo As AcadIdPair
Set objCol(0) = Obj1
Set objCol(1) = Obj2
retObj = CopyObjects(objCol)

oder

retObj = CopyObjects(objCol, , IdPObj)
Set ObjInfo = IdPObj(0)
```

Alternativ zu dieser globalen Kopierfunktion, können Sie auch die *Copy*-Anweisung der einzelnen Objekte verwenden, um diese zu duplizieren.

EndUndoMark/StartUndoMark

Mit diesen beiden Anweisungen teilen Sie AutoCAD den Start- und Endpunkt einer Zurück-Operation mit. Ein Beispiel zum besseren Verständnis: Angenommen, Sie erzeugen mittels einer VBA-Prozedur vier Linien. Wenn Sie nach Abschluss der Prozedur die Zurück-Funktion von AutoCAD ausführen, werden alle Linien in einer Operation wieder entfernt. Um AutoCAD zu zwingen, jede einzelne Linie in einem separaten Zurückvorgang zu integrieren, müssen Sie die Erstellungsfunktion innerhalb der Prozedur mit einer Start- und einer Endmarke einschließen, was folgendes Beispiel verdeutlicht.

```
Dim NewLine As AcadLine
Dim sPnt(2) As Double: ePnt(2) As Double
sPnt(0) = 0: sPnt(1) = 0
ePnt(0) = 0: ePnt(1) = 10
For i = 0 To 3
    ThisDrawing.StartUndoMark
    Set line = ThisDrawing.ModelSpace.AddLine(stPnt, endPnt)
    sPnt(0) = sPnt(0) + 5: ePnt(0) = ePnt(0) + 5
    ThisDrawing.EndUndoMark
Next
```

Export

Parameter: *ByVal FileName As String, ByVal Extension As String, ByVal SelectionSet As AcadSelectionSet*

Wie der Name bereits verrät, dient diese Prozedur dem Export der Zeichnung oder einzelner Zeichnungselemente in eine Datei der Formate *wmf*, *sat*, *eps*, *dxf* oder *bmp*. Mit dem ersten Parameter müssen Sie der Funktion den Namen der zu erstellenden Datei mitteilen. Der zweite erhält die Dateierweiterung der Exportdatei, was damit den Typ der zu erstellenden Datei festlegt. Im dritten Parameter müssen Sie einen Auswahlsatz festlegen, der die Objekte enthält, die exportiert werden sollen. Hier noch einige Regeln für die Verwendung des Auswahlsatzes:

- Wenn Sie die Daten in eine Datei der Formate *wmf* oder *bmp* exportieren, sollte der Auswahlsatz Objekte enthalten. Geben Sie einen leeren Auswahlsatz im Parameter an, wird der Benutzer in der AutoCAD-Eingabeaufforderung angewiesen, Objekte auszuwählen.
- Wenn Sie als Exportformat *eps* oder *dxf* verwenden, wird der Parameter *SelectionSet* ignoriert, egal ob in dem Auswahlsatz Objekte enthalten sind oder nicht. Es wird in diesem Fall immer die komplette Zeichnung exportiert.
- Auch beim Export in das *sat*-Format sollte ein mit Objekten gefüllter Auswahlsatz verwendet werden. Im Gegensatz zu den Formaten *wmf* und *bmp* wird hier aber keine Eingabeaufforderung in der AutoCAD-Befehlszeile angezeigt, wenn der Auswahlsatz leer ist. In diesem Fall wird keine Datei erstellt. Außerdem ignoriert AutoCAD Objekte im Auswahlsatz, die keine Volumenkörper oder Regionen sind.

```
Dim exportFile As String
Dim sset As AcadSelectionSet
exportFile = "C:\DXF-File"
If TypeName(ThisDrawing.SelectionSets("EXPORT")) = "Nothing" Then
    ThisDrawing.SelectionSets.Add "EXPORT"
End If
Set sset = ThisDrawing.SelectionSets("EXPORT")
ThisDrawing.Export exportFile, "DXF", sset
```

GetVariable

Parameter: *Name As String*

Rückgabedatentyp: *Variant*

Diese Funktion ruft den aktuellen Wert einer AutoCAD-Systemvariable ab. Den Namen der entsprechenden Variable geben Sie der Funktion als Parameter mit auf den Weg. Je nach Typ der Systemvariablen erhalten Sie dann auch Ergebnisse unterschiedlichen Datentyps. Aus diesem Grund muss die Variable, in die Sie den Wert schreiben lassen wollen, auch als Datentyp *Variant* deklariert werden.

```
Dim VarOut As Variant
```

Abruf der aktuellen Texthöhe:

```
VarOut = ThisDrawing.GetVariable("TEXTSIZE")
```

Die Variable *VarOut* enthält jetzt einen Wert des Datentyps *Double*.

Abruf des zuletzt eingegebenen Punkts:

```
VarOut = ThisDrawing.GetVariable("LASTPOINT")  
X = VarOut(0): Y = VarOut(1): Z = VarOut(2)
```

Die Variable *VarOut* enthält jetzt ein 3D-Array mit den entsprechenden Koordinaten.

Abruf der AutoCAD-Versionsnummer:

```
VarOut = ThisDrawing.GetVariable("ACADVER")
```

Die Variable *VarOut* enthält jetzt eine Zeichenfolge (Datentyp *String*).

Siehe auch Kapitel

HandleToObject

Parameter: *Handle As String*

Rückgabedatentyp: *Object*

Bei dieser Prozedur handelt es sich um eine der wichtigsten der Klasse *AcadDocument*. Mit ihr können Sie ein Objekt mit Hilfe der Referenz (Handle), die das Objekt besitzt, aus der Zeichnung abrufen. Ist kein Element in der Zeichnung vorhanden, das dem mitgegebenen Handle entspricht, gibt die Funktion *Nothing* zurück. Da jedes Objekt einer Zeichnung, egal ob Zeichnungselement (Linie, Polylinie, Text ...) oder Stilobjekt (Layer, Linientyp, Textstil ...), eine Handle-Eigenschaft besitzt, kann folglich jedes Element auch über diese Funktion abgerufen werden, ohne dass das Element direkt durch den Benutzer ausgewählt worden ist.

```
Dim Object As Object  
Set Object = ThisDrawing.HandleToObject(Handle)
```

Siehe auch *ObjectIdToObject*

Import

Parameter: *ByVal FileName As String, InsertionPoint, ByVal ScaleFactor As Double*

Rückgabedatentyp: *Object*

Neben der Exportfunktion bietet die Klasse *AcadDocument* natürlich auch eine Importfunktion, mit der Sie Zeichnungsdaten aus Dateien der Formate *sat*, *eps*, *dxf* oder *wmf* in das Dokument einlesen lassen können. Wenn Sie Daten im *wmf*-Format importieren, erstellt diese Funktion eine neue Blockreferenz, die alle importierten Daten enthält. Diese Blockreferenz können Sie dann an eine Objektvariable zur Weiterverarbeitung übergeben:

```
Set ImpObj = ThisDrawing.Import(importFile, InsPoint, ScaleFactor)
```

Bei allen anderen Formaten wird kein Objekt ausgegeben. Sie können sich deshalb die Zuweisung sparen:

```
ThisDrawing.Import importFile, InsPoint, ScaleFactor
```

Als Parameter verlangt die Funktion zunächst den zu importierenden Dateinamen, dann den Einfügapunkt für die neuen Zeichnungsdaten und schließlich noch einen Skalierfaktor, mit dem die Daten eingefügt werden sollen.

Siehe auch *Export*

LoadShapeFile

Parameter: *ByVal FullName As String*

Diese Prozedur weist AutoCAD an, die im Parameter festgelegte Symboldatei (*shx*) zu laden.

```
ThisDrawing.LoadShapeFile FileName
```

Siehe auch Kapitel

New

Parameter: *[ByVal TemplateFileName As String]*

Rückgabedatentyp: *AcadDocument*

Diese Funktion erstellt eine neue AutoCAD-Zeichnung und gibt dieses Dokument an eine Objektvariable zurück. Da die Funktion nur im SDI-Modus (Single Document Interface) verwendet werden kann, ist das Dokument immer im Klassenmodul *ThisDrawing* enthalten und Sie können die Zuweisung ignorieren.

```
Dim TempName As String  
TempName = ThisDrawing.Application.Preferences.Files. _  
    TemplateDwgPath & "\ansi-a.dwt"  
ThisDrawing.New TempName
```

oder

```
Set ActDoc = ThisDrawing.New(TempName)
```

Als Parameter können Sie auch wahlweise den Namen einer Vorlagendatei verwenden, die dann für das neue Dokument verwendet wird.

Um Zeichnungen zu erstellen, wenn sich AutoCAD im MDI-Modus (Multiple Document Interface) befindet, was ja normalerweise immer der Fall sein dürfte, müssen Sie die *Add*-Funktion der Dokumentauflistung (Klasse *AcadDocuments*) verwenden. Um zu kontrollieren, in welchem Modus sich AutoCAD gerade befindet, rufen Sie den Wert der Systemvariablen „SDI“ ab und reagieren Sie dementsprechend in Ihrem Programmcode.

ObjectIdToObject

Parameter: *ByVal ObjectId As Long*

Rückgabedatentyp: *AcadDocument*

Genau wie die Funktion *HandleToObject*, gibt auch diese das Objekt, das der Parameterangabe der Funktion entspricht, zurück. Der Unterschied besteht lediglich im Parameter, der zur Identifizierung des Objekts dient. Anstelle des Handle wird hier die *ObjectId* verwendet, die aber genauso in jedem AutoCAD-Element gespeichert wird und als Eigenschaft abrufbar ist.

```
Dim Object As Object  
Set Object = ThisDrawing.ObjectIdToObject(Handle)
```

Das Objekt kann nur abgerufen werden, wenn es sich tatsächlich in der Zeichnung befindet, also nicht gelöscht wurde oder sich in einer anderen geöffneten Zeichnung befindet.

Siehe auch *HandleToObject*

Open

Parameter: *ByVal FullName As String*

Rückgabedatentyp: *AcadDocument*

Auch diese Funktion kann nur angewendet werden, wenn sich AutoCAD im SDI-Modus (Single Document Interface) befindet. Sie weist AutoCAD an, die im Parameter *FullName* angegebene Datei in AutoCAD zu öffnen. Wenden Sie diese Funktion im MDI-Modus an, wird ein Fehler ausgelöst. Verwenden Sie dann vielmehr die *Open*-Funktion der Dokumentauflistung. Als Ergebnis liefert die Funktion ein Objekt vom Datentyp *AcadDocument*. Da die Funktion nur im SDI-Modus funktioniert, können Sie aber immer über das Klassenmodul *ThisDrawing* auf die Zeichnung zugreifen.

```
Dim FileName As String
FileName = ThisDrawing.Application.Path & "\Sample\Opera.dwg"
ThisDrawing.Open FileName
```

oder

```
Set ActDoc = ThisDrawing.Open(FileName)
```

PurgeAll

Wenn Sie diese Prozedur ausführen, werden alle unbenutzten Blöcke, Layer, Linientypen usw. aus der Zeichnung entfernt. Die Bereinigungsoperation wird natürlich nur auf das Dokument angewendet, das die Prozedur aufgerufen hat. Da dieser Anweisung keinerlei Parameter mitgegeben werden können, werden folglich alle Objekte bereinigt, als hätten Sie in der Befehlszeile `-Bereinigen *j` eingegeben.

```
ThisDrawing.PurgeAll
```

Regen

Parameter: *ByVal WichViewports As AcRegenType*

Mithilfe dieser Anweisung können Sie eine Regenerierung der kompletten Zeichnung oder zumindest des aktuellen Ansichtsfensters auslösen. Welchen Teil der Zeichnung Sie regenerieren lassen möchten, geben Sie im Parameter *WichViewports* an.

acActiveViewport – Diese Konstante legt fest, dass nur das aktuelle Ansichtsfenster regeneriert werden soll.

acAllViewports – Die Festlegung dieser Konstante ermöglicht die Regenerierung der kompletten Zeichnung.

```
ThisDrawing.Regan acActiveViewport
```

Save

Diese Prozedur speichert das aufrufende Dokument unter dem aktuellen Namen. Wurde die Zeichnung zuvor noch nicht gesichert, werden für die Speicherung die Eigenschaften *Path* und *Name* des Dokuments verwendet.

```
ThisDrawing.Save
```

SaveAs

Parameter: *ByVal FullFileName As String, [ByVal SaveType]*

Im Gegensatz zur vorherigen Prozedur müssen Sie hier den Namen der Datei angeben, unter dem die Zeichnung gespeichert werden soll. Sie entspricht also der Anweisung **SPEICHERN UNTER** in AutoCAD. Im ersten Parameter legen Sie den Namen der Datei fest, in der die Zeichnungsdaten gespeichert werden sollen. Der zweite Parameter ist optional und soll den Dateityp der Zeichnung nennen. Mögliche Konstanten, die Sie für diesen Parameter verwenden können, sind in der folgenden Tabelle aufgelistet.

Konstante	Dateityp	Wert
<i>acR12_DXF</i>	AutoCAD Release12/LT2 DXF (*.dxf)	1
<i>acR13_DWG</i>	AutoCAD Release13/LT95 DWG (*.dwg)	4
<i>acR13_DXF</i>	AutoCAD Release13/LT95 DXF (*.dxf)	5
<i>acR14_DWG</i>	AutoCAD Release14/LT97 DWG (*.dwg)	8
<i>acR14_DXF</i>	AutoCAD Release14/LT97 DXF (*.dxf)	9
<i>acR15_DWG</i>	AutoCAD 2000 DWG (*.dwg)	12
<i>acR15_DXF</i>	AutoCAD 2000 DXF (*.dxf)	13
<i>acR15_Template</i>	AutoCAD 2000 Vorlagendatei (*.dwt)	14
<i>acNative</i>	Wenn Sie diese Konstante wählen, wird die Zeichnung in dem Format gespeichert, das der ausführenden AutoCAD-Version entspricht.	12

Tabelle 6.3: Konstanten der SaveAs-Anweisung

Als Standardwert der Speicherung legt AutoCAD die Konstante *acR15_DWG* fest:

```
Dim FileName As String
FileName = GetDocName
ThisDrawing.SaveAs FileName, acR15_DWG
```

oder:

```
ThisDrawing.SaveAs FileName
```

SendCommand

Parameter: *ByVal Command As String*

Um eine Befehlszeichenfolge in der AutoCAD-Befehlszeile ausführen zu lassen, können Sie diese Anweisung der Dokumentklasse verwenden. Sie sendet die angegebene Zeichenfolge zur Weiterverarbeitung an AutoCAD. Die Anweisung verarbeitet alle AutoCAD-Befehle inklusive Lisp-Ausdrücken, als wären sie in der Befehlszeile eingegeben worden. Benutzen Sie die Konstante *vbCr*, um die Zeichenfolge abzuschließen, was gleichbedeutend mit der Betätigung der -Taste in der AutoCAD-Befehlszeile ist.

Wer sich bereits mit AutoLisp auskennt, der dürfte keinerlei Schwierigkeiten bei der Verwendung dieser Prozedur haben. Für alle anderen im Folgenden noch einige Beispiele und Anmerkungen für die Erstellung der Zeichenfolge, die an AutoCAD weitergegeben wird.

- **Übergabe von Parametern:** Alle AutoCAD-Befehle, die nicht mit einem Dialogfenster als Eingabemaske arbeiten, erwarten die Auswahl von Parametern, wie die Eingabe von Koordinaten oder Optionen. Sie können in der Zeichenfolge der *SendCommand*-Anweisung die Parameter bzw. Optionen mit übergeben, indem Sie ein Leerzeichen zwischen die einzelnen Parameter setzen:

```
SendCommand "-Layer SE 0 " & vbCr
```

oder:

```
SendCommand "-Layer SE 0 "
```

- **Übergabe von Koordinaten:** Punktkoordinaten müssen stets als Zeichenfolge übergeben werden. Das heißt, dass Sie einen gewählten Punkt erst in einzelne Zeichenfolgen umwandeln müssen (*Utility.RealToString*), da sonst ein Komma statt eines Punkts als Dezimaltrenner übergeben wird.

```
PtX = Utility.RealToString(Pt(0), acDecimal, 5)
PtY = Utility.RealToString(Pt(1), acDecimal, 5)
SendCommand "L " & PtX & "," & PtY
```

- **Verwendung von Zwischenabfragen:** Da VBA, anders als LISP, keine *PAUSE*-Anweisung in der Kommandozeile verwenden kann, müssen wir uns für diesen Fall eine andere Methode einfallen lassen. Nehmen wir einmal an, Sie möchten einen Kreis mithilfe des AutoCAD-Befehls *_Circle* erzeugen und die Koordinaten nach Ausführung des Befehls festlegen, den Radius aber über die *SendCommand*-Anweisung senden. Zu diesem Zweck müssten wir die *SendCommand*-Befehlszeile trennen. Der erste Teil bekommt die *_Circle*-Anweisung, dann wählen wir mithilfe der *GetPoint*-Funktion eine Koordinate, um diese schließlich samt Radius in einer zweiten *SendCommand*-Zeile zu übertragen.

```
ThisDrawing.SendCommand "_circle "
Pt = ThisDrawing.Utility.GetPoint(, "Punkt wählen")
ComStr = Utility.RealToString(Pt(0), acDecimal, 3) & "," & " _
        & Utility.RealToString(Pt(1), acDecimal, 3)
ThisDrawing.SendCommand ComStr & " "
```

SetVariable

Parameter: *ByVal Name As String, Value*

Diese Anweisung wird zum Ändern einzelner AutoCAD-Systemvariablen verwendet. Sie müssen dieser Prozedur zwei Parameter mitgeben. Im ersten natürlich den Namen der Systemvariablen, die Sie ändern wollen, und im zweiten den neuen Wert. Da die einzelnen Systemvariablen Einträge unterschiedlichen Datentyps enthalten, müssen Sie für die entsprechenden Variablen auch speziell deklarierte Werte übergeben, wie die folgenden Beispiele zeigen werden.

Ändern der aktuellen Texthöhe:

```
Dim TxtSize As Double
TxtSize = 2.5
ThisDrawing.SetVariable "TEXTSIZE", TxtSize
```

Festlegen eines neuen Rasterabstands für das aktuelle Ansichtsfenster:

```
Dim RstPnts(1) As Double
RstPnts(0) = 20: RstPnts(1) = 20
ThisDrawing.SetVariable "GRIDUNIT", RstPnts
```

Ändern des Schraffurmustersnamens:

```
Dim HPName As String
HPName = "ANGLE"
ThisDrawing.SetVariable "HPNAME", HPName
```

Sie können natürlich nur die Systemvariablen ändern, die nicht schreibgeschützt sind.

Siehe auch *GetVariable*.

WBlock

Parameter: *ByVal FileName As String, SelectionSet As AcadSelectionSet*

Diese Prozedur hat genau die gleiche Aufgabe zu erfüllen wie der AutoCAD-Befehl `WBlock`, also die Erzeugung einer neuen Zeichnungsdatei aus den in einem Auswahlset zusammengestellten Zeichnungselementen. Die Ausführung dieser Prozedur ist recht einfach. Zunächst stellen Sie einen Auswahlset zusammen, der dann im zweiten Parameter der Prozedur übergeben wird. Anschließend rufen Sie die Anweisung selbst auf und fügen im ersten Parameter den Zeichnungsnamen hinzu.

```
Dim FileName As String
Dim AcSSet As AcadSelectionSet
On Local Error Resume Next
If TypeName(ThisDrawing.SelectionSets("EXPORT")) = "Nothing" Then
    ThisDrawing.SelectionSets.Add "EXPORT"
End If
Set AcSSet = ThisDrawing.SelectionSets("EXPORT")
AcSSet.SelectOnScreen
FileName = InputBox("Legen Sie den Zeichnungsnamen fest")
ThisDrawing.WBlock FileName, AcSSet
```

6.3 Kommunikation des Benutzers mit der AutoCAD-Zeichnung

Die Klasse *AcadUtility* umfasst eine Vielzahl von Prozeduren, die Sie bei der Entwicklung von AutoCAD-Applikationen häufig verwenden werden. So enthält dieses Objekt Funktionen, mit denen Sie im Dokumentfenster einzelne Zeichnungsobjekte, Punkte, Winkel usw. wählen können. Diese Klasse ist besonders wichtig für die spätere Kommunikation zwischen dem Benutzer und der AutoCAD-Zeichnung. Schauen Sie sich also die Benutzung der einzelnen Funktionen genau an, da wir diese im weiteren Verlauf noch sehr häufig verwenden werden.

Sie sollten eine Objektvariable öffentlich deklarieren und in dieser Variablen dann einen Verweis auf die Klasse *AcadUtility* speichern, damit Sie die Funktionen einfacher benutzen können.

```
Public AcadUtil As AcadUtility
Set AcadUtil = ThisDrawing.Utility
```


Winkel zwischen zwei Punkten berechnen (AngleFromXAxis)

Diese Funktion liefert als Ergebnis den Winkel zwischen zwei Punkten. Diese Punkte müssen als Array (X, Y, Z) deklariert sein. Der Winkel, der zurückgegeben wird, hat die Maßeinheit Radiant und hängt weiterhin von der Einstellung der Systemvariablen für die Winkelrichtung (*ANGBASE*) ab.

```
Dim FromPoint(2) As Double
Dim ToPoint(2) As Double
Dim OutAngle As Double
FromPoint(0) = 0: FromPoint(2) = 0: FromPoint(2) = 0
ToPoint(0) = 10: ToPoint(2) = 10: ToPoint(2) = 0
OutAngle = ThisDrawing.Utility.AngleFromXAxis(FromPoint, ToPoint)
```

Zeichenfolge in einen Winkel umwandeln (AngleToReal)

Diese Funktion wandelt einen Winkel, der als Zeichenkette (String) mitgegeben wird, in einen Radiantwinkel um. Als zweiten Parameter müssen Sie noch die Einheit des Eingabewinkels festlegen. Zu diesem Zweck können Sie die Systemvariable *AUNITS*, die die aktuell eingestellte Winkeleinheit enthält, verwenden. Benutzen Sie hierzu die Funktion *ThisDrawing.GetVariable*.

Da der Parameter bereits eine Auflistung von Konstanten enthält, können Sie auch eine der entsprechenden Konstanten wählen.

acDegrees – Dezimalgrad

acDegreeMinuteSeconds – Grad Minuten Sekunden

acGrads – Neugrad

acRadians – Radiant

```
Dim InAngle As String
Dim OutAngle As Double
InAngle = "100.0000g"
OutAngle = ThisDrawing.Utility.AngleToReal(InAngle, _
    ThisDrawing.GetVariable("AUNITS"))
```

Winkel in ein Zeichenfolge umwandeln (AngleToString)

Diese Funktion bildet das Gegenstück zur vorhergehenden Funktion. Das bedeutet, Sie können einen Winkel, der als Zahl (*Double*) mitgegeben wird, in eine Zeichenkette (*String*) umwandeln. Als zweiten Parameter müssen Sie noch die Einheit des Eingabewinkels festlegen. Zu diesem Zweck können Sie die Systemvariable *AUNITS*, die die aktuell eingestellte Winkeleinheit enthält, verwenden. Zusätzlich enthält diese Funktion einen dritten Parameter, der die Genauigkeit (Anzahl der Nachkommastellen) festlegt. Auch hierfür gibt es eine Systemvariable. *AUPREC* liefert die aktuell eingestellte Winkelgenauigkeit. Sie können für den zweiten Parameter auch die zuvor beschriebenen Konstanten verwenden.

```
Dim InAngle As Double
Dim OutAngle As String
InAngle = 100.0000
OutAngle = ThisDrawing.Utility.AngleToReal(InAngle, acGrads _
    ThisDrawing.GetVariable("AUPREC"))
```

Array erstellen (CreateTypedArray)

Diese Funktion erstellt ein Array von Daten, wie sie bei der Erstellung von Polylinien, Linien usw. verwendet werden. Damit Sie diese Methode verwenden können, müssen Sie die Klasse *AcadUtility* einer Objektvariablen vom Typ *Object* zuweisen. Im ersten Parameter legen Sie die Variable fest, in der die Ergebnisse abgelegt werden sollen. Der zweite Parameter bestimmt den Datentyp der Eingabedaten. Hierzu können Sie folgende Konstanten verwenden:

- *vbBoolean*
- *vbInteger*
- *vbLong*
- *vbSingle*
- *vbDouble*

```
Dim AcadUtil As Object
Dim OutPline As Acad3DPolyline
Dim VertList As Variant
Set AcadUtil = ThisDrawing.Utility
AcadUtil.CreateTypedArray VertList, vbDouble, 0, 0, 0, 10, 10, 0
Set OutPline = ThisDrawing.ModelSpace.Add3DPoly(VertList)
```

Die vorherigen Codezeilen erzeugen aus der erstellten Koordinatenliste *VertList* eine 3D-Polylinie im Modellbereich. Die Variable *VertList* wird dabei durch die Funktion in folgendes Array umgewandelt:

```
VertList(0) = 0; VertList(1) = 0; VertList(2) = 0
VertList(3) = 10; VertList(4) = 10; VertList(5) = 0
```

Zeichenfolge in eine Strecke umwandeln (DistanceToReal)

Diese Funktion wandelt eine Dezimalzahl, die als Zeichenkette (*String*) mitgegeben wird, in eine Fließkommazahl um. Als zweiten Parameter müssen Sie noch die Einheit des Eingabewerts festlegen. Zu diesem Zweck können Sie die Systemvariable *AUPREC* verwenden, die die aktuell eingestellte Dezimaleinheit enthält. Benutzen Sie hierzu die Funktion *ThisDrawing.GetVariable*.

Da der Parameter bereits eine Auflistung von Konstanten enthält, können Sie auch eine der entsprechenden Konstanten wählen.

- *acDefaultUnits* – die im System eingestellte Einheit
- *acScientific*
- *acDecimal*
- *acEngineering*
- *acArchitectural*
- *acFractional*

```
Dim InValue As String
Dim OutValue As Double
InValue = "100.0000"
OutValue = ThisDrawing.Utility.DistanceToReal(InValue, acDefaultUnits)
```

Winkleingabe durch den Benutzer in der Zeichnung

Die Funktion *GetAngle*

Mit dieser Funktion ermöglichen Sie es dem Benutzer, einen Winkel in der Zeichnungsoberfläche zu bestimmen. Der Benutzer kann dabei den Winkel in der Zeichnung abgreifen oder einfach einen Winkel in der Befehlszeile eingeben. Es gelten die gleichen Bestimmungen, wie Sie es bereits aus der Arbeit mit AutoCAD kennen. Über den zweiten Parameter können Sie eine Zeichenfolge festlegen, die in der Befehlszeile von AutoCAD erscheint.

Um bestimmte Benutzereingaben zu ermitteln (z.B. den aktuellen Drehwinkel der Zeichnung als Winkel übernehmen), gibt es weitere Hilfsmittel, die auch bei allen anderen Benutzereingaben gelten. Daher werde ich diese spezielle Methode gesondert besprechen. Im vorliegenden Code wird deshalb einfach nur ein Winkel angefragt.

```
Dim Prompt As String
Dim OutAngle As Double
Prompt = "Legen Sie einen Winkel fest:"
OutAngle = ThisDrawing.Utility.GetAngle(, Prompt)
```

Als ersten Parameter können Sie zusätzlich die Koordinaten eines Punkts mitgeben, den Sie als Array (*Point(2) As Double*) deklarieren. Sie legen damit den Startpunkt des Winkels bereits fest.

```
Dim Prompt As String
Dim OutAngle As Double
Dim StartPoint(2) As Double
Prompt = "Legen Sie einen Winkel fest:"
StartPoint(0) = 100: StartPoint(1) = 150
OutAngle = ThisDrawing.Utility.GetAngle(StartPoint, Prompt)
```

Die Funktion *GetOrientation*

Diese Methode hat die gleiche Funktionsweise wie *GetAngle*. Allerdings wird bei dieser Funktion der Wert der Systemvariablen *ANGBASE* ignoriert.

```
Dim Prompt As String
Dim OutAngle As Double
Prompt = "Legen Sie eine Richtung fest:"
OutAngle = ThisDrawing.Utility.GetOrientation(, Prompt)
```

Als ersten Parameter können Sie zusätzlich die Koordinaten eines Punkts mitgeben, den Sie als Array (*Point(2) As Double*) deklarieren. Sie legen damit den Startpunkt des Winkels bereits fest.

```
Dim Prompt As String
Dim OutAngle As Double
Dim StartPoint(2) As Double
Prompt = "Legen Sie eine Richtung fest:"
StartPoint(0) = 100: StartPoint(1) = 150
OutAngle = ThisDrawing.Utility.GetOrientation(StartPoint, Prompt)
```

Die Ecke eines Rechtecks abfragen (GetCorner)

Mit dieser Funktion ermöglichen Sie es dem Benutzer, einen Punkt in der Zeichnung zu bestimmen. Dieser Punkt liegt einem Eingabepunkt, in Form eines Rechtecks, gegenüber. Im ersten Parameter geben Sie dieser Funktion den ersten Eckpunkt als Array (*Point(2) As Double*) mit. Über den zweiten Parameter können Sie eine Zeichenfolge festlegen, die in der Befehlszeile von AutoCAD erscheint.

```
Dim Prompt As String
Dim OutPoint As Variant
Dim StartPoint(2) As Double
Prompt = "Legen Sie die rechte obere Ecke des Rahmens fest:"
StartPoint(0) = 100: StartPoint(1) = 150
OutPoint = ThisDrawing.Utility.GetCorner(StartPoint, Prompt)
```

Nach erfolgreicher Ausführung erhalten Sie ein Array, das drei Koordinaten (X, Y, Z) enthält, zurück.

Abstandsermittlung durch Benutzereingabe (GetDistance)

Mit dieser Funktion ermöglichen Sie es dem Benutzer, einen Abstand zwischen zwei Punkten in der Zeichnungsoberfläche zu bestimmen. Der Benutzer kann dabei den Abstand in der Zeichnung abgreifen oder einfach einen Wert in der Befehlszeile eingeben. Es gelten die gleichen Bestimmungen, wie Sie es bereits von der Arbeit mit AutoCAD her kennen. Über den zweiten Parameter können Sie eine Zeichenfolge festlegen, die in der Befehlszeile von AutoCAD erscheint.

```
Dim Prompt As String
Dim OutValue As Double
Prompt = "Legen Sie eine Strecke fest:"
OutValue = ThisDrawing.Utility.GetDistance(, Prompt)
```

Als ersten Parameter können Sie zusätzlich die Koordinaten eines Punkts mitgeben, den Sie als Array (*Point(2) As Double*) deklarieren. Sie legen damit den Startpunkt der zu bestimmenden Strecke bereits fest.

```
Dim Prompt As String
Dim OutValue As Double
Dim StartPoint(2) As Double
Prompt = "Legen Sie eine Strecke fest:"
StartPoint(0) = 100: StartPoint(1) = 150
OutValue = ThisDrawing.Utility.GetDistance(StartPoint, Prompt)
```

Einzelne grafische Objekte durch den Benutzer abrufen (GetEntity)

Diese ist eine besondere Funktion, mit der Sie den Benutzer auffordern, ein AutoCAD-Objekt im Zeichnungsfenster auszuwählen. Anders als beim Auswahl Satz wird immer nur ein Objekt zurückgegeben. Zusätzlich zum AutoCAD-Element wird noch ein Punkt zurückgegeben, der der gepickten Koordinate entspricht. Hat der Benutzer kein Objekt gewählt, wird ein Fehler ausgelöst.

```
Dim Prompt As String
Dim OutEnt As Objekt
Dim OutPoint As Variant
Prompt = "Wählen Sie ein AutoCAD-Element:"
ThisDrawing.Utility.GetEntity OutEnt, OutPoint, Prompt
```

Um zu prüfen, welchem Elementtyp (Linien, Blockreferenz, Polylinie ...) das zurückgegebene Objekt entspricht, haben Sie zwei Möglichkeiten. Zum einen können Sie die Eigenschaft *ObjectName* des Objekts nutzen. Die zweite Möglichkeit ist die Prüfung über die VBA-Anweisung *TypeName()*. Diese Anweisung gibt dann als Ergebnis den Klassennamen des Objekts zurück. Auf diese Weise können Sie alle Klassennamen verwenden. Sie nehmen einfach den Namen der Klasse, wie er im Objektkatalog erscheint, und fügen zusätzlich ein „I“ vor der entsprechenden Bezeichnung ein.

```
Select Case TypeName(OutEnt)
    Case "IAcadLine"
        'Das gewählte Objekt ist eine Linie
    Case "IAcadBlockReference"
        'Das gewählte Objekt ist eine Blockreferenz
    Case "IAcadCircle"
        'Das gewählte Objekt ist ein Kreis
End Select
```

Verwendung von Schlüsselwörtern

Die Funktionen *GetInput* und *InitializeUserInput*

Diese Funktionen sind eng miteinander verknüpft. Um die Prozedur *GetInput* verwenden zu können, müssen Sie erst die Funktion *InitializeUserInput* ausführen, um eine Benutzerzeichenfolge festzulegen, die dem Benutzer bestimmte Schlüsselwörter zur Verfügung zu stellen. Sie finden diese Vorgehensweise bereits bei den verschiedensten AutoCAD-Befehlen.

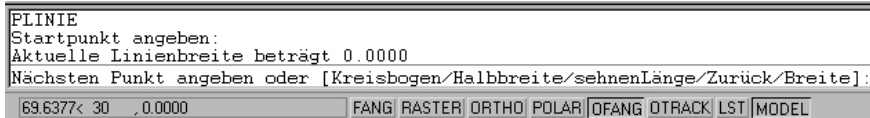


Abbildung 6.1: Beispiel einer Benutzerabfrage

Die Funktion *GetInput* gibt nach der Benutzereingabe das eingegebene Schlüsselwort als Zeichenfolge zurück. Somit sind Sie in der Lage, bei den verschiedensten Auswahlfunktionen (*GetPoint*, *GetAngle*, *GetCorner* ...) dem Benutzer die Eingabe von Schlüsselwörtern zu ermöglichen.

```
Dim keywordList As String
keywordList = "Exit Optionen Zurück"
ThisDrawing.Utility.InitializeUserInput 128, keywordList
```

Dieser Code initialisiert zunächst die entsprechenden Schlüsselwörter für die spätere Eingabe. Im ersten Parameter können Sie zusätzlich einen Bit-Wert setzen, wobei die entsprechenden Werte folgende Wirkung haben.

- 1 – Diese Option verhindert, dass der Benutzer eine *Null*-Zeichenfolge eingibt, also die Eingabe nur mit der Leertaste oder  abschließt.
- 2 – Diese Option legt fest, dass keine 0 vom Benutzer eingegeben werden kann.
- 4 – Diese Option bestimmt, dass keine negativen Zahlen vom Benutzer eingegeben werden können.

- 8 – Diese Option ermöglicht dem Benutzer die Eingabe eines Punkts auch außerhalb der Zeichnungsgrenzen. Die Prüfung der Zeichnungsgrenzen wird ignoriert, auch wenn die Systemvariable *LIMCHECK* aktiviert wurde.
- 16 – Diese Option wird derzeit noch nicht benutzt.
- 32 – Diese Option legt fest, dass bei Funktionen, die Linien verwenden (*GetDistance*, *GetCorner*), gestrichelte statt durchgezogener Linien verwendet werden. Wurde die Systemvariable *POPUPS* auf 0 gesetzt, ignoriert AutoCAD diese Option.
- 64 – Ignoriert die Z-Koordinate eines 3D-Arrays, das von der Funktion *GetDistance* zurückgegeben wird. So kann der Benutzer eine 2D-Strecke bestimmen, auch wenn Koordinaten mit unterschiedlichen Höhen gewählt wurden.
- 128 – Erlaubt eine beliebige Benutzereingabe.

Sie können die Bits für diesen Parameter auch kombinieren, indem Sie die entsprechenden Werte addieren. Die Kombination von 2 und 4 würde vorgeben, dass der Benutzer keine Zahlen eingeben darf, die kleiner oder gleich 0 sind. Der entsprechende Parameter wäre entsprechend $2 + 4 = 6$.

```
Dim OutPnt As Variant
Dim Prompt As String
Prompt = "Legen Sie einen Punkt fest (Exit Optionen Zurück):"
OutPnt = ThisDrawing.Utility.GetPoint(, Prompt)
If TypeName(OutPnt) = "Double()" Then
    MsgBox "Sie haben einen Punkt gewählt", vbOkOnly
Else
    If StrComp(Err.Description, "Benutzereingabe", 1) = 0 Then
        Dim inputString As String
        Err.Clear
        inputString = ThisDrawing.Utility.GetInput
        Select Case inputString
            Case "Exit"
                MsgBox "Exit wurde gewählt", vbOkOnly
            Case "Optionen"
                MsgBox "Optionen wurde gewählt", vbOkOnly
            Case "Zurück"
                MsgBox "Zurück wurde gewählt", vbOkOnly
        End Select
    End If
End If
```

Im zweiten Teil des Programms passiert Folgendes: Nach der Deklaration der Variablen wird durch die Funktion *GetPoint* ein Punkt abgefragt. Da wir eine Initialisierung der Benutzereingaben vorgenommen haben, kann der Benutzer außer des Punkts auch Texteingaben machen. Nach Abschluss der Funktion *GetPoint* kontrollieren wir, was der Benutzer eigentlich eingegeben hat. Hat der Benutzer einen Punkt gewählt, gibt die Anweisung *TypeName(OutPnt)* die Zeichenfolge *Double()* zurück, die anzeigt, dass die Variable Daten vom Typ *Double* enthält. Ist dies der Fall, hat der Benutzer kein Schlüsselwort eingegeben. Ist die Variable *OutPnt* nicht vom Typ *Double*, muss der Benutzer entweder ein Schlüsselwort eingegeben oder die Funktion abgebrochen haben. Um herauszufinden, was der Benutzer gemacht hat, prüfen wir die Variable *Description* des *Err*-Objekts auf ihren Inhalt. Das hat folgenden Grund: Sollte der Benutzer ein Schlüsselwort eingegeben haben, wird der Fehlercode -2145320928 („Benutzereingabe

ist ein Schlüsselwort“) ausgelöst. Ist dies der Fall, können wir damit fortfahren herauszufinden, welches Schlüsselwort verwendet wurde. Hierbei hilft uns nun die Funktion *GetInput*. Diese Funktion gibt genau das Wort zurück, das vom Benutzer gewählt wurde.

Der Benutzer muss das Wort nicht vollständig eingegeben haben. Es reicht für die Auswertung bereits ein Teil des Worts.

Tipp

Verwenden Sie Schlüsselwörter, die den gleichen Anfang besitzen, wird stets das erste Schlüsselwort ausgegeben.

```
keywordList = "Punkt1 Punkt2"
ThisDrawing.Utility.InitializeUserInput 128, keywordList
```

Gibt der Benutzer nur „P“ oder „Pu“ ein, wird als Ergebnis immer „Punkt1“ als eingegebenes Schlüsselwort zurückgegeben.

Zum Abschluss noch einige allgemeine Bemerkungen und Informationen zu den Funktionen des *Utility*-Objekts. Sie können die Funktionen *InitializeUserInput* und *GetInput* bei folgenden Funktionen verwenden: *GetAngle*, *GetCorner*, *GetDistance*, *GetEntity*, *GetInteger*, *GetKeyword*, *GetOrientation*, *GetPoint*, *GetReal*, *GetString* und *GetSubEntity*. Die Vorgehensweise bei Initialisierung und Abfrage der Schlüsselwörter ist dabei immer die gleiche.

Eingabe von Schlüsselwörtern (GetKeyword)

Diese Funktion fordert den Benutzer auf, ein Schlüsselwort in der AutoCAD-Befehlszeile einzugeben. Als Ergebnis wird dann das komplette Schlüsselwort zurückgegeben, das aus der Benutzereingabe resultiert. Die maximale Eingabelänge des Schlüsselworts liegt bei insgesamt 511 Zeichen. Um diese Funktion ausführen zu können, müssen Sie die Methode *InitializeUserInput* initialisieren. Hat der Benutzer eine Zeichenfolge eingegeben, aus der kein Schlüsselwort resultiert, zeigt AutoCAD eine Fehlermeldung an und die Prozedur wird erneut ausgeführt. Schließt der Benutzer die Eingabe anders als mit  ab, wird durch die Funktion eine Leerzeichenfolge (" ") zurückgegeben, es sei denn, Sie haben mit *InitializeUserInput* die Eingabe von Nullzeichenfolgen unterdrückt (siehe Parameter).

```
Dim keywordList As String
keywordList = "Linie Punkt Text"
ThisDrawing.Utility.InitializeUserInput 128, keywordList
Dim Prompt As String
Dim StrVal As Integer
Prompt = "Ihrer Auswahl(Linie, Punkt, Text):"
StrVal = ThisDrawing.Utility.GetKeyword(Prompt)
```

Ermittlung einzelner Punktkoordinaten durch den Benutzer (GetPoint)

Diese Funktion wartet in der AutoCAD-Befehlszeile auf die Eingabe eines Punkts (2D/3D) durch den Benutzer.

```
Dim Prompt As String
Dim OutPnt As Variant
Prompt = "Legen Sie den Startpunkt fest:"
OutPnt = ThisDrawing.Utility.GetPoint(, Prompt)
```

Als ersten Parameter können Sie zusätzlich die Koordinaten eines Punkts mitgeben, den Sie als Array (*Point(2) As Double*) deklarieren. Sie legen damit den Startpunkt der zu bestimmenden Strecke bereits fest.

```
Dim Prompt As String
Dim OutPnt As Variant
Dim StartPoint(2) as Double
Prompt = "Legen Sie den Startpunkt fest:"
OutPnt = ThisDrawing.Utility.GetPoint(, Prompt)
StartPoint(0) = OutPnt(0)
StartPoint(1) = OutPnt(1)
StartPoint(2) = OutPnt(2)
Prompt = "Legen Sie den Zielpunkt fest:"
OutPnt = ThisDrawing.Utility.GetPoint(StartPoint, Prompt)
```

Nach erfolgreicher Ausführung erhalten Sie ein Array zurück, das drei Koordinaten (X, Y, Z) enthält.

Ganzzahlen des Benutzers abrufen (GetInteger)

Diese Funktion fordert den Benutzer auf, eine Ganzzahl in der AutoCAD-Befehlszeile einzugeben. Die Zahlen werden als Datentyp *Integer* behandelt und müssen demzufolge im Bereich von -32768 bis +32767 liegen.

```
Dim Prompt As String
Dim IntVal As Integer
Prompt = "Geben Sie eine Zahl ein:"
IntVal = ThisDrawing.Utility.GetInteger(Prompt)
```

Reelle Werte (Double) des Benutzers abrufen (GetReal)

Diese Funktion fordert den Benutzer auf, eine reelle Zahl in der AutoCAD-Befehlszeile einzugeben. Als Ergebnis wird eine Zahl vom Datentyp *Double* zurückgegeben.

```
Dim Prompt As String
Dim OutVal As Double
Prompt = "Geben Sie eine Zahl ein:"
IntVal = ThisDrawing.Utility.GetReal(Prompt)
```

Benutzerzeichenfolge abrufen (GetString)

Diese Funktion fordert den Benutzer auf, eine Zeichenfolge in der AutoCAD-Befehlszeile einzugeben. Als Ergebnis wird diese Zeichenfolge zurückgegeben. Die maximale Länge der einzugebenden Zeichenfolge liegt bei insgesamt 132 Zeichen. Sie können es dem Benutzer ermöglichen, Leerzeichen (Leertaste) in der Zeichenfolge zu verwenden, indem Sie für den ersten Parameter folgende Werte setzen.

- **TRUE** – Der Benutzer kann in der Zeichenfolge Leerzeichen (Space) eingeben.
- **FALSE** – Der Benutzer darf keine Leerzeichen verwenden.

```
Dim StrVal As String
Dim Prompt As String
Prompt = "Geben Sie einen Text ein:"
StrVal = ThisDrawing.Utility.GetString(True, Prompt)
```


Sie können auch die VBA-Anweisung `InputBox` verwenden, um eine Zeichenfolge vom Benutzer abzufragen. Dabei wird die Zeichenfolge nicht in der AutoCAD-Befehlszeile ermittelt. Vielmehr blendet die Anweisung ein Dialogfenster zur Abfrage ein.

```
Dim StrVal As String
Dim Prompt As String
Prompt = "Geben Sie einen Text ein:"
StrVal = InputBox(Prompt, "Text eingeben", StrVal)
```

Die Funktion `InputBox` ermöglicht folgende Parameter:

- *Prompt* (erforderlich) – Zeigt eine Zeichenfolge im Dialogfenster an.
- *Titel* (optional) – Legt den Titel des Fensters fest.
- *Default* (optional) – Eine Vorgabezeichenfolge.
- *XPos* (optional) – Legt den Abstand vom linken Rand fest.
- *YPos* (optional) – Legt den Abstand vom oberen Rand fest.
- *Helpfile* (optional) – Legt die Hilfedatei fest.
- *Context* (optional) – Legt die Hilfetextkennung fest.

Subelemente abrufen (`GetSubEntity`)

Die Funktion `GetSubEntity` ähnelt in ihrer Funktionsweise der Prozedur `GetEntity`. Allerdings können Sie mit dieser Prozedur auch Subelemente von AutoCAD-Objekten ermitteln lassen. Subelemente wären z.B. die Objekte einer Blockreferenz. Diese Funktion unterstützt dabei folgende AutoCAD-Objekte:

3DFace	3DPolylinie	3DSolid	Arc	Attribute
AttributRef	BlockRef	Circle	Dim3PointAngular	DimAligned
DimAngular	DimDiametric	DimOrdinate	DimRadial	DimRotated
Ellipse	ExternalReference	Hatch	Leader	LWPolyline
Line	MInstertBlock	MLine	MText	Point
PolyfaceMesh	PolygonMesh	Polyline	PViewport	Raster
Ray	Region	Shape	Solid	Spline
Text	Tolerance	Trace	XLine	

Außerdem gelten folgende Übergabeparameter:

- *Object* – Diese Variable gibt das gewählte Element oder Subelement als Objektvariable zurück.
- *PickedPoint* – Gibt den Punkt zurück, der bei der Auswahl des Objekts gewählt wurde.
- *TransMatrix* – Gibt die Transformationsmatrix des Elements als Array (4, 4) zurück.
- *ContextData* – Gibt eine Liste von *ObjectIDs* zurück, die in dem gewählten Objekt enthalten sind.
- *Prompt* (optional) – Legt eine Zeichenfolge fest, die in der Befehlszeile von AutoCAD angezeigt werden soll.

```

Dim Prompt As String
Dim OutEnt As Objekt
Dim OutPnt As Variant
Dim TransM As Variant
Dim CData As Variant
Prompt = "Wählen Sie ein AutoCAD-Element:"
ThisDrawing.Utility.GetSubEntity OutEnt, OutPnt, TransM, CData, Prompt
If VarType(CData) = vbEmpty Then
    MsgBox "Das gewählte Objekt enthält keine Subelemente", vbOkOnly
Else
    MsgBox "Das gewählte Objekt enthält Subelemente", vbOkOnly
End If

```

Wenn Sie in der Variablen *ContextData* eine Liste mit *ObjectIDs* erhalten, können Sie jede *ObjectID* mit der Funktion *ObjectIDToObject* in ein AutoCAD-Objekt umwandeln.

Um zu prüfen, welchem Elementtyp (Linien, Blockreferenz, Polylinie ...) das zurückgegebene Objekt entspricht, haben Sie zwei Möglichkeiten. Zum einen können Sie die Eigenschaft *ObjectName* des Objekts nutzen. Die zweite Möglichkeit ist die Prüfung über die VBA-Anweisung *TypeName()*. Diese Anweisung gibt dann als Ergebnis den Klassennamen des Objekts zurück. Auf diese Weise können Sie alle Klassennamen verwenden. Sie nehmen einfach den Namen der Klasse, wie er im Objektkatalog erscheint, und fügen zusätzlich ein „I“ vor der entsprechenden Bezeichnung ein.

```

Select Case TypeName(OutEnt)
Case "IAcadLine"
    'Das gewählte Objekt ist eine Linie
Case "IAcadBlockReference"
    'Das gewählte Objekt ist eine Blockreferenz
Case "IAcadCircle"
    'Das gewählte Objekt ist ein Kreis
End Select

```

Datei aus dem Netz herunterladen (GetRemoteFile)

Mit dieser Funktion kann der Benutzer eine Datei aus dem Netz herunterladen. Hierzu muss nur die entsprechende URL der Datei als erster Parameter sowie das Ziel (wohin die Datei gespeichert werden soll) als zweiter Parameter festgelegt werden. Mit dem dritten Parameter legen Sie nun noch fest, ob der Cache des Systems genutzt werden soll oder die Datei jedes Mal neu heruntergeladen wird.

- **FALSE** – Der Cache wird genutzt, die Datei kann im Cache zwischengespeichert werden.
- **TRUE** – Die Datei wird stets neu geladen.

```

Dim URL As String
GETURL:
URL = InputBox("Geben Sie die vollständige URL der Datei ein. " & _
    , "Download-URL", URL)
URL = Trim(URL)
If URL = "" Then Exit Sub
If StrComp(URL, "BROWSER", vbTextCompare) = 0 Then
    ThisDrawing.Utility.LaunchBrowserDialog URL, _
        "AutoCAD Browser", "Open", "http://www.autodesk.com", _
        "ACADBROWSER", True

```

```

GoTo GETURL
End If
If Not (ThisDrawing.Utility.IsURL(URL)) Then
    MsgBox "Sie haben keine gültige Adresse eingegeben."
    GoTo GETURL
End If
ThisDrawing.Utility.GetRemoteFile URL, DestFile, True
MsgBox URL & " wurde nach: " & DestFile & " heruntergeladen", vbOkOnly

```

Dateien ins Netz übertragen (PutRemoteFile)

Diese Funktion ist das Gegenstück zur Prozedur *GetRemoteFile*. Der Benutzer kann eine Datei auf einem Remote-Computer ablegen. Hierzu muss nur die entsprechende URL angegeben werden, unter der die Datei abgelegt werden soll, sowie die Datei selbst.

```

Dim DURL As String, LFile As String
DURL = ftp://www.meinserver.de/dateien/
LFile = "c:\Daten\autocad\projekte\dokument.dwg"
ThisDrawing.Utility.PutRemoteFile DURL, LFile
MsgBox "Datei: " & LFile & " wurde nach: " & DURL & " übertragen"

```

Die Funktionen IsRemoteFile und IsURL

Diese beiden Funktionen haben im Einzelnen folgende Aufgabe: Die erste Funktion *IsRemoteFile* soll prüfen, ob die angegebene Datei heruntergeladen wurde. Ist dies der Fall, wird die entsprechende URL zurückgegeben. Die Funktion kann folgende Ergebnisse liefern:

- **TRUE** – Die angegebene Datei wurde heruntergeladen.
- **FALSE** – Die angegebene Datei hat kein URL-Gegenstück.

Die Funktion *IsURL* ist das Gegenstück zu *IsRemoteFile*. Soll heißen, die Funktion prüft die angegebene URL auf Gültigkeit.

- **TRUE** – Die angegebene URL ist gültig.
- **FALSE** – Die angegebene URL ist ungültig.

Webbrowser starten (LaunchBrowserDialog)

Über diese Funktion ermöglichen Sie es dem Benutzer, das Dialogfenster des Webbrowsers zu starten. Der Benutzer kann sich dann zu jedem beliebigen URL bewegen und eine URL auswählen. Außerdem muss diese Funktion vor den Prozeduren *GetRemoteFile*, *PutRemoteFile*, *IsRemoteFile* und *IsURL* gestartet werden.

```

Dim URL As String
URL = InputBox("Geben Sie eine Adresse ein.", "Adresse", URL)
URL = Trim(URL)
If URL = "" Then Exit Sub
ThisDrawing.Utility.LaunchBrowserDialog URL, "AutoCAD Browser", _
    "Open", "http://www.autodesk.com", "ACADBROWSER", True

```

Die Funktion hat folgende Parameter:

- **SelectedURL** – Legt die ausgewählte URL fest.
- **DialogTitel** – Legt den Titel des Dialogfensters fest.

- *OpenButtonCaption* – Legt den Text der Schaltfläche OK/ÖFFNEN fest.
- *StartPageURL* – Legt die Startseite des Browsers fest.
- *RegistryRootKey* – Legt den Root-Namen fest, unter dem Browserinformationen in der Registrierung gespeichert werden sollen. Solche Informationen wären Größe, Position und andere Dialogeinstellungen. Wenn Sie eine Leerzeichenfolge übergeben, wird keine Speicherung vorgenommen.
- *OpenButtonAlwaysEnabled* – Wenn Sie für diesen Parameter *TRUE* übergeben, wird die Schaltfläche ÖFFNEN aktiviert. Ist der Parameter *FALSE* wird die Schaltfläche deaktiviert.

Als Ergebnis der Funktion erhalten Sie entweder *TRUE*, dann wurde die Funktion erfolgreich ausgeführt. Andernfalls erhalten Sie den Wert *FALSE* und die Funktion wurde nicht erfolgreich ausgeführt.

Koordinaten aus polaren Werten berechnen (PolarPoint)

Mithilfe der Funktion *PolarPoint* können Sie die Koordinaten eines Punkts aus einem Winkel und einer Strecke, ausgehend von einem Stützpunkt, berechnen lassen. Der Stützpunkt wird als Array (*Point(2) As Double*), der Winkel und die Strecke als *Double* übergeben. Als Ergebnis liefert die Funktion ein 3D-Array des berechneten Punkts (X, Y, Z).

```
Dim OutPnt As Variant
Dim StartPnt(2) As Double
Dim InAngle As Double
Dim InDist As Double
InAngle = 0
InDist = 100
OutPnt = ThisDrawing.Utility.PolarPoint(StartPnt, InAngle, InDist)
```

Sie können den Winkel und die Strecke ebenfalls durch AutoCAD-Funktionen bestimmen lassen und diese dann mit der Funktion *PolarPoint* verschachteln.

```
OutPnt = ThisDrawing.Utility.PolarPoint(StartPnt, _
ThisDrawing.Utility.AngleFromXAxis(Point1, Point2), InDist)
```

Zeichenfolgen in der AutoCAD-Befehlszeile ausgeben (Prompt)

Dies ist eine recht einfache Funktion. Sie hat lediglich die Aufgabe, eine Zeichenfolge in der AutoCAD-Befehlszeile auszugeben. Als zusätzliche Information kann ich Ihnen eigentlich nur Folgendes mit auf den Weg geben. Verwenden Sie die Konstanten *vbCr*, *vbCrLf*, *vbNewLine*, *vbBack* und *vbTab* für eine weitere Formatierung des Ausgabestrings. Wollen Sie zum Beispiel, dass die Zeile wieder an der Position der letzten Ausgabe beginnt, so dass die letzte Zeile von der neuen überlagert wird, stellen Sie dem Prompt ein *vbCr* voran. Die folgenden Schleifen haben in AutoCAD alle eine andere Wirkung. Aber testen Sie es doch einmal selbst.

```
'Alle Werte hintereinander
For i = 0 To 1000
    ThisDrawing.Utility.Prompt "Zeile " & CStr(i)
Next
,Alle Werte übereinander
For i = 0 To 1000
    ThisDrawing.Utility.Prompt vbCrLf & "Zeile " & CStr(i)
Next
```

```
'Alle Werte untereinander
For i = 0 To 1000
    ThisDrawing.Utility.Prompt vbCrLf & "Zeile " & CStr(i)
NextDie
```

Dezimalzahlen in Zeichenfolgen umwandeln (RealToString)

Diese Funktion wandelt eine Dezimalzahl, die als erster Parameter mitgegeben wird, in eine Zeichenfolge um. Als zweiten Parameter legen Sie die Einheit des Ausgabewerts fest. Zu diesem Zweck können Sie die Systemvariable *LUPREC* verwenden, die die aktuell eingestellte Dezimaleinheit enthält. Benutzen Sie hierzu die Funktion *ThisDrawing.GetVariable*.

Da der Parameter bereits eine Auflistung von Konstanten enthält, können Sie auch eine der entsprechenden Konstanten wählen.

Konstante	Beschreibung	Wert
<i>acDefaultUnits</i>	Die im System eingestellte Einheit	-1
<i>acScientific</i>	Wissenschaftlich	1
<i>acDecimal</i>	Dezimal	2
<i>acEngineering</i>	Engineering	3
<i>acArchitectural</i>	Architectural	4
<i>AcFractional</i>	Bruch	5

Tabelle 6.4: Konstanten der Dezimaleinheiten

Zusätzlich enthält diese Funktion einen dritten Parameter, der die Genauigkeit (Anzahl der Nachkommastellen) festlegt. Auch hierfür gibt es eine Systemvariable. *LUPREC* liefert die aktuell eingestellte Dezimalgenauigkeit.

```
Dim InVal As Double
Dim OutVal As String
InVal = 100.000
OutVal = ThisDrawing.Utility.RealToString(InVal, acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
```

Koordinatenumrechnung (TranslateCoordinates)

Diese Prozedur wandelt Koordinaten von einem System (Quelle) in ein anderes System (Ziel) um. Es kommt recht häufig vor, dass Sie Koordinaten im WKS haben und diese auf das aktuelle BKS umrechnen oder WKS-Koordinaten in den Papierbereich übertragen müssen. Zu diesem Zweck wurde diese Funktion entwickelt. Sie beinhaltet insgesamt fünf Parameter mit folgender Bedeutung: Der erste Parameter enthält die Koordinaten (X, Y, Z) des Ausgangssystems. Dieser wird, wie immer, als Array (*Point(2) As Double*) an die Funktion übergeben. Der zweite und der dritte Parameter geben jeweils das Quellsystem und das Zielsystem an. Sie können dabei folgende Konstanten verwenden.

Konstante	Beschreibung	Wert
<i>acWorld</i>	Weltkoordinatensystem	0
<i>acUCS</i>	Benutzerkoordinatensystem	1

Tabelle 6.5: Konstanten der Koordinatensystemdefinition

Konstante	Beschreibung	Wert
<i>acDisplayDCS</i>	Aktuelles Koordinatensystem im Modellbereich	2
<i>acPaperSpaceDCS</i>	Koordinatensystem des Papierbereichs	3
<i>acOCS</i>	Objektkoordinatensystem	4

Tabelle 6.5: Konstanten der Koordinatensystemdefinition (Forts.)

Im vierten Parameter können Sie noch festlegen, ob die Ausgangsdaten (Punkt im Quellsystem) als Punkt oder als Vektorkoordinaten interpretiert werden sollen.

- *TRUE* – Ursprungspunkt wird als Vektor interpretiert.
- *FALSE* – Ursprungspunkt wird als Punkt interpretiert.

Der fünfte Parameter kann, muss aber nicht mitgegeben werden. Er stellt die dreidimensionalen normalen Einheitenvektor dar.

Als erstes Beispiel für diese Funktion übertragen wir einen Punkt vom WKS in den Papierbereich.

```
Dim AcPnt(2) As Double
Dim Cpoint As Variant
ThisDrawing.MSpace = True
AcPnt(0) = 1000
AcPnt(1) = 1000
AcPnt(2) = 0
CPoint = ThisDrawing.Utility.TranslateCoordinates(AcPnt, acWorld, acDisplayDCS, False)
AcPnt(0) = CPoint(0)
AcPnt(1) = CPoint(1)
AcPnt(2) = CPoint(2)
CPoint = ThisDrawing.Utility.TranslateCoordinates(AcPnt, _
    acDisplayDCS, acPaperSpaceDCS, False)
```

Im zweiten Beispiel werden die Koordinaten, die im WKS vorliegen, in das aktuelle BKS übertragen.

```
Dim VarPoint As Variant
Dim InsPnt(2) As Double
VarPoint = ThisDrawing.Utility.GetPoint(, "wählen Sie einen Punkt:")
InsPnt(0) = VarPoint(0)
InsPnt(1) = VarPoint(1)
InsPnt(2) = VarPoint(2)
VarPoint = ThisDrawing.Utility.TranslateCoordinates(InsPnt, acWorld, acUCS, False)
InsPnt(0) = VarPoint(0)
InsPnt(1) = VarPoint(1)
InsPnt(2) = VarPoint(2)
```

Sie können keine Objektkoordinaten direkt ineinander umwandeln (OCS → OCS). Gehen Sie hierfür den Umweg über das WKS.

1. Lesen Sie die X- und Y-Koordinaten des Objekts aus.
2. Legen Sie für die Z-Koordinate die Erhebung des Objekts fest.
3. Ermitteln Sie den dreidimensionalen normalen Einheitenvektor des Objekts (z.B. Parameter *Normal* einer Polylinie).

6.4 Der Modellbereich/Papierbereich

Sowohl der Modellbereich als auch der Papierbereich dienen als Container grafischer AutoCAD-Objekte, wie z.B. Linien, Polylinien oder auch komplexe Objekte wie Volumenkörper. Für den Papierbereich ist es zwar nicht allzu sinnvoll, 3D-Objekte hinzuzufügen, aber durchaus möglich. Wie unterscheiden sich nun diese beiden Bereiche? Nun vom Modellbereich gibt es nur einen in einer AutoCAD-Zeichnung. Dieser enthält alle Daten, auf die Sie in einer Zeichnung zugreifen können, wenn Sie in den Registereintrag *Modell* umschalten.



Abbildung 6.2: Register Modellbereich

Vom Papierbereich dagegen kann es mehrere in einer Zeichnung geben. Jedes Papierlayout stellt dem Objekt *PaperSpace* andere Daten zur Verfügung. Sie müssen also darauf achten, welches Layout aktuell eingestellt wurde, bevor Sie auf die Daten des Papierbereichs zugreifen.

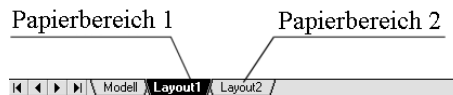


Abbildung 6.3: Register der Papierbereiche

Wenn nun gerade der Modellbereich das aktive Layout stellt, dann enthält das *PaperSpace*-Objekt die Daten des zuletzt gewählten Papierlayouts zur Verfügung. Sollte noch kein Papierlayout direkt in der Zeichnungssitzung gewählt worden sein, so verweist das *PaperSpace*-Objekt auf das erste Papierlayout.

6.4.1 Zugriff auf den Modell-/Papierbereich

Um auf die gesamte dreidimensionale Geometrie des Modellbereichs zugreifen zu können, stellt jede Zeichnungsklasse die Objekteigenschaft *ModelSpace* zur Verfügung. Diese entspricht dem Datentyp *AcadModelSpace* und enthält alle Eigenschaften und Methoden, um grafische Objekte der Zeichnung hinzuzufügen oder bereits vorhandene Objekte abzurufen. Der Aufbau entspricht in etwa der eines Blocks, was auch logisch ist, da der Modellbereich einer Zeichnung zum Datencontainer eines Blocks wird, wenn die Zeichnung als Block in eine andere Zeichnung eingefügt wird. Wenn Sie auf den Modellbereich der aktuellen Zeichnung zugreifen wollen, können Sie dies mit folgender Codezeile tun:

```
Dim MoSpace As AcadModelSpace
Set MoSpace = ThisDrawing.ModelSpace
```

Der Zugriff auf den Modellbereich eines bestimmten Dokuments der Dokumentauflistung funktioniert folgendermaßen:

```
Dim MoSpace As AcadModelSpace
Set MoSpace = Documents(n).ModelSpace
```

Sie brauchen auch hier nicht unbedingt einen Verweis auf den Modellbereich zu erstellen, sondern können auch direkt mit der Objektvariablen arbeiten.

```
MsgBox ThisDrawing.ModelSpace.Count & "Elemente im Modellbereich"
```

Für den Zugriff auf den Papierbereich trifft die gleiche Vorgehensweise zu, nur dass Sie einige Punkte beachten müssen. Da eine Zeichnung nicht nur einen Papierbereich, sondern eine Vielzahl davon enthalten kann, müssen Sie darauf achten, auf welchen Sie gerade zugreifen, um nicht die Daten in einem falschen Bereich abzulegen. Entscheidend dafür, welcher Papierbereich gerade aktiv ist, ist die Einstellung der Layouts. Jedes Papierlayout speichert seine eigenen Papierbereichsdaten.

```
Dim PaSpace As AcadPaperSpace
Set PaSpace = ThisDrawing.PaperSpace
```

Der Zugriff auf den Modellbereich eines bestimmten Dokuments der Dokumentauflistung funktioniert folgendermaßen:

```
Dim PaSpace As AcadPaperSpace
Set PaSpace = Documents(n).PaperSpace
```

Egal wie, ob indirekt über einen Verweis oder direkt über die Eigenschaft des Zeichnungsobjekts, Sie können nun über diese Objekte auf die Daten des Modell- oder Papierbereichs zugreifen oder grafische Objekte hinzufügen. Dazu stellen die Objekte *ModelSpace* und *PaperSpace* verschiedene Eigenschaften und Methoden zur Verfügung.

6.4.2 Grafische Objekte im Modell-/Papierbereich erstellen

Um grafische Objekte in den Modell- bzw. Papierbereich einzufügen, stellen die Klassen *AcadModelSpace* für den Modell- und *AcadPaperSpace* für den Papierbereich die verschiedensten *Add*-Funktionen zur Verfügung. Jede dieser Funktionen erstellt dabei jeweils ein Objekt, (bis auf die Funktion *AddRegion*, die in dieser Hinsicht einen Sonderfall darstellt) das durch eine ganz bestimmte Klasse definiert wird. Der Aufruf der entsprechenden *Add*-Funktion verläuft in etwa immer auf die gleiche Art und Weise. Zunächst werden die Parameter, die die entsprechende *Add*-Funktion zur Erstellung des grafischen Objekts benötigt, zusammengestellt und dann zusammen mit der Funktion aufgerufen. Als Ergebnis können Sie sich dann auch eine Objektvariable mit einem Verweis auf das erstellte AutoCAD-Objekt zurückgeben lassen. Wenn Sie im Objektkatalog einmal den Definitionsbereich der Klasse *AcadModelSpace* einblenden oder auch den der Klasse *AcadPaperSpace*, ganz wie Sie wollen, erhalten Sie einen Überblick über alle *Add*-Funktionen sowie deren Parameter.

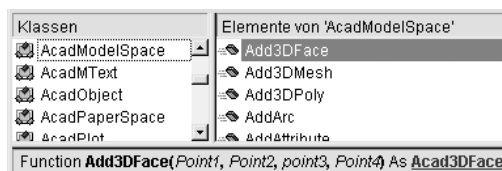


Abbildung 6.4: Ausschnitt der Klasse *AcadModelSpace*

Sie erhalten die Funktionen dieser Klasse auch, wenn Sie im Codefenster die Zeile *ThisDrawing.ModelSpace*. eingeben.

Als erstes kleines Beispiel, anhand dessen Sie bereits erkennen können, wie die *Add*-Funktionen des Modell- und Papierbereichs im Allgemeinen gehandhabt werden, soll uns die Erzeugung einer einfachen Linie dienen. Erstellen Sie zu diesem Zweck eine Prozedur im Klassenmodul *ThisDrawing*. Die nachfolgenden Zeilen müssen Sie dann in den neuen *Sub – End Sub*-Bereich einfügen.

Zu Beginn erfolgt wie immer die Deklaration der einzelnen verwendeten Variablen:

```
Dim NewLine As AcadLine
Dim StartPt(2) As Double
Dim EndPt(2) As Double
```

In der Variablen *NewLine* werden wir das Ergebnis der Erstellungsfunktion speichern. Es handelt sich dabei um eine Objektvariable vom Typ *AcadLine*, da dieser Objekttyp von der Funktion *AddLine* zurückgegeben wird. Sie können auch Objektvariablen vom Typ *Object*, *AcadObject*, *AcadEntity* oder *Variant* verwenden, da diese Objekttypen der Klasse *AcadLine* sozusagen übergeordnet sind. Abschließend werden noch die beiden Arrays *StartPt* und *EndPt* deklariert. Diese beiden Arrays sollen als Übergabeparameter für den Start- sowie den Endpunkt der neuen Linie dienen, die ja zwingend erforderlich sind. Da sowohl der Start- als auch der Endpunkt einer Linie einer dreidimensionalen Koordinate entspricht, müssen diese beiden Übergabevariablen als dreidimensionale Arrays definiert werden. Diesen Variablen werden nun noch die jeweiligen X-, Y- und Z-Werte zugewiesen:

```
StartPt(0) = 0
StartPt(1) = 0
StartPt(2) = 0
EndPt(0) = 100
EndPt(1) = 100
EndPt(2) = 0
```

Abschließend wird dann nur noch die Linie im Modellbereich mit den angegebenen Koordinaten erzeugt und ein Verweis auf die erstellte Linie an die Objektvariable *NewLine* übergeben.

```
Set NewLine = ModelSpace.AddLine(StartPt, EndPoint)
```

Sie können nun mit der Variablen *NewLine* weiterarbeiten, um z.B. den Layer oder den Linientyp der neuen Linie anzupassen.

Ein wenig komplexer gestaltet sich dagegen die Vorgehensweise bei der Erstellung eines einzelnen Textes. Schauen wir uns dies im nächsten Beispiel einmal an.

Beginnen Sie eine neu eingefügte Prozedur mit der Deklaration der einzelnen verwendeten Variablen:

```
Dim NewText As AcadText
Dim InsPt(2) As Double
Dim Height As Double
Dim TextString As String
```

Die Objektvariable *NewText* soll uns wiederum als Ergebnisvariable für die Funktion *AddText* dienen, mit der später der Text in den Modellbereich eingefügt werden soll. Die restlichen drei Variablen werden wiederum als Übergabeparameter, die ja auch wieder erforderlich sind, in den entsprechenden Datentypen deklariert.

Die Variable *TextString* soll die Zeichenfolge des neuen Textes beinhalten und an die Funktion *AddText* weitergeben. Diese Zeichenfolge könnte dabei mit folgendem Code abgefragt werden:

```
TextString = InputBox("Textzeichenfolge eingeben:", "Neuer Text")
If TextString = "" Then Exit Sub
```

Mithilfe dieser Funktion kann die Zeichenfolge des neuen Textes über ein Dialogfenster festgelegt werden. Sie können aber auch andere Funktionen wie z.B. die *GetString*-Funktion der *Utility*-Klasse verwenden, um eine entsprechende Zeichenfolge abzufragen.

Als Nächstes werden noch die Koordinaten des Einfügekpunkts an das Array *InsPoint* und die Texthöhe an die Variable *Height* übergeben:

```
InsPoint(0) = 10
InsPoint(1) = 10
InsPoint(2) = 0
Height = GetVariable("TextSize")
```

Die Texthöhe kann zu diesem Zweck aus der AutoCAD-Systemvariablen *TEXTSIZE* übernommen werden. Sie können hierfür die Funktion *GetVariable* auf das jeweilige AutoCAD-Dokument, in diesem Fall das aktuelle Dokument, anwenden.

Nachdem alle notwendigen Parametervariablen mit den vorgesehenen Werten gefüllt wurden, kann der Text mit folgender Funktion im Modellbereich erzeugt werden:

```
Set NewText = ModelSpace.AddText(TextString, InsPoint, Height)
```

Bevor wir uns nun den einzelnen *Add*-Funktionen der Klassen *AcadModelSpace* und *AcadPaperSpace* zuwenden, als drittes und abschließendes Beispiel noch eine Prozedur zur Erstellung einer 3D-Polylinie. Legen Sie zu diesem Zweck wiederum eine Prozedur im Klassenmodul *ThisDrawing* an und fügen Sie den nachfolgend beschriebenen Code der neuen Prozedur hinzu.

Als Erstes folgt wieder die Deklaration der einzelnen Variablen. Für die Erstellung einer 3D-Polylinie wird ein Array von 3D-Koordinaten benötigt. Dieses Array kann fest oder dynamisch deklariert werden. Wenn Sie das Array mit einer festen Anzahl an Koordinaten deklarieren möchten, benutzen Sie hierzu die folgende Codezeile:

```
Dim VertList(11) As Double
```

oder

```
Dim VertList(0 To 11) As Double
```

In diesem Fall legen wir einfach eine Anzahl von vier Stützpunkten für die zu erzeugende 3D-Polylinie fest. Daraus ergibt sich eine Gesamtzahl von zwölf Einzelkoordinaten (vier Stützpunkte * drei Koordinaten pro Stützpunkt X, Y, Z). Da das Array mit dem Index 0 beginnt und dieser die erste Einzelkoordinate (X des ersten Stützpunkts) beinhaltet, wird die Variable in einem Bereich von 0 bis 11 definiert.

Eine zweite Möglichkeit der Dimensionierung des Koordinatenarrays wäre die dynamische Deklaration. Diese Art der Deklaration kommt dann in Betracht, wenn Sie die Koordinaten vor Erstellung der 3D-Polylinie bestimmen. Die Koordinaten könnten mittels einer Variablen an die Prozedur übergeben worden sein:

```
Public Sub Create3DPoly(CoordList)
ReDim VertList(UBound(CoordList)) As Double
```

oder:

```
ReDim VertList(0 To UBound(CoordList)) As Double
```

In beiden Fällen wird die Obergrenze des Arrays *CoordList* mithilfe der Funktion *UBound()* ermittelt und als Obergrenze für die neue Variable *VertList* festgelegt. Diese Variable wird anschließend mit der Funktion *Add3DPoly* weiterverwendet. Nun könnten Sie zu Recht fragen: „Warum übergeben wir die Koordinaten der Variablen *CoordList* nicht direkt an die Funktion?“ Nun dies funktioniert so nicht, da die Variable *CoordList* vom Typ *Variant* ist und die Funktion *Add3DPoly* eine Variable vom Typ *Double* als Übergabeparameter erwartet.

Eine andere Möglichkeit der Koordinatenfestlegung wäre die Ermittlung in der Prozedur selbst. Zu diesem Zweck bedienen wir uns der Funktion *GetPoint* der *Utility*-Klasse von AutoCAD. Bevor wir aber mit der eigentlichen Koordinatenbestimmung beginnen, sollten noch folgende Variablendeklarationen dem Code vorangestellt werden:

```
Dim varPoint As Variant
Dim LastPt(2) As Double
Dim VertList
Dim Prompt As String
```

Was danach folgt, ist der eigentliche Code der Koordinatenabfrage für die neue 3D-Polylinie. Um einen ungewollten Programmabbruch während der Programmausführung zu vermeiden, sollten Sie dem Code noch folgende Zeile voranstellen.

```
On Local Error Resume Next
```

Diese Zeile wird verhindern, dass das Programm abbricht, wenn Sie die rechte Maustaste betätigen, da AutoCAD bei dieser Aktion einen Fehler („Benutzereingabe ist ein Schlüsselwort“) auslöst. Was dieser Fehler, der im eigentlichen Sinne ja kein Fehler, sondern ein Hinweis sein soll, bedeutet, können Sie im Kapitel 6.3 nachlesen. Kommen wir aber nun zum Listing über die Koordinatenwahl für die 3D-Polylinie:

```
Do
    VarPoint = Empty
    If TypeName(vertList) = "Empty" Then
        Prompt = "Startpunkt:"
        VarPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Else
        Prompt = "nächster Punkt:"
        VarPoint = ThisDrawing.Utility.GetPoint(LastPt, Prompt)
    End If
    If TypeName(VarPoint) = "Double()" Then
        If TypeName(vertList) = "Empty" Then
            ReDim vertList(2) As Double
        Else
            ReDim Preserve vertList(UBound(vertList) + 3) As Double
        End If
        vertList(UBound(vertList) - 2) = VarPoint(0)
        vertList(UBound(vertList) - 1) = VarPoint(1)
        vertList(UBound(vertList)) = VarPoint(2)
        LastPt(0) = VarPoint(0)
        LastPt(1) = VarPoint(1)
        LastPt(2) = VarPoint(2)
```

```

Else
    Exit Do
End If
Loop

```

Der gesamte Code der Koordinatenwahl läuft in einer *Do...Loop*-Schleife ab, und zwar so lange, bis der Benutzer keine Koordinate mehr auswählt. Wenn Sie diesen Code weiterverwenden wollen, sollten Sie noch einige Ergänzungen in Bezug auf die Kriterien zum Verlassen der Schleife vornehmen. Bestimmte Funktionen in AutoCAD, z.B. das Aufrufen von transparenten Befehlen, die nicht über die AutoCAD-Befehlszeile ausgeführt werden, können zu ungewollten Abbrüchen der Funktion *GetPoint* führen.

Zu Beginn der Schleife wird mithilfe der Funktion *TypeName()* geprüft, ob das Array *vertList*, das ja die Koordinaten der 3D-Polylinie aufnehmen soll, bereits ein Koordinatenpaar enthält. Ist dies nicht der Fall, so gibt die Funktion *TypeName()* als Ergebnis den Wert „Empty“ zurück und der erste Punkt der 3D-Polylinie wird abgefragt. Ansonsten wird immer der nächste Stützpunkt abgefragt und vom letzten gewählten Punkt wird ein Gummiband zum Cursor gezogen. Nachdem der Benutzer eine gültige Koordinate ausgewählt hat, wird dies wiederum mit der Funktion *TypeName* geprüft. Enthält die Variable *VarPoint* eine Koordinate, so wird als Ergebnis der Prüfung „Double()“ zurückgegeben und der Code wird mit den folgenden Zeilen fortgesetzt:

```

If TypeName(vertList) = "Empty" Then
    ReDim vertList(2) As Double
Else
    ReDim Preserve vertList(UBound(vertList) + 3) As Double
End If

```

In diesen Codezeilen wird geprüft, wie bereits weiter oben beschrieben, ob bereits Koordinaten gewählt wurden. Ist die Koordinatenliste leer, wird sie als dreidimensionales Array (0-2) initialisiert:

```
ReDim vertList(2) As Double
```

Im anderen Fall wird folgende Codezeile ausgeführt:

```
ReDim Preserve vertList(UBound(vertList) + 3) As Double
```

Mit dem Schlüsselwort wird die Deklarationsanweisung angewiesen, alle bereits dem Array hinzugefügten Koordinaten in der Liste unberührt zu lassen und nur die obere Grenze des Arrays neu zu setzen. Abschließend werden noch die ermittelten Koordinaten an das Array angehängt und als letzter ermittelter Stützpunkt gesetzt.

Nachdem die Koordinaten der Stützpunkte für die neue 3D-Polylinie ermittelt wurden, bleibt nur noch die Erzeugung der Polylinie im Modellbereich. Dies wird mit folgenden Codezeilen erledigt:

```

If TypeName(VertList) = "Double()" Then
    If UBound(VertList) > 2 Then
        Set NewPoly = ThisDrawing.Add3DPoly(VertList)
    End If
End If

```

Hier wird zunächst geprüft ob das Array *VertList* überhaupt Koordinaten enthält. Da eine Polylinie ja mindestens zwei Stützpunkte braucht, benötigen wir noch eine Abfrage, die nur dann das Erzeugen der Polylinie zulässt, wenn das Array mindestens diese zwei Stützpunkte enthält. Diese Aufgabe übernimmt die Funktion *UBound()*, die die obere Grenze des Arrays zurückgibt. Enthält das Array also mehr als einen Stützpunkt, ist die obere Grenze immer größer 2.

Nachdem ich mit einigen Beispielen das Grundprinzip der *Add*-Funktionen erläutert habe, kommen wir abschließend noch zu zwei Funktionen, die ebenfalls Objekte in den Modellbereich der Zeichnung einfügen, sich aber in der Syntax von den anderen *Add*-Funktionen unterscheiden. Zum einen gibt es die Funktion zum Einfügen einer Blockreferenz oder einer Zeichnungsdatei in den Modellbereich. Hierbei handelt es sich um die Funktion *InsertBlock*. Schauen wir uns anhand eines Beispiels einmal an, wie diese Funktion verwendet werden kann. Nach der Deklaration der einzelnen Variable

```
Dim InsPoint(2) As Double
Dim BlkName As String
Dim XS As Double, YS As Double, ZS As Double
Dim Rotation As Double
Dim VarPoint As Variant
```

werden zunächst die einzelnen Parameter, die für die Funktion verwendet werden, abgefragt bzw. festgelegt. Der Blockname kann dabei wieder mithilfe der *InputBox*-Funktion abgefragt werden. Der Benutzer kann dabei einen Blocknamen, der bereits in der Zeichnung definiert wurde, oder einen Zeichnungsnamen samt Verzeichnis eingeben.

```
BlkName = InputBox("Block- oder Zeichnungsname", "Einfügen")
If BlkName = "" Then Exit Sub
```

Gibt der Benutzer einen Namen mit Verzeichnis ein, wird der entsprechende Block in die Zeichnung geladen. Ist der Block bereits vorhanden, wird er durch die externe Zeichnung neu definiert. Als Nächstes wird der Einfügepunkt der Blockreferenz ermittelt.

```
VarPoint = ThisDrawing.Utility.GetPoint(, "Einfügepunkt wählen:")
If TypeName(VarPoint) = "Empty" Then Exit Sub
InsPoint(0) = VarPoint(0)
InsPoint(1) = VarPoint(1)
InsPoint(2) = VarPoint(2)
```

Alle anderen Parameter können wir in diesem Fall einfach festlegen.

```
XS = 1: YS = 1: ZS = 1
Rotation = 0
```

Die Variablen *XS*, *YS* und *ZS* legen die jeweiligen Skalierfaktoren in den entsprechenden Richtungen fest. Die Variable *Rotation* bestimmt den Drehwinkel des einzufügenden Blocks. Nachdem die Vorarbeit erledigt wurde, kann die Blockreferenz mit der Funktion *InsertBlock* in den Modellbereich eingefügt werden.

```
Set NewBlk = ThisDrawing.ModelSpace.InsertBlock(InsPoint, _
    BlkName, XS, YS, ZS, Rotation)
```

Als Ergebnis der Einfügung wird die Blockreferenz in die Objektvariable *NewBlk* übergeben, so dass Sie die Eigenschaften der Blockreferenz weiterbearbeiten können. Was Sie alles mit solch einer Blockreferenz anstellen können, erfahren Sie im Abschnitt über die Blockreferenzen.

Als letztes Beispiel in diesem Abschnitt betrachten wir noch die Funktion *AttachExternalReference*, mit der externe Referenzen in die Zeichnung eingebunden werden können. Diese Funktion benötigt eine Vielzahl von Parametern, die zunächst wieder deklariert werden.

```
Dim PathName As String
Dim XName As String
Dim XS As Double, YS As Double, ZS As Double
Dim Rotation As Double
Dim VarPoint As Variant
```

Füllen wir diese Variable nun wieder mit den entsprechenden Werten. Als Erstes wird der Zeichnungsname abgefragt.

```
PathName = InputBox("Zeichnungsname", "XRef einfügen")
If PathName = "" Then Exit Sub
```

Der Benutzer muss hier den vollständigen Namen der Zeichnung, also inklusive Verzeichnis und Dateierweiterung, eingeben. Als zweiter Parameter wird der eigentliche Name der externen Referenz festgelegt. Dieser kann, genau wie der erste Parameter, mithilfe der Funktion *InputBox* ermittelt werden.

```
XName = InputBox("XRefname", "XRef einfügen")
If XName = "" Then Exit Sub
```

Als Nächstes wird wieder der Einfügepunkt für die externe Referenz ermittelt. Hierzu nutzen wir wieder die *Utility*-Funktion *GetPoint*:

```
VarPoint = ThisDrawing.Utility.GetPoint(, "Einfügepunkt wählen:")
If TypeName(VarPoint) = "Empty" Then Exit Sub
InsPoint(0) = VarPoint(0)
InsPoint(1) = VarPoint(1)
InsPoint(2) = VarPoint(2)
```

Allen anderen Parametern werden wieder feste Werte zugeordnet:

```
XS = 1: YS = 1: ZS = 1
Rotation = 0
```

Zum Ende dieser Prozedur kann die externe Referenz mit folgender Codezeile eingefügt werden.

```
Set XR = ModelSpace.AttachExternalReference(PathName, _
    XName, InsPoint, XS, YS, ZS, Rotation, True)
```

Damit wäre die externe Referenz in der aktuellen Zeichnung eingefügt.

3D-Flächen erstellen

Um eine 3D-Fläche zu erstellen, benötigen wir zunächst die vier Kontrollpunkte der neuen 3D-Fläche. Diese könnten z.B. über die *GetPoint*-Funktion des *Utility*-Objekts ermittelt werden. Die ermittelten Eckpunkte, jeder entspricht dabei einem 3D-Array, werden jeder für sich als Parameter der Funktion hinzugefügt.

Add3DFace

Parameter: *Point1*, *Point2*, *Point3*, *Point4*

Rückgabotyp: *Acad3DFace*

```
Public Sub Sample_Add3DFace()
Dim New3DF As Acad3DFace
Point1 = ThisDrawing.Utility.GetPoint(, "erster Punkt:")
Point2 = ThisDrawing.Utility.GetPoint(Point1, "zweiter Punkt:")
Point3 = ThisDrawing.Utility.GetPoint(Point2, "dritter Punkt:")
Point4 = ThisDrawing.Utility.GetPoint(Point3, "vierter Punkt:")
Set New3DF = ThisDrawing.ModelSpace.Add3DFace(Point1, Point2, _
    Point3, Point4)
End Sub
```

Siehe auch *Kapitel 8.3.3*

3D-Netz erstellen

Da ein 3D-Netz aus einer bestimmten Anzahl an Feldern besteht, verlangt diese Funktion einige Parameter, die die Definition der einzelnen Kontrollpunkte bestimmen. Der erste Parameter bestimmt die Anzahl der Kontrollpunkte in M-Richtung, der zweite die der Kontrollpunkte in N-Richtung. Die Anzahl der Kontrollpunkte muss sowohl in M- als auch in N-Richtung im Bereich zwischen 2 und 256 liegen. Die einzelnen Koordinaten werden in einem Array gespeichert, das folgendermaßen deklariert ist:

```
Dim PointsMatrix((M * N * 3) - 1) As Double
```

Dieses Array bildet schließlich den dritten Parameter der *Add*-Funktion.

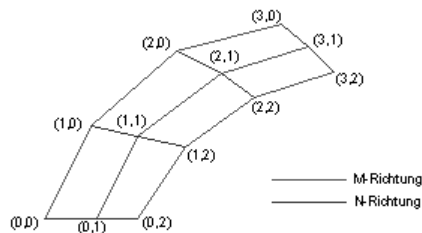


Abbildung 6.5: 3D-Netz

Add3DMesh

Parameter: *M As Long*, *N As Long*, *PointsMatrix*

Rückgabotyp: *AcadPolygonMesh*

```
Public Sub Sample_Add3DMesh()
Dim New3DM As AcadPolygonMesh
Dim M As Integer, N As Integer
Dim PtnMatrix, Prompt As String
On Local Error Resume Next
M = ThisDrawing.Utility.GetInteger("Anzahl der Punkte in M:")
```

```

N = ThisDrawing.Utility.GetInteger("Anzahl der Punkte in N:")
If N < 2 Or N > 256 Or M < 2 Or M > 256 Then
    MsgBox "Ungültige Anzahl an Kontrollpunkten", vbCritical
Else
    ReDim PtnMatrix((M * N * 3) - 1) As Double
    Prompt = "erster Punkt des Netzes:"
    x = 0
    For i = 1 To M * N
        varpoint = ThisDrawing.Utility.GetPoint(, Prompt)
        PtnMatrix(x) = varpoint(0)
        x = x + 1
        PtnMatrix(x) = varpoint(1)
        x = x + 1
        PtnMatrix(x) = varpoint(2)
        x = x + 1
        Prompt = "nächster Punkt des Netzes:"
    Next
    Set New3DM = ThisDrawing.ModelSpace.Add3DMesh(M, N, PtnMatrix)
End If
End Sub

```

Siehe auch *Kapitel 8.3.22*

Polylinien erstellen

AutoCAD stellt insgesamt drei Arten von Polylinien zur Verfügung. Die erste ist die 3D-Polylinie, die vollständige 3D-Arrays interpretieren kann. Das bedeutet, dass jeder Stützpunkt räumlich dargestellt werden kann. Als Parameter der Funktion, die dem Hinzufügen von 3D-Polylinien dient, müssen Sie ein Array dieser Stützpunkte mitliefern, woraus sich auch schon die Definition des Arrays ergibt.

```
Dim PointArray((N * 3) - 1) As Double
```

Der Wert N steht hierbei für die Anzahl der Stützpunkte, wobei jeder Stützpunkt drei Koordinaten enthält. Von der so berechneten Gesamtanzahl an Stützpunktkoordinaten muss noch 1 abgezogen werden, da das Array mit dem Feldindex 0 beginnt.

3D-Polylinien erstellen

Add3DPolylinie

Parameter: *PointsArray*

Rückgabebetyp: *Acad3DPolyline*

```

Public Sub Sample_Add3DPoly()
    Dim New3DP As Acad3DPolyline
    Dim PtnMatrix(5) As Double, Prompt As String
    Dim LastPnt(2) As Double, VarPoints
    On Local Error Resume Next
    Prompt = "erster Punkt der 3D-Polylinie:"
    x = 0
    Do
        varPoint = Empty

```



```

If x = 0 Then
    varPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(varPoint) = True Then Exit Do
    PtnMatrix(0) = varPoint(0)
    PtnMatrix(1) = varPoint(1)
    PtnMatrix(2) = varPoint(2)
    x = 3
    Prompt = "nächster Punkt der 3D-Polylinie:"
ElseIf x = 3 Then
    varPoint = ThisDrawing.Utility.GetPoint(LastPnt, Prompt)
    If IsEmpty(varPoint) = True Then Exit Do
    PtnMatrix(3) = varPoint(0)
    PtnMatrix(4) = varPoint(1)
    PtnMatrix(5) = varPoint(2)
    Set New3DP = ThisDrawing.ModelSpace.Add3DPoly(PtnMatrix)
    x = 6
Else
    varPoint = ThisDrawing.Utility.GetPoint(LastPnt, Prompt)
    If IsEmpty(varPoint) = True Then Exit Do
    New3DP.AppendVertex varPoint
End If
LastPnt(0) = varPoint(0)
LastPnt(1) = varPoint(1)
LastPnt(2) = varPoint(2)
Loop
End Sub

```

Siehe auch *Kapitel 8.3.4*

2D-Polylinien erstellen

Die zweite Art von Polylinien, die erstellt werden können, sind die 2D-Polylinien. Bei dieser Art von Polylinie spielen die Höhen nur eine untergeordnete Rolle und sollen nur die räumliche Position der gesamten Polylinie darstellen. AutoCAD unterscheidet zwei Arten von 2D-Polylinien: zum einen die „normale“ Polylinie, die sich noch aus vorangegangenen AutoCAD-Versionen erhalten hat. Diese Polylinien speichern 3D-Arrays pro Stützpunkt, genau wie die 3D-Polylinien, weshalb die Definition des Arrays, das zur Erstellung der Polylinie erforderlich ist, auf die gleiche Weise erfolgt:

```
Dim PointArray((N * 3) - 1) As Double
```

Allerdings werden die Höhen aus den Stützpunktkoordinaten bei der Darstellung der Polylinie ignoriert. Die eigentliche Höhe wird in der Polylinieneigenschaft *Elevation* gespeichert.

AddPolylinie

Parameter: *VerticesList*

Rückgabotyp: *AcadPolyline*

```

Public Sub Sample_AddPolyline()
    Dim New2DP As AcadPolyline
    Dim PtnMatrix(5) As Double, Prompt As String

```

```

Dim LastPnt(2) As Double, VarPoints, PlHeight As Double
On Local Error Resume Next
Prompt = "Geben Sie die Höhe der 2D-Polylinie an:"
PlHeight = ThisDrawing.Utility.GetReal(Prompt)
Prompt = "erster Punkt der 2D-Polylinie:"
x = 0
Do
    varPoint = Empty
    If x = 0 Then
        varPoint = ThisDrawing.Utility.GetPoint(, Prompt)
        If IsEmpty(varPoint) = True Then Exit Do
        PtnMatrix(0) = varPoint(0)
        PtnMatrix(1) = varPoint(1)
        PtnMatrix(2) = 0
        x = 3
        Prompt = "nächster Punkt der 2D-Polylinie:"
    ElseIf x = 3 Then
        varPoint = ThisDrawing.Utility.GetPoint(LastPnt, Prompt)
        If IsEmpty(varPoint) = True Then Exit Do
        PtnMatrix(3) = varPoint(0)
        PtnMatrix(4) = varPoint(1)
        PtnMatrix(5) = 0
        Set New2DP = ThisDrawing.ModelSpace.AddPolyline(PtnMatrix)
        New2DP.Elevation = PlHeight
        x = 6
    Else
        varPoint = ThisDrawing.Utility.GetPoint(LastPnt, Prompt)
        If IsEmpty(varPoint) = True Then Exit Do
        varPoint(2) = 0
        New2DP.AppendVertex varPoint
    End If
    LastPnt(0) = varPoint(0)
    LastPnt(1) = varPoint(1)
    LastPnt(2) = varPoint(2)
Loop
End Sub

```

Siehe auch *Kapitel 8.3.12*

Bei der zweiten Art handelt es sich um die optimierte Polylinie. Diese benötigt weniger Speicherplatz, da für die Stützpunkte nur die Lagekoordinaten (X, Y) verwendet werden. Die Höhe selbst wird, wie bei der „normalen“ 2D-Polylinie, in der Eigenschaft *Elevation* gespeichert. Daraus ergibt sich folgende Definition des Stützpunkt-Arrays für die Übergabe an die *Add*-Funktion:

```
Dim PointArray((N * 2) - 1) As Double
```

Der Unterschied zu den vorangegangenen Definitionen besteht lediglich darin, dass bei diesem Array nur zwei Koordinaten pro Stützpunkt berücksichtigt werden müssen.

Optimierte Polylinien erstellen

AddLightWeightPolyline

Parameter: *VerticesList*

Rückgabebetyp: *AcadLWPolyline*

```
Public Sub Sample_AddLWPolyline()
Dim NewLWP As AcadLWPolyline, NewPoint(1) As Double
Dim PtnMatrix(3) As Double, Prompt As String
Dim LastPnt(2) As Double, VarPoints, PlHeight As Double
On Local Error Resume Next
Prompt = "Geben Sie die Höhe der LW-Polylinie an:"
PlHeight = ThisDrawing.Utility.GetReal(Prompt)
Prompt = "erster Punkt der LW-Polylinie:"
x = 0
Do
    varPoint = Empty
    If x = 0 Then
        varPoint = ThisDrawing.Utility.GetPoint(, Prompt)
        If IsEmpty(varPoint) = True Then Exit Do
        PtnMatrix(0) = varPoint(0)
        PtnMatrix(1) = varPoint(1)
        x = 2
        Prompt = "nächster Punkt der 2D-Polylinie:"
    ElseIf x = 2 Then
        varPoint = ThisDrawing.Utility.GetPoint(LastPnt, Prompt)
        If IsEmpty(varPoint) = True Then Exit Do
        PtnMatrix(2) = varPoint(0)
        PtnMatrix(3) = varPoint(1)
        Set NewLWP = ThisDrawing.ModelSpace. _
            AddLightWeightPolyline(PtnMatrix)
        NewLWP.Elevation = PlHeight
        x = 4
    Else
        varPoint = ThisDrawing.Utility.GetPoint(LastPnt, Prompt)
        If IsEmpty(varPoint) = True Then Exit Do
        NewPoint(0) = varPoint(0)
        NewPoint(1) = varPoint(1)
        NewLWP.AddVertex (UBound(NewLWP.Coordinates) + 1) / 2, _
            NewPoint
    End If
    LastPnt(0) = varPoint(0)
    LastPnt(1) = varPoint(1)
    LastPnt(2) = varPoint(2)
Loop
End Sub
```

Siehe auch *Kapitel 8.3.12*

Kreisbogen erstellen

Anders als mit AutoCAD selbst, wo Sie einen Bogen auf unterschiedliche Weise erstellen können, steht Ihnen in VBA leider nur eine Funktion zur Verfügung. Diese Funktion hat festgelegte Parameter, die Sie alle mit Werten belegen müssen, was die Flexibilität bei der Erstellung eines Bogens erheblich einschränkt. Sie müssen für die Erstellung des Bogens folgende Werte an die *Add*-Funktion übergeben: den Mittelpunkt, der als 3D-Array definiert wird, den Radius, Start- und Endwinkel, die jeweils vom Mittelpunkt in Richtung Start- und Endpunkt berechnet werden. Daraus folgt, dass für die Erstellung von Kreisbögen Ihre Kreativität gefragt ist, um die einzelnen Parameter auf den verschiedensten Wegen bestimmen zu können.

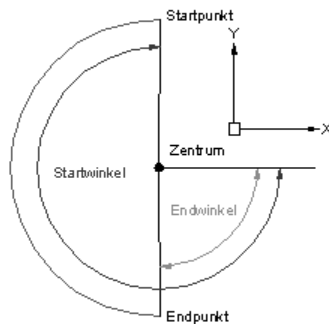


Abbildung 6.6: Kreisbogen

Für die Erstellung eines Bogens werden im Folgenden zwei Möglichkeiten aufgezeigt, wie die Parameter auf unterschiedliche Art und Weise ermittelt werden könnten.

AddArc

Parameter: *Center*, *Radius As Double*, *StartAngle As Double*, *EndAngle As Double*

Rückgabebetyp: *AcadArc*

1. Variante: Ermittlung der Parameter durch direkte Benutzereingaben aller Parameter

```
Public Sub Sample_AddArc1()
Dim NewArc As AcadArc
Dim Center As Variant, Radius As Double
Dim StartAngle As Double, EndAngle As Double
On Local Error Resume Next
Prompt = "Zentrumspunkt des Bogens:"
Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Center) = True Then Exit Sub
Prompt = "Radius des Bogens:"
Radius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Radius = 0 Then Exit Sub
Prompt = "Startwinkel des Bogens:"
StartAngle = ThisDrawing.Utility.GetAngle(, Prompt)
Prompt = "Endwinkel des Bogens:"
EndAngle = ThisDrawing.Utility.GetAngle(, Prompt)
Set NewArc = ThisDrawing.ModelSpace.AddArc(Center, Radius, _
    StartAngle, EndAngle)
End Sub
```

2. Variante: Ermittlung der Parameter durch Berechnung von Radius, Start- und Endwinkel nach Auswahl von Zentrum, Start- und Endpunkt durch den Benutzer

```
Public Sub Sample_AddArc2()
Dim NewArc As AcadArc
Dim Center As Variant, Radius As Double
Dim StartAngle As Double, EndAngle As Double
On Local Error Resume Next
Prompt = "Zentrumspunkt des Bogens:"
Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Center) = True Then Exit Sub
Prompt = "Startpunkt des Bogens:"
StartPt = ThisDrawing.Utility.GetPoint(Center, Prompt)
Prompt = "Endpunkt des Bogens:"
EndPt = ThisDrawing.Utility.GetPoint(Center, Prompt)
Radius = Sqr((Cdbl(Center(0)) - Cdbl(StartPt(0))) ^ 2 + _
(Cdbl(Center(1)) - Cdbl(StartPt(1))) ^ 2)
StartAngle = ThisDrawing.Utility.AngleFromXAxis(Center, StartPt)
EndAngle = ThisDrawing.Utility.AngleFromXAxis(Center, EndPt)
Set NewArc = ThisDrawing.ModelSpace.AddArc(Center, Radius, _
StartAngle, EndAngle)
End Sub
```

Siehe auch *Kapitel 8.3.5*

Attributdefinition erstellen

Attributdefinitionen sollten im Modell- bzw. Papierbereich nur dann eingefügt werden, wenn Sie die Zeichnung später als Blockdefinition in eine andere Zeichnung einfügen wollen.

Die Funktion selbst enthält bereits eine ganze Reihe von Definitionsparametern, so dass das Attribut nur wenig Nacharbeit nach der Erstellung erfordert. Im ersten Parameter wird die Höhe des Attributtextes festgelegt, der zweite enthält den Modus des Attributs, wobei Sie folgende Konstanten verwenden können:

- *acAttributeModeNormal* – Entspricht der normalen Darstellung, also dem Ausschluss aller anderen Attributmodi.
- *acAttributeModeInvisible* – Das Attribut wird unsichtbar.
- *acAttributeModeConstant* – Vergibt an Attribute einen festen Wert.
- *acAttributeModeVerify* – Fordert beim Einfügen des Blocks auf, die Richtigkeit des Attributwerts zu prüfen.
- *acAttributeModePreset* – Setzt das Attribut auf den Vorgabewert, wenn Sie einen Block einfügen, der ein voreingestelltes Attribut enthält.

Sie können die einzelnen Konstanten auch kombinieren, wenn Sie zwei verschiedene Modi setzen möchten, indem Sie diese addieren:

```
Modus = AcAttributeModeInvisible + AcAttributeModeConstant
```

Im Parameter *Prompt* wird die Zeichenfolge festgelegt, die beim Einfügen des Blocks als Anfrage in der Befehlszeile angezeigt wird. Wenn Sie im Modus den Wert *acAttributeModeConstant* festgelegt haben, wird die Eingabeaufforderung ignoriert. Als Nächstes wird noch der Einfügepunkt benötigt, der die Position des Attributs in der Zeichnung bestimmt. Dieser

Punkt muss wieder als 3D-Array definiert sein. Die letzten beiden Parameter bestimmen zum einen die Attributbezeichnung, die zur Identifizierung des Attributs bestimmt ist. Sie können für diesen Parameter alle Zeichen außer Leer- und Ausrufezeichen verwenden. Zum anderen bleibt noch der Vorgabewert für das Attribut, der im Parameter *Value* festgelegt wird.

AddAttribute

Parameter: *Height As Double, Mode As AcadAttributeMode, Prompt As String, InsertionPoint, Tag As String, Value As String*

Rückgabetyt: *AcadAttribute*

```
Public Sub Sample_AddAttribut()
Dim NewAttDef As AcadAttribute
Dim Insertion As Variant, Height As Double
Dim Prompt As String
Dim Modus As AcadAttributeMode, AttPrompt As String
Dim TagString As String, Value As String
On Local Error Resume Next
Prompt = "Einfügepunkt:"
Insertion = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Insertion) = True Then Exit Sub
Prompt = "Attributhöhe:"
Height = ThisDrawing.Utility.GetDistance(Insertion, Prompt)
If Height = 0 Then Exit Sub
Prompt = "Text in der Eingabeaufforderung:"
AttPrompt = ThisDrawing.Utility.GetString(1, Prompt)
Prompt = "Attributbezeichnung:"
TagString = ThisDrawing.Utility.GetString(0, Prompt)
Prompt = "Vorgabetext des Attributes:"
Value = ThisDrawing.Utility.GetString(1, Prompt)
Set NewAttDef = ThisDrawing.ModelSpace.AddAttribute(Height, _
    AcadAttributeModeNormal, AttPrompt, Insertion, TagString, Value)
End Sub
```

Siehe auch *Kapitel 8.3.27*

Kreis erstellen

Um dem Modell- oder Papierbereich einen Kreis hinzuzufügen, benötigt die Funktion *AddCircle*, die diese Aufgabe erledigt, lediglich den Mittelpunkt als 3D-Array und den Radius des Kreises. Das gestaltet die Erzeugung eines Kreises recht simpel.

AddCircle

Parameter: *Center, Radius As Double*

Rückgabetyt: *AcadCircle*

```
Public Sub Sample_AddCircle()
Dim NewCircle As AcadCircle
Dim Center As Variant, Radius As Double
On Local Error Resume Next
Prompt = "Zentrumspunkt des Kreises:"
Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Center) = True Then Exit Sub
```

```
Prompt = "Radius:"
Radius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Radius = 0 Then Exit Sub
Set NewCircle = ThisDrawing.ModelSpace.AddCircle(Center, Radius)
End Sub
```

Siehe auch *Kapitel 7.3.6*

Benutzerdefiniertes Objekt erstellen

Ein benutzerdefiniertes Objekt kann nur in Verbindung mit der dazugehörigen Arx-Anwendung eingefügt werden und erfordert als Parameter den Klassennamen der Anwendung. Bevor Sie ein Objekt, das von einer ARX-Applikation definiert wird, in AutoCAD einfügen, muss diese Applikation geladen sein, da sonst die Funktion einen Fehler zurückgibt und das Objekt nicht erstellt wird.

AddCustomObject

Parameter: *ClassName As String*

Rückgabebetyp: *Object*

```
Public Sub Sample_AddCustomObject
Dim AppObj As AcadObject
ThisDrawing.Application.LoadArx "MyARXApp.arx"
Set AppObj = ThisDrawing.ModelSpace.AddCustomObject("MyArxObject")
End Sub
```

Winkelbemaßung erstellen

Für ein Objekt, das einen Winkel bemaßt, stellen die Klassen *AcadModelSpace* und *AcadPaperSpace* zwei Funktionen (*AddDim3PointAngular* und *AddDimAngular*) zur Verfügung. Beide Funktionen erstellen eine Winkelbemaßung mithilfe von drei Punkten, deren eingeschlossener Winkel den Bemaßungstext ergibt. Der vierte Parameter bestimmt hingegen den Einfügepunkt des Bemaßungstextes, was gleichzeitig die Position des Bogens zur Winkelanzeige definiert. Beide Funktionen erstellen jeweils Bemaßungsobjekte, die zwar bezüglich des Aussehens identisch sind, aber jeweils durch eine eigene Klasse definiert werden.

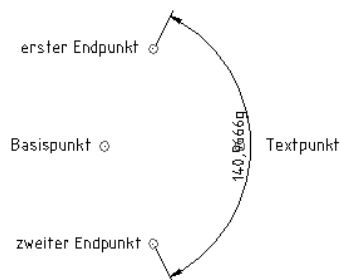


Abbildung 6.7: Winkelbemaßung

AddDim3PointAngularParameter: *AngleVertex, FirstEndPoint, SecondEndPoint, TextPoint*Rückgabebetyp: *AcadDim3PointAngular*

```
Public Sub Sample_AddDim3PointAngular()
Dim NewDimObj As AcadDim3PointAngular
Dim BasePoint As Variant, StartPnt As Variant, EndPnt As Variant
Dim TextPoint As Variant
On Local Error Resume Next
Prompt = "Basispunkt der Winkelbemaßung:"
BasePoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(BasePoint) = True Then Exit Sub
Prompt = "Erste Seite:"
StartPnt = ThisDrawing.Utility.GetPoint(BasePoint, Prompt)
If IsEmpty(StartPnt) = True Then Exit Sub
Prompt = "Zweite Seite:"
EndPnt = ThisDrawing.Utility.GetPoint(BasePoint, Prompt)
If IsEmpty(EndPnt) = True Then Exit Sub
Prompt = "Texteinfügekpunkt:"
TextPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(TextPoint) = True Then Exit Sub
Set NewDimObj = ThisDrawing.ModelSpace.AddDim3PointAngular( _
    BasePoint, StartPnt, EndPnt, TextPoint)
End Sub
```

Siehe auch *Kapitel 8.3.8***AddDimAngular**Parameter: *AngleVertex, FirstEndPoint, SecondEndPoint, TextPoint*Rückgabebetyp: *AcadDimAngular*

```
Public Sub Sample_AddDimAngular()
Dim NewDimObj As AcadDimAngular
Dim BasePoint As Variant, StartPnt As Variant, EndPnt As Variant
Dim TextPoint As Variant
On Local Error Resume Next
Prompt = "Basispunkt der Winkelbemaßung:"
BasePoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(BasePoint) = True Then Exit Sub
Prompt = "Erste Seite:"
StartPnt = ThisDrawing.Utility.GetPoint(BasePoint, Prompt)
If IsEmpty(StartPnt) = True Then Exit Sub
Prompt = "Zweite Seite:"
EndPnt = ThisDrawing.Utility.GetPoint(BasePoint, Prompt)
If IsEmpty(EndPnt) = True Then Exit Sub
Prompt = "Texteinfügekpunkt:"
TextPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(TextPoint) = True Then Exit Sub
Set NewDimObj = ThisDrawing.ModelSpace.AddDimAngular(BasePoint, _
    StartPnt, EndPnt, TextPoint)
End Sub
```

Siehe auch *Kapitel 8.3.8*

Ausgerichtetes Bemaßungsobjekt erstellen

Bei dem ausgerichteten Bemaßungsobjekt handelt es sich um eine Linearbemaßung zwischen zwei definierten Punkten. Diese beiden Basispunkte müssen auch als Parameter der *Add*-Funktion mitgegeben werden. Außerdem ist wieder die Position des Bemaßungstextes erforderlich. Alle drei Punkte müssen als 3D-Array definiert sein und könnten beispielsweise über die *GetPoint*-Funktion des *Utility*-Objekts abgefragt werden, wie das unten stehende Beispiel zeigt.

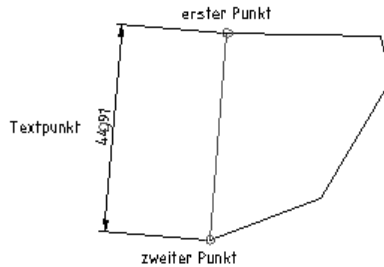


Abbildung 6.8: Ausgerichtete Bemaßung

AddDimAligned

Parameter: *ExtLine1Point*, *ExtLine2Point*, *TextPosition*

Rückgabetyt: *AcadDimAligned*

```
Public Sub Sample_AddDimAligned()
Dim NewDimObj As AcadDimAligned
Dim LinePnt1 As Variant, LinePnt2 As Variant
Dim TextPoint As Variant
On Local Error Resume Next
Prompt = "Anfangspunkt der ersten Hilfslinie:"
LinePnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(LinePnt1) = True Then Exit Sub
Prompt = "Anfangspunkt der zweiten Hilfslinie:"
LinePnt2 = ThisDrawing.Utility.GetPoint(LinePnt1, Prompt)
If IsEmpty(LinePnt2) = True Then Exit Sub
Prompt = "Texteinfügepunkt:"
TextPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(TextPoint) = True Then Exit Sub
Set NewDimObj = ThisDrawing.ModelSpace.AddDimAligned(LinePnt1, _
LinePnt2, TextPoint)
End Sub
```

Siehe auch *Kapitel 8.3.8*

Durchmesserbemaßung erstellen

Um den Durchmesser eines Kreises oder eines Bogens zu bemaßen, werden drei Parameter benötigt. Der erste bestimmt den ersten Ausgangspunkt des Kreises, der zweite legt den Durchmesser selbst und die Richtung der Bemaßungslinie fest und ist ebenfalls als 3D-Array definiert. Der dritte Parameter legt schließlich die Länge der Führungslinie fest. Da mit dieser

Art der Punktangaben nur schwerlich der Mittelpunkt des Kreises und somit der genaue Durchmesser zu bestimmen ist, sollten Sie den ersten Punkt über ein Zentrum berechnen lassen. Dazu sind im folgenden zwei Beispiele angegeben.

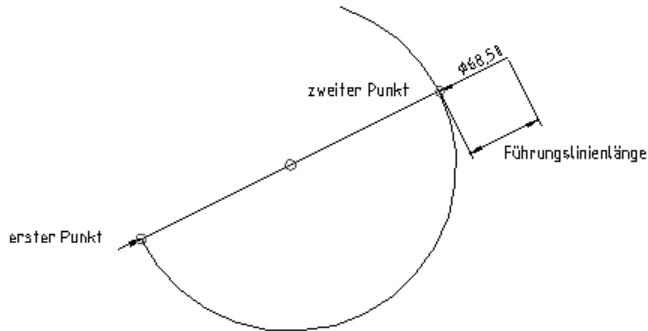


Abbildung 6.9: Durchmesserbemaßung

AddDimDiametric

Parameter: *ChordPoint*, *FarChordPoint*, *LeaderLength* As Double

Rückgabebetyp: *AcadDimDiametric*

1. Variante: Durchmesserbemaßung über Auswahl des Zentrums punkts

```
Public Sub Sample_AddDimDiametric1()
    Dim NewDimObj As AcadDimDiametric, Radius As Double
    Dim FirstPnt As Variant, SecondPnt As Variant
    Dim LeaderLength As Double, Center As Variant
    On Local Error Resume Next
    Prompt = "Zentrumspunkt der Durchmesserbemaßung:"
    Center = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(Center) = True Then Exit Sub
    Prompt = "Punkt auf dem Kreisbogen:"
    SecondPnt = ThisDrawing.Utility.GetPoint(Center, Prompt)
    If IsEmpty(SecondPnt) = True Then Exit Sub
    Radius = Sqr((Cdbl(Center(0)) - Cdbl(SecondPnt(0))) ^ 2 + _
        (Cdbl(Center(1)) - Cdbl(SecondPnt(1))) ^ 2)
    Angle = ThisDrawing.Utility.AngleFromXAxis(SecondPnt, Center)
    FirstPnt = ThisDrawing.Utility.PolarPoint(SecondPnt, Angle, _
        Radius * 2)
    Prompt = "Länge der Führungslinie:"
    LeaderLength = ThisDrawing.Utility.GetDistance(FirstPnt, Prompt)
    If LeaderLength = True Then Exit Sub
    Set NewDimObj = ThisDrawing.ModelSpace.AddDimDiametric(FirstPnt, _
        SecondPnt, LeaderLength)
End Sub
```

2. Variante: Durchmesserbemaßung über Auswahl eines Kreises oder Bogens

```
Public Sub Sample_AddDimDiametric2()
Dim NewDimObj As AcadDimDiametric, Radius As Double
Dim FirstPnt As Variant, SecondPnt As Variant
Dim LeaderLength As Double, Center As Variant
Dim Object As Object
On Local Error Resume Next
Prompt = "zu bemessenden Kreis oder Bogen wählen:"
ThisDrawing.Utility.GetEntity Object, Pickedpoint, Prompt
If TypeName(Object) = "IAcadArc" Or TypeName(Object) = _
    "IAcadCircle" Then
    Center = Object.Center
    Prompt = "Richtung der Bemaßungslinie:"
    Angle = ThisDrawing.Utility.GetOrientation(Center, Prompt)
    SecondPnt = ThisDrawing.Utility.PolarPoint(Center, Angle, _
        Object.Radius)
    Angle = ThisDrawing.Utility.AngleFromXAxis(SecondPnt, Center)
    FirstPnt = ThisDrawing.Utility.PolarPoint(SecondPnt, Angle, _
        Object.Radius * 2)
    Prompt = "Länge der Führungslinie:"
    LeaderLength = ThisDrawing.Utility.GetDistance(FirstPnt, _
        Prompt)
    If LeaderLength = True Then Exit Sub
    Set NewDimObj = ThisDrawing.ModelSpace.AddDimDiametric( _
        FirstPnt, SecondPnt, LeaderLength)
End If
End Sub
```

Siehe auch *Kapitel 8.3.8*

Koordinatenbemaßung erstellen

Zur Bemaßung der X- oder Y-Koordinate eines gewählten Punkts erfordert die Funktion *AddDimOrdinate* drei Angaben. Zum einen den Punkt, dessen Koordinaten bemaßt werden sollen, und zum anderen den Endpunkt der Führungslinie. Die Definitionen dieser beiden Punkte sind in unten stehender Abbildung zu erkennen. Beide Punkte sind wie üblich als 3D-Array an die Funktion zu übergeben. Der dritte Parameter bestimmt schließlich, welcher der beiden Koordinaten X oder Y bemaßt werden soll. Geben Sie im Parameter den Wert *True* ein, wird die X-Koordinate bemaßt. Der Wert *False* veranlasst die Funktion, die Y-Koordinate für die Bemaßung zu verwenden.

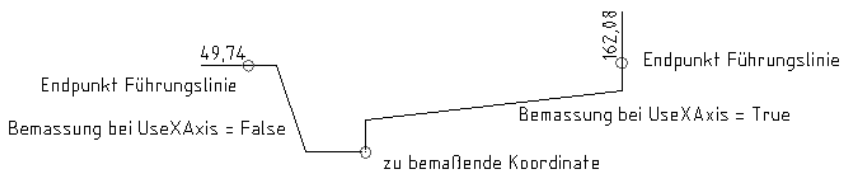


Abbildung 6.10: Koordinatenbemaßung

AddDimOrdinate

Parameter: *DefinitionPoint*, *LeaderEndPoint*, *UseXAxis As Long*

Rückgabebetyp: *AcadDimOrdinate*

1. Variante: Bemaßung der X-Ordinate

```
Public Sub Sample_AddDimOrdinateX()
Dim NewDimObj As AcadDimOrdinate
Dim SecondPnt As Variant, BasePnt As Variant
On Local Error Resume Next
Prompt = "Zu bemassende Koordinate wählen:"
BasePnt = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(BasePnt) = True Then Exit Sub
Prompt = "Endpunkt der Führungslinie:"
SecondPnt = ThisDrawing.Utility.GetPoint(BasePnt, Prompt)
If IsEmpty(SecondPnt) = True Then Exit Sub
Set NewDimObj = ThisDrawing.ModelSpace.AddDimOrdinate(BasePnt, _
    SecondPnt, True)
End Sub
```

2. Variante: Bemaßung der Y-Ordinate

```
Public Sub Sample_AddDimOrdinateY()
Dim NewDimObj As AcadDimOrdinate
Dim SecondPnt As Variant, BasePnt As Variant
On Local Error Resume Next
Prompt = "Zu bemassende Koordinate wählen:"
BasePnt = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(BasePnt) = True Then Exit Sub
Prompt = "Endpunkt der Führungslinie:"
SecondPnt = ThisDrawing.Utility.GetPoint(BasePnt, Prompt)
If IsEmpty(SecondPnt) = True Then Exit Sub
Set NewDimObj = ThisDrawing.ModelSpace.AddDimOrdinate(BasePnt, _
    SecondPnt, False)
End Sub
```

Siehe auch *Kapitel 8.3.8*

Radiusbemaßung erstellen

Neben dem Bemaßungsobjekt für Durchmesser, deren Erzeugung wir bereits besprochen haben, stellt AutoCAD auch eine Funktion für die Erstellung eines Bemaßungsobjekts für Radien zur Verfügung. Mit dieser Funktion ist jedoch die direkte Verwendung des Zentrums nicht nur möglich, sie ist sogar zwingend erforderlich. Der zweite und dritte Parameter entspricht wieder dem der Durchmesserbemaßung, der zweite übernimmt also den Radius selbst und die Richtung der Bemaßungslinie und der letzte bestimmt die Länge der Führungslinie.

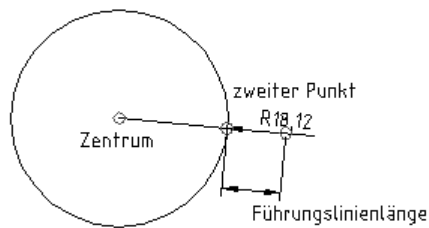


Abbildung 6.11: Radiusbemaßung

AddDimRadial

Parameter: *Center, ChordPoint, LeaderLength* As Double

Rückgabebetyp: *AcadDimRadial*

1. Variante: Radiusbemaßung über Auswahl des Zentrums punkts

```
Public Sub Sample_AddDimRadial1()
    Dim NewDimObj As AcadDimRadial
    Dim SecondPnt As Variant
    Dim LeaderLength As Double, Center As Variant
    On Local Error Resume Next
    Prompt = "Zentrumspunkt der Radiusbemaßung:"
    Center = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(Center) = True Then Exit Sub
    Prompt = "Punkt auf dem Kreisbogen:"
    SecondPnt = ThisDrawing.Utility.GetPoint(Center, Prompt)
    If IsEmpty(SecondPnt) = True Then Exit Sub
    Prompt = "Länge der Führungslinie:"
    LeaderLength = ThisDrawing.Utility.GetDistance(SecondPnt, Prompt)
    If LeaderLength = True Then Exit Sub
    Set NewDimObj = ThisDrawing.ModelSpace.AddDimRadial(Center, _
        SecondPnt, LeaderLength)
End Sub
```

2. Variante: Radiusbemaßung über Auswahl eines Kreises oder Bogens

```
Public Sub Sample_AddDimRadial2()
    Dim NewDimObj As AcadDimRadial
    Dim SecondPnt As Variant
    Dim LeaderLength As Double, Center As Variant
    Dim Object As Object
    On Local Error Resume Next
    Prompt = "zu bemassenden Kreis oder Bogen wählen:"
    ThisDrawing.Utility.GetEntity Object, Pickedpoint, Prompt
    If TypeName(Object) = "IAcadArc" Or TypeName(Object) = _
        "IAcadCircle" Then
        Center = Object.Center
        Prompt = "Richtung der Bemassungslinie:"
        Angle = ThisDrawing.Utility.GetOrientation(Center, Prompt)
        SecondPnt = ThisDrawing.Utility.PolarPoint(Center, Angle, _
```

```

Object.Radius)
Prompt = "Länge der Führungslinie:"
LeaderLength = ThisDrawing.Utility.GetDistance(SecondPnt, _
    Prompt)
If LeaderLength = True Then Exit Sub
Set NewDimObj = ThisDrawing.ModelSpace.AddDimRadial(Center, _
    SecondPnt, LeaderLength)
End If
End Sub

```

Siehe auch *Kapitel 8.3.8*

Gedrehte lineare Bemaßung erstellen

Die gedrehte lineare Bemaßung kann, im Gegensatz zur ausgerichteten Bemaßung, zur bemaßten Linie gedreht dargestellt werden. Zur Erzeugung eines solchen Objekts wird die Funktion *AddDimRotated* verwendet, die Objekte der Klasse *AcadDimRotated* erzeugt. Die ersten beiden Parameter der Funktion sind identisch mit denen der *Add*-Funktion für die ausgerichtete Bemaßung und definieren somit den Start- und Endpunkt der zu bemaßenden Strecke. Der dritte Parameter bestimmt die Länge der ersten Führungslinie. Mit dem dritten Parameter legen Sie die Richtung der Bemaßungslinie fest, woraus sich die Länge und Richtung der zweiten Führungslinie ergibt.



Abbildung 6.12: Gedrehte lineare Bemaßung

AddDimRotated

Parameter: *ExtLine1Point*, *ExtLine2Point*, *DimLineLocation*, *RotationAngle As Double*

Rückgabetyt: *AcadDimRotated*

1. Variante: lineare Bemaßung ausgerichtet zum Zeichnungsrand

```

Public Sub Sample_AddDimRotated1()
Dim NewDimObj As AcadDimRotated, DimLinePnt As Variant
Dim ExtLinePnt1 As Variant, ExtLinePnt2 As Variant
On Local Error Resume Next
Prompt = "Anfangspunkt der ersten Hilfslinie:"
ExtLinePnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(ExtLinePnt1) = True Then Exit Sub
Prompt = "Anfangspunkt der zweiten Hilfslinie:"
ExtLinePnt2 = ThisDrawing.Utility.GetPoint(ExtLinePnt1, Prompt)
If IsEmpty(ExtLinePnt2) = True Then Exit Sub
Prompt = "Führungslinienposition:"
DimLinePnt = ThisDrawing.Utility.GetPoint(ExtLinePnt1, Prompt)

```

```

If IsEmpty(DimLinePnt) = True Then Exit Sub
Set NewDimObj = ThisDrawing.ModelSpace.AddDimRotated(ExtLinePnt1 _
    , ExtLinePnt2, DimLinePnt, _
    ThisDrawing.GetVariable("VIEWTWIST"))
End Sub

```

2. Variante: lineare Bemaßung mit Ausrichtung durch den Benutzer

```

Public Sub Sample_AddDimRotated2()
Dim NewDimObj As AcadDimRotated, DimLinePnt As Variant
Dim ExtLinePnt1 As Variant, ExtLinePnt2 As Variant
Dim Angle As Double
On Local Error Resume Next
Prompt = "Anfangspunkt der ersten Hilfslinie:"
ExtLinePnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(ExtLinePnt1) = True Then Exit Sub
Prompt = "Anfangspunkt der zweiten Hilfslinie:"
ExtLinePnt2 = ThisDrawing.Utility.GetPoint(ExtLinePnt1, Prompt)
If IsEmpty(ExtLinePnt2) = True Then Exit Sub
Prompt = "Führungslinienposition:"
DimLinePnt = ThisDrawing.Utility.GetPoint(ExtLinePnt1, Prompt)
If IsEmpty(DimLinePnt) = True Then Exit Sub
Prompt = "Richtung der Bemaßungslinie:"
Angle = ThisDrawing.Utility.GetOrientation(DimLinePnt, Prompt)
Set NewDimObj = ThisDrawing.ModelSpace.AddDimRotated( _
    ExtLinePnt1, ExtLinePnt2, DimLinePnt, Angle)
End Sub

```

Siehe auch *Kapitel 8.3.8*

Ellipse erstellen

Das Hinzufügen von Ellipsen geschieht in VBA über die Funktion *AddEllipse*. Diese Funktion erfordert drei Parameter, wobei der erste den Mittelpunkt der Ellipse bestimmt. Dieser Punkt wird wie immer als 3D-Array übergeben. Der zweite Parameter gibt die Position des Endpunkts der Hauptachse relativ zum Mittelpunkt an, wie es in der unteren Abbildung zu sehen ist. Der dritte und somit letzte Parameter legt das Verhältnis von Nebenachse zur Hauptachse fest. Damit ergibt sich der Wert aus:

$\text{RadiusRatio} = \text{NebenRadius} / \text{HauptRadius}$

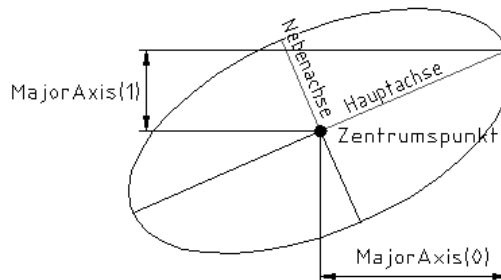


Abbildung 6.13: Ellipse

Sie können für das Achsenverhältnis Werte, die größer als 0 sind und den Wert 1.0 nicht überschreiten, verwenden. Der Wert 1.0 definiert folglich einen Kreis, wobei das Objekt trotzdem der Klasse *AcadEllipse* entspricht.

AddEllipse

Parameter: *Center, MajorAxis, RadiusRatio As Double*

Rückgabetyt: *AcadEllipse*

```
Public Sub Sample_AddEllipse()
Dim NewDimObj As AcadEllipse
Dim Center As Variant, MajorAxis As Variant
Dim Radius As Double, RadRatio As Double
On Local Error Resume Next
Prompt = "Zentrumspunkt der Ellipse:"
Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(center) = True Then Exit Sub
Prompt = "Achsendpunkt der Ellipse:"
MajorAxis = ThisDrawing.Utility.GetPoint(center, Prompt)
If IsEmpty(MajorAxis) = True Then Exit Sub
Prompt = "Abstand zur anderen Achse:"
Radius = ThisDrawing.Utility.GetDistance(center, Prompt)
If Radius = 0 Then Exit Sub
RadRatio = Radius / Sqr((Cdbl(Center(0)) - Cdbl(MajorAxis(0))) _
    ^ 2 + (Cdbl(Center(1)) - Cdbl(MajorAxis(1))) ^ 2)
MajorAxis(0) = MajorAxis(0) - Center(0)
MajorAxis(1) = MajorAxis(1) - Center(1)
MajorAxis(2) = MajorAxis(2) - Center(2)
Set NewDimObj = ThisDrawing.ModelSpace.AddEllipse(Center, _
    MajorAxis, RadRatio)
End Sub
```

Siehe auch *Kapitel 8.3.8*

Schraffur erstellen

Das Hinzufügen von Schraffuren unterteilt sich in AutoCAD-VBA in zwei Arbeitsgänge. Zunächst wird das eigentliche Schraffurobjekt erzeugt. Anschließend wird diesem Schraffurobjekt eine äußere Umgrenzung, die aus einer Liste von Objekten besteht, hinzugefügt. Erst dieser zweite Schritt komplettiert das Schraffurobjekt, so dass es in der Zeichnung sichtbar wird. Das Hinzufügen eines Schraffurobjekts zum Modell- bzw. Papierbereich allein reicht folglich nicht, um eine Schraffur zu erstellen.

Fangen wir also mit der Erzeugung des Schraffurobjekts an. Als Parameter verlangt die Funktion *AddHatch* den Mustertyp, wobei Sie folgende Konstanten verwenden können:

- *acHatchPatternTypePredefined* – Ein vordefiniertes Schraffurmuster der Datei *acad.pat* soll verwendet werden.
- *acHatchPatternTypeUserDefined* – Ein Schraffurmuster mit der aktuellen Linienart soll verwendet werden.
- *acHatchPatternTypeCustomDefined* – Ein vordefiniertes Schraffurmuster aus einer anderen Musterdatei soll anstatt des Musters der Datei *acad.pat* verwendet werden.

Im zweiten Parameter legen Sie den Namen des Schraffurmusters fest, z.B. „Angle“ oder „Solid“. Welchen Namen Sie verwenden, hängt davon, welche Musterdatei Sie verwenden. Als dritten und letzten Parameter der Funktion müssen Sie noch festlegen, ob Sie eine assoziative oder nicht assoziative Schraffur erstellen wollen. Erstellen Sie eine assoziative Schraffur, bleibt die Verbindung zu den Objekten erhalten, die als äußere oder innere Schleife dem Schraffurobjekt hinzugefügt wurden. Nachdem alle Parameter z.B. durch den Benutzer festgelegt wurden, kann die Schraffur zunächst erstellt werden.

```
Set NewHatch = ThisDrawing.ModelSpace.AddHatch(PatternType, _
    "Angle", True)
```

Im zweiten Schritt muss die Zuordnung von AutoCAD-Objekten erfolgen, die die Umgrenzung der Schraffur bilden sollen. Die Klasse *AcadHatch* bietet dafür mehrere Funktionen an. Für welche Sie sich entscheiden, bleibt natürlich Ihnen überlassen. Nehmen wir fürs Erste die Anweisung *AppendOuterLoop* für die äußere Umgrenzung. Was wir nun für diese Umgrenzung benötigen, sind Objekte, die in einem Array zusammengefasst an die Funktion übergeben werden. Die Ermittlung dieser Objekte könnte z.B. mithilfe eines Auswahlgesetzes durch den Benutzer erfolgen.

```
SSet.SelectOnScreen
```

Nachdem die Objekte ermittelt wurden, müssen diese noch an ein Array übergeben werden, dessen Felder mit dem Datentyp *AcadEntity* deklariert wurden.

```
ReDim LoopObjs(SSet.Count - 1) As AcadEntity
```

Da das Array mit dem Feld 0 beginnt, deklarieren wir die Obergrenze wieder mit der Anzahl der gewählten Objekte – 1. Bleibt nur noch die Übertragung der einzelnen Objekte an das Array und der abschließende Aufruf der *AppendOuterLoops*-Funktion:

```
For i = 0 To SSet.Count - 1
    Set LoopObjs(i) = SSet.Item(i)
Next
NewHatch.AppendOuterLoop LoopObjs
```

Wichtig bei der Erstellung der Objektverweise in das Array ist das Schlüsselwort *Set*, da es sich um Objekte und nicht um Werte handelt.

Tipp

Nachdem Sie ein Schraffurobjekt erstellt haben, müssen Sie erst eine Schraffurumgrenzung mithilfe einer der Schleifenfunktionen erstellen, bevor Sie weitere Änderungen, z.B. Layer, Winkel oder Linientyp, vornehmen, da es sonst zu unvorhersehbaren Fehlern kommen kann.

AddHatch

Parameter: *PatternType* As Long, *PatternName* As String, *Associativity* As Boolean

Rückgabotyp: *AcadHatch*

```
Public Sub Sample_AddHatch()
    Dim NewHatch As AcadHatch
    Dim HPName As String
    Dim LoopObjs, SSet As AcadSelectionSet
    On Local Error Resume Next
```

```

HPName = ThisDrawing.GetVariable("HPNAME")
Prompt = "Schraffurmuster festlegen <" & HPName & ">:"
HPName = ThisDrawing.Utility.GetString(0, Prompt)
If HPName = "" Then
    HPName = ThisDrawing.GetVariable("HPNAME")
End If
If TypeName(ThisDrawing.SelectionSets("HATCH")) = "Nothing" Then
    ThisDrawing.SelectionSets.Add "HATCH"
End If
Set SSet = ThisDrawing.SelectionSets("HATCH")
SSet.SelectOnScreen
If SSet.Count > 0 Then
    ReDim LoopObjs(SSet.Count - 1) As AcadEntity
    For i = 0 To SSet.Count - 1
        Set LoopObjs(i) = SSet.Item(i)
    Next
    Set NewHatch = ThisDrawing.ModelSpace.AddHatch( _
        acHatchPatternTypePreDefined, HPName, True) NewHatch.AppendOuterLoop LoopObjs
End If
End Sub

```

Siehe auch *Kapitel 8.3.10*

Führungslinie erstellen

Mithilfe von Führungslinien können Sie Anmerkungen mit einem Objekt verbinden. Normalerweise handelt es sich bei den Anmerkungen um einen Text, es können aber auch Blockreferenzen oder Form- und Lagetoleranzen mit diesem Objekt verbunden sein. Zur Erstellung einer Führungslinie können Sie die Funktion *AddLeader* verwenden. Auch diese Funktion verlangt zur Erstellung des Objekts, genau wie alle anderen, einige Parameter. Im ersten Parameter legen Sie ein Array mit 3D-Koordinaten fest, die die einzelnen Stützpunkte darstellen, über die die Führungslinien laufen sollen. Sie müssen mindestens zwei Stützpunkte angeben, alle weiteren sind optional.



Abbildung 6.14: Führungslinie

Der Parameter *Annotation* übergibt das Anmerkungsobjekt der Führung. Dies kann ein *Tolerance*-Objekt (*Kapitel 8.3.26*), ein *MText* (*Kapitel 8.3.16*) oder eine Blockreferenz (*Kapitel 8.3.29*) sein. Wenn Sie kein Anmerkungsobjekt verwenden wollen, müssen Sie ein leeres Objekt zuordnen.

```

Set AnnObj = Nothing
ThisDrawing.ModelSpace.AddLeader PtArray, AnnObj, LeaderType

```

Der letzte Parameter legt schließlich die Darstellung des Führungsobjekts fest. Sie können die Führungslinien als Linienzug oder als Spline mit oder ohne Pfeil darstellen lassen. Folgende Konstanten sind für diesen Parameter durch AutoCAD vordefiniert.

Darstellung	Konstante	Beschreibung
	<i>AcLineNoArrow</i>	Führungsobjekt als Linienzug und ohne Pfeil
	<i>AcLineWithArrow</i>	Führungsobjekt als Linienzug und mit Pfeil
	<i>AcSplineNoArrow</i>	Führungsobjekt als Spline und ohne Pfeil
	<i>AcSplineWithArrow</i>	Führungsobjekt als Spline und mit Pfeil

AddLeader

Parameter: *PointsArray*, *Annotation As AcadEntity*, *Type As AcLeaderType*

Rückgabebetyp: *AcadLeader*

1. Variante: Führungsobjekt ohne Anmerknungsobjekt als Linienzug mit Pfeil

```
Public Sub Sample_AddLeader1()
Dim NewLeader As AcadLeader
Dim LeaderObj As AcadEntity
Dim PtArray, Prompt As String, LastPt(2) As Double
On Local Error Resume Next
Do
    varPoint = Empty
    If IsEmpty(PtArray) = True Then
        Prompt = "ersten Führungspunkt wählen:"
        varPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Else
        Prompt = "nächsten Führungspunkt wählen:"
        varPoint = ThisDrawing.Utility.GetPoint(LastPt, Prompt)
    End If
    If IsEmpty(varPoint) = True Then Exit Do
    If IsEmpty(PtArray) = True Then
        ReDim PtArray(2) As Double
        PtArray(0) = varPoint(0): PtArray(1) = varPoint(1)
        PtArray(2) = varPoint(2)
    Else
        ReDim Preserve PtArray(UBound(PtArray) + 3) As Double
        PtArray(UBound(PtArray) - 2) = varPoint(0)
        PtArray(UBound(PtArray) - 1) = varPoint(1)
        PtArray(UBound(PtArray)) = varPoint(2)
    End If
    LastPt(0) = varPoint(0): LastPt(1) = varPoint(1)
    LastPt(2) = varPoint(2)

```

```

Loop
If IsEmpty(PtArray) = True Then Exit Sub
If UBound(PtArray) < 5 Then Exit Sub
Set LeaderObj = Nothing
Set NewLeader = ThisDrawing.ModelSpace.AddLeader(PtArray, _
    LeaderObj, acLineWithArrow)
End Sub

```

2. Variante: Führungsobjekt mit *MText* als Anmerkungsobjekt als Spline mit Pfeil

```

Public Sub Sample_AddLeader2()
Dim NewLeader As AcadLeader, NewMText As AcadMText
Dim PtArray, Prompt As String, LastPt(2) As Double
On Local Error Resume Next
Do
    varPoint = Empty
    If IsEmpty(PtArray) = True Then
        Prompt = "ersten Führungspunkt wählen:"
        varPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Else
        Prompt = "nächsten Führungspunkt wählen:"
        varPoint = ThisDrawing.Utility.GetPoint(LastPt, Prompt)
    End If
    If IsEmpty(varPoint) = True Then Exit Do
    If IsEmpty(PtArray) = True Then
        ReDim PtArray(2) As Double
        PtArray(0) = varPoint(0): PtArray(1) = varPoint(1)
        PtArray(2) = varPoint(2)
    Else
        ReDim Preserve PtArray(UBound(PtArray) + 3) As Double
        PtArray(UBound(PtArray) - 2) = varPoint(0)
        PtArray(UBound(PtArray) - 1) = varPoint(1)
        PtArray(UBound(PtArray)) = varPoint(2)
    End If
    LastPt(0) = varPoint(0): LastPt(1) = varPoint(1)
    LastPt(2) = varPoint(2)
Loop
If IsEmpty(PtArray) = True Then Exit Sub
If UBound(PtArray) < 5 Then Exit Sub
Prompt = "Anmerkungstext eingeben:"
TextString = ThisDrawing.Utility.GetString(1, Prompt)
Set NewMText = ThisDrawing.ModelSpace.AddMText(LastPt, 10, _
    TextString)
Set NewLeader = ThisDrawing.ModelSpace.AddLeader(PtArray, _
    NewMText, acSplineWithArrow)
End Sub

```

Siehe auch *Kapitel 8.3.9*

Linien erstellen

Eine Linie, die durch ihren Anfangs- und Endpunkt definiert wird, zum Modell bzw. Papierbereich hinzuzufügen, ist eine recht einfache Sache. Da eine Linie für ihre Darstellung außer den beiden Endpunkten keine weiteren Informationen benötigt, benötigt die Funktion *AddLine*

auch nur diese beiden Parameter. Sowohl der Start- als auch der Endpunkt der Linie werden wieder jeweils als 3D-Array mit den drei Feldern für X-, Y- und Z-Koordinate definiert und mit der Prozedur *AddLine* ausgeführt. Das so erzeugte Objekt entspricht dem Datentyp *AcadLine* und kann anschließend weiter angepasst werden.

AddLine

Parameter: *StartPoint*, *EndPoint*

Rückgabety: *AcadLine*

```
Public Sub Sample_AddLine()
Dim NewLine As AcadLine
Dim StartPt As Variant, Prompt As String, EndPt As Variant
Dim LastPt(2) As Double
On Local Error Resume Next
Do
    If IsEmpty(StartPt) = True Then
        Prompt = "Startpunkt wählen:"
        StartPt = ThisDrawing.Utility.GetPoint(, Prompt)
        If IsEmpty(StartPt) = True Then Exit Sub
        LastPt(0) = StartPt(0)
        LastPt(1) = StartPt(1)
        LastPt(2) = StartPt(2)
    Else
        EndPt = Empty
        Prompt = "nächsten Linienpunkt wählen:"
        EndPt = ThisDrawing.Utility.GetPoint(LastPt, Prompt)
        If IsEmpty(EndPt) = True Then Exit Sub
        Set NewLine = ThisDrawing.ModelSpace.AddLine(LastPt, EndPt)
        LastPt(0) = EndPt(0)
        LastPt(1) = EndPt(1)
        LastPt(2) = EndPt(2)
    End If
Loop
End Sub
```

Siehe auch *Kapitel 7.3.11*

Mehrere Instanzen eines Blocks in einer rechteckigen Anordnung erstellen

Sie können eine rechteckige Anordnung eines Blocks erzeugen, indem Sie die Anweisung *ArrayRectangular* auf eine Blockreferenz anwenden oder ein Objekt der Klasse *AcadMInsertBlock* dem Modell- oder Papierbereich hinzufügen. Der Unterschied besteht darin, dass *AcadMInsertBlock* ein zusammenhängendes Objekt darstellt und, anders als eine normale Blockreferenz, nicht aufgelöst werden kann. Um solch ein Objekt mit VBA zu erzeugen, müssen Sie die Funktion *AddMInsertBlock* des entsprechenden Bereichs verwenden. Diese Funktion verlangt eine ganze Reihe von Übergabeparametern, die alle einen bestimmten Einfluss auf die Darstellung des Objekts nehmen. Schauen wir uns diese Parameter einmal im Einzelnen an:

InsertionPoint – Legt den Einfügepunkt des Multiblocks fest. Der Punkt muss als 3D-Array definiert sein und die X-, Y- und Z-Koordinate enthalten. Von diesem Einfügepunkt ausgehend, bauen alle weiteren Parameter auf.

Name – Übergibt den Namen des Blocks, mit dem der Multiblock erzeugt werden soll. Sie müssen einen Blocknamen verwenden, der in der Blockauflistung enthalten ist. Sie können nicht, wie bei einer Blockreferenz, den Verzeichnisnamen vor den Blocknamen stellen, um diesen in die Zeichnung zu laden.

XScale, *YScale*, *ZScale* – Diese drei Parameter legen den Skalierfaktor der einzelnen Blockelemente, nicht aber den des gesamten Multiblocks fest.

Rotation – Der Drehwinkel, mit dem das Objekt eingefügt werden soll, wird mit diesem Parameter festgelegt. Der Winkel wird immer ausgehend vom Einfügepunkt in Spaltenrichtung angebracht.

NumRows – Legt die Anzahl der Zeilen fest. Wenn Sie z.B. den Wert 2 für diesen Parameter vergeben, also zwei Zeilen erzeugen wollen, dann erhalten Sie zwei übereinander liegende Reihen des angegebenen Blocks.

NumColumns – Legt die Anzahl der Spalten fest. Wenn Sie z.B. den Wert 2 für diesen Parameter vergeben, also zwei Spalten erzeugen wollen, dann erhalten Sie zwei nebeneinander liegende Reihen des angegebenen Blocks.

RowSpacing, *ColumnSpacing* – Legt den Abstand zwischen den einzelnen Blockdarstellungen in Zeile (*RowSpacing*) oder Spalte (*ColumnSpacing*) fest. Leider hat man bei der Erstellung der AutoCAD-Typenbibliothek nicht auf den Datentyp dieser Parameter geachtet, so dass nur Ganzzahlen übergeben werden können. Diese lassen sich aber nachträglich korrigieren, denn die Eigenschaften *RowSpacing* und *ColumnSpacing* der Klasse *AcadMInsertBlock* sind im korrekten Datentyp (*Double*) deklariert worden.

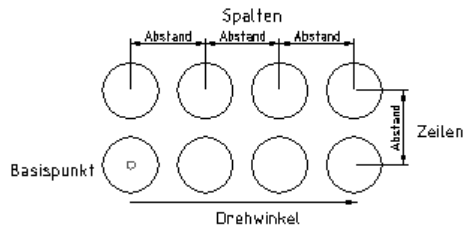


Abbildung 6.15: Multiblockreferenz

AddMInsertBlock

Parameter: *InsertionPoint*, *Name As String*, *Xscale As Double*, *Yscale As Double*, *Zscale As Double*, *Rotation As Double*, *NumRows As Long*, *NumColumns As Long*, *RowSpacing As Long*, *ColumnSpacing As Long*

Rückgabetyyp: *AcadMInsertBlock*

```
Public Sub Sample_AddMInsertBlock()
    Dim NewBlkRef As AcadMInsertBlock
    Dim InsPoint As Variant, Prompt As String, Angle As Double
    Dim Center(2) As Double, Rows As Long, Cols As Long
    Dim NewBlk As AcadBlock, RowSpace As Double, ColSpace As Double
    On Local Error Resume Next
    If TypeName(ThisDrawing.Blocks("NBlock")) = "Nothing" Then
        Set NewBlk = ThisDrawing.Blocks.Add(Center, "NBlock")
    End If
End Sub
```

```

Center(0) = 0: Center(2) = 0: Center(2) = 0
NewBlk.AddCircle Center, 1
End If
Prompt = "Einfügapunkt wählen:"
InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(InsPoint) = True Then Exit Sub
Prompt = "Drehwinkel wählen:"
Angle = ThisDrawing.Utility.GetOrientation(InsPoint, Prompt)
Prompt = "Anzahl Zeilen angeben:"
Rows = ThisDrawing.Utility.GetInteger(Prompt)
If Rows = 0 Then Rows = 1
If Rows > 1 Then
    Prompt = "Abstand zwischen den Zeilen angeben(---):"
    RowSpace = ThisDrawing.Utility.GetReal(Prompt)
    If RowSpace = 0 Then RowSpace = 1
End If
Prompt = "Anzahl Spalten angeben:"
Cols = ThisDrawing.Utility.GetInteger(Prompt)
If Cols = 0 Then Cols = 1
If Cols > 1 Then
    Prompt = "Abstand zwischen den Spalten angeben(|||):"
    ColSpace = ThisDrawing.Utility.GetReal(Prompt)
    If ColSpace = 0 Then ColSpace = 1
End If
Set NewBlkRef = ThisDrawing.ModelSpace.AddMInsertBlock( _
    InsPoint, "NBlock", 1, 1, 1, Angle, Rows, Cols, RowSpace, _
    ColSpace)
End Sub

```

Siehe auch *Kapitel 8.3.30*.

Multilinien erstellen

Die Multiline ist eine Art parallele Polylinie. Sie wird über mehrere Stützpunkte gezeichnet und bildet so einen zusammenhängenden Linienzug, der bis zu 16 parallele Linien enthalten kann. Da die *StyleName*-Eigenschaft, die den Multiliniestil bestimmt, schreibgeschützt ist, müssen Sie den Stil festlegen, bevor Sie eine Multiline in den entsprechenden Bereich (Modell/ Papier) einfügen, da eine spätere Änderung am Multilinienobjekt nicht möglich ist. Sie können den Namen des Multiliniestils über die Systemvariable *CMLSTYLE* festlegen oder abrufen. Außerdem können Sie mit der Systemvariablen *CMLJUST* die Ausrichtung und mit *CMLSCALE* die Skalierung der Multilinen festlegen.

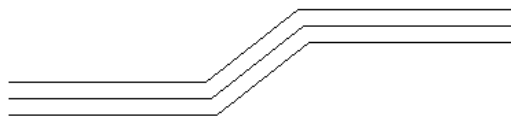


Abbildung 6.16: Multiline

Um nun die Multiline über VBA zu erzeugen, muss für den erforderlichen Parameter der Funktion *AddMLine* ein 3D-Array mit Koordinaten bestimmt werden, das die Stützpunkte der Multiline festlegt. Da die Klasse *AcadMLine*, die aus dieser Funktion resultiert, über

keine *AppendVertex*-Funktion oder ähnlich verfügt, müssen Sie die Koordinatenliste der Multilinie mit der *ReDim*-Anweisung neu dimensionieren. Wie das gemacht wird, entnehmen Sie dem nachfolgenden Beispiel.

AddMLine

Parameter: *VertexList*

Rückgabetypp: *AcadMLine*

```
Public Sub Sample_AddMLine()
Dim NewMLine As AcadMLine
Dim StartPt As Variant, Prompt As String, EndPt As Variant
Dim LastPt(2) As Double, MStilName As String, VertList
Dim MStilScale As Double
On Local Error Resume Next
MStilName = ThisDrawing.GetVariable("CMLSTYLE")
Prompt = "Multiliniestil <" & MStilName & ">:"
MStilName = ThisDrawing.Utility.GetString(0, Prompt)
ThisDrawing.SetVariable "CMLSTYLE", MStilName
MStilScale = ThisDrawing.GetVariable("CMLSCALE")
Prompt = "Multilinienskalierung <" & _
    ThisDrawing.Utility.RealToString(MStilScale, acDefaultUnits, _
    3) & ">:"
MStilScale = ThisDrawing.Utility.GetReal(Prompt)
ThisDrawing.SetVariable "CMLSCALE", MStilScale
Do
    If IsEmpty(StartPt) = True Then
        Prompt = "Startpunkt wählen:"
        StartPt = ThisDrawing.Utility.GetPoint(, Prompt)
        If IsEmpty(StartPt) = True Then Exit Sub
        LastPt(0) = StartPt(0)
        LastPt(1) = StartPt(1)
        LastPt(2) = StartPt(2)
    Else
        EndPt = Empty
        Prompt = "nächsten Linienpunkt wählen:"
        EndPt = ThisDrawing.Utility.GetPoint(LastPt, Prompt)
        If IsEmpty(EndPt) = True Then Exit Sub
        If TypeName(NewMLine) = "Nothing" Then
            ReDim VertList(5) As Double
            VertList(0) = StartPt(0)
            VertList(1) = StartPt(1)
            VertList(2) = StartPt(2)
            VertList(3) = EndPt(0)
            VertList(4) = EndPt(1)
            VertList(5) = EndPt(2)
            Set NewMLine = ThisDrawing.ModelSpace.AddMLine(VertList)
        Else
            ReDim Preserve VertList(UBound(NewMLine.Coordinates) _
                + 3) As Double
            VertList(UBound(VertList) - 2) = EndPt(0)
            VertList(UBound(VertList) - 1) = EndPt(1)
            VertList(UBound(VertList)) = EndPt(2)
            NewMLine.Coordinates = VertList
        End If
    End If
Loop
```



```

        NewMLine.Update
    End If
    LastPt(0) = EndPt(0)
    LastPt(1) = EndPt(1)
    LastPt(2) = EndPt(2)
End If
Loop
End Sub

```

Siehe auch *Kapitel 8.3.13*

Mehrzeiligen Text erstellen

Um einen mehrzeiligen Text für lange, komplexe Textpassagen zu erstellen, steht Ihnen sowohl im Modell- als auch im Papierbereich die Funktion *AddMText* zur Verfügung. Unter Verwendung der folgenden Parameter erstellt diese Funktion ein Objekt der Klasse *AcadMText*. *InsertionPoint* übergibt den Einfügapunkt für das *MText*-Objekt und muss wie immer als 3D-Array erstellt werden. Mit dem zweiten Parameter legen Sie die Breite des Textobjekts fest. Der dritte Parameter bestimmt schließlich die Textzeichenfolge, die im Absatztext dargestellt werden soll.

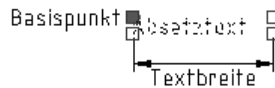


Abbildung 6.17: *MText*-Objekt

AddMText

Parameter: *InsertionPoint*, *Width As Double*, *Text As String*

Rückgabetyt: *AcadMText*

```

Public Sub Sample_AddMText()
    Dim NewMText As AcadMText
    Dim InsPoint As Variant, Prompt As String
    Dim TextWidth As Double, TextString As String
    On Local Error Resume Next
    Prompt = "Einfügapunkt des Absatztextes:"
    InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(InsPoint) = True Then Exit Sub
    Prompt = "Breite des Absatztextes:"
    TextWidth = ThisDrawing.Utility.GetDistance(InsPoint, Prompt)
    If TextWidth = 0 Then Exit Sub
    Prompt = "Text eingeben:"
    TextString = ThisDrawing.Utility.GetString(1, Prompt)
    If TextString = "" Then Exit Sub
    Set NewMText = ThisDrawing.ModelSpace.AddMText(InsPoint, _
        TextWidth, TextString)
End Sub

```

Siehe auch *Kapitel 8.3.16*

Punktobjekt erstellen

Ein Punkt stellt ein sehr einfaches grafisches Objekt in AutoCAD dar, das daher auch nur geringe Anpassungsmöglichkeiten besitzt. Aus diesem Grund benötigt die Funktion *AddPoint*, die einen solchen Punkt im Modell- oder Papierbereich erzeugt, nur einen einzigen Parameter. Mit diesem Parameter legen Sie den Einfügepunkt des Punktobjekts fest, weshalb für den Parameter wieder ein 3D-Array erforderlich ist, das die X-, Y- und Z-Koordinate des Einfügepunkts übernimmt.

AddPoint

Parameter: *Point*

Rückgabetyt: *AcadPoint*

```
Public Sub Sample_AddPoint()
Dim NewPoint As AcadPoint
Dim InsPoint As Variant
On Local Error Resume Next
Prompt = "Einfügepunkt wählen:"
Do
    InsPoint = Empty
    InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(InsPoint) = True Then Exit Sub
    Set NewPoint = ThisDrawing.ModelSpace.AddPoint(InsPoint)
Loop
End Sub
```

Siehe auch *Kapitel 8.3.14*

Vielflächennetz erstellen

Die Erstellung eines Vielflächennetzes erfolgt ähnlich wie die eines rechteckigen Netzes. Sie geben zum Erstellen eines Vielflächennetzes die Koordinaten für die Kontrollpunkte an. Anschließend definieren Sie die einzelnen Flächen, indem Sie für jeden der Scheitelpunkte eine Nummer eingeben. Beim Erstellen des Vielflächennetzes können Sie bestimmte Kanten als unsichtbar definieren oder sie Layern oder Farben zuordnen. Der erste Parameter, den Sie an die *Add*-Funktion übergeben müssen, enthält alle Stützpunkte des Netzes. Mit dem zweiten Parameter fügen Sie diese Stützpunkte zu einzelnen Flächen zusammen, die jeweils aus vier Punkten bestehen müssen.

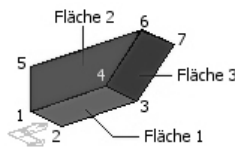


Abbildung 6.18: Vielflächennetz

Um die einzelnen Netze des Bilds zusammenzustellen, müsste die Definition der Flächenliste wie folgt vorgenommen werden:

3 Flächen mal 4 Stützpunkte pro Fläche ergibt:

```
Dim FaceList(0 To 11) As Integer
```

1. Fläche:

```

FaceList(0) = 1 'von Stützpunkt 1
FaceList(1) = 2 'nach Stützpunkt 2
FaceList(2) = 3 'nach Stützpunkt 3
FaceList(3) = 4 'nach Stützpunkt 4

```

2. Fläche:

```

FaceList(4) = 1 'von Stützpunkt 1
FaceList(5) = 4 'nach Stützpunkt 4
FaceList(6) = 6 'nach Stützpunkt 6
FaceList(7) = 5 'nach Stützpunkt 5

```

3. Fläche:

```

FaceList(8) = 4 'von Stützpunkt 4
FaceList(9) = 3 'nach Stützpunkt 3
FaceList(10) = 7 'nach Stützpunkt 7
FaceList(11) = 6 'nach Stützpunkt 6

```

AddPolyfaceMesh

Parameter: *VertexList*, *FaceList*

Rückgabetypp: *AcadPolyfaceMesh*

```

Public Sub Sample_AddPolyfaceMesh()
Dim NewPFMesh As AcadPolyfaceMesh
Dim InsPoint As Variant, Prompt As String
Dim Pnt, VertList, FaceList, i As Integer
On Local Error Resume Next
Pnt1 = Empty
Prompt = "Position 1. Kontrollpunkt:"
Pnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Pnt1) = True Then Exit Sub
ReDim VertList(2) As Double
VertList(0) = Pnt1(0): VertList(1) = Pnt1(1)
VertList(2) = Pnt1(2)
i = 2
Do
    Pnt1 = Empty
    Prompt = "Position " & CStr(i) & ". Kontrollpunkt:"
    Pnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(Pnt1) = True Then Exit Do
    ReDim Preserve VertList(UBound(VertList) + 3) As Double
    VertList(UBound(VertList) - 2) = Pnt1(0)
    VertList(UBound(VertList) - 1) = Pnt1(1)
    VertList(UBound(VertList)) = Pnt1(2)
    i = i + 1
Loop
ReDim FaceList(3) As Integer
k = 1: f = 1: i = 0
Do
    ThisDrawing.Utility.InitializeUserInput 128, "eXit"
    Prompt = "Kontrollpunkt " & CStr(k) & " der " & CStr(f) & _
        ". Fläche oder <eXit>:"

```

```

getfacenumber:
  FaceList(i) = ThisDrawing.Utility.GetInteger(Prompt)
  If ThisDrawing.Utility.GetInput = "eXit" Then Exit Do
  If FaceList(i) < -((UBound(VertList) + 1) / 3) Or FaceList(i) _
    > ((UBound(VertList) + 1) / 3) Then
    Prompt = "Sie müssen einen Wert zwischen " & _
      CStr(-((UBound(VertList) + 1) / 3)) & " und " & _
      CStr((UBound(VertList) + 1) / 3) & " eingeben"
    ThisDrawing.Utility.Prompt Prompt
    GoTo getfacenumber
  End If
  i = i + 1
  If k = 4 Then
    k = 1: f = f + 1
    ReDim Preserve FaceList(UBound(FaceList) + 4) As Integer
  Else
    k = k + 1
  End If
Loop
Set NewPFMesh = ThisDrawing.ModelSpace.AddPolyfaceMesh(VertList, _
  FaceList)
End Sub

```

Siehe auch *Kapitel 8.3.21*

Papierbereichsansichtsfenster erstellen

(nur im Papierbereich möglich)

Ansichtsfenster sind Ausschnitte des Modellbereichs, die im Papierbereich dargestellt werden. Da dieses Objekt nur im Papierbereich erstellt werden kann, können Sie die *Add*-Funktion auch nur über das *PaperSpace*-Objekt aufrufen, da diese Funktion in der Klasse *AcadModelSpace* nicht definiert wurde.

```
ThisDrawing.PaperSpace.AddViewport Center, Width, Height
```

Wenn Sie nur das Ansichtsfenster erzeugen wollen, können Sie dies auch tun, wenn sich die Zeichnung im Modellbereich befindet. Zum Erstellen benötigt die Funktion den Mittelpunkt des Fensters und die Breite und Höhe des Ansichtsfensters. Der Zentrumsunkt nimmt wieder die drei Koordinaten (X, Y und Z) auf und muss aus diesem Grund als 3D-Array definiert sein. Für die Breite und Höhe können Sie positive reelle Zahlen verwenden.

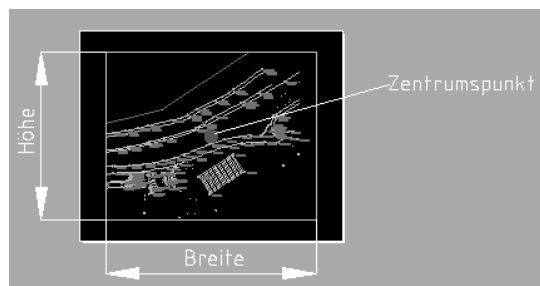


Abbildung 6.19: Papierbereichsansichtsfenster

AddPViewportParameter: *Center, Width As Double, Height As Double*Rückgabotyp: *AcadPViewport*

```

Public Sub Sample_AddPViewport()
Dim NVP As AcadPViewport
Dim NV As AcadView
Dim Center(2) As Double
Dim Prompt As String, FrameObj As AcadPolyline
Dim Pnt1 As Variant, Pnt2 As Variant
Dim Pnt3 As Variant, Pnt4 As Variant
Dim VertList(11) As Double, Diagonale As Double
Dim LPoint(2) As Double, RPoint(2) As Double
Dim Height As Double, Width As Double
Dim LayoutName As String
Const PI = 3.14159265358979
On Local Error Resume Next
Prompt = "linke untere Ecke des Bereiches:"
Pnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Pnt1) = True Then Exit Sub
Prompt = "rechte obere Ecke des Bereiches:"
Pnt3 = ThisDrawing.Utility.GetCorner(Pnt1, Prompt)
If IsEmpty(Pnt3) = True Then Exit Sub
Angle = ThisDrawing.GetVariable("VIEWTWIST") + _
    ThisDrawing.Utility.AngleFromXAxis(Pnt1, Pnt3)
Diagonale = Sqr((Cdbl(Pnt1(0)) - Cdbl(Pnt3(0))) ^ 2 + _
    (Cdbl(Pnt1(1)) - Cdbl(Pnt3(1))) ^ 2)
Height = Abs(Sin(Angle) * Diagonale)
Width = Sqr(Diagonale ^ 2 - Height ^ 2)
Angle = 2 * PI - ThisDrawing.GetVariable("VIEWTWIST")
Pnt2 = ThisDrawing.Utility.PolarPoint(Pnt1, Angle, Width)
Pnt4 = ThisDrawing.Utility.PolarPoint(Pnt3, Angle + PI, Width)
VertList(0) = Pnt1(0)
VertList(1) = Pnt1(1)
VertList(2) = 0
VertList(3) = Pnt2(0)
VertList(4) = Pnt2(1)
VertList(5) = 0
VertList(6) = Pnt3(0)
VertList(7) = Pnt3(1)
VertList(8) = 0
VertList(9) = Pnt4(0)
VertList(10) = Pnt4(1)
VertList(11) = 0
Set FrameObj = ThisDrawing.ModelSpace.AddPolyline(VertList)
FrameObj.Closed = True
Prompt = "Gewählten Bereich übertragen in Layout:"
LayoutName = ThisDrawing.Utility.GetString(1, Prompt)
ThisDrawing.ActiveLayout = ThisDrawing.Layouts(LayoutName)
LP = FrameObj.Coordinate(0)
LPoint(0) = LP(0)
LPoint(1) = LP(1)
LPoint(2) = LP(2)

```

```
RP = FrameObj.Coordinate(1)
RPoint(0) = RP(0)
RPoint(1) = RP(1)
RPoint(2) = RP(2)
ThisDrawing.ActiveLayout.PaperUnits = acMillimeters
Width = Sqr((Cdbl(LP(0)) - Cdbl(RP(0))) ^ 2 + (Cdbl(LP(1)) - Cdbl(RP(1))) ^ 2)
LP = FrameObj.Coordinate(2)
Height = Sqr((Cdbl(LP(0)) - Cdbl(RP(0))) ^ 2 + (Cdbl(LP(1)) - Cdbl(RP(1))) ^ 2)
ThisDrawing.ActiveLayout.SetCustomScale 1, Cdbl(TextBox1.Text) _
/ 1000
ThisDrawing.Regen acActiveViewport
ActFrame.Elevation = 0
ThisDrawing.MSpace = False
varPoint = ThisDrawing.ActiveLayout.Block.Item(0).Center
Center(0) = Center(0) / 2
Center(1) = Center(1) / 2
Set NVP = ThisDrawing.PaperSpace.AddPViewport(Center, Width, _
Height)
NVP.ViewportOn = True
NVP.TwistAngle = (2 * PI) - ThisDrawing.Utility.AngleFromXAxis( _
LPoint, RPoint)
ThisDrawing.MSpace = True
ThisDrawing.Application.ZoomWindow LPoint, RPoint
ThisDrawing.ActivePViewport = NVP
ThisDrawing.MSpace = False
End Sub
```

Siehe auch *Kapitel 7.12.2*

Pixelbilder einfügen

Um ein Pixel- oder auch Rasterbild in eine Zeichnung einzufügen, benötigen Sie für die *Add-* Funktion zunächst einmal den Namen (inklusive des Verzeichnisses) des Bildes, das eingefügt werden soll. Diesen könnten Sie z.B. mit dem *CommonDialog*-Steuerelement von Microsoft durch den Benutzer auswählen lassen. Folgende Bildformate werden dabei von AutoCAD unterstützt.

Typ (Dateiendung)	Beschreibung
BMP (.BMP, .RLE, .DIB)	Windows- und OS/2-Bitmap-Format
CALS-I (.RST, .GP4, .MIL, .CAL, .CG4)	Computer Aided Acquisition and Logistics Support
FLIC (.flc, .fli)	Digitales Autodesk-Animationsformat
Geospot (.bil)	Corporation-Format mit geografischen Referenzdaten
GeoTIFF (.tif)	TIFF mit geografischen Referenzdaten
IG4 (.ig4)	Image Systems Group 4
IGS (.igs)	Image Systems Gray Scale
JFIF (.jpg)	JPEG File Interchange Format

Tabelle 6.6: Unterstützte Bildformate

Typ (Dateiendung)	Beschreibung
JPEG (.jpg)	Joint Photographic Experts Group
PCX (.pcx)	PC Paintbrush Exchange
PICT (.pct)	Macintosh PICT1, PICT2
PNG (.png)	Portable Network Graphics
RLC (.rlc)	RLC-Format
TARGA (.tga)	TrueVision-Bilddateiformat
TIFF (.tif)	Tagged Image File Format

Tabelle 6.6: Unterstützte Bildformate (Forts.)

Sie können die Dateierweiterungen der Filtereigenschaft des Steuerelements übergeben, um nur die entsprechenden Dateien anzeigen zu lassen, die den Dateierweiterungen entsprechen. Als Zweites benötigen Sie einen Einfügekpunkt für das Bild. Dieser Punkt muss wieder als 3D-Array definiert sein. Im dritten Parameter legen Sie schließlich den Skalierfaktor für das Bild fest. Dieser Faktor passt die Größe des Bilds in Bezug auf die Originalwerte und ausgehend vom Einfügekpunkt an. Der letzte Parameter bestimmt den Drehwinkel des Bilds.

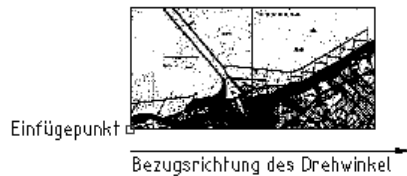


Abbildung 6.20: Pixelbild

AddRaster

Parameter: *ImageFileName* As String, *InsertionPoint*, *ScaleFactor* As Double, *RotationAngle* As Double

Rückgabotyp: *AcadRaster*

```
Public Sub Sample_AddRaster()
Dim NewRaster As AcadRasterImage
Dim InsPoint As Variant, Prompt As String
Dim ScaleFactor As Double, Angle As Double
Dim ComDialog As CommonDialog
On Local Error Resume Next
Set ComDialog = New CommonDialog
ComDialog.CancelError = True
ComDialog.Filter = "Rasterdateien|*.bmp;*.rlc;*.dib;*.rst;*.gp4" _
    & " ;*.mil;*.cal;*.cg4;*.flc;*.fli;*.bil;*.tif;*.ig4;*.igs" _
    & " ;*.jpg ;*.pcx;*.pct;*.png;*.rlc;*.tga|"
ComDialog.DialogTitle = "Rasterdatei einfügen"
ComDialog.ShowOpen
If Err = 0 Then
    Prompt = "Einfügekpunkt des Rasterbildes:"
```

```

InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
Prompt = "Einfügepunkt des Rasterbildes"
If IsEmpty(InsPoint) = True Then Exit Sub
Prompt = "Drehwinkel des Rasterbildes:"
Angle = ThisDrawing.Utility.GetOrientation(InsPoint, Prompt)
Prompt = "Skalierfaktor des Rasterbildes<1.000>:"
ScaleFactor = ThisDrawing.Utility.GetReal(Prompt)
If ScaleFactor = 0 Then ScaleFactor = 1
Err.Clear
Set NewRaster = ThisDrawing.ModelSpace.AddRaster( _
    ComDialog.FileName, InsPoint, ScaleFactor, Angle)
If Err <> 0 Then MsgBox Err.Description, vbCritical, "Fehler"
End If
End Sub

```

Siehe auch *Kapitel 8.3.23*

Strahl erstellen

Für die Erzeugung einer einseitig begrenzten Linie stellen sowohl das *Document*-Objekt *ModelSpace* als auch das Objekt *PaperSpace* die Funktion *AddRay* zur Verfügung. Für die Verwendung dieser Funktion benötigen Sie lediglich den Startpunkt des Strahls und einen weiteren Punkt, der die Richtung des Strahls festlegt. Dabei müssen diese beiden Punkte einem 3D-Array entsprechen. Nachdem Sie mit der *Add*-Funktion ein Objekt der Klasse *AcadRay* erzeugt haben, wird in AutoCAD eine Linie gezeichnet, die ausgehend vom angegebenen Startpunkt durch den zweiten Punkt verläuft.

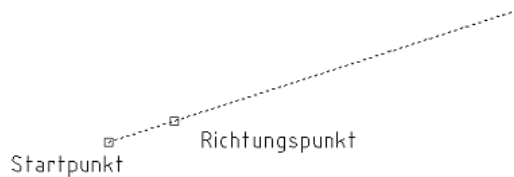


Abbildung 6.21: Strahl-Objekt

AddRay

Parameter: *Point1*, *Point2*

Rückgabety: *AcadRay*

```

Public Sub Sample_AddRay()
Dim NewRay As AcadRay
Dim StartPt As Variant, Prompt As String, EndPt As Variant
Dim LastPt(2) As Double
On Local Error Resume Next
Do
    If IsEmpty(StartPt) = True Then
        Prompt = "Startpunkt wählen:"
        StartPt = ThisDrawing.Utility.GetPoint(, Prompt)
        If IsEmpty(StartPt) = True Then Exit Sub
    Else

```



```

EndPt = Empty
Prompt = "Richtung des Strahls festlegen:"
EndPt = ThisDrawing.Utility.GetPoint(StartPt, Prompt)
If IsEmpty(EndPt) = True Then Exit Sub
Set NewRay = ThisDrawing.ModelSpace.AddRay(StartPt, EndPt)
End If
Loop
End Sub

```

Siehe auch *Kapitel 8.3.17*

Region(en) erstellen

Regionen oder begrenzte plane Flächen können erstellt werden, indem Sie der Funktion *AddRegion* ein Array mit grafischen Objekten übergeben, die eine oder mehrere *geschlossene* Umgrenzung(en) bilden. Für die Erstellung der Umgrenzung(en) können Sie Linien, Kreise, Bögen, elliptische Bögen und Splines verwenden. Das Besondere an dieser Funktion ist, dass nicht nur ein einzelnes Objekt erstellt werden kann, sondern gleichzeitig mehrere Regionen. Wie viele Regionen durch die Funktion erstellt werden, hängt davon ab, wie viele geschlossene Umgrenzungen durch die Objekte des Arrays gebildet werden können. Sie können auf die einzelnen erstellten Regionen zugreifen, indem Sie auf das entsprechende Objekt des zurückgegebenen Arrays zugreifen:

```
NewRegions(0).Layer = "0"
```

oder durch Erstellung eines Verweises an eine Objektvariable:

```
Set NewRegion = NewRegions(0)
```

Die Anzahl der erstellten Regionen könnten Sie z.B. durch die *UBound()*-Funktion ermitteln, die die Obergrenze des Arrays zurückliefert. Daraus folgt, dass ein Abruf aller Regionen über eine *For...Next*-Schleife erfolgen kann:

```

For i = 0 To UBound(NewRegions)
    Set NewRegion = NewRegions(i)
Next

```

Tipp

Nachdem Sie die Region(en) erfolgreich erstellt haben, sollten Sie die Objekte, die zur Bildung der Region(en) gedient haben, löschen, da dies nicht automatisch von AutoCAD vorgenommen wird.

AddRegion

Parameter: *ObjectList*

Rückgabetyt: *Variant* (Array mit den erstellten Regionen)

```

Public Sub Sample_AddRegion()
Dim NewRegions
Dim SSet As AcadSelectionSet
Dim ObjList
On Local Error Resume Next
Set SSet = ThisDrawing.ActiveSelectionSet
SSet.SelectOnScreen
If SSet.Count > 0 Then

```

```

ReDim ObjList(SSet.Count - 1) As AcadEntity
For i = 0 To SSet.Count - 1
    Set ObjList(i) = SSet.Item(i)
Next
NewRegions = ThisDrawing.ModelSpace.AddRegion(ObjList)
If IsEmpty(NewRegion) = True Then
    MsgBox Err.Description
Else
    MsgBox UBound(NewRegions) + 1 & " Region(en) erstellt!", _
        vbInformation, "Regionen"
End If
End If
End Sub

```

Siehe auch *Kapitel 8.3.24*

Symbolobjekt erstellen

Symbole sind Objekte, die auf Vorlagen aus einer Symboldefinitionsdatei (*shp*) in AutoCAD erstellt werden können. Voraussetzung für die erfolgreiche Erstellung ist also eine vorhandene Definition des Symbols, dessen Namen Sie im ersten Parameter der Funktion übergeben. Wenn Sie einen Symbolnamen verwenden, der zu einer externen Referenz (XRef) gehört, die der aktuellen Zeichnung zugewiesen ist, kennzeichnet AutoCAD die Symboldatei als extern abhängig. Sie können extern abhängige Symbole nur dann in der aktuellen Zeichnung verwenden, wenn diese vorher neu geladen wurden.

Der zweite Parameter legt den Einfügepunkt des Symbolobjekts fest und muss als 3D-Array definiert sein. Mit den letzten beiden Parametern übergeben Sie der Funktion den Skalierfaktor, der einen Wert > 0 enthalten muss, und den Drehwinkel des Symbolobjekts. Nachdem Sie das Symbol erfolgreich erstellt haben, erhalten Sie ein Objekt der Klasse *AcadShape*, dessen Eigenschaften Sie weiter anpassen können.

AddShape

Parameter: *Name As String, InsertionPoint, ScaleFactor As Double, RotationAngle As Double*

Rückgabebetyp: *AcadShape*

```

Public Sub Sample_AddShape()
Dim NewShape As AcadShape
Dim InsPoint As Variant, Prompt As String
Dim ScaleFactor As Double, Angle As Double, ShapeName As String
On Local Error Resume Next
Prompt = "Einfügepunkt wählen:"
InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(InsPoint) = True Then Exit Sub
Prompt = "Symbolname eingeben:"
ShapeName = ThisDrawing.Utility.GetString(0, Prompt)
Prompt = "Skalierfaktor:"
ScaleFactor = ThisDrawing.Utility.GetDistance(InsPoint, Prompt)
If ScaleFactor = 0 Then ScaleFactor = 1
Prompt = "Drehwinkel:"
Angle = ThisDrawing.Utility.GetOrientation(InsPoint, Prompt)

```

```

Do
    Set NewShape = ThisDrawing.ModelSpace.AddShape(ShapeName, _
        InsPoint, ScaleFactor, Angle)
    If TypeName(NewShape) = "Nothing" Then Exit Sub
    InsPoint = Empty
    Prompt = "Einfügepunkt wählen:"
    InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(InsPoint) = True Then Exit Sub
Loop
End Sub

```

Siehe auch *Kapitel 8.3.15*

2D-Volumenkörperpolygon erstellen

Ein Solid-Objekt mit VBA in AutoCAD zu erzeugen, ist wieder eine ganz simple Angelegenheit. Was Sie hierfür benötigen, sind lediglich die vier Eckpunkte, die das Solid-Objekt begrenzen. Alle vier Punkte müssen jeweils als 3D-Array definiert sein. Worauf der Benutzer achten muss, ist die Lage der Stützpunkte. Diese sind anders als z.B. bei Polylinien nicht nacheinander, sondern gegenüberliegend positioniert, wie in nachfolgender Abbildung zu erkennen ist.

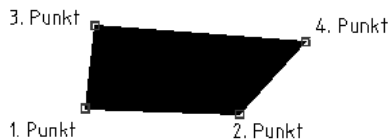


Abbildung 6.22: 2D-Volumenkörperpolygon

Um die Flächenfüllung eines Volumenkörperpolygons ein- bzw. auszuschalten, müssen Sie den Wert der Systemvariablen FILLMODE entsprechend anpassen:

- 1 – Flächen werden gefüllt.
- 0 – Nur die Verbindungen zwischen den einzelnen Stützpunkten werden angezeigt.

AddSolid

Parameter: *Point1, Point2, Point3, Point4*

Rückgabety: *AcadSolid*

1. Variante: Erstellung eines Solid-Objekts dem AutoCAD-Befehl *Solid* nachempfunden

```

Public Sub Sample_AddSolid1()
    Dim NewSolid As AcadSolid
    Dim InsPoint As Variant, Prompt As String
    Dim Pnt1, Pnt2, Pnt3, Pnt4
    On Local Error Resume Next
    Pnt1 = Empty
    Prompt = "Erster Stützpunkt:"
    Pnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(Pnt1) = True Then Exit Sub

```

```

Pnt2 = Empty
Prompt = "Zweiter Stützpunkt:"
Pnt2 = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Pnt2) = True Then Exit Sub
Do
    Pnt3 = Empty
    Prompt = "Dritter Stützpunkt:"
    Pnt3 = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(Pnt3) = True Then Exit Sub
    Pnt4 = Empty
    Prompt = "Vierter Stützpunkt:"
    Pnt4 = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(Pnt4) = True Then Exit Sub
    Set NewSolid = ThisDrawing.ModelSpace.AddSolid(Pnt1, Pnt2, _
        Pnt3, Pnt4)
    Pnt1(0) = Pnt3(0): Pnt1(1) = Pnt3(1): Pnt1(2) = Pnt3(2)
    Pnt2(0) = Pnt4(0): Pnt2(1) = Pnt4(1): Pnt2(2) = Pnt4(2)
Loop
End Sub

```

2.Variante: Erstellung des Solid-Objekts der Erzeugung einer Polylinie nachgestellt

```

Public Sub Sample_AddSolid2()
Dim NewSolid As AcadSolid
Dim InsPoint As Variant, Prompt As String
Dim Pnt1, Pnt2, Pnt3, Pnt4
On Local Error Resume Next
Pnt1 = Empty
Prompt = "Erster Stützpunkt:"
Pnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Pnt1) = True Then Exit Sub
Pnt2 = Empty
Prompt = "Zweiter Stützpunkt:"
Pnt2 = ThisDrawing.Utility.GetPoint(Pnt1, Prompt)
If IsEmpty(Pnt2) = True Then Exit Sub
Do
    Pnt4 = Empty
    Prompt = "Dritter Stützpunkt:"
    Pnt4 = ThisDrawing.Utility.GetPoint(Pnt2, Prompt)
    If IsEmpty(Pnt4) = True Then Exit Sub
    Pnt3 = Empty
    Prompt = "Vierter Stützpunkt:"
    Pnt3 = ThisDrawing.Utility.GetPoint(Pnt4, Prompt)
    If IsEmpty(Pnt3) = True Then Exit Sub
    Set NewSolid = ThisDrawing.ModelSpace.AddSolid(Pnt1, Pnt2, _
        Pnt3, Pnt4)
    Pnt1(0) = Pnt3(0): Pnt1(1) = Pnt3(1): Pnt1(2) = Pnt3(2)
    Pnt2(0) = Pnt4(0): Pnt2(1) = Pnt4(1): Pnt2(2) = Pnt4(2)
Loop
End Sub

```

Siehe auch *Kapitel 8.3.20*

Splines erzeugen

Splines sind quadratische oder kubische NURBS-Kurven (Non-Uniform Rational B-Spline), die anhand von Stützpunkten, Kontrollpunkten, Tangentenvektoren im Start- und Endsteuertpunkt definiert werden. Durch Anpassung verschiedener Eigenschaften kann die Darstellung der Splines optimiert werden. Für die Erzeugung eines Spline stellen sowohl *AcadModelSpace* als auch *AcadPaperSpace* die Funktion *AddSpline* zur Verfügung. Sie müssen dieser Funktion im ersten Parameter ein Array mit 3D-Koordinaten mitgeben, das die Liste der Steuertpunkte definiert. Um einen Spline zu erstellen, muss diese Liste mindestens zwei Steuertpunkte enthalten. Zusätzlich müssen Sie noch die Start- und Endtangente des Spline bestimmen. Diese werden jeweils als Richtungsvektoren, ausgehend vom Start- bzw. Endpunkt, in 3D-Arrays als zweiter und dritter Parameter übergeben.

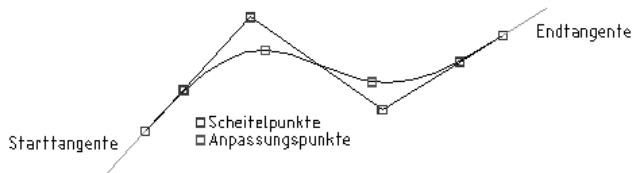


Abbildung 6.23: Spline

AddSpline

Parameter: *PointsArray*, *StartTangent*, *EndTangent*

Rückgabebetyp: *AcadSpline*

```
Public Sub Sample_AddSpline()
Dim NewSpline As AcadSpline
Dim StartPt As Variant, Prompt As String, EndPt As Variant
Dim LastPt(2) As Double, VertList
Dim StartTan, EndTan
On Local Error Resume Next
Do
    If IsEmpty(StartPt) = True Then
        Prompt = "Startpunkt wählen:"
        StartPt = ThisDrawing.Utility.GetPoint(, Prompt)
        If IsEmpty(StartPt) = True Then Exit Sub
        LastPt(0) = StartPt(0)
        LastPt(1) = StartPt(1)
        LastPt(2) = StartPt(2)
    Else
        EndPt = Empty
        Prompt = "nächsten Steuertpunkt wählen:"
        EndPt = ThisDrawing.Utility.GetPoint(LastPt, Prompt)
        If IsEmpty(VertList) = True Then
            ReDim VertList(5) As Double
            VertList(0) = StartPt(0)
            VertList(1) = StartPt(1)
            VertList(2) = StartPt(2)
            VertList(3) = EndPt(0)
```

```

VertList(4) = EndPt(1)
VertList(5) = EndPt(2)
Else
  If IsEmpty(EndPt) = True Then
    Prompt = "Starttangente:"
    StartTan = ThisDrawing.Utility.GetPoint(StartPt, _
      Prompt)
    If IsEmpty(StartTan) = True Then
      ReDim StartTan(2) As Double
      StartTan(0) = 0
      StartTan(1) = 0
      StartTan(2) = 0
    Else
      StartTan(0) = StartTan(0) - StartPt(0)
      StartTan(1) = StartTan(1) - StartPt(1)
      StartTan(2) = StartTan(2) - StartPt(2)
    End If
    Prompt = "Endtangente:"
    EndTan = ThisDrawing.Utility.GetPoint(LastPt, Prompt)
    If IsEmpty(EndTan) = True Then
      ReDim EndTan(2) As Double
      EndTan(0) = 0
      EndTan(1) = 0
      EndTan(2) = 0
    Else
      EndTan(0) = EndTan(0) - EndPt(0)
      EndTan(1) = EndTan(1) - EndPt(1)
      EndTan(2) = EndTan(2) - EndPt(2)
    End If
    Set NewSpline = ThisDrawing.ModelSpace.AddSpline( _
      VertList, StartTan, EndTan)
    Exit Do
  End If
  ReDim Preserve VertList(UBound(VertList) + 3) As Double
  VertList(UBound(VertList) - 2) = EndPt(0)
  VertList(UBound(VertList) - 1) = EndPt(1)
  VertList(UBound(VertList)) = EndPt(2)
End If
LastPt(0) = EndPt(0)
LastPt(1) = EndPt(1)
LastPt(2) = EndPt(2)
End If
Loop
End Sub

```

Siehe auch *Kapitel 8.3.16*

Einzeiligen Text erzeugen

Einzeiliger Text beinhaltet nur eine Zeichenfolge, die in einer einzigen Zeile angezeigt wird. Zur Erzeugung des Textobjekts benötigen wir natürlich die Zeichenfolge, die in der Zeichnung angezeigt werden soll. Diese wird im ersten Parameter übergeben. Wie immer wird auch der Einfügekpunkt benötigt, der wiederum als 3D-Array definiert wird und die Position des Textes bestimmt. Als Letztes erfordert die *Add*-Funktion noch eine Texthöhe, deren Wert

natürlich größer als 0 sein muss, damit der Text erzeugt werden kann. Sie können für die Vorgabe einer Texthöhe den Wert der Systemvariablen *TEXTSIZE* verwenden, die ja die aktuelle Texthöhe bestimmt. Alle anderen Eigenschaften, wie Drehwinkel, Textstil oder Ausrichtung können Sie an dem Objekt anpassen, das als Ergebnis der Funktion zurückgegeben wird.



Abbildung 6.24: Einzeiliger Text

AddText

Parameter: *TextString As String, InsertionPoint, Height As Double*

Rückgabebetyp: *AcadText*

```
Public Sub Sample_AddText()
Dim NewText As AcadText
Dim InsPoint As Variant, Prompt As String
Dim TextString As String, TextHeight As Double, Rotation As Double
On Local Error Resume Next
Prompt = "Einfügapunkt:"
InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(InsPoint) = True Then Exit Sub
TextHeight = ThisDrawing.GetVariable("TEXTSIZE")
Prompt = "Texthöhe <" & ThisDrawing.Utility.RealToString( _
    TextHeight, acDefaultUnits, 3) & ">:"
TextHeight = ThisDrawing.Utility.GetDistance(InsPoint, Prompt)
If TextHeight = 0 Then
    TextHeight = ThisDrawing.GetVariable("TEXTSIZE")
End If
Rotation = ThisDrawing.GetVariable("VIEWTWIST")
Prompt = "Drehwinkel <" & ThisDrawing.Utility.AngleToString( _
    Rotation, ThisDrawing.GetVariable("AUNITS"), 4) & ">:"
Rotation = ThisDrawing.Utility.GetOrientation(InsPoint, Prompt)
Prompt = "Text eingeben:"
TextString = ThisDrawing.Utility.GetString(1, Prompt)
If TextString = "" Then Exit Sub
Set NewText = ThisDrawing.ModelSpace.AddText(TextString, _
    InsPoint, TextHeight)
NewText.Rotation = Rotation
End Sub
```

Siehe auch *Kapitel 8.3.16*

Geometrische Toleranz erstellen

Toleranzobjekte werden in Zusammenhang mit anderen grafischen Objekten verwendet und sollen die maximal zulässige Abweichung von Form oder Profil, Ausrichtung, Position und Lauf der Objekte von der exakten Geometrie einer Zeichnung definieren. Die Zeichenfolge,

die Sie im ersten Parameter der Funktion angeben, bestimmt den eigentlichen Aufbau des Objekts. Geben Sie hier einen einfachen Text an, ohne den Zusatz von Steuerzeichen, wird ein umrahmter Text erzeugt. Die Definition der einzelnen Steuerzeichen können Sie der folgenden Tabelle entnehmen.

Definition	Beschreibung										
{\Fgdt;}	Bestimmt eine Symboldefinition. Der dem Semikolon folgende Buchstabe legt das verwendete Symbol fest, z.B. {\Fgdt;n} für ein Durchmesserzeichen										
a	b	c	d	e	f	g	h	i	j		
											
k	l	m	n	p	r	s	t	u			
											
%%v	Legt eine senkrechte Trennungslinie fest.										
vbLf	Diese Konstante ermöglicht einen Zeilenumbruch, womit die Definition in der nächsten Zeile fortgesetzt wird.										
	<table><tr><td>erste Zeile</td></tr><tr><td>zweite Zeile</td></tr></table>									erste Zeile	zweite Zeile
erste Zeile											
zweite Zeile											

Tabelle 6.7: Steuerzeichen für geometrische Toleranzen

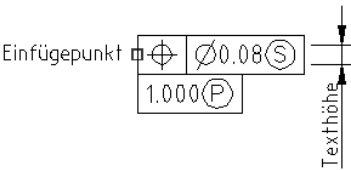


Abbildung 6.25: Geometrische Toleranz

Die Darstellung dieses Bilds wird nach der Definition aus der vorangegangenen Tabelle durch folgende Zeichenfolge bestimmt:

```
"{\Fgdt;j}%v{\Fgdt;n}0.08{\Fgdt;s}" & vbLf & "1.000{\Fgdt;p}"
```

Diese setzt sich wie folgt zusammen:

{\Fgdt;j}	%v	{\Fgdt;n}	0.08	{\Fgdt;s}	VbLf	1.000	{\Fgdt;n}
			0.08		Fortsetzung mit der nächsten Zeile	1.000	

Als zweite Vorgabe wird der Einfügepunkt des Toleranzobjekts benötigt. Dieser ist wie gehabt als 3D-Array zu definieren. Der dritte und letzte Parameter bestimmt die Richtung oder den Drehwinkel, mit der/dem das Toleranzobjekt eingefügt werden soll. Allerdings wird hier kein direkter Winkel, sondern vielmehr ein dreidimensionaler Richtungsvektor, ausgehend vom Einfügepunkt, verlangt.

AddTolerance

Parameter: *Text As String, InsertionPoint, Direction*

Rückgabetyt: *AcadTolerance*

```
Public Sub Sample_AddTolerance()
Dim NewTolerance As AcadTolerance
Dim InsPoint As Variant, Prompt As String
Dim TextString As String, Direction
On Local Error Resume Next
Prompt = "Einfügepunkt:"
InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(InsPoint) = True Then Exit Sub
Prompt = "Richtung:"
Direction = ThisDrawing.Utility.GetPoint(InsPoint, Prompt)
If IsEmpty(Direction) = True Then
    ReDim Direction(2) As Double
    Direction(0) = 0
    Direction(1) = 0
    Direction(2) = 0
Else
    Direction(0) = Direction(0) - InsPoint(0)
    Direction(1) = Direction(1) - InsPoint(1)
    Direction(2) = Direction(2) - InsPoint(2)
End If
Prompt = "Toleranztext:"
TextString = ThisDrawing.Utility.GetString(1, Prompt)
Set NewTolerance = ThisDrawing.ModelSpace.AddTolerance( _
    TextString, InsPoint, Direction)
End Sub
```

Siehe auch *Kapitel 8.3.26*

Spurobjekt oder Band erzeugen

Ein Band ist in der Darstellung und Definition der Scheitelpunkte ähnlich einer 2D-Solidfläche. Der Unterschied bei der Erstellung besteht darin, dass die Koordinaten der vier Scheitelpunkte nicht in vier 3D-Arrays, sondern in einem Array, das aus zwölf Feldern besteht, übergeben werden.

```
Dim PointsArray(11) As Double
```

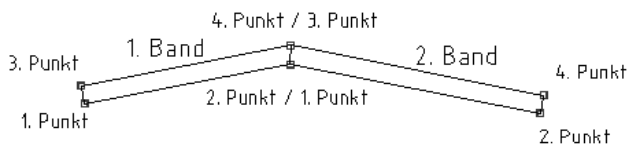


Abbildung 6.26: Spurobjekt (Band)

Um die Flächenfüllung eines Bands ein- bzw. auszuschalten, müssen Sie den Wert der Systemvariablen *FILLMODE* entsprechend anpassen:

- 1 – Flächen werden gefüllt.
- 0 – Nur die Verbindungen zwischen den einzelnen Stützpunkten werden angezeigt.

Das Band wird zwar im Stile der 2D-Solidflächen erstellt. Wenn Sie aber die Erstellung dem AutoCAD-Befehl anpassen möchten, indem Sie eine Mittellinie erzeugen, und die Stützpunkte anhand der Bandbreite berechnen lassen wollen, können Sie auf den Wert der Systemvariablen *TRACEWID* zurückgreifen. Diese hat die aktuelle Bandbreite gespeichert.

AddTrace

Parameter: *PointsArray*

Rückgabebetyp: *AcadTrace*

```
Public Sub Sample_AddTrace()
Dim NewTrace As AcadTrace
Dim InsPoint As Variant, Prompt As String
Dim Pnt, VertList(11) As Double
On Local Error Resume Next
Pnt1 = Empty
Prompt = "Erster Stützpunkt:"
Pnt1 = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Pnt1) = True Then Exit Sub
Pnt2 = Empty
Prompt = "Zweiter Stützpunkt:"
Pnt2 = ThisDrawing.Utility.GetPoint(Pnt1, Prompt)
If IsEmpty(Pnt2) = True Then Exit Sub
Do
    Pnt4 = Empty
    Prompt = "Dritter Stützpunkt:"
    Pnt4 = ThisDrawing.Utility.GetPoint(Pnt2, Prompt)
    If IsEmpty(Pnt4) = True Then Exit Sub
    Pnt3 = Empty
    Prompt = "Vierter Stützpunkt:"
    Pnt3 = ThisDrawing.Utility.GetPoint(Pnt4, Prompt)
    If IsEmpty(Pnt3) = True Then Exit Sub
    For i = 0 To 2
        VertList(i) = Pnt1(i)
        VertList(i + 3) = Pnt2(i)
        VertList(i + 6) = Pnt3(i)
        VertList(i + 9) = Pnt4(i)
    Next
    Set NewTrace = ThisDrawing.ModelSpace.AddTrace(VertList)
    Pnt1(0) = Pnt3(0): Pnt1(1) = Pnt3(1): Pnt1(2) = Pnt3(2)
    Pnt2(0) = Pnt4(0): Pnt2(1) = Pnt4(1): Pnt2(2) = Pnt4(2)
Loop
End Sub
```

Siehe auch *Kapitel 8.3.19*

Konstruktionslinie erzeugen

Eine Konstruktionslinie wird ähnlich erstellt wie ein Strahl. Auch hier benötigen Sie einen Basispunkt und einen Richtungsvektor. Der Unterschied zu Strahlen besteht lediglich in der Darstellung der Konstruktionslinien. Während Strahlen in einer Richtung begrenzt sind, werden Konstruktionslinien generell in beiden Richtungen unendlich angezeigt.

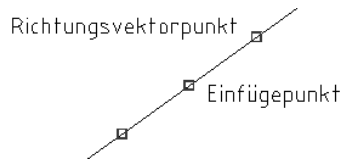


Abbildung 6.27: Konstruktionslinie

Bei der *AddXLine*-Funktion müssen, anders als bei Strahlen, beide Punkt-Arrays Koordinaten im WKS enthalten. Der Richtungsvektor, den Sie später in den Eigenschaften des Konstruktionslinien-Objekts finden, wird durch die *Add*-Funktion aus den beiden Punkten bestimmt.

AddXLine

Parameter: *Point1*, *Point2*

Rückgabety: *Acad3DSolid*

```
Public Sub Sample_AddXLine()
Dim NewXLine As AcadXline
Dim StartPt As Variant, Prompt As String, EndPt As Variant
On Local Error Resume Next
Do
  If IsEmpty(StartPt) = True Then
    Prompt = "Punkt angeben:"
    StartPt = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(StartPt) = True Then Exit Sub
  Else
    EndPt = Empty
    Prompt = "Durch Punkt:"
    EndPt = ThisDrawing.Utility.GetPoint(StartPt, Prompt)
    If IsEmpty(EndPt) = True Then Exit Sub
    Set NewXLine = ThisDrawing.ModelSpace.AddXline(StartPt, _
      EndPt)
  End If
Loop
End Sub
```

Siehe auch *Kapitel 8.3.18*

3D-Volumenkörper erstellen (3D-Solid)

AutoCAD bietet die Möglichkeit, die unterschiedlichsten Grundformen von 3D-Volumenkörpern zu erstellen. Jeder dieser Volumenkörper wird nach erfolgreicher Ausführung der entsprechenden *Add*-Funktion durch die Klasse *Acad3DSolid* dargestellt. Sie können z.B. Kugeln, Quader, Keile oder Kegel erzeugen, wobei sowohl die Klasse *AcadModelSpace* als

auch *AcadPaperSpace* für jede Grundform, die Sie bereits aus AutoCAD kennen, eine eigene *Add*-Funktion bereitstellt. Im Folgenden werden alle diese Funktionen aufgelistet und jeweils mit einem Beispiel beschrieben.

Quaderförmigen 3D-Volumenkörper erstellen

Um einen 3D-Volumenkörper in Form eines Quaders dem Modell- bzw. Papierbereich hinzuzufügen, stellen die Klassen *AcadModelSpace* und *AcadPaperSpace* die Funktion *AddBox* zur Verfügung. Als Ergebnis der Funktion erhalten Sie ein Objekt vom Typ *Acad3DSolid*, was im Zusammenhang mit anderen *Add*-Funktionen noch öfter der Fall sein wird. Als Parameter zur Erstellung des Quaders verlangt die Funktion neben dem Einfügepunkt, der wiederum als 3D-Array definiert werden muss, noch die Angaben für Kantenlänge, Höhe und Breite des Quaders.

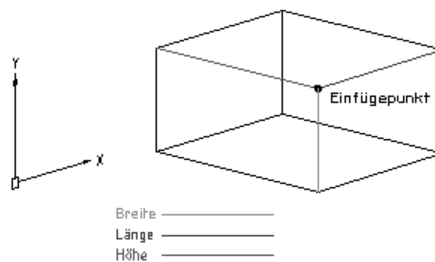


Abbildung 6.28: Quaderförmiger Volumenkörper

AddBox

Parameter: *Origin*, *Length As Double*, *Width As Double*, *Height As Double*

Rückgabetyt: *Acad3DSolid*

```
Public Sub Sample_AddBox()
Dim NewBox As Acad3DSolid
Dim Insertion As Variant, Height As Double
Dim Length As Double, Width As Double
On Local Error Resume Next
Prompt = "Einfügepunkt des Quaders:"
Insertion = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Insertion) = True Then Exit Sub
Prompt = "Quaderhöhe:"
Height = ThisDrawing.Utility.GetDistance(Insertion, Prompt)
If Height = 0 Then Exit Sub
Prompt = "Quaderlänge:"
Length = ThisDrawing.Utility.GetDistance(Insertion, Prompt)
If Length = 0 Then Length = Height
Prompt = "Quaderbreits:"
Width = ThisDrawing.Utility.GetDistance(Insertion, Prompt)
If Width = 0 Then Width = Height
Set NewBox = ThisDrawing.ModelSpace.AddBox(Insertion, Length, _
    Width, Height)
End Sub
```

Siehe auch *Kapitel 8.3.32*

Kegelförmigen 3D-Volumenkörper erstellen

Um einen 3D-Volumenkörper in Form eines Kegels dem Modell- bzw. Papierbereich hinzuzufügen, können Sie die Funktion *AddCone* verwenden. Genau wie die Quader-Funktion erstellt auch diese Prozedur ein Objekt vom Typ *Acad3DSolid*. Allerdings werden zur Erstellung eines Kegels wesentlich weniger Parameter erfordert. Sie müssen lediglich den Mittelpunkt und den Radius der Grundfläche und die Kegelhöhe festlegen.

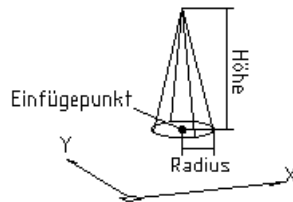


Abbildung 6.29: Kegelförmiger Volumenkörper

AddCone

Parameter: *Center, BaseRadius As Double, Height As Double*

Rückgabtyp: *Acad3DSolid*

```
Public Sub Sample_AddCone()
Dim NewCone As Acad3DSolid
Dim Center As Variant, Radius As Double, Height As Double
On Local Error Resume Next
Prompt = "Zentrumspunkt der Grundfläche:"
Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Center) = True Then Exit Sub
Prompt = "Radius der Grundfläche:"
Radius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Radius = 0 Then Exit Sub
Prompt = "Kegelhöhe:"
Height = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Height = 0 Then Exit Sub
Set NewCone = ThisDrawing.ModelSpace.AddCone(Center, Radius, _
    Height)
End Sub
```

Siehe auch *Kapitel 8.3.32*

Zylinderförmigen 3D-Volumenkörper erstellen

Auch beim Hinzufügen eines Zylinders erhalten Sie ein Objekt vom Typ *Acad3DSolid*. Als Grundlagen für die Erstellung des Zylinders gelten die gleichen Parameter wie für einen Kegel. Allerdings wird der Radius der Grundfläche auch auf die entgegengesetzte Seite übertragen, damit ein Zylinder entstehen kann.

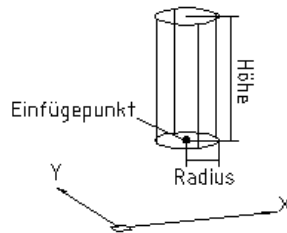


Abbildung 6.30: Zylinderförmiger Volumenkörper

AddCylinderParameter: *Center, Radius As Double, Height As Double*Rückgabetypp: *Acad3DSolid*

```

Public Sub Sample_AddCylinder()
Dim NewCylinder As Acad3DSolid
Dim Center As Variant, Radius As Double, Height As Double
On Local Error Resume Next
Prompt = "Einfügapunkt des Zylinders:"
Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Center) = True Then Exit Sub
Prompt = "Radius des Zylinders:"
Radius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Radius = 0 Then Exit Sub
Prompt = "Zylinderhöhe:"
Height = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Height = 0 Then Exit Sub
Set NewCylinder = ThisDrawing.ModelSpace.AddCylinder(Center, _
    Radius, Height)
End Sub

```

Siehe auch *Kapitel 8.3.32***Kegelförmigen 3D-Volumenkörper mit elliptischer Grundfläche erstellen**

Die Funktion *AddEllipticalCone* erzeugt, ähnlich wie die Funktion *AddCone*, ein 3D-Solid-Objekt in Form eines Kegels. Der Unterschied zum normalen Kegel besteht lediglich darin, dass dieser Kegel eine elliptische Grundfläche besitzt. Aus diesem Grund ist es folglich klar, dass für die Grundfläche ein zusätzlicher Parameter erforderlich ist. Der erste definiert wiederum den Einfügapunkt und somit das Zentrum der Grundfläche. Der zweite und dritte Parameter bestimmen den Radius der Haupt- und Nebenachse der elliptischen Grundfläche. Mit dem vierten legen Sie wieder die Kegelhöhe fest.

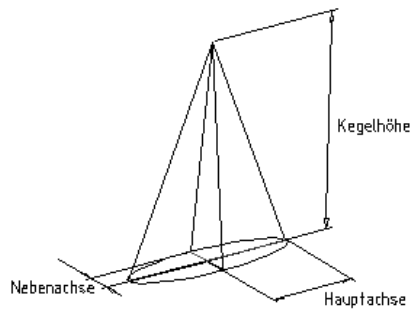


Abbildung 6.31: Kegelförmiger Volumenkörper

AddEllipticalCone

Parameter: *Center*, *MajorRadius As Double*, *MinorRadius As Double*, *Height As Double*

Rückgabety: *Acad3DSolid*

```
Public Sub Sample_AddEllipticalCone()
Dim NewCone As Acad3DSolid
Dim Center As Variant, Height As Double
Dim MajorRadius As Double, MinorRadius As Double
On Local Error Resume Next
Prompt = "Zentrumspunkt der Grundfläche:"
Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Center) = True Then Exit Sub
Prompt = "Radius der Hauptachse der Grundfläche:"
MajorRadius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If MajorRadius = 0 Then Exit Sub
Prompt = "Radius der Nebenachse der Grundfläche:"
MinorRadius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If MinorRadius = 0 Then Exit Sub
Prompt = "Kegelhöhe:"
Height = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Height = 0 Then Exit Sub
Set NewCone = ThisDrawing.ModelSpace.AddEllipticalCone(Center, _
    MajorRadius, MinorRadius, Height)
End Sub
```

Siehe auch *Kapitel 8.3.32*.

Zylinderförmigen 3D-Volumenkörper mit elliptischer Grundfläche erstellen

Bei der Erzeugung dieses Zylindertyps können Sie sich an der Erstellung eines Zylinders mit runder Grundfläche orientieren. Der Funktion läuft auf ähnliche Weise ab und erfordert wiederum einen zusätzlichen Parameter zur Definition der elliptischen Flächen. Der erste bestimmt den Zentrumspunkt des Zylinders, der zweite und dritte den Radius der Haupt- und Nebenachse, der letzte die Höhe des Zylinders. Die Definition der einzelnen Parameter können Sie in Abbildung 6.32 sehen.

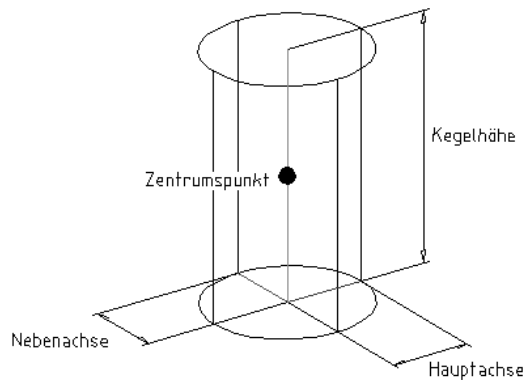


Abbildung 6.32: Zylinderförmiger Volumenkörper

AddEllipticalCylinder

Parameter: *Center*, *MajorRadius As Double*, *MinorRadius As Double*, *Height As Double*

Rückgabotyp: *Acad3DSolid*

```
Public Sub Sample_AddEllipticalCylinder()
Dim NewCone As Acad3DSolid
Dim Center As Variant, Height As Double
Dim MajorRadius As Double, MinorRadius As Double
On Local Error Resume Next
Prompt = "Zentrumspunkt der Grundfläche:"
Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Center) = True Then Exit Sub
Prompt = "Radius der Hauptachse der Grundfläche:"
MajorRadius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If MajorRadius = 0 Then Exit Sub
Prompt = "Radius der Nebenachse der Grundfläche:"
MinorRadius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If MinorRadius = 0 Then Exit Sub
Prompt = "Zylinderhöhe:"
Height = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Height = 0 Then Exit Sub
Set NewCone = ThisDrawing.ModelSpace.AddEllipticalCylinder( _
    Center, MajorRadius, MinorRadius, Height)
End Sub
```

Siehe auch *Kapitel 8.3.32*

Extrusionsvolumenkörper anhand von Profil, Höhe und Verjüngungswinkel hinzufügen

Um aus zweidimensionalen Objekten einen Volumenkörper zu generieren, stellt AutoCAD-VBA zwei Funktionen zur Verfügung. Mit der ersten Funktion können Sie durch Angabe einer Höhe und eines Verjüngungswinkels einen Volumenkörper auf Grundlage einer Region erstellen lassen. Der erste Parameter enthält das Region-Objekt, auf dessen Form der Volumenkörper aufgebaut werden soll. Der zweite Parameter bestimmt die Höhe, in der der Körper aufgebaut werden soll. Mit dem letzten Parameter können Sie einen Winkel festlegen, der den

Volumenkörper vom Basisobjekt (Region) aus verjüngt oder verbreitert. Sie können für den Winkel einen Wert zwischen -90° und 90° angeben. Wenn Sie für den Winkel den Wert 0 festlegen, wird der Volumenkörper lotrecht aufgebaut. Negative Werte verjüngen das Objekt, positive haben den gegenteiligen Effekt.

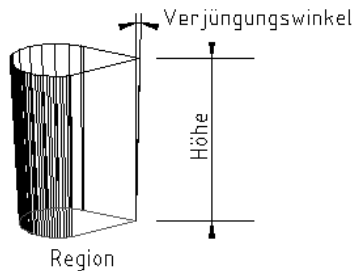


Abbildung 6.33: Extrusionsvolumenkörper

AddExtrudedSolid

Parameter: *Profile As AcadRegion, Height As Double, TaperAngle As Double*

Rückgabebetyp: *Acad3DSolid*

```
Public Sub Sample_AddExtrudedSolid()
Dim NewExtSolid As Acad3DSolid
Dim Object As Object, Prompt As String, PickedPt
Dim Height As Double, Angle As Double
On Local Error Resume Next
Prompt = "Wählen Sie eine Region:"
ThisDrawing.Utility.GetEntity Object, PickedPt, Prompt
If TypeName(Object) = "IAcadRegion" Then
    Prompt = "Höhe des Volumenkörpers:"
    Height = ThisDrawing.Utility.GetDistance(, Prompt)
    If Height <= 0 Then Exit Sub
    Prompt = "Verjüngungswinkel:"
    Angle = ThisDrawing.Utility.GetAngle(, Prompt)
    Set NewExtSolid = ThisDrawing.ModelSpace.AddExtrudedSolid( _
        Object, Height, Angle)
    If Err <> 0 Then MsgBox Err.Description, vbCritical, "Fehler"
End If
End Sub
```

Siehe auch *Kapitel 8.3.32*

Extrusionsvolumenkörper anhand eines Pfads

Die zweite Möglichkeit einen Volumenkörper aus einer Region, die die Grundfläche bildet, zu erstellen, ist die Vorgabe des Pfads, über den der Volumenkörper extrudiert werden soll. Dieser Pfad kann durch Linien, Kreise, Bögen, Ellipsen, elliptische Bögen, Polylinien oder Splines bestimmt werden. Der Pfad darf allerdings nicht auf derselben Ebene liegen wie die Region und er darf auch keine Bereiche mit starken Kurven haben. Die Funktion *AddExtrudedSolid*

AlongPath, die diese Aufgabe ausführt, verlangt im ersten Parameter wieder die Region, die die Grundfläche des Volumenkörpers bilden wird. Im zweiten Parameter müssen Sie das Objekt übergeben, das den Pfad bestimmt.

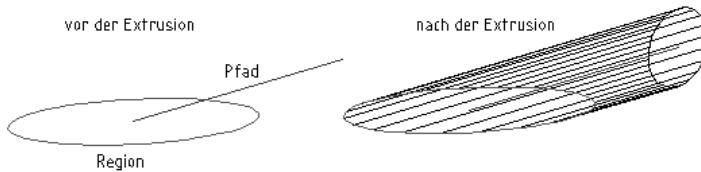


Abbildung 6.34: Extrusionsvolumenkörper

AddExtrudedSolidAlongPath

Parameter: *Profile As AcadRegion, Path As Object*

Rückgabetyt: *Acad3DSolid*

```
Public Sub Sample_AddExtrudedSolidAlongPath()
Dim NewExtSolid As Acad3DSolid
Dim Object As Object, Prompt As String, PickedPt
Dim PathObj As Object
On Local Error Resume Next
Prompt = "Wählen Sie eine Region:"
ThisDrawing.Utility.GetEntity Object, PickedPt, Prompt
If TypeName(Object) = "IAcadRegion" Then
    Prompt = "Objekt an dem entlang extrudiert werden soll:"
    ThisDrawing.Utility.GetEntity PathObj, PickedPt, Prompt
    If TypeName(PathObj) = "Nothing" Then Exit Sub
    Set NewExtSolid = ThisDrawing.ModelSpace. _
        AddExtrudedSolidAlongPath(Object, PathObj)
    If Err <> 0 Then MsgBox Err.Description, vbCritical, "Fehler"
End If
End Sub
```

Siehe auch *Kapitel 8.3.32*

Rotationsvolumenkörper erstellen

Mit der Funktion zur Erstellung eines Rotationsvolumenkörpers haben Sie eine hervorragende Möglichkeit, auf einfachste Weise aus einer 2D-Fläche einen komplexen Volumenkörper erzeugen zu lassen. Was zur Verwendung der Funktion benötigt wird, ist lediglich eine Grundfläche, die einem Objekt der Klasse *AcadRegion* entsprechen muss. Auf dieser Region wird dann der Volumenkörper, unter Angabe einer Rotationsachse, aufgebaut. Diese Achse legen Sie in den Parametern zwei und drei fest. *AxisPoint* definiert den Startpunkt *AxisDir* den Richtungsvektor der Achse. Beide Parameter müssen als 3D-Arrays definiert werden, wobei der Parameter *AxisDir* die Richtungskordinaten in Bezug auf den Startpunkt der Achse enthält. Das bedeutet, dass Sie die ermittelten Punktkordinaten, falls Sie einen Punkt ermittelt haben, erst in einen Richtungsvektor umrechnen müssen.

Beispiel:

Gewählter Startpunkt der Achse:

```
AxisPoint1(0) = 100
AxisPoint1(1) = 150
AxisPoint1(2) = 10
```

Gewählter Richtungspunkt der Achse:

```
AxisPoint2(0) = 220
AxisPoint2 (1) = 180
AxisPoint2 (2) = 10
```

Richtungsvektor wird wie folgt berechnet:

```
AxisDir(0) = AxisPoint2(0) - AxisPoint1(0)
AxisDir(1) = AxisPoint2(1) - AxisPoint1(1)
AxisDir(2) = AxisPoint2(2) - AxisPoint1(2)
```

und ergibt demnach:

```
AxisDir(0) = 120
AxisDir(1) = 30
AxisDir(2) = 0
```

Als letzten Parameter übergeben Sie der Funktion den Rotationswinkel. Dieser Winkel legt fest, wie viel Volumenkörper entstehen soll. Wenn Sie einen Wert manuell und nicht durch eine VBA-Winkelfunktion bestimmen, müssen Sie darauf achten, dass der Wert in Radiant angegeben wird. Das bedeutet, dass Sie für einen vollständigen Volumenkörper einen Wert von 6.28 oder $2 * \pi$ angeben, für einen halben Volumenkörper 3.14 oder π usw.

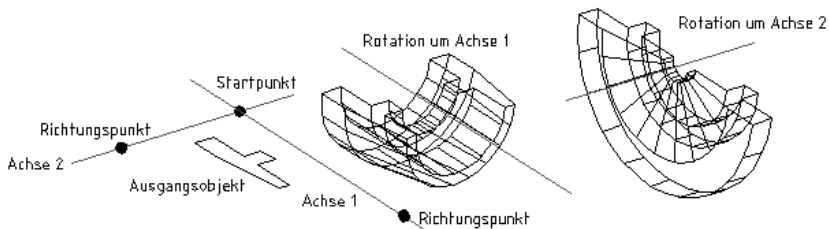


Abbildung 6.35: Rotationsvolumenkörper

AddRevolvedSolid

Parameter: *Profile As AcadRegion, AxisPoint, AxisDir, Angle As Double*

Rückgabety: *Acad3DSolid*

```
Public Sub Sample_AddRevolvedSolid()
Dim NewRevSolid As Acad3DSolid
Dim Object As Object, Prompt As String, PickedPt
Dim AxisPt, AxisDir, Angle As Double
Const PI = 3.14159265358979
On Local Error Resume Next
```

```

Prompt = "Wählen Sie eine Region:"
ThisDrawing.Utility.GetEntity Object, PickedPt, Prompt
If TypeName(Object) = "IAcadRegion" Then
    Prompt = "Achse durch Punkt:"
    AxisPt = ThisDrawing.Utility.GetPoint(, Prompt)
    If IsEmpty(AxisPt) = True Then Exit Sub
    Prompt = "Achsrichtung:"
    AxisDir = ThisDrawing.Utility.GetPoint(AxisPt, Prompt)
    If IsEmpty(AxisDir) = True Then Exit Sub
    AxisDir(0) = AxisDir(0) - AxisPt(0)
    AxisDir(1) = AxisDir(1) - AxisPt(1)
    AxisDir(2) = AxisDir(2) - AxisPt(2)
    Prompt = "Rotationswinkel angeben <360>°:"
    Angle = ThisDrawing.Utility.GetAngle(, Prompt)
    If Angle = 0 Then
        Angle = 2 * PI
    End If
    Err.Clear
    Set NewRevSolid = ThisDrawing.ModelSpace. _
        AddRevolvedSolid(Object, AxisPt, AxisDir, Angle)
    If Err <> 0 Then MsgBox Err.Description, vbCritical, "Fehler"
End If
End Sub

```

Siehe auch *Kapitel 8.3.32*

3D-Solid in Form einer Kugel erzeugen

Genau wie bei der Erzeugung eines Kreises, werden auch bei einer Kugel der Zentrumsunkt und der Radius verlangt, nur dass in diesem Fall ein kugelförmiges 3DSolid-Objekt modelliert wird. Die Dichte der Kugel, also wie rund die Kugel erzeugt wird, ist abhängig von der Einstellung der Systemvariablen *ISOLINES*.



Abbildung 6.36: Kugel

AddSphere

Parameter: *Center, Radius As Double*

Rückgabetyt: *Acad3DSolid*

```

Public Sub Sample_AddSphere()
    Dim NewSphere As Acad3DSolid
    Dim InsPoint As Variant, Prompt As String
    Dim Center, Radius As Double, IsoLines As Long
    On Local Error Resume Next
    Prompt = "Mittelpunkt der Kugel:"

```

```

Center = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(Center) = True Then Exit Sub
IsoLines = ThisDrawing.GetVariable("ISOLINES")
Prompt = "Anzahl der Isolinien für die Oberfläche:"
IsoLines = ThisDrawing.Utility.GetInteger(Prompt)
ThisDrawing.SetVariable "ISOLINES", IsoLines
Prompt = "Radius der Kugel angeben:"
Radius = ThisDrawing.Utility.GetDistance(Center, Prompt)
If Radius = 0 Then Exit Sub
Set NewSphere = ThisDrawing.ModelSpace.AddSphere(Center, Radius)
End Sub

```

Siehe auch *Kapitel 8.3.32*

Ringförmigen Volumenkörper erstellen

Ein ringförmiger Volumenkörper oder auch Torus kann durch die *AddTorus*-Funktion erstellt werden. Diese benötigt im ersten Parameter die Angabe eines 3D-Arrays, das den Mittelpunkt des Torus bestimmt. Des Weiteren wird das Objekt von zwei Radiuswerten definiert, die im zweiten und dritten Parameter der *Add*-Funktion festgelegt werden. Der *TorusRadius* bezeichnet dabei den Gesamtdurchmesser des Rings, der *TubeRadius* den Abstand vom Mittelpunkt des Torus zum Mittelpunkt des Rings.

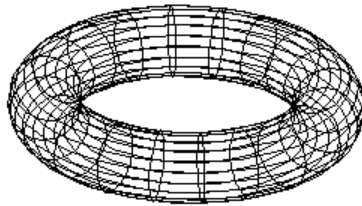


Abbildung 6.37: Ringförmigen Volumenkörper

Wenn Sie für beide Radien positive Werte angeben und der *TubeRadius* größer ist als der des *TorusRadius*, erhalten Sie als Ergebnis eine Kugel, die an jedem Pol eine Vertiefung aufweist.

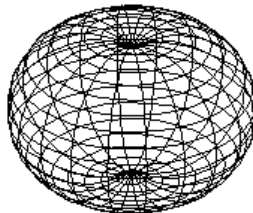


Abbildung 6.38: *TorusRadius* kleiner als der *TubeRadius*

Wenn der Radius des Torus negativ ist und der *TubeRadius* einen größeren positiven Wert als der Radius des Torus aufweist, sieht das Ergebnis wie eine Kugel mit spitzen Polen aus.

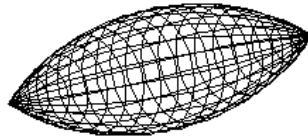


Abbildung 6.39: TorusRadius kleiner als 0, TubeRadius größer als 0

AddTorus

Parameter: *Center, TorusRadius As Double, TubeRadius As Double*

Rückgabetyt: *Acad3DSolid*

```
Public Sub Sample_AddTorus()
Dim NewTorus As Acad3DSolid
Dim InsPoint As Variant, Prompt As String
Dim TorusRad As Double, TubeRad As Double
On Local Error Resume Next
Prompt = "Mittelpunkt des Torus:"
InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(InsPoint) = True Then Exit Sub
Prompt = "Radius des Torus angeben:"
TorusRad = ThisDrawing.Utility.GetDistance(InsPoint, Prompt)
Prompt = "Radius des Rohres angeben:"
TubeRad = ThisDrawing.Utility.GetDistance(, Prompt)
Set NewTorus = ThisDrawing.ModelSpace.AddTorus(InsPoint, _
    TorusRad, TubeRad)
End Sub
```

Siehe auch *Kapitel 8.3.32*

3D-Volumenkörper in Form eines Keils erzeugen

Die nächste Form eines 3D-Volumenkörpers, für den VBA eine spezielle *Add*-Funktion zur Verfügung stellt, ist die eines Keils. Die Keilbasis, also die Grundfläche, liegt bei der Erzeugung parallel zum aktuellen BKS. Als Parameter verlangt die *Add*-Funktion zunächst den Einfügepunkt des Keils, der genau das Zentrum der Grundfläche des Volumenkörpers bildet. Dieser wird, wie immer, als 3D-Array an die Funktion übergeben und enthält die X-, Y- und Z-Koordinate. Zusätzlich zur Erzeugung des Keils benötigen Sie die Länge und Breite der Grundfläche sowie die Höhe der stumpfen Seite des Keils. Die Höhe kann dabei sowohl positive als auch negative Werte enthalten. Je nach Wert wird dann der Keil in die entsprechende Richtung gezeichnet.

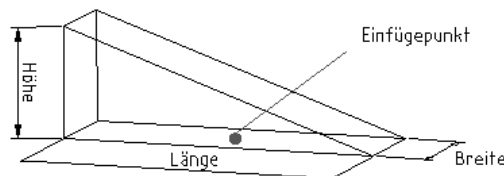


Abbildung 6.40: Keil

AddWedge

Parameter: *Center, Length As Double, Width As Double, Height As Double*

Rückgabety: *Acad3DSolid*

```
Public Sub Sample_AddWedge()
Dim NewWedge As Acad3DSolid
Dim InsPoint As Variant, Prompt As String
Dim Width As Double, Height As Double, Length As Double
On Local Error Resume Next
Prompt = "Einfügapunkt:"
InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(InsPoint) = True Then Exit Sub
Prompt = "Länge der Grundfläche:"
Length = ThisDrawing.Utility.GetDistance(InsPoint, Prompt)
If Length <= 0 Then Exit Sub
Prompt = "Breite der Grundfläche:"
Width = ThisDrawing.Utility.GetDistance(InsPoint, Prompt)
If Width <= 0 Then Exit Sub
Prompt = "Höhe des Keiles:"
Height = ThisDrawing.Utility.GetDistance(InsPoint, Prompt)
If Height = 0 Then Exit Sub
Set NewWedge = ThisDrawing.ModelSpace.AddWedge(InsPoint, _
    Length, Width, Height)
End Sub
```

Siehe auch *Kapitel 8.3.32*

Blockreferenz einfügen

Eine Blockreferenz ist eine grafisch eingefügte Blockdefinition, die Sie in der Blockauflistung der entsprechenden Zeichnung finden können. Sie ist also nicht zu verwechseln mit der eigentlichen Blockdefinition (Datentyp *AcadBlock*). Im ersten Parameter der Funktion übergeben Sie den Einfügapunkt der Blockreferenz. Dieser muss einem 3D-Array mit den X-, Y- und Z-Koordinaten entsprechen und bestimmt somit die Position, an der der Block in der Zeichnung eingefügt wird. Wo sich der Bezugspunkt in der Blockdefinition befindet, ist abhängig von der Eigenschaft *Origin* der jeweiligen Blockdefinition. Da eine Blockdefinition mit einem eindeutigen Namen versehen ist, ist es dieser, der auch die Darstellung der Blockreferenz bestimmt. Der Name des einzufügenden Blocks wird im zweiten Parameter übergeben. Wenn Sie einen Block einfügen wollen, der noch nicht in der Zeichnung definiert wurde, sondern sich als Zeichnungsdatei auf Ihrem System befindet, müssen Sie die gesamte Verzeichnisangabe der Datei dem Namen voranstellen.

Einfügen aus der Blockauflistung:

```
Name = "Block1"
```

Einlesen als Blockdefinition aus einer Datei:

```
Name = "C:\Block1.dwg"
```

Der Block wird dann zunächst in die Blockauflistung eingelesen und erst dann in die Zeichnung eingefügt. Wenn Sie auf diese Weise eine Blockreferenz einfügen, deren Definition in der Blockauflistung bereits vorhanden ist, wird die Definition des Blocks durch die der angegebenen Zeichnungsdatei ersetzt. Die nächsten drei Parameter bestimmen den Skalierfaktor der Blockreferenz in X-, Y- und Z-Richtung und der letzte den Drehwinkel.

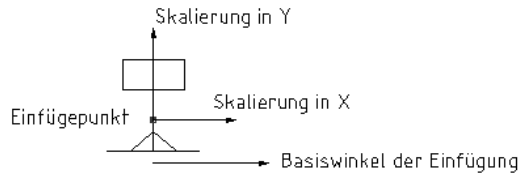


Abbildung 6.41: Blockreferenz

InsertBlock

Parameter: *InsertionPoint*, *Name As String*, *Xscale As Double*, *Yscale As Double*, *Zscale As Double*, *Rotation As Double*

Rückgabetyp: *AcadBlockReference*

```
Public Sub Sample_InsertBlock()
Dim NewBlockRef As AcadBlockReference
Dim InsPoint As Variant, Prompt As String
Dim Name As String, Rotation As Double
Dim XScale As Double, YScale As Double, ZScale As Double
Dim ComDialog As MSComDlg.CommonDialog
On Local Error Resume Next
Prompt = "Name der Blockreferenz:"
Set ComDialog = New MSComDlg.CommonDialog
Do
    Name = ThisDrawing.Utility.GetString(0, Prompt)
    If Name = "" Then Exit Sub
    If TypeName(ThisDrawing.Blocks(Name)) = "Nothing" Then
        ComDialog.CancelError = True
        ComDialog.DialogTitle = "Block einfügen"
        ComDialog.Filter = "Zeichnungsdateien (*.dwg)|*.dwg|"
        Err.Clear
        ComDialog.ShowOpen
        If Err = 0 Then
            Name = ComDialog.FileName
            Exit Do
        End If
    Else
        Exit Do
    End If
Loop
Prompt = "Einfügepunkt:"
InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
If IsEmpty(InsPoint) = True Then Exit Sub
Prompt = "Skalierung in X <1.000>:"
XScale = ThisDrawing.Utility.GetReal(Prompt)
If XScale <= 0 Then XScale = 1
Prompt = "Skalierung in Y <" & _
    ThisDrawing.Utility.RealToString(XScale, acDefaultUnits, _
    3) & ">:"
YScale = ThisDrawing.Utility.GetReal(Prompt)
If YScale <= 0 Then YScale = XScale
ZScale = XScale
```



```
Prompt = "Drehwinkel:"
Rotation = ThisDrawing.Utility.GetAngle(InsPoint, Prompt)
Set NewBlockRef = ThisDrawing.ModelSpace.InsertBlock(InsPoint, _
    Name, XScale, YScale, ZScale, Rotation)
End Sub
```

Siehe auch *Kapitel 8.3.29*

Externe Referenzen einfügen

Eine andere Variante eines Blocks ist die externe Referenz. Diese wird zwar genau wie die „normalen“ Blockdefinitionen in der Blockauflistung der Zeichnung geführt, die Daten der externen Zeichnung werden allerdings nicht in der Zeichnung gespeichert. Sie bleiben in der separaten Zeichnungsdatei, womit die Darstellung der Referenz immer dem aktuell gespeicherten Stand der externen Zeichnungsdatei entspricht. Um nun eine externe Referenz in den Modell- oder Papierbereich einzufügen, werden wieder einige Parameter von der Funktion verlangt, die die Darstellung der Referenz beeinflussen. Mit dem ersten Parameter übergeben Sie den Dateinamen (inklusive Verzeichnisangabe) der Zeichnung, die zugeordnet werden soll. Der zweite Parameter verlangt zwar ebenfalls eine Namensangabe, bei der es sich allerdings um den Namen der Blockdefinition handelt, unter der die externe Referenz in der Blockauflistung eingetragen werden soll. Sie müssen für diesen Parameter einen Namen verwenden, der noch nicht in der Blockauflistung der Zeichnung verwendet wird, da sonst ein Fehler ausgelöst wird. Die nächsten fünf Parameter haben die gleiche Bedeutung wie bei der *InsertBlock*-Funktion, die im Abschnitt zuvor bereits besprochen wurde. *InsertionPoint* übergibt den Einfügepunkt, *XScale*, *YScale* und *ZScale* die Skalierung und *Rotation* den aktuellen Drehwinkel der externen Referenz. Mit dem letzten Parameter (*bOverlay*) legen Sie fest, ob die Referenz überlagert oder zugeordnet werden soll.

- *TRUE* – die externe Referenz überlagern.
- *FALSE* – die externe Referenz zuordnen.

Wenn Sie die Überlagerung der externen Referenz aktivieren, werden alle verschachtelten Überlagerungen der Datei ignoriert und folglich nicht geladen und angezeigt.

AttachExternalReference

Parameter: *PathName As String, Name As String, InsertionPoint, Xscale As Double, Yscale As Double, Zscale As Double, Rotation As Double, bOverlay As Boolean*

Rückgabetyp: *AcadExternalReference*

```
Public Sub Sample_AttachExternalReference()
Dim NewXRef As AcadExternalReference
Dim InsPoint As Variant, Prompt As String
Dim FileName As String, Name As String, Rotation As Double
Dim XScale As Double, YScale As Double, ZScale As Double
Dim ComDialog As MSComDlg.CommonDialog, bOverlay As Boolean
On Local Error Resume Next
Set ComDialog = New MSComDlg.CommonDialog
ComDialog.CancelError = True
ComDialog.DialogTitle = "Block einfügen"
ComDialog.Filter = "Zeichnungsdateien (*.dwg)|*.dwg|"
Err.Clear
ComDialog.ShowOpen
```

```

If Err = 0 Then
  For i = Len(ComDialog.FileName) - 4 To 1 Step -1
    If Mid(ComDialog.FileName, i, 1) = "\" Then Exit For
    FileName = Mid(ComDialog.FileName, i, 1) & FileName
  Next
  Prompt = "Name der externen Referenz <" & FileName & ">:"
  Name = ThisDrawing.Utility.GetString(0, Prompt)
  If Name = "" Then
    Name = FileName
  End If
  FileName = ComDialog.FileName
  Prompt = "Einfügepunkt:"
  InsPoint = ThisDrawing.Utility.GetPoint(, Prompt)
  If IsEmpty(InsPoint) = True Then Exit Sub
  Prompt = "Skalierung in X <1.000>:"
  XScale = ThisDrawing.Utility.GetReal(Prompt)
  If XScale <= 0 Then XScale = 1
  Prompt = "Skalierung in Y <" & ThisDrawing.Utility. _
    RealToString(XScale, acDefaultUnits, 3) & ">:"
  YScale = ThisDrawing.Utility.GetReal(Prompt)
  If YScale <= 0 Then YScale = XScale
  ZScale = XScale
  Prompt = "Drehwinkel:"
  Rotation = ThisDrawing.Utility.GetAngle(InsPoint, Prompt)
  Select Case MsgBox("Soll die externen Referenz " & _
    "zugeordnet werden?", vbYesNoCancel, "Zuordnen")
    Case vbYes
      bOverlay = False
    Case vbNo
      bOverlay = True
    Case Else
      Exit Sub
  End Select
  Set NewXRef = ThisDrawing.ModelSpace. _
    AttachExternalReference(FileName, Name, InsPoint, XScale, _
      YScale, ZScale, Rotation, bOverlay)
End If
End Sub

```

Siehe auch *Kapitel 8.3.31*

6.4.3 Modell- und Papierbereich verwalten

Neben der Möglichkeit, mithilfe der *Add*-Funktionen die verschiedensten Objekte dem Modell- oder Papierbereich hinzuzufügen, bieten diese Objekte auch noch andere Eigenschaften und Methoden, mit denen Sie die Daten verwalten können. Wenn Sie sich die Elemente der Klasse *AcadModelSpace* oder *AcadPaperSpace* einmal im Objektkatalog auflisten lassen, werden Sie dort noch einige Funktionen und Eigenschaften finden, von denen allerdings nicht jede relevant für die beiden Bereiche ist. Diese Eigenschaften sind lediglich vorhanden, weil Blockdefinitionen auf derselben Grundlage beruhen wie *ModelSpace* und *PaperSpace*. Gehen wir also die interessanten Elemente durch.

Die wichtigsten Eigenschaften des Modell- und Papierbereichs

Application

(schreibgeschützt)

Datentyp: *AcadApplication*

Mit dieser Eigenschaft können Sie das Anwendungsobjekt zurückgeben lassen. Dieses Anwendungsobjekt entspricht der AutoCAD-Sitzung, in der die Zeichnung geöffnet wurde.

```
Set AcadApp = ThisDrawing.ModelSpace.Application
```

Siehe auch *Kapitel 5*

Count

(schreibgeschützt)

Datentyp: *Long*

Hier können Sie die Anzahl der grafischen Objekte ermitteln, die im Modell- oder Papierbereich erzeugt wurden, je nachdem, welches der beiden Objekte Sie ansprechen.

```
Count = ThisDrawing.ModelSpace.Count  
MsgBox "Der Modellbereich enthält " & CStr(Count) & " Objekte"
```

Die *Count*-Eigenschaft können Sie nicht direkt beeinflussen, sie ändert sich nur, wenn Sie grafische Objekte in den Modell- oder Papierbereich einfügen oder aus diesem entfernen, indem Sie sie löschen. Die *Count*-Eigenschaft des *PaperSpace*-Objekts ändert sich ebenfalls, wenn Sie das aktive Papierlayout wechseln.

Document

(schreibgeschützt)

Datentyp: *AcadDocument*

Gibt das Zeichnungsobjekt zurück, zu dem der Modell- oder Papierbereich gehört.

HasExtensionDictionary

(schreibgeschützt)

Datentyp: *Boolean*

Mit dieser Eigenschaft könnten Sie überprüfen, ob dem Modell- oder Papierbereich ein Erweiterungswörterbuch zugeordnet wurde (*TRUE*) oder nicht (*FALSE*).

Siehe auch *Kapitel 7.14*

Layout

(schreibgeschützt)

Datentyp: *AcadLayout*

Über die *Layout*-Eigenschaft können Sie das Layout abrufen, das diesem Objekt zugeordnet ist. Für den Modellbereich wird das immer das Layout „Model“ sein. Für den Papierbereich ist diese Eigenschaft schon interessanter. Sie können auf diese Weise auf das Papierlayout zugreifen, das dem *PaperSpace*-Objekt zugeordnet ist, wenn sich die Zeichnung gerade im Modellbereich befindet:

```
Set ActLayout = PaperSpace.Layout
```

Siehe auch *Kapitel 7.6*

Name

(schreibgeschützt)

Datentyp: *String*

Da die Objekte *ModelSpace* und *PaperSpace* einem Block entsprechen und auch in der Blockauflistung geführt werden, ist diesen beiden Bereichen auch ein Name zugeordnet, der sie identifiziert. Für den Modellbereich lautet der Name „*Model_Space“, für den Papierbereich „*Paper_Space“. Sie werden die Bereiche auch in der Blockauflistung unter diesem Namen finden.

Origin

Datentyp: *Variant (3D-Array – Origin(0 To 2) As Double)*

Diese Eigenschaft speichert die Ursprungskoordinaten des Modell- oder Papierbereichs. Diese Koordinaten sind ausschlaggebend für die Positionierung der Zeichnung, wenn diese als Block oder externe Referenz in einer anderen Zeichnung referenziert wird. Diese Eigenschaft ist gleichzusetzen mit dem AutoCAD-Befehl `BASIS`.

```
Dim InsPoint(2) As Double
InsPoint(0) = 0
InsPoint(1) = 0
InsPoint(2) = 0
ModelSpace.Origin = InsPoint
```

Die wichtigsten Methoden des Modell- und Papierbereichs

GetExtensionDictionary

Rückgabedatentyp: *AcadDictionary*

Diese Funktion ruft das Erweiterungswörterbuch ab, das dem Modell- oder Papierbereich zugeordnet wurde.

```
Set ExtDict = ModelSpace.GetExtensionDictionary
```

Siehe auch *Kapitel 7.14*

GetXData

Parameter: *AppName As String, XDataType, XDataValue*

Mit *GetXData* können Sie alle erweiterten Daten oder die, die einem bestimmten Anwendungsnamen entsprechen, abrufen. Die Daten werden im Array *XDataValue* zurückgegeben, die Datentypen im Array *XDataType*.

```
Dim XDataType As Variant
Dim XDataValue As Variant
ModelSpace.GetXData "", XDataType, XDataValue
```

Siehe auch *Kapitel 7.1.2*

Item

Parameter: *Index*

Rückgabedatentyp: *AcadEntity*

Die *Item*-Funktion gibt das grafische Objekt an einer bestimmten Position innerhalb der *ModelSpace*- oder *PaperSpace*-Objektauflistung wieder:

```
Dim ActEnt As AcadEntity  
Set ActEnt = ThisDrawing.ModelSpace.Item(n)
```

oder:

```
Set ActEnt = ThisDrawing.ModelSpace(n)
```

Normalerweise ist der Datentyp des zurückgegebenen Objekts vorab nicht bekannt, weshalb als Rückgabetyt auch die Klasse *AcadEntity* verwendet wird, da es sich in jedem Fall um ein grafisches Objekt handelt. Wenn Ihnen allerdings im Voraus bekannt ist, um welchen Typ es sich bei dem zurückgegebenen Objekt handelt, können Sie die Deklaration auch präziser vornehmen.

```
Dim ActArc As AcadArc  
Set ActArc = ThisDrawing.ModelSpace.Item(n)
```

Sollten die Datentypen bei der Rückgabe allerdings nicht dem Datentyp der Objektvariablen entsprechen, löst AutoCAD einen Fehler aus. Aus diesem Grund ist es immer empfehlenswert, eine Variable vom Typ *Variant*, *Object*, *AcadObject* oder *AcadEntity* zu verwenden.

SetXData

Parameter: *XDataType*, *XDataValue*

Mit *SetXData* können Sie zusätzliche Daten ab dem Gruppencodebereich 1000 im Modell- oder Papierbereichsobjekt speichern.

```
Dim XDataType As Integer  
Dim XDataValue As Variant  
ModelSpace.SetXData XDataType, XDataValue
```

Siehe auch *Kapitel 7.1.2*

7 Verwaltung der Zeichnungsstruktur

Die Struktur einer Zeichnung ist im Allgemeinen durch ihre Layer, Linientypen, Textstile oder Blöcke gekennzeichnet. Anhand dieser Daten erhält die Zeichnung ihre typische Darstellung und Funktionsweise. Im vorliegenden Kapitel beschäftigen wir uns ausgiebig mit den Auflistungen, in denen die einzelnen Objekte enthalten sind.

Kapitelübersicht

- 7.1 Bearbeiten von AutoCAD-Objekten
- 7.2 Verwenden von Gruppen
- 7.3 Objektverbindungen (Hyperlinks)
- 7.4 Layern verwalten
- 7.5 Linientypen verwalten
- 7.6 Layouts
- 7.7 Verwaltung der Textstile
- 7.8 Benutzerkoordinatensystem (BKS)
- 7.9 Anwendungen in AutoCAD registrieren
- 7.10 Blockdefinitionen verwalten
- 7.11 Benannte Ansichten
- 7.12 Benannte Ansichtsfenster
- 7.13 Bemaßungsstile
- 7.14 Erweiterte Daten (Erweiterungswörterbuch)

7.1 Bearbeiten von AutoCAD-Objekten

7.1.1 Eigenschaften aller AutoCAD-Objekte

Jedes AutoCAD-Objekt, egal ob ein grafisches Objekt (z.B. Linien, Polylinien, Blockreferenzen), Strukturobjekte wie Layer, Linientypen usw. oder die Auflistungsobjekte, in denen sie gespeichert sind, enthält Eigenschaften und Methoden, die alle bestimmte Grundfunktionen erfüllen.

Referenz/Handle

Über die Eigenschaft *Handle* erhalten Sie einen alphanumerischen Wert, der ein Objekt in der Zeichnung eindeutig identifiziert. Dabei handelt es sich um eine hexadezimale Zahl, die dem Objekt zugeordnet wird, nachdem es in der Zeichnung erzeugt wurde. Die Referenz bleibt auch bestehen, wenn Sie die Zeichnung in einem *dxf*-File abspeichern. Nur wenn Sie die Zeichnung oder einen Ausschnitt daraus mithilfe des *Wblock*-Befehls extrahieren, erhalten die Elemente neue Referenzen.

```
ObjRef = Object.Handle
```

ObjectID

Die *ObjectID* enthält eine Ganzzahl, die ein Objekt in AutoCAD eindeutig identifiziert. Im Gegensatz zur Referenz (Handle) wird dieser Wert allerdings bei jedem Zeichnungsstart neu bestimmt. Aus diesem Grund können Sie auch keine dauerhafte Verbindung zwischen zwei Objekten über die *ObjectID* herstellen, da sich der Wert ja ständig ändert.

HasExtensionDictionary

Ob einem Objekt erweiterte Daten zugeordnet wurden oder nicht, können Sie mit dieser Eigenschaft kontrollieren. Ist die Eigenschaft auf den Wert *TRUE* gesetzt, wurde dem Objekt ein Erweiterungswörterbuch, das die erweiterten Daten enthält, zugeordnet. Sie können dann mithilfe der Funktion *GetExtensionDictionary* diese erweiterten Daten auslesen.

Siehe auch *Kapitel 7.14*

Datentyp eines Objekts ermitteln

Häufig ist es erforderlich, den Datentyp eines Objekts zu ermitteln, damit bestimmte Operationen ausgeführt oder umgangen werden. Da ja bekanntlich viele Wege nach Rom führen, stehen uns auch für diese Aufgabe mehrere Wege offen. Zum einen gibt es die Eigenschaft *ObjectName*, die jedes AutoCAD-Objekt besitzt. Diese gibt den Klassennamen zurück, so wie er auch in *dxf*-Dateien abgespeichert wird. Eine Linie besitzt z.B. den Klassennamen „AcDbLine“, womit sich eindeutig ermitteln lässt, um was für ein Objekt es sich handelt. Denselben Wert würden Sie mit der Eigenschaft *EntityName* erhalten, der aber nicht im Objektkatalog auftaucht. Eine weitere Möglichkeit ist die Abfrage der Eigenschaft *EntityType*, die ebenfalls nicht im Objektkatalog geführt wird, aber dennoch existiert. Diese gibt statt des Namens eine Ganzzahl zurück. Jeder Wert entspricht dabei einem bestimmten Objekttyp. Nun brauchen wir uns die einzelnen Zahlen nicht zu notieren, denn auch dafür gibt es eine Konstantenauflistung (*AcEntityName*), in der für jeden Datentyp eine Konstante erzeugt wurde. Bleiben wir einmal bei der Linie. Diese würde in der *EntityType*-Eigenschaft den Wert 19 enthalten. Wenn wir uns die Konstanten für die Linie herausuchen, würde die Konstante *acLine* am besten passen und genau diese enthält ebenfalls den Wert 19. Damit würde sich folgende Abfrage für eine Linie ergeben:

```
If Object.EntityType = acLine Then  
End If
```

Die dritte Möglichkeit ist die Abfrage des Objekttyps mithilfe der *TypeName*-Funktion. Diese Funktion gibt den genauen Klassennamen zurück, so wie er im Objektkatalog zu finden ist. Handelt es sich bei dem entsprechenden Objekt um die besagte Linie, würde die Funktion

```
MsgBox TypeName(Object)
```


den Text „IAcadLine“ ausgeben.

Egal, welche Funktion bzw. Eigenschaft Sie verwenden, alle eignen sich zur Bestimmung des Datentyps eines Objekts. Es ist nur eine reine Gewohnheitsfrage, mit welcher Abfrage Sie arbeiten.

Eigentümerobjekt ermitteln

Bis auf das *Application*-Objekt besitzt jedes Objekt einen so genannten Container, also ein übergeordnetes Objekt. Um dieses übergeordnete Objekt zu ermitteln, müssen Sie lediglich zwei Schritte ausführen. Zuerst rufen Sie mit der Eigenschaft *OwnerID* die *ObjectID* des übergeordneten Objekts ab um dann mithilfe der *ObjectIdToObject*-Funktion einen Verweis auf eben dieses Objekt zu erstellen. Sie können die beiden Schritte auch kombinieren, um Platz zu sparen.

```
Set OwnerObj = ThisDrawing.ObjectIdToObject(Object.OwnerID)
```

Nachdem Sie das Eigentümerobjekt abgerufen haben, könnten Sie nun mit *ObjectName* prüfen, um welches Objekt es sich handelt, z.B. um den Modellbereich, den Papierbereich oder einen Block usw.

7.1.2 Methoden aller AutoCAD-Objekte

Erweiterungswörterbuch abrufen

Das Erweiterungswörterbuch ist eine optionale Sequenz, in der die Referenz auf ein Wörterbuchobjekt gespeichert werden kann. Mithilfe des Erweiterungswörterbuchs können beliebige Datenbankobjekte mit anderen verbunden werden. Damit Sie auf dieses Objekt zugreifen können, enthält jede Objektklasse die Funktion *GetExtensionDictionary*.

```
Set EDict = Object.GetExtensionDictionary
```

Siehe auch *Kapitel 7.12*

Erweiterte Daten

Erweiterte Daten sind Zusatzinformationen, die in jedem Objekt gespeichert werden können. Dabei handelt es sich nicht, wie beim Erweiterungswörterbuch, um Objekte, sondern um Arrays, die eine Vielzahl von Einzeldaten enthalten können. Dieser Datenbereich wird normalerweise nur von Applikationen und nicht von AutoCAD selbst benutzt. AutoCAD verwaltet lediglich diesen Bereich.

Die erweiterten Daten werden in zwei Felder unterteilt. Das erste Feld enthält die Codes, die den Typ des jeweiligen Datensatzes bestimmen. Das zweite Feld legt den eigentlichen Wert des Datensatzes fest.

In der folgenden Tabelle sind alle Codes aufgeführt, die Sie in erweiterten Daten verwenden können.

Code	Datentyp/Feldtyp	Beschreibung
1000	String Zeichenfolge	Zeichenfolgen in erweiterten Daten können bis zu 255 Byte umfassen (das 256. Byte ist für das Nullzeichen reserviert).
1001	String Anwendungsname	Unter diesem Code wird der Anwendungsname (kann bis zu 31 Byte umfassen) abgelegt, dem die folgenden erweiterten Daten zugeordnet sind. Hinweis: Die 1001-Gruppe darf nur zu Beginn einer Gruppenauflistung in die erweiterten Daten eingefügt werden. AutoCAD identifiziert anhand dieses Namens den Beginn einer Gruppe mit erweiterten Daten einer Anwendung.
1002	String Listenanfang „{“ Listenende „}“	Mithilfe der geschweiften Klammern können die Anwendungen die Daten in Listenform gliedern. Die öffnende Klammer leitet eine Liste ein, die schließende Klammer beendet sie. Listen können verschachtelt sein. Beim Lesen der erweiterten Daten für eine bestimmte Anwendung überprüft AutoCAD, ob die Klammern paarweise vorliegen.
1003	String Layername	Name des Layers, der mit den erweiterten Daten verbunden ist.
1004	Binäre Daten	Binäre Daten sind in <i>Blöcken</i> mit variabler Länge gegliedert. Die maximale Länge dieser Blöcke beträgt 127 Byte. In DXF-ASCII-Dateien werden binäre Daten als Folgen von Hexadezimalziffern dargestellt (je zwei Ziffern pro binäres Byte).
1005	Datenbankreferenz	Referenzen von Elementen in der Zeichnungsdatenbank. Anmerkung: Wenn Sie eine Zeichnung mit Referenzen und Referenzen erweiterter Daten mit dem Befehl <code>EINFÜGE</code>, <code>EINFÜGE *</code>, <code>XREF Binden</code>, <code>XBINDEN</code> oder <code>ÖFFNEN</code> (partielles Öffnen) in eine andere Zeichnung einfügen, werden die Referenzen der erweiterten Daten auf dieselbe Weise wie die zugehörigen Elementreferenzen übertragen, so dass die Bindungen erhalten bleiben. Dies gilt auch für den Vorgang <code>URSPRUNG</code> (bei Blöcken) und bei allen anderen AutoCAD-Aktionen. Wenn der Befehl <code>PRÜFUNG</code> eine Referenz erweiterter Daten feststellt, die keiner Elementreferenz in der Zeichnungsdatei entspricht, wird dies als Fehler angesehen. Beim Beheben dieser Fehler setzt <code>PRÜFUNG</code> die Referenz auf 0.
1010, 1020, 1030	Double 3 reelle Zahlen	Drei reelle Werte in der Reihenfolge X, Y, Z. Diese Werte können als Punkt- oder Vektor-Datensatz verwendet werden. AutoCAD nimmt keine Änderungen an diesen Werten vor.

Tabelle 7.1: Codetabelle für erweiterte Daten

Code	Datentyp/Feldtyp	Beschreibung
1011, 1021, 1031	Double Weltbereichsposition	Im Gegensatz zu einfachen 3D-Punkten werden die Weltbereichskoordinaten dieses Punkts gemeinsam mit dem übergeordneten Element, zu dem die erweiterten Daten gehören, versetzt, skaliert, gedreht und gespiegelt. Darüber hinaus wird die Weltbereichsposition gestreckt, wenn der Befehl <code>STRECKEN</code> auf das übergeordnete Element angewendet wird und dieser Punkt im ausgewählten Fenster liegt.
1012, 1022, 1032	Double Weltbereichsverschiebung	3D-Punkt, der gemeinsam mit dem übergeordneten Element skaliert, gedreht und gespiegelt wird (nicht jedoch versetzt oder gestreckt).
1013, 1023, 1033	Double Weltrichtung	3D-Punkt, der gemeinsam mit dem übergeordneten Element gedreht und gespiegelt wird (nicht jedoch versetzt, skaliert oder gestreckt).
1040	Double Reelle Zahl	Reeller Wert
1041	Double Abstand	Reeller Wert, der gemeinsam mit dem übergeordneten Element skaliert wird.
1042	Double Skalierfaktor	Reeller Wert, der gemeinsam mit dem übergeordneten Element skaliert wird. Der Unterschied zwischen Abstand und Skalierfaktor ist anwendungsdefiniert.
1070	Long Ganzzahl	16-Bit-Ganzzahl (mit oder ohne Vorzeichen)

Tabelle 7.1: Codetabelle für erweiterte Daten (Forts.)

Wenn Sie erweiterte Daten speichern, muss der Anwendungsname nicht in der *Registered-Application*-Auflistung registriert worden sein. Es ist nur sinnvoll, ihn dort zu registrieren, damit eine Verbindung der erweiterten Daten zu einer Anwendung besteht.

Siehe auch *Kapitel 7.14*

Erweiterte Daten lesen

Um die erweiterten Daten eines Objekts auszulesen, stellt Ihnen jedes Objekt die Funktion *GetXData* zur Verfügung. Mit dieser Funktion können Sie entweder alle oder nur erweiterte Daten einer bestimmten Anwendung abrufen. Die Funktion verlangt dazu die Angabe von drei Parametern. Im ersten legen Sie den Namen der Anwendung fest, der die entsprechenden Daten zugeordnet sind. Wenn Sie statt des Anwendungsnamens eine leere Zeichenfolge angeben, werden alle erweiterten Daten des Objekts abgerufen. Für die beiden anderen Parameter müssen Sie lediglich jeweils eine Variable vom Typ *Variant* festlegen. Diese Variablen enthalten nach dem Aufruf der Prozedur die erweiterten Daten des Objekts.

```
Dim XTypeOut As Variant
Dim XDataOut As Variant
Object.GetXData "MyApp", XTypeOut, XDataOut
```

Die Variable *TypeOut* enthält nach dem Aufruf der Prozedur ein Array mit den Gruppencodes der Datensätze, die Variable *DataOut* die dazugehörigen Werte. Sie können die einzelnen Daten der Variablen anschließend mithilfe einer *For...Next*-Schleife durchlaufen.

```
For i = 0 To Ubound(XTypeOut)
  XCode = XTypeOut(i)
  XData = XDataOut(i)
  Select Case XCode
    Case 1000
    Case 1001
  End Select
Next
```

Die erweiterten Daten eines Ansichtsfensters könnten beispielsweise folgenden Inhalt besitzen.

CodeArray	Code	DatenArray	Datenwert
XTypeOut(0)	1001	XDataOut(0)	ACAD
XTypeOut(1)	1000	XDataOut(1)	MVIEW
XTypeOut(2)	1002	XDataOut(2)	{
XTypeOut(3)	1070	XDataOut(3)	16
XTypeOut(4)	1010	XDataOut(4)(0)	0
		XDataOut(4)(1)	0
		XDataOut(4)(2)	0
		XDataOut(5)(0)	0
XTypeOut(5)	1010	XDataOut(5)(1)	0
		XDataOut(5)(2)	1
		XDataOut(6)	0
XTypeOut(6)	1040	XDataOut(6)	0
XTypeOut(7)	1040	XDataOut(7)	2897,50620811083
.	.	.	.
.	.	.	.
.	.	.	.
XTypeOut(87)	1003	XDataOut(87)	UEBERFENSTER
XTypeOut(88)	1002	XDataOut(88)	}
XTypeOut(89)	1002	XDataOut(89)	}

Tabelle 7.2: Beispiel eines erweiterten Datensatzes

Erweiterte Daten schreiben

Das Schreiben der erweiterten Daten erfolgt auf ähnliche Weise wie das Lesen. Sie müssen hierfür lediglich ein Array mit den Codes und ein Array mit den Daten erstellen und anschließend mit der Anweisung *SetXData* in die erweiterten Daten des Objekts schreiben. Da die Codes immer vom Typ Integer sind, können Sie das Array der Codes ebenfalls mit diesem Datentyp deklarieren. Bei dem Array, das die Daten an sich enthält, ist das etwas anders. Da die erweiterten Daten unterschiedliche Datentypen (*String*, *Double*, *Array* ...) besitzen können, muss das Array, das diese Daten enthält, vom Datentyp *Variant* sein, da zunächst ja nicht klar ist, welche Daten in den einzelnen Feldern abgelegt werden.

```
Dim XTypeIn(n) As Integer
Dim XDataIn(n) As Variant
```

Die Anzahl der Datenfelder (n) hängt natürlich davon ab, welche und wie viele erweiterte Daten Sie abspeichern wollen. Fakt ist: Das erste Feld ist mit dem Anwendungsnamen belegt.

```
XTypeIn(0) = 1001
XDataIn(0) = "MyApp"
```

In die übrigen Felder können Sie dann die verschiedensten Daten schreiben, wobei Sie sich nur an die Tabelle 7.1 halten müssen. Da die Daten in einem Array abgelegt werden, spielt es keine Rolle, wie oft Sie einen bestimmten Datentyp verwenden. Sie könnten z.B. x-mal den Code 1000 (Zeichenfolge) hintereinander verwenden. Allerdings würde ich empfehlen, die Daten in diesem Fall in Listenform, also mit öffnender „{“ und schließender „}“ Klammer, abzuspeichern.

Nachdem die beiden Arrays erstellt wurden, müssen sie nur noch abgespeichert werden. Dazu rufen Sie die Anweisung *SetXData* über das entsprechende Objekt auf, dem die erweiterten Daten zugeordnet werden sollen. Als Parameter werden die beiden Arrays verwendet und schon ist der Datensatz abgespeichert.

```
Object.SetXData XTypeIn, XDataIn
```

Tipp

Sie können für einen Anwendungsnamen nur einen erweiterten Datensatz abspeichern. Wenn Sie erweiterte Daten unter einem Anwendungsnamen speichern, der bereits existiert, werden die vorhandenen Daten durch die neuen ersetzt.

Erweiterte Daten löschen

Da es keine Funktion zum Löschen eines erweiterten Datensatzes gibt, müssen wir an dieser Stelle improvisieren. Wie Sie ja bereits wissen, überschreiben die erweiterten Daten eines Anwendungsnamens die bereits vorhandenen erweiterten Daten eben dieser Anwendung. Diesen Umstand können wir nutzen, um die erweiterten Daten komplett zu löschen. Der Trick dabei ist, als erweiterten Datensatz nur den Anwendungsnamen zu übergeben. Dadurch wird der komplette erweiterte Datensatz entfernt.

```
Dim XTypeIn(0) As Integer
Dim XDataIn(0) As Variant

XTypeIn(0) = 1001
XDataIn(0) = "MyApp"
Object.SetXData XTypeIn, XDataIn
```

7.1.3 Ereignisprozeduren aller AutoCAD-Objekte

Sobald ein Objekt – egal ob grafisches Element, Objektauflistung oder das darin verwaltete Objekt – geändert wird, löst dies ein Ereignis in AutoCAD aus. Im Normalfall können Sie darauf mit den Ereignisprozeduren *ObjectModified*, *ObjectErased* und *ObjectAdded* der Dokumentklasse (*Kapitel 6.2.1*) reagieren. In einigen Fällen kann es aber notwendig sein, ein bestimmtes Objekt direkt zu überwachen. Nehmen wir einmal an, Sie lassen die Daten eines Objekts in einem ungebundenen Formular anzeigen. Der Benutzer kann ja in der Zeichnung weiterarbeiten und das Objekt dort anpassen. Um nun auf diese Änderungen zu reagieren, besitzt jedes Objekt die Ereignisprozedur *Modified*. Dieses Ereignis bezieht sich ganz speziell auf das entsprechende Objekt. Allerdings muss dieses Ereignis erst initialisiert werden, bevor

Sie die Prozedur verwenden können. Dazu müssen Sie zunächst eine Objektvariable mit den Ereignissen der Klasse deklarieren, indem Sie das Schlüsselwort *WithEvents* vor der Objektvariablen positionieren. Außerdem müssen Sie in diesem Fall auch die genaue Objektklasse angeben. Für eine Objektvariable vom Typ *AcadBlockReference* würde die Deklarationszeile folgendermaßen ausschauen:

```
Public WithEvents MyObject As AcadBlockReference
```

Damit wurde VBA angewiesen, auch die Ereignisprozeduren der Klasse zu verwenden. Eine Einschränkung gibt es allerdings auf diesem Gebiet. Sie können Objekte mit Ereignisprozeduren nur in Klassenmodulen (z.B. *ThisDrawing* oder Formularcode), nicht aber auf Modulebene deklarieren. Damit wäre der erste Schritt bereits abgeschlossen und Sie können den entsprechenden Code, der bei Auslösung des Ereignisses ausgeführt werden soll, in die Ereignisprozedur der Objektvariablen eintragen. Um nun die Prozedur nutzen zu können, muss die Objektvariable auch initialisiert werden, was bedeutet, dass Sie einen Verweis auf ein ganz bestimmtes Objekt der Variablen zuweisen müssen. Dazu stehen verschiedene Möglichkeiten zur Verfügung. Entweder Sie erstellen ein Objekt mit einer *Add*-Anweisung des entsprechenden Containers und speichern den zurückgegebenen Verweis in der Variablen oder Sie wählen ein vorhandenes Objekt über die entsprechende Funktion des jeweiligen Containers. Egal, wie Sie diesen Verweis ermitteln, die Ereignisprozedur wird nun nur ausgelöst, wenn das Objekt, auf das die Variable verweist, irgendwie geändert wird.

Anwendungsbeispiel

(Beispielprojekt: 07\Modified.dvb)

Als kleines Anwendungsbeispiel zu diesem Thema werden wir die Daten einer Polylinie, die zuvor durch den Benutzer gewählt wurde, in einem Formular darstellen. Ändert der Benutzer diese Polylinie – egal, welche Eigenschaft – löst dies das Ereignis *Modified* aus und die Formulardaten werden aktualisiert.

Startprozedur

In der Startprozedur passiert zunächst nichts Weltbewegendes. Der Benutzer wird lediglich aufgefordert, eine Polylinie in der Zeichnung auszuwählen.

```
Public Sub SelectPolyline()  
On Error Resume Next  
Set Util = New MyUtility  
Util.GetEntity Object, PickedPoint, "Polylinie wählen"
```

Hat er dies getan, wird geprüft, ob es sich bei dem gewählten Objekt auch tatsächlich um eine Polylinie handelt. Ist dies der Fall, wird der Verweis auf das Objekt an die Formularvariable *MyObject* übergeben. Anschließend wird eine Formularprozedur aufgerufen, die die Daten dieser Polylinie an die Steuerelemente des Formulars übertragen soll, und danach das Formular ungebunden eingeblendet.

```
Select Case TypeName(Object)  
Case "IAcadPolyline", "IAcad3DPolyline", "IAcadLWPolyline"  
Set ModifiedFrm.MyObject = Object  
ModifiedFrm.initFormular  
ModifiedFrm.Show 0  
End Select  
End Sub
```

Der Code der Initialisierungsprozedur soll uns an dieser Stelle erst einmal nicht interessieren. Vielmehr soll das Augenmerk auf die Ereignisprozedur gerichtet werden. Der erste Schritt muss hierbei die Deklaration der Objektvariablen im Formular sein. Da in AutoCAD verschiedene Formen von Polylinien existieren, die auch durch verschiedene Klassen abgebildet werden, müssen wir als Datentyp eine Klasse wählen, die alle Polylinien abbilden und außerdem das *Modified*-Ereignis auslösen kann. Eine dieser Klassen ist *AcadEntity* (siehe auch *Kapitel 8.3.2*).

```
Public WithEvents MyObject As AcadEntity
```

Durch die Deklaration der Variablen mit dem Schlüsselwort *WithEvents* können Sie nun auf die Ereignisprozedur *Modified* zugreifen. In diese Ereignisprozedur wird der Code eingetragen, der bei Auslösung des Ereignisses ausgeführt werden soll. Da *Modified* auch ausgelöst wird, wenn das entsprechende Objekt gelöscht wurde, müssen wir zunächst prüfen, ob das Objekt in der Zeichnung überhaupt noch existiert. Dazu müssen Sie nur auf irgendeine Eigenschaft des Objekts zugreifen.

```
Private Sub MyObject_Modified(ByVal pObject As IAcadObject)
On Error Resume Next
Handle = pObject.Handle
```

Wurde es gelöscht, löst AutoCAD den Fehler Nummer -2145386420 (<Objekt wurde gelöscht>) aus, auf den wir dann nur reagieren müssen.

```
If Err.Description = "Objekt wurde gelöscht" Then
    MsgBox Err.Description, vbCritical
    Unload Me
Else
```

Andernfalls wird die Initialisierungsprozedur erneut aufgerufen, um den Inhalt der Steuerelemente mit den aktuellen Daten des Objekts zu aktualisieren.

```
    InitFormular
End If
End Sub
```

Da die Prozedur als Argument die Objektvariable *pObject* liefert, in der ebenfalls ein Verweis auf das Objekt, das geändert wurde, gespeichert ist, könnten wir auch diese Variable statt *MyObject* zur weiteren Verarbeitung innerhalb der Ereignisprozedur verwenden.

7.1.4 Arbeiten mit Auflistungen

Bevor wir mit den einzelnen Auflistungsobjekten der AutoCAD-Objektbibliothek fortfahren, wollen wir uns zunächst noch mit einigen Eigenschaften und Prozeduren befassen, die in allen Auflistungsobjekten verfügbar sind. Diese Eigenschaften dienen im Allgemeinen der Verwaltung der Objekte, die in diesen Auflistungen enthalten sind.

Eigenschaften

Mit der *Count*-Eigenschaft können Sie die Anzahl der Objekte ermitteln, die in der angesprochenen Auflistung enthalten sind. Enthält die Auflistung keine Objekte, was bei einigen der Fall sein kann, gibt *Count* den Wert 0 zurück. Sie können die *Count*-Eigenschaft beispielsweise als Endwert einer *For..Next*-Schleife verwenden, um alle Objekte der Auflistung zu durchlaufen.

```
For i= 1 To Blocks.Count
    Set xBlock = Blocks.Item(i)
Next
```

Neben der *Count*-Eigenschaft ist auch die Eigenschaft *Name* von Bedeutung. Jedes Objekt das in einer Auflistung enthalten ist, besitzt diese Eigenschaft. Sie speichert den Namen des Objekts und muss eindeutig definiert sein, was bedeutet, dass es innerhalb der Auflistung kein weiteres Objekt mit demselben Namen geben darf und auch nicht gibt. Wenn Sie versuchen, einem Objekt einen bereits vergebenen Namen zuzuweisen, löst AutoCAD einen Fehler aus. Im Normalfall kann der Wert der *Name*-Eigenschaft geändert werden, indem Sie einfach eine neue Zeichenfolge festlegen.

```
Object.Name = "neuer Name"
```

Ein Name darf maximal 256 Zeichen enthalten, wobei alle Zeichen bis auf folgende zulässig sind:

```
< > / \ : ? * | , = " `
```

Methoden

Neben den zuvor besprochenen Eigenschaften spielen auch zwei Methoden eine wichtige Rolle bei der Verwaltung der Objektauflistungen.

Mit der *Add*-Anweisung fügen Sie der Auflistung ein neues Objekt hinzu. Bis auf einige Ausnahmen, die wie immer die Regel bestätigen, benötigt diese Anweisung nur die Angabe des Namens des hinzuzufügenden Objekts im Parameter der Funktion. Die Zeichenfolge, die Sie hier festlegen, findet sich dann in der *Name*-Eigenschaft des erstellten Objekts wieder. Das bedeutet natürlich, dass in der Auflistung noch kein Objekt mit demselben Namen existieren darf. Wurde die Funktion erfolgreich ausgeführt, gibt diese einen Verweis auf das erstellte Objekt zurück. Sie könnten diesen Verweis dann in einer Objektvariablen zur weiteren Verarbeitung speichern:

```
Set NewLayer = Layers.Add("neuer Layer")
```

müssen es aber nicht:

```
Layers.Add "neuer Layer"
```

Die zweite Methode, die in Zusammenhang mit Objektauflistungen wichtig ist, ist die *Item*-Funktion. Mit dieser Funktion können Sie einen Verweis auf ein Objekt aus der Auflistung unter Angabe der Position oder des Namens abrufen und in einer Objektvariablen speichern. Da es sich bei dieser Eigenschaft um die Standardeigenschaft handelt, brauchen Sie also die Eigenschaft beim Aufruf des Objekts nicht extra zu benennen.

```
Set ActLayer = Layers.Item(n)
```

oder

```
Set ActLayer = Layers.Item("0")
```

oder

```
Set ActLayer = Layers(n)
```

oder

```
Set ActLayer = Layers("0")
```


Wenn Sie den Namen eines Objekts dazu verwenden, ein Objekt abzurufen, müssen Sie darauf achten, dass die Variable, die Sie für den Parameter verwenden, vom Datentyp *String* ist (falls Sie den Namen nicht direkt als Zeichenfolge festlegen). In manchen Fällen kann die Verwendung einer *Variant*-Variablen nicht zum gewünschten Ergebnis führen, obwohl in der Variablen eine Zeichenfolge gespeichert wurde. In diesem Fall sollten Sie sicher gehen, dass die *Item*-Funktion eine *String*-Variable erhält, und die Variable in die *CStr()*-Funktion einpacken.

```
Dim LayerName
LayerName = "0"
Set ActLayer = Layers.Item(CStr(LayerName))
```

Je nachdem, welche Auflistung Sie ansprechen, erhalten Sie Verweise unterschiedlichen Datentyps. Die *Layers*-Auflistung gibt Objektverweise vom Datentyp *AcadLayer* zurück, die *Block*-auflistung hingegen Objektverweise des Datentyps *AcadBlock*.

Obwohl die Auflistung auch über eine *Delete*-Anweisung verfügt, können Sie diese nicht verwenden, sondern müssen auf die *Delete*-Anweisung der in der Auflistung enthaltenen Objekte zurückgreifen, um diese zu löschen.

7.2 Verwendung von Gruppen

Mit einer Gruppe können Sie so eine Art benannten Auswahl Satz von Objekten definieren, die im Gegensatz zu unbenannten Auswahl Sätzen (*SelectionSets*) mit der Zeichnung gespeichert werden. Die Gruppendefinitionen sind ebenso abrufbar, wenn Sie die Zeichnung, die diese enthalten, als externe Referenz verwenden oder in eine andere Zeichnung einfügen. Sie können jedoch erst dann direkt auf Gruppen zugreifen, die in einer externen Referenz oder einem Block definiert wurden, wenn Sie die externe Referenz oder Blöcke aufgelöst haben.

7.2.1 Gruppenuflistung

In der Gruppenuflistung werden alle in der Zeichnung verfügbaren Gruppendefinitionen verwaltet. Diese Auflistung wird in VBA durch die Klasse *AcadGroups* dargestellt und ähnelt in ihrer Funktionsweise der anderer Auflistungen in VBA. Sie können Gruppen hinzufügen, indem Sie die *Add*-Funktion unter Angabe des Gruppennamens verwenden. Um vorhandene Gruppen aus der Auflistung abzurufen, benutzen Sie wieder die *Item*-Funktion. Zugriff auf die Gruppenuflistung erhalten Sie, indem Sie das Gruppenobjekt der Zeichnung abrufen, was Sie mit folgender Codezeile erreichen:

```
Set ActGroups = ThisDrawing.Groups
```

Diese Codezeile funktioniert natürlich nur mit dem jeweils aktuellen Dokument. Wenn Sie die Gruppen einer anderen Zeichnung bearbeiten wollen, müssen Sie diese über die Dokumentauflistung des Anwendungsobjekts abrufen.

```
Set ActGroups = Documents(n).Groups
```

Durch diese Zuweisung entsteht ein Objekt des Typs *AcadGroups*, das wiederum Elemente zur Weiterverarbeitung einzelner Gruppendefinitionen enthält.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.3: Eigenschaften der Gruppenauflistung

Eigenschaft	Beschreibung
<i>Add</i>	Diese Anweisung fügt eine Gruppe der Auflistung hinzu.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.4: Methoden der Gruppenauflistung

Gruppen hinzufügen/vorhandene Gruppen abrufen

Wie wir bereits wissen, stellt die Klasse *AcadGroups* die *Add*-Funktion zur Verfügung, um der Gruppenauflistung eine neue Gruppe hinzuzufügen. Wie bei allen anderen Auflistungen müssen Sie auch bei dieser Funktion den Namen der neuen Gruppe im Parameter festlegen. Der Name der Gruppe muss eindeutig sein, was bedeutet, dass noch keine Gruppe mit diesem Namen existiert. Folgende Zeilen erzeugen z.B. eine Gruppe mit dem Namen „Gruppe1“ und fügen diese der Auflistung hinzu. Als Ergebnis liefert die Funktion ein Objekt der Klasse *AcadGroup*, das Sie einer Variablen zuweisen können.

```
Dim NewGroup As AcadGroup
Set NewGroup = ThisDrawing.Groups.Add("Gruppe1")
```

Wenn Sie auf eine bereits definierte Gruppe zugreifen wollen, benutzen Sie wie gehabt die *Item*-Funktion der Gruppenauflistung.

```
Dim CrGroup As AcadGroup
Set CrGroup = ThisDrawing.Groups.Item(n)
```

oder

```
Set CrGroup = ThisDrawing.Groups(n)
```

Mit beiden Schreibweisen erstellen Sie einen Verweis auf die n-te Gruppe des Auflistungsbereichs. Welche Sie benutzen, bleibt natürlich Ihnen überlassen.

Wie bei Layern oder Blöcken können Sie auch mit der *Item*-Funktion der Gruppenauflistung die Gruppennamen verwenden, um auf eine einzelne Gruppe zuzugreifen.

```
Dim GroupName As String
Dim CrGroup As AcadGroup
GroupName = "Gruppe1"
Set CrGroup = ThisDrawing.Groups(GroupName)
```

7.2.2 Einzelne Gruppen verarbeiten

Wie Sie einen Verweis auf einzelne Gruppen erstellen, haben wir ja bereits geklärt. Bleibt nur noch die Verarbeitung eines einzelnen Gruppenobjekts. Wie jedes andere Objekt enthält auch eine Gruppe Eigenschaften und Methoden, mit denen Sie deren Daten bearbeiten können.

Gruppenelemente verwalten

Da das Gruppenobjekt dazu dient, grafische Objekte zu einer Auswahl zusammenzufassen, enthält dieses auch Eigenschaften und Methoden, mit denen Sie diese Aufgaben realisieren können. Mit der *AppendItems*-Anweisung fügen Sie der Gruppe grafische Objekte hinzu. Diese Anweisung verlangt, anders als eine „normale“ *Add*-Anweisung, ein Array mit den Objekten, die der Gruppe hinzugefügt werden sollen. Auch wenn Sie nur ein Objekt hinzufügen wollen, müssen Sie dieses in einem Array platzieren. Die Felder dieses Objekts müssen mit dem Datentyp *AcadEntity* deklariert werden, da mit dieser Klasse alle grafischen Objekte von AutoCAD abgebildet werden können. Die Größe der Arrays hängt davon ab, wie viele Objekte Sie der Gruppe hinzufügen wollen.

```
Dim ObjList(0) As AcadEntity
Set ObjList(0) = ModelSpace.Item(0)
ActGroup.AppendItems ObjList
```

Da es sich bei der Liste der Objekte normalerweise so lange um ein Array unbekannter Größe handelt, bis der Benutzer die Objektauswahl abgeschlossen hat, wird das Datenfeld auch erst zu diesem Zeitpunkt dimensioniert, z.B. so:

```
ReDim ObjList(AcSSet.Count - 1) As AcadEntity
```

Um Objekte wieder aus der Gruppe zu entfernen, müssen Sie die *RemoveItems*-Anweisung verwenden. Im Parameter dieser Anweisung müssen Sie, genau wie mit *AppendItems*, ein Array der Objekte angeben, die aus der Gruppe entfernt werden sollen.

```
Dim Objs(0) As AcadEntity
Set Objs(0) = ActGroup.Item(0)
ActGroup.RemoveItems Objs
```

Zugriff auf einzelne Objekte erhalten Sie wie bei den Auflistungen mittels der *Item*-Anweisung und mit der *Count*-Eigenschaft können Sie die Anzahl der Objekte abrufen, die in der Gruppe zusammengefasst wurden.

Weitere Eigenschaften und Methoden

Das Gruppenobjekt enthält zusätzlich zu den bereits besprochenen Eigenschaften und Methoden und denen, die sowieso jedes AutoCAD-Objekt besitzt, auch noch die allgemeinen Eigenschaften der grafischen Objekte. Dazu gehören unter anderem die *Layer*- und die *Linientyp*- oder *PlotStyleName*-Eigenschaft. Im Gegensatz zu den grafischen Objekten können die Werte für diese Eigenschaften beim Gruppenobjekt allerdings nur geschrieben werden und gelten auch nicht direkt für das Gruppenobjekt. Der Wert, den Sie der entsprechenden Eigenschaft zuweisen, ändert immer die gleichnamige Eigenschaft der der Gruppe zugeordneten grafischen Objekte. Im Einzelnen bedeutet dies: Ändern Sie z.B. die *Layer*-Eigenschaft, ändern sich die Layer der in der Gruppe enthaltenen Objekte. Ändern Sie die *Color*-Eigenschaft, ändert sich die Farbe der Objekte usw. Aus diesem Grund kann der aktuelle Wert der Eigenschaft auch nicht ermittelt werden, da die Eigenschaften eigentlich nur als Durchgangsstationen für die gleichnamigen Eigenschaften der Gruppenelemente gedacht sind.

Genauso verhält es sich auch mit den Methoden *Highlight* und *Update*.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Layer</i> (schreibgeschützt)	Zeigt den Layer des Objekts an.
<i>Linetype</i> (schreibgeschützt)	Zeigt den Linientyp des Objekts an.
<i>LinetypeScale</i> (schreibgeschützt)	Zeigt den aktuellen Linientypfaktor des Objekts an.
<i>Lineweight</i> (schreibgeschützt)	Zeigt die aktuellen Linienstärke für das Objekt an.
<i>Name</i>	Bestimmt den Namen der Gruppe
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektamen finden Sie im Anhang.

Tabelle 7.5: Weitere Eigenschaften der Gruppenobjekte

Eigenschaft	Beschreibung
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den <i>PlotStilName</i> für das Objekt zurück
<i>Visible</i>	Zeigt die Sichtbarkeit des Objekts an.

Tabelle 7.5: Weitere Eigenschaften der Gruppenobjekte (Forts.)

Eigenschaft	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.6: Weitere Methoden der Gruppenobjekte

Anwendungsbeispiel

(Beispielprojekt: 07\AcadGroups.dvb)

Aber schauen wir uns doch einfach mal ein Beispielprojekt zum Thema Gruppen an. In diesem Programm sind Funktionen enthalten, die das Hinzufügen bzw. Entfernen von Objekten verdeutlichen. Ziel des Programms wird es sein, die in der Zeichnung bereits definierten Gruppen in einem Formular aufzulisten und dann Gruppen gegebenenfalls hinzuzufügen oder zu entfernen. Außerdem soll es dem Benutzer möglich sein, Objekte, die einer bestimmten Gruppe angehören, aus dieser zu entfernen und natürlich auch neue Objekte der Gruppe hinzuzufügen.

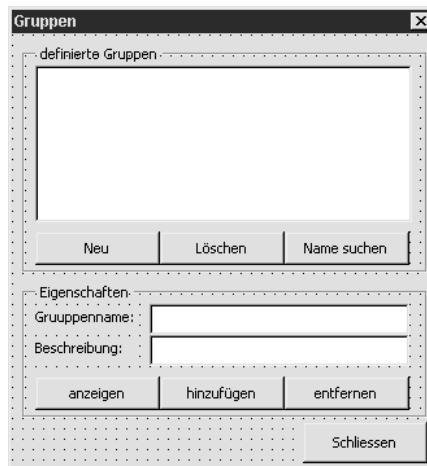


Abbildung 7.1: Formularentwurf

Nachdem Sie ein neues Projekt angelegt haben, fügen Sie diesem ein Formular hinzu, auf dem Sie dann ein Listenfeld, einige Schaltflächen und zwei Textfelder positionieren. Nachdem das Formular fertig gestellt wurde, sollte es der Abbildung 7.1 entsprechen. Benennen müssen Sie das Formular übrigens mit „GroupsFrm“, damit der Code anschließend korrekt funktioniert. Die Namen der Steuerelemente können so belassen werden, wie sie beim Einfügen derselben durch VBA erzeugt wurden. Sie müssen im Formularcode dann nur darauf achten, dass die Zuordnung richtig vorgenommen wird.

Dem Klassenmodul *ThisDrawing* wird nun folgende Startprozedur hinzugefügt:

```
Public Sub showGroups()  
GroupsFrm.initGroupList  
GroupsFrm.Show  
End Sub
```

Die Funktionsweise der Prozedur dürfte ja eigentlich bekannt sein, es folgt also nur eine kurze Zusammenfassung: zunächst eigene Initialisierungsprozedur aufrufen und danach Formular starten.

Die Initialisierungsroutine *initGroupList* soll dann auch die erste Prozedur des Formularcodes werden. Da der Liste ja nur die Namen der Gruppen hinzugefügt werden, ist diese auch recht simpel aufgebaut.

```
Public Sub initGroupList()  
Dim ActGroup As AcadGroup  
On Local Error Resume Next  
ListBox1.Clear  
For Each ActGroup In ThisDrawing.Groups  
    ListBox1.AddItem ActGroup.Name  
Next  
End Sub
```

Nach Deklaration der Variablen *ActGroup*, die die jeweils aktuelle Gruppe aufnehmen soll, braucht nur noch das *ListBox*-Steuerelement von eventuell vorhandenen Einträgen bereinigt zu werden. Danach wird die Auflistung der Gruppen mittels *For...Each*-Schleife durchlaufen und bei jedem Durchgang die jeweilige Gruppe an die Variable *ActGroup* übergeben. Der Name der Gruppe, auf die die Variable gerade verweist, wird der *ListBox* hinzugefügt.

Setzen wir nun den Formularcode mit den einzelnen Ereignisprozeduren fort. Als Erstes wollen wir auf die Auswahl eines Gruppennamens in der *ListBox* reagieren. Wenn das der Fall ist, sollen der Name und die Beschreibung der Gruppe in die Textfelder übertragen werden. Beim Namen ist das ja relativ einfach, da ja nur der *ListBox*-Eintrag in das Textfeld geschrieben werden muss:

```
Private Sub ListBox1_Click()  
Dim ActGroup As AcadGroup  
TextBox1.Text = ListBox1.Text
```

Die Beschreibung ist schon ein wenig komplizierter, da sie nicht über die Eigenschaften des Gruppenobjekts abrufbar ist. Wir müssen hier also einen anderen Weg, und zwar den über den *XRecord*, gehen.

```
End Sub
```

Das Zurückschreiben des Namens erfolgt, wenn der Benutzer den Inhalt von *TextBox1* geändert hat. Zunächst prüfen wir, ob der Name der Gruppe wirklich geändert wurde, indem der Inhalt des Textfelds mit dem der Liste verglichen wird.

```
Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim GroupName As String
Dim ActGroup As AcadGroup
On Local Error Resume Next
If TextBox1.Text <> ListBox1.Text Then
    GroupName = TextBox1.Text
    If TypeName(ThisDrawing.Groups(GroupName)) = "Nothing" Then
```

Ist das nicht der Fall, wurde ein neuer Name eingegeben und wir können die Gruppe aus der Gruppenuflistung abrufen, die in der *ListBox* gewählt wurde. Anschließend wird dort der neue Name eingetragen und gleichzeitig der Inhalt der *ListBox* angepasst.

```
        Set ActGroup = ThisDrawing.Groups(ListBox1.Text)
        ActGroup.Name = GroupName
        ListBox1.List(ListBox1.ListIndex) = GroupName
    Else
        MsgBox "Gruppenname " & GroupName & _
            " ist bereits vorhanden", vbCritical
        Cancel = True
    End If
End If
End Sub
```

Über die Schaltfläche werden wir die Elemente, die der gewählten Gruppe zugeordnet sind, in der Zeichnung hervorheben. Nachdem der Benutzer eine Taste betätigt hat, wird die Hervorhebung wieder aufgehoben.

```
Private Sub CommandButton6_Click()
Dim ActGroup As AcadGroup
On Local Error Resume Next
Me.Hide
Set ActGroup = ThisDrawing.Groups(ListBox1.Text)
For Each Obj In ActGroup
    Obj.Highlight True
Next
ThisDrawing.Utility.GetString 0, "weiter mit Tastendruck"
For Each Obj In ActGroup
    Obj.Highlight False
    Obj.Update
Next
Me.Show
End Sub
```

Die nächste Ereignisprozedur wird den Namen einer Gruppe ermitteln, indem der Benutzer ein Objekt in der Zeichnung wählt und dieses dann in den einzelnen Gruppen sucht. Wird das Objekt in einer Gruppe gefunden, wird der Name dieser Gruppe im Listenfeld markiert.

```
Private Sub CommandButton7_Click()
Dim Object As Object
Dim ActGroup As AcadGroup
```

```
On Local Error Resume Next
Me.Hide
ThisDrawing.Utility.GetEntity Object, PP
```

Nachdem der Benutzer ein Objekt in der Zeichnung gewählt hat, was wir mit der *TypeName*-Funktion prüfen, werden alle Gruppen nacheinander durchsucht.

```
If TypeName(Object) <> "Nothing" Then
    For i = 0 To ListBox1.ListCount - 1
```

Sie können auch die *Count*-Eigenschaft der Gruppenuflistung statt der der *ListBox* verwenden, um die Gruppen zu durchlaufen. Auf jeden Fall wird anschließend jede Gruppe nach dem gewählten Objekt durchsucht. Stimmt das Objekt mit dem der Gruppe überein, kann der *ListIndex* der *ListBox* auf den aktuellen Schleifenwert gesetzt und die Funktion vorzeitig verlassen werden.

```
        Set ActGroup = ThisDrawing.Groups(i)
        For Each Obj In ActGroup
            If Obj = Object Then
                ListBox1.ListIndex = i
                Exit Sub
            End If
        Next
    Next
End If
Me.Show
End Sub
```

Schließlich brauchen wir noch die Ereignisprozeduren zum Hinzufügen und Entfernen von Objekten zur und aus der Gruppe. Das Hinzufügen sollte kein Problem darstellen., denn wir brauchen ja nur das Formular auszublenzen, einen Auswahlatz zu definieren und diesen vom Benutzer mit Objekten füllen zu lassen, um danach die gewählten Objekte der Gruppen hinzuzufügen. Anschließend wird das Formular wieder eingeblendet.

```
Private Sub CommandButton4_Click()
Dim AcSSet As AcadSelectionSet
Dim ActGroup As AcadGroup
Dim ObjList
On Local Error Resume Next
Me.Hide
If TypeName(ThisDrawing.SelectionSets("Auswahl")) = "Nothing" Then
    ThisDrawing.SelectionSets.Add "Auswahl"
End If
Set AcSSet = ThisDrawing.SelectionSets("Auswahl")
AcSSet.SelectOnScreen
If AcSSet.Count <> 0 Then
```

Nach der Definition des Auswahlatzes und der Objektauswahl durch den Benutzer können wir die Objekte des Auswahlatzes an die Gruppe übergeben. Dazu dimensionieren wir ein Array mit der Größe der Anzahl der im Auswahlatz enthaltenen Objekte. Da das erste Objekt im Feld 0 des Arrays abgelegt wird, können wir die Obergrenze des Arrays auf Anzahl -1 setzen. Danach wird in jedes Feld des Arrays jeweils ein Objektverweis abgelegt.


```

ReDim ObjList(AcSSet.Count - 1) As AcadEntity
For i = 0 To AcSSet.Count - 1
    Set ObjList(i) = AcSSet.Item(i)
Next

```

Nachdem das Array komplett gefüllt wurde, braucht nur noch die gewählte Gruppe abgerufen zu werden, um dieser mithilfe der *AppendItems*-Anweisung die Objekte hinzuzufügen. Sie brauchen dabei nicht zu kontrollieren, ob das Objekt bereits in der Gruppe enthalten ist, da jedes Objekt nur einmal aufgenommen werden kann.

```

Set ActGroup = ThisDrawing.Groups(ListBox1.Text)
ActGroup.AppendItems ObjList
End If
AcSSet.Clear
Me.Show
End Sub

```

Das Entfernen mehrerer Objekte aus einer Gruppe ist schon etwas aufwendiger, da die komplette Gruppe selektiert wird, wenn der Benutzer die Objekt über einen Auswahlsatz markiert. Wir müssen die Gruppe also auflösen, dürfen aber die Objekte nicht verlieren. Darum speichern wir sie temporär in einem Array und ordnen sie nach der Benutzerauswahl der Gruppe wieder zu.

```

Private Sub CommandButton5_Click()
Dim AcSSet As AcadSelectionSet
Dim ActGroup As AcadGroup
Dim TempGrp, ObjList
On Local Error Resume Next
Me.Hide

```

Nachdem das Formular ausgeblendet wurde, folgt der Teil, der die Objekte der Gruppe an ein „temporäres“ Array übergibt. Dazu dimensionieren wir das Array mit der Anzahl der Gruppenobjekte und übergeben danach die einzelnen Objekte an das Array.

```

Set ActGroup = ThisDrawing.Groups(ListBox1.Text)
ReDim TempGrp(ActGroup.Count - 1) As AcadEntity
For i = 0 To ActGroup.Count - 1
    Set TempGrp(i) = ActGroup.Item(i)
Next

```

Anschließend können alle Objekte aus der Gruppe entfernt werden, damit der Benutzer die Objekte einzeln auswählen kann.

```

ActGroup.RemoveItems TempGrp
If TypeName(ThisDrawing.SelectionSets("Auswahl")) = "Nothing" Then
    ThisDrawing.SelectionSets.Add "Auswahl"
End If
Set AcSSet = ThisDrawing.SelectionSets("Auswahl")
AcSSet.SelectOnScreen

```

Bevor nun die gewählten Objekte aus dem Auswahlsatz in ein Array übertragen werden, um diese anschließend mithilfe der *RemoveItems*-Anweisung aus der Gruppe zu entfernen, müssen die Objekte des „temporären“ Arrays wieder in die Gruppe zurückgeschrieben werden.

```

ActGroup.AppendItems TempGrp
If AcSSet.Count <> 0 Then
    ReDim ObjList(AcSSet.Count - 1) As AcadEntity
    For i = 0 To AcSSet.Count - 1
        Set ObjList(i) = AcSSet.Item(i)
    Next
    ActGroup.RemoveItems ObjList
End If
AcSSet.Clear
Me.Show
End Sub

```

7.3 Objektverbindungen (Hyperlinks)

Wie Sie vielleicht wissen, kann jedes grafische Objekt in AutoCAD mit anderen Dateien, beispielsweise AutoCAD-Zeichnungen oder Textdateien, verknüpft werden. Diese Verknüpfung wird in den so genannten Hyperlinks gespeichert. Sie könnten so z.B. einem Zeichnungsobjekt eine andere Zeichnung, eine Videodatei oder ein Textdokument zuweisen, das gestartet wird, nachdem die Verbindung geöffnet wurde. Zusätzlich können Sie für das Zieldokument eine benannte Position angeben, beispielsweise einen AutoCAD-Ausschnitt oder ein Lesezeichen in einem Textverarbeitungsprogramm, um direkt an diese Position zu gelangen, nachdem das entsprechende Dokument geöffnet wurde.

Sie können in AutoCAD-Zeichnungen sowohl absolute als auch relative Hyperlinks erstellen. Absolute Hyperlinks enthalten den vollständigen Pfad eines Dateiverzeichnisses. Relative Hyperlinks enthalten nur einen Teil des Verzeichnisses, der relativ zum vorgegebenen Verzeichnis liegt. Da ein Zeichnungsobjekt mehrere Hyperlinks aufnehmen kann, müssen diese in einer Auflistung gesammelt werden.

7.3.1 Hyperlinks-Auflistung

Diese Ansammlung von Hyperlinks wird durch die Klasse *AcadHyperlinks* dargestellt. Über die Eigenschaften und Methoden dieser Klasse können Sie somit auf jeden einzelnen Hyperlink eines Objekts zugreifen, Verknüpfungen hinzufügen oder entfernen.

Um der Auflistung eine neue Objektverbindung hinzuzufügen, müssen Sie die *Add*-Anweisung der Auflistung verwenden. Im Gegensatz zu anderen *Add*-Funktionen können Sie hier weitere Informationen übergeben. Die Parameter sind optional und müssen darum auch nicht mit einem Wert belegt werden. Falls Sie die Parameter mit einem Wert füllen wollen, können Sie im zweiten Parameter die Beschreibung des Links und im dritten eine Position innerhalb des verknüpften Dokuments (z.B. ein Lesezeichen) festlegen:

```
Hyperlinks.Add "Link1", "neuer Link", "Marke"
```

oder auch nur:

```
Hyperlinks.Add "Link1"
```

Ein vorhandener Hyperlink lässt sich wie gehabt mit der *Item*-Anweisung abrufen. Allerdings können Sie hier nur die Position des Links innerhalb der Auflistung angeben, da einzelne Hyperlink-Objekte über keine *Name*-Eigenschaft verfügen.

```
Set ActLink = Hyperlinks.Item(n)
```

oder:

```
Set ActLink = Hyperlinks(n)
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.

Tabelle 7.7: Eigenschaften der Hyperlinksaufliistung

Eigenschaft	Beschreibung
<i>Add</i>	Diese Anweisung fügt der Auflistung einen Hyperlink hinzu.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.

Tabelle 7.8: Methoden der Hyperlinksaufliistung

Eigenschaften und Prozeduren eines einzelnen Hyperlinks

In der *URL*-Eigenschaft ist die Verknüpfung zu einer Datei gespeichert. Sie können auf lokale, im Netzwerk oder Internet/Intranet gespeicherte Dateien verweisen. Da der komplette Pfad zum verknüpften Dokument mitunter sehr lang sein kann und Sie somit keine sofortige Übersicht darüber haben, um welche Verbindung es sich handelt, ist es sinnvoll, eine Beschreibung des Hyperlinks festzulegen. Diese Beschreibung wird in der Eigenschaft *URLDescription* gespeichert. In der *URLNamedLocation*-Eigenschaft wird die Position innerhalb des verknüpften Dokuments gespeichert. Bei dieser Position könnte es sich z.B. um einen benannten Ausschnitt einer Zeichnung oder um ein Lesezeichen innerhalb eines Textdokuments handeln. Um eine Verknüpfung aus der Auflistung zu entfernen, können Sie wie gewohnt die *Delete*-Anweisung verwenden.

Hyperlinks verwalten

Da weder ein Hyperlink noch deren Auflistung ein Objekt in AutoCAD darstellen, sondern es sich vielmehr um erweiterte Daten eines grafischen Objekts handelt, enthalten diese Klassen auch keine Eigenschaften, wie *Handle* oder *ObjectID*. Um auf die Verknüpfungsdaten zugreifen zu können, müssen Sie in jedem Fall erst das Objekt in der Zeichnung auswählen, dessen Verknüpfungsdaten Sie verarbeiten wollen.

Anwendungsbeispiel

(Beispielprojekt: 07\AcadHyperlinks.dvb)

Im nun folgenden Beispiel werden wir in einem Formular alle Verknüpfungen eines grafischen Objekts, das durch den Benutzer ausgewählt werden kann, in einer Liste anzeigen lassen und dem Benutzer die Möglichkeit geben, die Verbindungen zu bearbeiten. Wenn Sie das Pro-

gramm selbst erstellen möchten, dann fügen Sie zunächst einem neuen Projekt zwei Formulare hinzu. Dem ersten (siehe Abbildung 7.2) fügen Sie fünf Schaltflächen, drei Textboxen und ein *ListView*-Steuerelement hinzu. Das Formular selbst benennen Sie in *HyperlinkFrm* um. Das zweite Formular (Abbildung 7.3) wird dem Hinzufügen von Verknüpfungen zum aktuell gewählten Objekt dienen. Da ein Hyperlink drei verschiedene Zeichenfolgeneigenschaften besitzt, müssen wir auch drei Textboxen hinzufügen, jeweils eine für eine Zeichenfolge. Außerdem werden eine *SPEICHERN*- und eine *ABBRECHEN*-Schaltfläche benötigt. Dem Formular geben Sie dann noch den Namen *NewHLFrm*.

Zuvor noch einige detaillierte Einstellungen zu den einzelnen Steuerelementen der Formulare:

- *HyperlinksFrm* – dem *ListView*-Steuerelement müssen zwei Spalten hinzugefügt werden. Die erste Spalte erhält die Überschrift *BESCHREIBUNG*, die zweite *VERKNÜPFUNG*. Außerdem muss die Eigenschaft *View* auf den Wert *4-Report* eingestellt werden sowie die Eigenschaften *MultiSelect* und *FullRowSelect* auf *TRUE*.
- *NewHLFrm* – die Eigenschaft *Default* der *SPEICHERN*-Schaltfläche und die *Cancel*-Eigenschaft von *ABBRECHEN* können Sie auf den Wert *TRUE* setzen. Das hat zur Folge, dass während der Laufzeit des Programms die -Taste die *SPEICHERN*-Schaltfläche und die -Taste die *ABBRECHEN*-Schaltfläche auslöst, als hätte der Benutzer mit der Maus darauf geklickt.

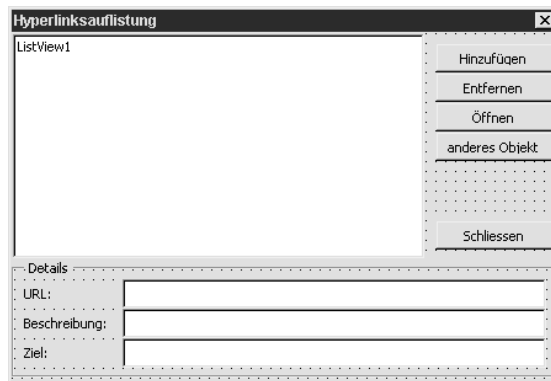


Abbildung 7.2: Hauptformular des Hyperlinkprogramms

Kommen wir nun wieder zum eigentlichen Hauptteil des Programms, dem Quellcode. Aufgeteilt ist dieser wieder in Modul- und Formularcode. Beginnen wir mit der Startprozedur, die wieder im Klassenmodul *ThisDrawing* platziert wird.

Die erste zu erledigende Aufgabe ist die Abfrage eines AutoCAD-Elements, dessen Verknüpfungsdaten angezeigt werden sollen. Das erledigt die Funktion *GetEntity* der Klasse *AcadUtility*. Da diese als Parameter eine Variable vom Typ *Object* erfordert, wird diese Variable eingangs der Prozedur deklariert.

```
Public Sub GetEntityHyperlinks()
Dim Object As Object
On Local Error Resume Next
Utility.GetEntity Object, PickedPoint, "Wählen Sie ein Objekt"
If TypeName(Object) <> "Nothing" Then
```

Nach der Ermittlung des AutoCAD-Elements und der anschließenden Prüfung, ob überhaupt ein Objekt gewählt wurde, starten wir die Funktion *InitLinkList* des Hauptformulars und übergeben gleichzeitig das Objekt an das Formular. Diese Prozedur wird nun alle weiteren Aufgaben übernehmen, so dass nur noch das Starten des Formulars vorzunehmen ist.

```
HyperlinkFrm.InitLinkList Object
HyperlinkFrm.Show
End If
End Sub
```

Damit wäre der Modulcode bereits abgeschlossen und wir können uns dem Formularcode zuwenden. Der erste Schritt ist die Deklaration einer öffentlichen Variable, die innerhalb des Formulars das ausgewählte Element aufnehmen wird. Diese Variable kann als Datentyp *Acad-Entity* deklariert werden, da es sich bei dem ausgewählten Objekt ja in jedem Fall um ein grafisches AutoCAD-Element handeln wird, zu dem dieser Objekttyp immer passt.

```
Public Object As AcadEntity
```

Bei der folgenden Funktion handelt es sich um die bereits eingangs erwähnte Initialisierungsroutine, die die vom ausgewählten Objekt verwendeten Verknüpfungen in dem *ListView*-Steuerelement unterbringen soll.

```
Public Sub InitLinkList(SelEntity As Object)
Dim HLink As AcadHyperlink
Dim LV As MSComctlLib.ListView
On Local Error Resume Next
```

Nachdem zwei Objektvariablen deklariert wurden, kann zunächst das *ListView*-Steuerelement von vorhandenen Einträgen bereinigt werden (ist nicht zwingend erforderlich).

```
ListView1.ListItems.Clear
```

Anschließend wird die *Hyperlinks*-Auflistung des mitgelieferten Objekts Eintrag für Eintrag durchlaufen. Für jeden Verknüpfungseintrag wird ein *ListView*-Element erzeugt, welches in der ersten Spalte den beschreibenden Namen und in der Zusatzspalte (*ListSubItem(1)*) den Link an sich enthalten wird.

```
For Each HLink In SelEntity.Hyperlinks
Set LV = ListView1.ListItems.Add(, , HLink.URLDescription)
LV.ListSubItems.Add , , HLink.URL
Next
```

Zum Ende der Prozedur wird das ausgewählte Objekt an die Variable *Object* übergeben, die ja bereits zu Beginn des Formularcodes deklariert wurde und von jeder Zeile innerhalb des Formulars aus erreichbar ist. Die beide Variablen *Object* und *SelEntity* sind zwar unterschiedlich deklariert (*Object As AcadEntity* und *SelEntity As Object*). Aber keine Angst, die beißen sich nicht, da es sich bei den gelieferten Objekten immer um grafische AutoCAD-Elemente handelt.

```
Set Object = SelEntity
End Sub
```

Dieser Code dürfte bereits hinlänglich bekannt sein. Er dient natürlich dem Schließen des Formulars und ist somit die Ereignisprozedur der SCHLIESSEN-Schaltfläche.

```
Private Sub CommandButton1_Click()
Me.Hide
End Sub
```

Wenn der Benutzer den Wunsch hat, eine Verknüpfung zum ausgewählten Element hinzuzufügen, kommt die nächste Ereignisprozedur zum Einsatz. Diese ruft im ersten Schritt das entsprechende Formular (*NewHLFrm*), das Sie ja bereits gestaltet haben, auf und wartet dann auf die entsprechenden Reaktionen des Benutzers.

```
Private Sub CommandButton2_Click()
Dim HLink As AcadHyperlink
Dim LV As MSComctlLib.ListItem
On Local Error Resume Next
NewHLFrm.Show
If NewHLFrm.OK = True Then
```

Hat dieser die Schaltfläche **SPEICHERN** im Formular *NewHLFrm* betätigt, so enthält die Variable *OK* logischerweise den Wert *TRUE* (das wurde ja so festgelegt) und wir können durch Hinzufügen des Links die Prozedur abschließen. Dazu fügen wir mithilfe der *Add*-Anweisung den Link, dessen URL, Beschreibung und Dokumentmarke der Hyperlinkauflistung hinzu.

```
Set HLink = Object.Hyperlinks.Add(NewHLFrm.TextBox1.Text)
HLink.URLDescription = NewHLFrm.TextBox2.Text
HLink.URLNamedLocation = NewHLFrm.TextBox3.Text
```

Was noch bleibt, ist der *ListView*-Eintrag. Dieser wird wie gehabt mittels folgender Zeilen erstellt.

```
Set LV = ListView1.ListItems.Add(, , HLink.URLDescription)
LV.ListSubItems.Add , , HLink.URL
End If
End Sub
```

Folgt als Nächstes die Reaktion auf die Aufforderung, einen Verknüpfungseintrag zu entfernen:

```
Private Sub CommandButton3_Click()
Dim HLink As AcadHyperlink
On Local Error Resume Next
```

Zuerst wieder die Sicherheitsabfrage, ob die Verknüpfung wirklich gelöscht werden soll oder vielleicht doch nicht:

```
If MsgBox("Gewählte Dateiverknüpfungen entfernen?", vbYesNo, _
"löschen") = vbYes Then
```

Da die Eigenschaft *MultiSelect* des *ListView*-Steuerelements auf *TRUE* gesetzt wurde, ist es dem Benutzer folglich möglich, mehrere Verknüpfungseinträge auf einmal auszuwählen. Diesen Umstand müssen wir nun bei der Entfernung der einzelnen Hyperlinks berücksichtigen. Da die *Count*-Eigenschaft angepasst wird, sobald ein Element aus der Liste entfernt wird, können wir diese nur mit dem letzten Eintrag beginnend zur ersten Zeile durchlaufen, um mehrere Elemente entfernen zu können.

```
With ListView1.ListItems
For i = .Count To 1 Step -1
If .Item(i).Selected = True Then
```

Wurde ein markierter Eintrag vorgefunden, so kann der entsprechende Link aus der Hyperlinkauflistung entfernt werden. Bei der Entfernung gehen wir davon aus, dass die Reihenfolge in der Auflistung der des *ListView*-Steuerelements entspricht. Allerdings beginnt die Auflistung mit dem Index 0 und das erste *ListView*-Element mit dem Index 1. Darum ziehen wir von der *ListView*-Position noch den Wert 1 ab.

```

        Set HLink = Object.Hyperlinks(i - 1)
        HLink.Delete
        .Remove I
    End If
Next
End With
End If
End Sub

```

Der Code folgender Prozedur soll das verbundene Dokument des Links öffnen, wenn der Benutzer die Schaltfläche **ÖFFNEN** gewählt hat. Zu diesem verbinden wir einfach die Zeichenfolge der *URL*-Eigenschaft mit dem *Shell*-Befehl von AutoCAD und übergeben ihn an die *SendCommand*-Anweisung des Dokuments (*Kapitel 6.2.3*). Sie könnten auch eine andere Methode zum Aufruf des Dokuments verwenden

```

Private Sub CommandButton4_Click()
On Local Error Resume Next
Me.Hide
ThisDrawing.SendCommand "shell " & _
    ListView1.SelectedItem.ListSubItems(1).Text & vbCrLf
End Sub

```

Der Code der Schaltfläche **ANDERES OBJEKT** findet sich bereits in der Startprozedur des Programms wieder. Lediglich das Aus- und wieder Einblenden des Formulars wurde hinzugefügt. Das zeitweilige Ausblenden ist notwendig, da sonst keine Auswahl im AutoCAD-Zeichnungsfenster vorgenommen werden kann.

```

Private Sub CommandButton5_Click()
Dim Object As Object
On Local Error Resume Next
Me.Hide
ThisDrawing.Utility.GetEntity Object, PickedPoint, _
    "Wählen Sie ein Objekt"
If TypeName(Object) <> "Nothing" Then
    initLinkList Object
End If
Me.Show
End Sub

```

Mit dem Code des *ItemClick*-Ereignisses werden wir den Inhalt der Textboxen der gewählten *ListView*-Zeile anpassen.

```

Private Sub ListView1_ItemClick(ByVal Item As _
MSComctlLib.ListItem)
Dim HL As AcadHyperlink
On Local Error Resume Next

```

Nachdem ein Verweis des Hyperlinks an die Variable *HL* übergeben wurde, können alle drei Eigenschaften in die jeweilige Textbox eingetragen werden. Sie könnten zwar die Eigenschaft *SelectedItem* des *ListView*-Steuerelements verwenden, was aber in dieser Prozedur nicht notwendig ist, da die Variable *Item* bereits die ausgewählte Zeile enthält.

```

Set HL = Object.Hyperlinks(Item.Index - 1)
TextBox1.Text = HL.URL

```

```

TextBox2.Text = HL.URLDescription
TextBox3.Text = HL.URLNamedLocation
End Sub

```

Mit den folgenden vier Prozeduren ändern wir die Zuordnung der verschiedenen Eintragungen des aktuell ausgewählten Hyperlinks. Die erste reagiert dabei auf die direkte Änderung der URL-Beschreibung innerhalb des *ListView*-Steuerelements. Wenn der Benutzer den linken Spalteneintrag mit der Maus anwählt, so ändert sich der Eintrag in ein Eingabefeld und es kann eine sofortige Eingabe erfolgen. Nachdem der Benutzer eine Änderung dieses Eintrags vorgenommen und diese bestätigt hat, wird dieses Ereignis ausgelöst. Wir müssen nun diese Änderung auf den Link übertragen.

```

Private Sub ListView1_AfterLabelEdit(Cancel As Integer, _
NewString As String)
Dim HL As AcadHyperlink
On Local Error Resume Next

```

Dazu holen wir den gewählten Link aus der Auflistung und übergeben der Eigenschaft den Inhalt der Variablen *NewString*, die ja dem Namen nach zu urteilen den neuen Text der URL-Beschreibung enthält.

```

Set HL = Object.Hyperlinks(ListView1.SelectedItem.Index - 1)
HL.URLDescription = NewString
ListView1_ItemClick ListView1.SelectedItem
End Sub

```

Bleiben die Ereignisprozeduren der Textboxen. Wurde der Inhalt von einer der Textboxen geändert, so wird der dazugehörige Code ausgeführt. In jeder dieser Prozeduren wird zunächst ein Verweis auf den Link, mithilfe der Listenposition des ausgewählten Elements, aus der Hyperlinkauflistung an die Variable *HL* übertragen. Dann wird nur noch der jeweilige Wert angepasst und, falls erforderlich, die Zeile des *ListView*-Steuerelements aktualisiert.

Die erste Textbox ändert die URL der gewählten Verknüpfung:

```

Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim HL As AcadHyperlink
On Local Error Resume Next
Set HL = Object.Hyperlinks(ListView1.SelectedItem.Index - 1)
HL.URL = TextBox1.Text
ListView1.SelectedItem.ListSubItems(1).Text = HL.URL
End Sub

```

Die zweite ändert die URL-Beschreibung:

```

Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim HL As AcadHyperlink
On Local Error Resume Next
Set HL = Object.Hyperlinks(ListView1.SelectedItem.Index - 1)
HL.URLDescription = TextBox2.Text
ListView1.SelectedItem.Text = HL.URLDescription
End Sub

```


Und schließlich die dritte, die URL-Marke innerhalb des unter URL festgelegten Dokuments:

```
Private Sub TextBox3_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim HL As AcadHyperlink
On Local Error Resume Next
Set HL = Object.Hyperlinks(ListView1.SelectedItem.Index - 1)
HL.URLNamedLocation = TextBox3.Text
End Sub
```

Sie könnten auch die Ereignisprozedur *AfterUpdate* statt *BeforeUpdate* zur Änderung der Eigenschaften verwenden. Allerdings hat es einen Vorteil, die *BeforeUpdate*-Prozeduren zu verwenden. Sie können bei Bedarf die Änderung abbrechen, indem Sie die Variable *Cancel* auf *TRUE* setzen.

Kommen wir nun noch zum zweiten Formular, mit dem der Benutzer eine neue Verknüpfung erstellen kann.

Abbildung 7.3: Formular zum Erstellen eines Hyperlinks

Auch dieses Formular enthält eine öffentliche, die allseits bekannte Variable *OK*, deren Inhalt darüber entscheidet, ob die Verknüpfung erzeugt wird oder nicht.

```
Public OK As Boolean
```

Der Code der Ereignisprozeduren der Schaltflächen entspricht wieder unserer üblichen Vorgehensweise. Wird die Schaltfläche *SPEICHERN* betätigt, wird die Variable *OK* auf *TRUE* gesetzt, im anderen Fall erhält *OK* den Wert *FALSE*. Danach wird jeweils das Formular ausgeblendet, um mit dem Programm fortsetzen zu können.

```
Private Sub CommandButton1_Click()
OK = True
Me.hide
End Sub
Private Sub CommandButton2_Click()
OK = False
Me.hide
End Sub
```

Abschließend noch eine kurze Anmerkung zum Thema *Hyperlinks*, die vor allem diejenigen interessieren wird, die gerne mit der DXF-Schnittstelle arbeiten bzw. wissen möchten, wo genau die Daten der einzelnen Verknüpfungen in der Zeichnung gespeichert werden. Da die Klassen *AcadHyperlinks* und *AcadHyperlink* kein AutoCAD-Objekt darstellen, werden die Daten direkt an das jeweilige AutoCAD-Objekt angehängt und zwar in den *XDaten* (erweiterte Daten). Sie könnten die Daten also genauso gut mit den Funktionen *GetXData* abrufen und *SetXData* zurückschreiben. Der *AppName*, unter dem diese Daten jeweils gespeichert sind, lautet *PE_URL* und hat bei einem einzelnen Link folgenden Aufbau.

DXF-Gruppencode	DXF-Daten	Beispieldatensatz
1001	Applikationsname	PE_URL
1000	URL	„D:\Pro- jekte\Zeichnung1.dwg“
1002	Steuerzeichenkette (zum Öffnen der zusätzlichen Informationszeilen)	{
1000	URL-Beschreibung	„Zeichnung1“
1000	URL-Marke	,Model
1002	Steuerzeichenkette (zum Schließen der zusätzlichen Informationszeilen)	}

Tabelle 7.9: Erweiterte Daten

Hängt eine weitere Verknüpfung an dem Objekt, so wird die Liste mit der nächsten URL fortgesetzt, ohne den Applikationsnamen erneut zu verwenden. Wenn Sie mehr Informationen zur Verwendung der erweiterten Daten benötigen, können Sie diese im *Kapitel 7.1.2* nachschlagen.

7.4 Verwalten von Layern

Jedes Zeichenobjekt wird auf einem bestimmten Layer abgelegt. Dies kann entweder der Vorgabelayer sein oder ein Layer, den Sie selbst erstellen und benennen. Wie Sie die einzelnen Layer einer Zeichnung verwalten, das erfahren Sie in diesem Abschnitt.

7.4.1 Layer-Auflistung

Sämtliche Layer sind in einer Layer-Auflistung zusammengefasst, über die Sie auf die einzelnen Layer zugreifen können. Zunächst folgt eine Beschreibung aller Eigenschaften und Methoden, die solch eine Layer-Auflistung zur Verwaltung der einzelnen Layer anbietet.

Layer hinzufügen/vorhandene Layer abrufen

Um nun einen Layer zur Auflistung und damit zur Zeichnung hinzuzufügen, stellt Ihnen die Klasse *AcadLayers* die *Add*-Anweisung zur Verfügung. Als Parameter müssen Sie lediglich den Namen des neuen Layers an die Prozedur übergeben. Für alle anderen Eigenschaften, wie Farbe, Linienstärke, Linientyp usw., werden zunächst die Grundeinstellungen (Farbe Weiß, eingeschaltet, getaut ...) verwendet. Sie können diese Einstellungen im weiteren Verlauf an Ihre Bedürfnisse anpassen.

```
Dim NewLayer As AcadLayer
Set NewLayer = ThisDrawing.Layers.Add("neuer Layer")
```

Der so erzeugte Layer wurde an die Objektvariable *NewLayer* übergeben. Sie könnten nun diese Variable benutzen, um die Eigenschaften des Layers anzupassen.

Um einen vorhandenen Layer aus der Layer-Auflistung auszuwählen, müssen Sie die *Item*-Funktion der Klasse *AcadLayers* verwenden.

```
Dim CrLayer As AcadLayer
Set CrLayer = ThisDrawing.Layers.Item(n)
```

oder:

```
Set CrLayer = ThisDrawing.Layers(n)
```

Beide Schreibweisen erzielen das gleiche Ergebnis, sie geben den Layer, der an Position n in der Layer-Auflistung steht, an die Objektvariable weiter. Die Methode, den Layer über den Index auszulesen, eignet sich vor allem dann, wenn Sie die Layer-Auflistung über eine *For..Next*-Schleife durchlaufen müssen.

```
For I = 0 To ThisDrawing.Layers.Count -1
    Set CrLayer = ThisDrawing.Layers(i)
Next
```

Mit der *For..Each*-Schleife können Sie ebenfalls die Layer-Auflistung durchlaufen lassen und so nacheinander alle Layer ansprechen. Dabei wiederholt diese Anweisung die Schleife so oft, solange sich Layer ermitteln lassen. Das heißt, wenn sich in der Layer-Auflistung drei Layer befinden, wird die Schleife dreimal wiederholt und bei jedem Durchlauf wird das entsprechende Element an die Objektvariable übergeben. Sie brauchen also die Übergabe des Elements nicht explizit durchzuführen.

```
For Each CrLayer In ThisDrawing.Layers
next
```

Die *Item*-Anweisung bietet allerdings noch eine weitere Möglichkeit, einen bestimmten Layer anzusprechen. Häufig ist es nicht nur sinnvoller, sondern auch schneller, einen bestimmten Layer über seinen Namen aus der Auflistung herauszuholen. Geben Sie zu diesem Zweck einfach den Namen statt des Index als Parameter der *Item*-Funktion an. Bevor Sie dies tun können, müssen Sie aber in jedem Fall eine Variable vom Datentyp *String* deklarieren, an die der Name zunächst übergeben wird. Anschließend wird dann diese Variable als Parameter in der *Item*-Funktion verwendet. Die Deklaration der Variable ist zwingend erforderlich, damit die Funktion ein Ergebnis zurückliefern kann.

```
Dim LayerName As String
Dim CrLayer As AcadLayer
LayerName = "0"
Set CrLayer = ThisDrawing.Layers(LayerName)
```

Egal, wie Sie einen Layer abrufen, als Ergebnis erhalten Sie, wenn der Abruf korrekt war, eine Objektvariable des Typs *AcadLayer*, die weiterverarbeitet werden kann.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.

Tabelle 7.10: Eigenschaften der Layer-Auflistung

Eigenschaft	Beschreibung
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.10: Eigenschaften der Layer-Auflistung (Forts.)

Eigenschaft	Beschreibung
<i>Add</i>	Diese Anweisung fügt eine Gruppe der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.11: Methoden der Layer-Auflistung

7.4.2 Einzelne Layer verarbeiten

Über die Klasse *AcadLayer* können Sie (fast) alle Eigenschaften eines Layers beeinflussen, die auch im AutoCAD zur Verfügung stehen. Wenn ich dabei „fast“ sage, so meine ich damit die Eigenschaft `IN ANSICHTSFENSTER FRIEREN`. Diese Eigenschaft steht unter VBA leider (noch) nicht zur Verfügung. Um trotzdem die Layer im Ansichtsfenster zu frieren, müssen wir einen Umweg einschlagen. Aber dazu später mehr. Schauen wir uns zunächst einmal die Eigenschaften und Methoden an, die ein Layer-Objekt zur Verfügung stellt.

Anmerkungen zu den Layer-Eigenschaften und -Methoden

Wenn Sie sich die Klasse *AcadLayer* betrachten, so dürfte im Allgemeinen klar sein, welche Eigenschaften welche Funktion eines Layers steuern. Es bedarf also keiner langen Erklärungen, sondern vielmehr nur einiger Anmerkungen zu einzelnen Funktionen.

Die *Color*-Eigenschaft legt die Farbe des Layers fest oder gibt diese zurück. Die Farbwerte liegen zwischen 0 und 256. Einige Farbwerte sind bereits in der Konstantenauf Auflistung *Acad_Color* und *AcColor* deklariert.

Konstante	Beschreibung	Wert
<i>acByBlock</i>	Farbe von Block	0
<i>acByLayer</i>	Farbe von Layer	256
<i>acRed</i>	Farbe Rot	1
<i>acYellow</i>	Farbe Gelb	2
<i>acGreen</i>	Farbe Grün	3
<i>acCyan</i>	Farbe Cyan	4
<i>acBlue</i>	Farbe Blau	5
<i>acMagenta</i>	Farbe Magenta	6
<i>acWhite</i>	Farbe Weiß/Schwarz, je nach AutoCAD-Hintergrundfarbe	7

Tabelle 7.12: Farbkonstanten

```
ActLayer.Color = AcWhite
```

oder

```
ActLayer.Color = 7
```

Für diese Eigenschaft gelten besondere Regeln, da es sich um die Farbeigenschaft der Layer handelt. Sie können für einen Layer die Konstanten *acByLayer* und *acByBlock* nicht verwenden. Wenn Sie versuchen, einem Layer die Farben *acByLayer* oder *acByBlock* zuzuordnen, wird jedes Mal ein Fehler ausgelöst und der Layer behält seine ursprüngliche Farbeinstellung. Die Farbe, die Sie für den Layer festlegen, wird für Objekte verwendet, für die die Farbeigenschaft *acByLayer* eingestellt wurde (Kapitel 8.3.1).

Ob der Layer gefroren werden soll oder nicht, bestimmt die Eigenschaft *Freeze*. Wenn Sie der Eigenschaft den Wert *TRUE* zuweisen, wird der Layer gefroren, *FALSE* taut den Layer. Sie können allerdings keinen aktuellen Layer frieren oder einen gefrorenen Layer zum aktuellen machen. Zuvor müsste der Layer erst einmal getaut werden.

Sperren bzw. entsperren können Sie die Layer mit der *Lock*-Eigenschaft. *Plottable* legt fest, ob die grafischen Objekte, die auf diesem Layer abgelegt wurden, geplottet werden können (*TRUE*) oder nicht (*FALSE*). Mit der *ViewportDefault*-Eigenschaft bestimmen Sie, ob der Layer in neuen verschiebbaren Ansichtsfenstern gefroren werden soll oder nicht. Ist die Eigenschaft auf den Wert *TRUE* gesetzt, wird der Layer in neuen Ansichtsfenstern gefroren, andernfalls getaut.

Die *LayerOn*-Eigenschaft legt fest, ob der Layer ein- oder ausgeschaltet werden soll, oder gibt die aktuelle Einstellung zurück. *TRUE* schaltet den Layer ein, *FALSE* schaltet ihn aus. Der Linientyp des Layers ist in der *Linetype*-Eigenschaft (Kapitel 7.5.1) gespeichert, die Linienstärke in *Lineweight* (Kapitel 8.3.1) und der Plotstil in *PlotStyleName*.

Layer zum aktuellen Layer machen

Wie mache ich einen bestimmten Layer zum aktuellen Layer? Folgende Codezeilen helfen bei der Beantwortung der Frage weiter:

```
Dim ActLayer As AcadLayer
On Local Error Resume Next

Set ActLayer = ThisDrawing.Layers("0")
ActLayer.Freeze = False
ThisDrawing.ActiveLayer = ActLayer
```

In diesen Codezeilen wird der Layer „0“ ausgewählt und die Eigenschaft *Freeze* auf den Wert *False* gesetzt, um ihn gegebenenfalls zu tauen. Anschließend wird der Layer an die Dokumenteigenschaft *ActiveLayer* übergeben, die den aktuellen Layer beinhaltet. Sollten Sie einen gefrorenen Layer an die Eigenschaft *ActiveLayer* weitergeben, so wird ein Fehler ausgelöst. Beachten müssen Sie lediglich, dass bei der Übergabe des Layers das Schlüsselwort *Set* keine Verwendung findet. Stammt der gewählte Layer nicht aus der aktiven Zeichnung, sondern aus irgendeiner anderen der Dokumentauflistung, so können Sie folgende Codezeile immer verwenden – egal, in welcher Zeichnung sich der entsprechende Layer befindet.

```
ActLayer.Document.ActiveLayer = ActLayer
```

In dieser Codezeile wird auf die Eigenschaft *Document* des jeweiligen Layers zurückgegriffen und somit immer die richtige Zeichnung getroffen.

Um einen Layer aus der Zeichnung zu entfernen, verwenden Sie die *Delete*-Anweisung des entsprechenden Layers. Damit dieser Vorgang erfolgreich ausgeführt werden kann, müssen Sie die gleichen Bedingungen beachten wie auch im „normalen“ AutoCAD. Das heißt, es können keine Layer gelöscht werden, die noch Objekte enthalten. Gleiches gilt für Layer, die von Blockdefinitionen referenziert werden, sowie die Layer *DEFPOINTS* und *0*, auch wenn sie keine sichtbaren Objekte enthalten. Aber das wissen Sie bereits (siehe auch Dokumenteigenschaft *PurgeAll*).

```
Dim ActLayer As AcadLayer
On Local Error Resume Next

Set ActLayer = ThisDrawing.Layers("XYZ")
ActLayer.Delete
```

Sollte der Layer nicht gelöscht werden können, so löst AutoCAD einen Fehler aus (*Objekt wird von einem anderen Objekt/von anderen Objekten referenziert*).

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Bestimmt die Farbe der Objekte, deren <i>Color</i> -Eigenschaft auf <i>vonLayer</i> eingestellt wurde.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Freeze</i>	Legt fest, ob der Layer gefroren ist oder nicht.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>LayerOn</i>	Legt fest, ob der Layer eingeschaltet ist oder nicht.
<i>Linetype</i>	Bestimmt den Linientyp der Objekte, deren <i>LineType</i> -Eigenschaft auf <i>vonLayer</i> eingestellt wurde.

Tabelle 7.13: Eigenschaften der Layer-Objekte

Eigenschaft	Beschreibung
<i>Lineweight</i>	Bestimmt die Linienstärke der Objekte, deren <i>LineType</i> -Eigenschaft auf <i>vonLayer</i> eingestellt wurde.
<i>Lock</i>	Legt fest, ob der Layer gesperrt ist oder nicht.
<i>Name</i>	Bestimmt den Namen des Layers.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Bestimmt den Plotstil der Objekte, deren <i>PlotStyleName</i> -Eigenschaft auf <i>vonLayer</i> eingestellt wurde.
<i>Plottable</i>	Legt fest, ob die grafischen Objekte des Layers geplottet werden können oder nicht.
<i>ViewportDefault</i>	Bestimmt, ob der Layer in neuen Ansichtsfenstern gefroren werden soll oder nicht.

Tabelle 7.13: Eigenschaften der Layer-Objekte (Forts.)

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.14: Methoden der Layer-Objekte

Anwendungsbeispiel

(Beispielprojekt: 07\AcadLayers.dvb)

Im folgenden Beispielprojekt werden wir die Layer einschließlich der Einstellungen in einem Formular anzeigen lassen. Die Darstellung erfolgt dabei in einem *ListView*-Steuerelement, so dass wir eine ähnliche Anzeige wie beim AutoCAD-Layermanager erhalten. Die Symbole zur Darstellung der Grundfarben und der Anzeige der Werte der Eigenschaften *LayerOn*, *Freeze*, *Lock*, *Plottable* und *ViewportDefault* sind in dem Steuerelement *ImageList1* abgelegt. Auf diese Symbolliste werden wir dann zur Laufzeit zugreifen und damit den entsprechenden Status der einzelnen Layer darstellen.

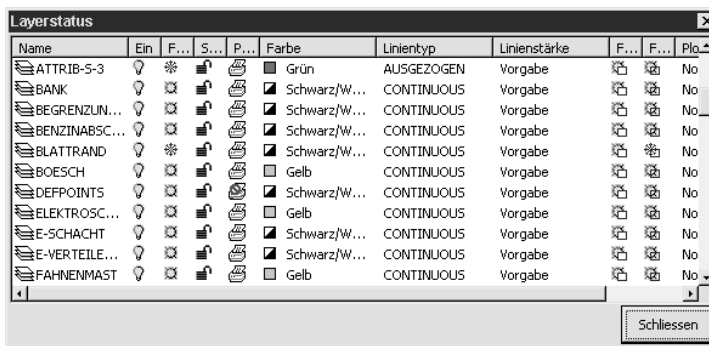


Abbildung 7.4: Layer-Formular



Abbildung 7.5: Symbole des ImageList-Steuerelements

Startprozedur

Mit der ersten Anweisung wird eine Prozedur aufgerufen, die das *Listview*-Steuerelement initialisiert und die Layer in diese Liste lädt. Da sich diese Prozedur innerhalb des Formularcodes befindet, wird der Container *LayerStatusFrm* dem Aufruf vorangestellt. Nachdem diese Prozedur abgeschlossen wurde, wird dann nur noch das Formular selbst durch die *Show*-Anweisung gestartet.

```
Public Sub ShowLayerStatus()
On Local Error Resume Next

LayerStatusFrm.initLayerList
LayerStatusFrm.Show

End Sub
```

Initialisierungsprozedur

Nachdem die notwendigen Variablen deklariert wurden, müssen wir zunächst das Abbildungslisten-Steuerelement (*ImageList1*), das die großen und kleinen Symbole für das *Listview*-Steuerelement verwaltet, eben diesem *Listview*-Steuerelement zuordnen. Um dies zu erreichen, wird das komplette *ImageList*-Steuerelement an die Eigenschaften *Icons* (große Symbole) und *SmallIcons* (kleine Symbole) übergeben.

```
Public Sub InitLayerList()
Dim LV As MSComctlLib.ListView
Dim Desc As String
Dim AcDraw As AcadDocument
Dim mCol As Collection
On Local Error Resume Next
Set ListView1.Icons = ImageList1
Set ListView1.SmallIcons = ImageList1
```


Im nächsten Schritt werden die einzelnen Spalten von *ListView1* initialisiert. Sie könnten diesen Schritt auch zur Entwurfzeit vornehmen und die Spalten in der Eigenschaftsseite des Steuerelements anlegen. Allerdings müssen Sie hierbei beachten, dass die letzten beiden Spalten erst nach Überprüfung der Einstellung der AutoCAD-Zeichnung erstellt werden: die Spalte *Frieren im aktuellen AF*, wenn sich die Zeichnung im Papierbereich befindet, da sich diese Eigenschaft auf die Papierbereichs-Ansichtsfenster bezieht, und die Spalte *Plotstil*, wenn die Systemvariable *PSTYLEMODE* den Wert 0 zurückgibt, was ja bedeutet, dass die Zeichnung benannte Plotstiltabellen verwendet.

```
With ListView1.ColumnHeaders
    .Clear
    .Add , , "Name", 125
    .Add , , "Ein", 20
    .Add , , "Frieren", 20
    .Add , , "Sperren", 20
    .Add , , "Plotten", 20
    .Add , , "Farbe"
    .Add , , "Linientyp"
    .Add , , "Linienstärke"
    .Add , , " Frieren in neuem AF ", 20
    If ThisDrawing.GetVariable("PSTYLEMODE") = 0 Then
        .Add , , "Frieren im aktuellen AF", 20
    End If
    If ThisDrawing.GetVariable("PSTYLEMODE") = 0 Then
        .Add , , "Plotstil"
    End If
End With
```

Im nächsten Schritt erstellen wir eine Auflistung aller Layer, die im aktiven Ansichtsfenster des Papierbereichs, falls eines existiert, gefroren sind. Da die VBA-Schnittstelle derzeit keine Möglichkeit bietet, die gefrorenen Layer eines Ansichtsfensters zu ermitteln, müssen wir noch auf diese Methode ausweichen. Dabei kommt uns zugute, dass AutoCAD diese Layer als erweiterte Daten unter dem Namen „Acad“ in den Abschnitt „MVIEW“ des Ansichtsfensters speichert. Abgelegt werden diese Layer-Namen unter dem Code 1003.

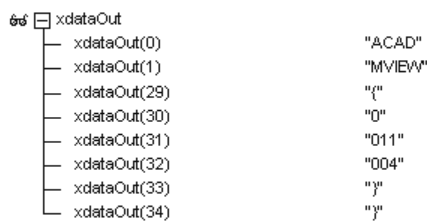


Abbildung 7.6: Erweiterte Daten eines Ansichtsfensters

```
Set tItem = ThisDrawing.ActivePViewport
tItem.GetXData "ACAD", xtypeOut, xdataOut
```

Zunächst wird das aktive Ansichtsfenster abgerufen und an die Variable *tItem* übergeben. Mithilfe der Anweisung *GetXData* (siehe Kapitel 7.1.2) werden nun die erweiterten Daten abgerufen, die durch die Anwendung AutoCAD in dem Ansichtsfenster gespeichert wurden. Die Codes werden an das Array *xtypeOut*, die Daten selbst an das Array *xdataOut* übergeben.

```

Set mCol = New Collection
If xdataOut(1) = "MVIEW" Then
    For y = LBound(xtypeOut) To UBound(xdataOut)
        If xtypeOut(y) = 1003 Then
            mCol.Add CStr(xdataOut(y)), UCase(CStr(xdataOut(y)))
        End If
    Next
End If

```

Nachdem die Variable *mCol* initialisiert wurde, können wir über eine *For...Next*-Scheife das Array *xdataOut* nach vorhandenen Layer-Namen durchsuchen. Jedes Mal, wenn im Array *xtypeOut* der Gruppencode 1003 auftaucht, enthält das Array *xDataOut* an der selben Position einen Layernamen.

xtypeOut(30) = 10003 → *xdataOut*(30) enthält einen Layer-Namen.

Jeder gefundene Layer-Name wird nun zur Auflistung *mCol* hinzugefügt. Diese Auflistung werden wir im späteren Verlauf noch verwenden.

```

ListView1.ListItems.Clear
Set AllLayers = ThisDrawing.Layers

```

Bevor die Layer-Auflistung des aktiven Dokuments an die Variable *AllLayers* übergeben wird, haben wir noch dafür gesorgt, dass das *ListView*-Steuerelement von allen eventuell noch vorhandenen Eintragungen bereinigt wird. In der nun folgenden *For...Next*-Schleife werden alle Layer nacheinander aus der Layer-Auflistung ausgelesen und dem *ListView*-Steuerelement hinzugefügt.

```

For y = 0 To AllLayers.Count - 1
    Set L = AllLayers(y)

```

Also Schleife öffnen und bei jedem Durchlauf den entsprechenden Layer an die Variable *L* übergeben:

```

    Set LV = ListView1.ListItems.Add(Text:=L.Name)

```

Die erste Aktion, die wir jetzt durchführen, ist das Hinzufügen des Layer-Namens zum *ListView*-Steuerelement. Da alle Listeneinträge in der Eigenschaft *ListItems* des Steuerelements verwaltet werden, müssen wir den neuen Eintrag auch dieser Auflistung hinzufügen. Wir werden nur den Parameter *Text* verwenden, um den Layer-Namen zu übergeben. Der Verweis auf das so erzeugte *Listview*-Element wird an die Variable *LV* übergeben, mit der anschließend weitergearbeitet wird.

Der nächste Schritt ist die Zuordnung des korrekten Symbols für den Eintrag. Falls es sich bei dem Layer um den aktuellen Layer der Zeichnung handelt, wird das zweite Symbol aus *ImageList1* zugeordnet, andernfalls das erste Symbol.

```

LV.Icon = IIf(ThisDrawing.GetVariable("CLAYER") = L.Name, 2, 1)
LV.SmallIcon = IIf(ThisDrawing.GetVariable("CLAYER") = L.Name _
    , 2, 1)

```

Bleibt jetzt nur noch die Zuordnung der einzelnen Eigenschaftswerte. Diese werden über die Zuweisung von Unterelementen, den *ListSubItems*, nacheinander hinzugefügt, so dass eine Zeile wie in Abbildung 7.7 entsteht.

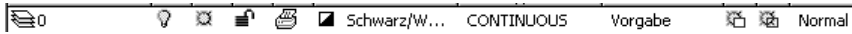


Abbildung 7.7: Abbildung eines Layers im ListView-Steurelement

Nun zum Code, der dies für uns erledigen soll. In der ersten *Add*-Anweisung wird das Symbol für den Layer-Status ein- oder ausgeschaltet festgelegt. Ist der entsprechende Layer eingeschaltet, verwenden wir Symbol 3 der *ImageList*-Symbolauflistung, ansonsten Symbol 7. Das Gleiche gilt für die Eigenschaften *Freeze*, *Lock* und *Plottable*, nur dass hier natürlich andere Symbole verwendet werden (siehe Abbildung 7.5). Für die Farbeigenschaft werden wir nicht nur die Farbe anzeigen lassen, sondern auch den Farbnamen. Aus diesem Grund wird zusätzlich zur *ReportIcon*-Eigenschaft des *ListSubItems*-Elements auch die *Text*-Eigenschaft belegt. Das Symbol und den Farbnamen ermitteln die Funktionen *GetColorImage* und *GetColorName*, die weiter unten aufgelistet und beschrieben sind. Den Linientyp und die Linienstärke lassen wir uns nur als Text anzeigen. Dabei wird der jeweilige Linienstärketext durch die Funktion *GetLineWeightText* ermittelt, da es sich bei dieser Layer-Eigenschaft ja um einen numerischen Wert handelt. Der Linientyp kann hingegen direkt übernommen werden. Bleiben noch die Eigenschaften aus dem Ansichtsfenster. Die Eigenschaft *IN NEUEM ANSICHTSFENSTER FRIEREN* kann wieder direkt ermittelt werden, da die Eigenschaft *ViewportDefault* den aktuellen Status beinhaltet.

```
With LV.ListSubItems
    .Add ReportIcon:=IIf(L.LayerOn = True, 3, 7)
    .Add ReportIcon:=IIf(L.Freeze = True, 8, 4)
    .Add ReportIcon:=IIf(L.Lock = True, 9, 5)
    .Add ReportIcon:=IIf(L.Plottable = True, 6, 10)
    .Add Text:=GetColorName(L.Color), _
        ReportIcon:=GetColorImage(L.Color)
    .Add Text:=L.Linetype
    .Add Text:=GetLineWeightText(L.LineWeight)
    .Add ReportIcon:=IIf(L.ViewportDefault = True, 22, 21)
```

Der Wert der Eigenschaft *im AF gefroren* ist ja in der Klasse *AcadLayer* nicht vorhanden, so dass hierbei nur die Funktion *IsFroozen* weiterhelfen kann. Auch hier müssen Sie zunächst wieder die Einstellung der Zeichnung beachten. Befindet sich die Zeichnung im Papierbereich, wird der Status des Layers mit der Funktion *IsFroozen* geprüft. Gibt diese den Wert *TRUE* zurück, ist der Layer im aktuellen Ansichtsfenster gefroren und es wird dementsprechend das Symbol 20 dargestellt. Ist der Layer hingegen im aktuellen Ansichtsfenster getaut, lassen wir das Symbol 19 anzeigen.

```
If ThisDrawing.ActiveSpace = acPaperSpace Then
    .Add ReportIcon:=IIf(IsFroozen(mCol, L.Name) = True, _
        20, 19)
End If
```

Zum Schluss wartet noch die Eigenschaft *Plotstil*. Diese Eigenschaft werden wir nur anzeigen lassen, wenn die Variable *PSTYLEMODE* auf 0 gestellt wurde, da ja nur in diesem Fall der Plotstil Verwendung findet.

```
If ThisDrawing.GetVariable("PSTYLEMODE") = 0 Then
    .Add Text:=L.PlotStyleName
End If
End With
Next
End Sub
```

Bleiben abschließend nur noch die Funktionen, die innerhalb der Prozedur *InitLayerList* aufgerufen werden. Beginnen wollen wir die Beschreibung mit der Prozedur *GetColorName*. Diese Funktion wird einen Farbwert, wie ihn die Eigenschaft *Color* ausgibt, in einen Text umwandeln. Dabei wird mittels einer *Select Case*-Anweisung die übergebene Farbe geprüft und der entsprechende Farbname als Ergebnis zurückgegeben.

```
Public Function GetColorName(Color As Integer) As String
On Local Error Resume Next
Select Case Color
    Case 1
        GetColorName = "Rot"
    Case 2
        GetColorName = "Gelb"
    Case 3
        GetColorName = "Grün"
    Case 4
        GetColorName = "Cyan"
    Case 5
        GetColorName = "Blau"
    Case 6
        GetColorName = "Magenta"
    Case 7
        GetColorName = "Schwarz/Weiss"
    Case Else
        GetColorName = "Farbe " & CStr(Color)
End Select
End Function
```

Die zweite Funktion soll die Symbolnummer zurückgeben, die dem angegebenen Farbwert entspricht. Diese Nummer gibt die Position des Farbsymbols im *ImageList*-Steuerelement an (siehe Abbildung 7.5).

```
Public Function GetColorImage(Color As Integer) As Integer
On Local Error Resume Next
Select Case Color
    Case 1 to 7
        GetColorImage = Color + 10
    Case Else
        GetColorImage = 18
End Select
End Function
```

In der nächsten Funktion wird die Linienstärke, die ja ebenfalls als numerischer Wert ausgegeben wird, in einen passenden Text umgewandelt. Diese Funktion erzeugt den Text je nach Einstellung der Systemvariablen *LWUNITS*. Diese Variable gibt den Status der Einheiteneinstellung der Linienstärken zurück, also ob diese in Zoll (0) oder in Millimetern (1) dargestellt werden. Dementsprechend ist dann auch der Rückgabewert der Funktion *GetLineWeightText*. Unterscheiden brauchen wir hier nur zwischen den Werten für „Vorgabe“ und den Zahlen an sich, da die Werte für *vonLayer* und *vonBlock* in der Eigenschaft *LineWeight* nicht akzeptiert werden.

```
Public Function GetLineWeightText(LineWeight As Integer) As String
Dim MUnit As Integer
On Local Error Resume Next
MUnit = ThisDrawing.GetVariable("LWUNITS")
Select Case LineWeight
```

```
Case -3
    GetLineWeightText = "Vorgabe"
Case 0
```

Da die Werte, die die Linienstärke darstellen, einem Vielfachen der Linienstärke entsprechen, ist die Umrechnung eine einfache Angelegenheit.

```
GetLineWeightText = IIf(MUnit = 1, Format(LineWeight / 100 _
    , "0.00 mm"), Format(LineWeight / 2530 , "0.000 ''"))
End Select
End Function
```

Kommen wir nun zu der Funktion, die den Layer-Status im Ansichtsfenster ermitteln soll. Wie Sie noch wissen, haben wir weiter vorne eine Auflistung mit Layern erstellt, die im aktuellen Ansichtsfenster gefroren sind. Diese Auflistung kommt nun in der folgenden Funktion zur Anwendung. Diese Funktion überprüft die Auflistung, die als erster Parameter übergeben wird, auf Vorhandensein des Layer-Namens, der im zweiten Parameter schlummert. Die Auflistung wird nun bei jeder Abfrage durchlaufen und jeder Name, der in der Auflistung vorhanden ist, wird mit dem gesuchten Layer-Namen verglichen. Ist der Layer-Name in der Liste enthalten, so ist der entsprechende Layer im Ansichtsfenster gefroren, andernfalls nicht.

```
Public Function IsFroozen(mCol As Collection, _
    LayerName As String) As Boolean
Dim LName As String
On Local Error Resume Next
For I = 0 To In mCol.Count
    LName = mCol.Item(I)
    If UCase(LName) = UCase(LayerName) Then
        IsFroozen = True
        Exit Function
    End If
Next
End Function
```

Sie können den Namen auch direkt in der Auflistung abfragen, wenn Sie bei der Erstellung der Auflistung den Parameter *Key* verwendet haben.

```
mCol.Add CStr(xdataOut(y)), UCase(CStr(xdataOut(y)))

Public Function IsFroozen(mCol As Collection, _
    LayerName As String) As Boolean
On Local Error Resume Next
If TypeName(mCol.Item(LayerName)) <> "String" Then
    IsFroozen = False
Else
    IsFroozen = True
End If
End Function
```

Bleibt zum Schluss nur noch das Beenden des Programms. Dies erledigen wir über das *Click*-Ereignis der Schaltfläche SCHLIESSEN. Dort wird nur das Formular mithilfe der *Hide*-Anweisung ausgeblendet.

```
Private Sub CommandButton1_Click()
Me.Hide
End Sub
```

7.4.3 Der Layerstatusmanager

Ab der Version 2000i enthält die VBA-Typenbibliothek die Klasse *AcadLayerStateManager*. Wie der Name dieser Klasse bereits verrät, können Sie mithilfe der Funktionen dieser Klasse den Status der Layer verwalten. Da diese Klasse nicht direkt mit einer Layer-Auflistung oder einem Dokument verbunden ist, müssen Sie die Verbindung vor Benutzung der Funktionen dieses Objekts erst manuell herstellen. Zu diesem Zweck stellt *AcadLayerStateManager* die Anweisung *SetDatabase* zur Verfügung. Diese Anweisung ordnet dem Objekt die Datenbank des entsprechenden Dokuments zu. Auf diese Datenbank bezieht sich dann der Layerstatusmanager.

```
Dim StateObj As AcadLayerStateManager
On Local Error Resume Next
Set StateObj = New AcadLayerStateManager
StateObj.SetDatabase ThisDrawing.Database
```

Statt des aktuellen Dokuments können Sie natürlich auch jedes andere Dokument zuordnen. Die *Mask*-Eigenschaft legt fest, welche Layer-Eigenschaften wiederhergestellt werden sollen. Alle Eigenschaften sind als Bitcode vergeben und können somit miteinander kombiniert werden. Sie finden alle Bitcode-Zuordnungen in der nachfolgenden Tabelle.

Konstante	Beschreibung	Wert
<i>acLsAll</i>	Alle Eigenschaften wiederherstellen	65535
<i>acLsNone</i>	Keine Eigenschaft wiederherstellen	0
<i>acLsOn</i>	Layer ein- oder ausschalten	1
<i>acLsFrozen</i>	Layer frieren oder tauen	2
<i>acLsLocked</i>	Layer sperren oder entsperren	4
<i>acLsPlot</i>	Plotten ein oder aus	8
<i>acLsNewViewport</i>	Layer in neuem Ansichtsfenster frieren oder tauen	16
<i>acLsColor</i>	Farbe	32
<i>acLsLineType</i>	Linientyp	64
<i>acLsLineWeight</i>	Linienstärke	128
<i>acLsPlotStyle</i>	Plotstil	256

Tabelle 7.15: Konstanten zur Steuerung der Mask-Eigenschaft

Um mehrere Einstellungen zu kombinieren, müssen Sie lediglich die einzelnen Werte aufaddieren.

```
StateObj.Mask("Status") = acLsOn + acLsColor
```

Um den aktuellen Status abzurufen, rufen Sie den aktuellen Wert ab und werten diesen dann beispielsweise mit einem *And*-Operator aus.

```
Dim AcSetting As AcLayerStateMask
AcSetting = StateObj.Mask("Status")
```

Nachdem Sie die Datenbank zugeordnet und die Maske festgelegt haben, können Sie über verschiedene Anweisungen bestimmte Aktionen mit dem Layer-Status-Objekt ausführen.

Mithilfe der *Save*-Anweisung können Sie die aktuellen Layer-Einstellungen sowie die Attributmaske unter dem angegebenen Statusnamen in der Datenbank speichern.

```
Dim eMask As AcLayerStateMask  
eMask = acLsAll  
StateObj.Save "Status", eMask
```

Das Gegenstück zur *Save*-Anweisung ist *Restore*. Diese stellt den als Parameter festgelegten Status der Layer wieder her. Bei der Benutzung dieser Prozedur gilt es zu beachten, dass die aktuellen Einstellungen der Eigenschaft *Mask* verwendet werden.

```
StateObj.Restore "Status"
```

Mit der *Rename*-Anweisung können Sie den angegebenen Layer-Status umbenennen. Als Parameter müssen Sie den alten und den neuen Statusnamen angeben. Wenn Sie versuchen, den Status mit einem Namen zu benennen, der bereits vergeben ist, löst AutoCAD einen Fehler aus.

```
StateObj.Rename "Status1", "Status2"
```

Um einen Layer-Status aus der Zeichnung zu entfernen, müssen Sie nur die *Delete*-Anweisung unter Angabe des Namens des Status ausführen.

```
StateObj.Delete "MeinStatus"
```

Um einen bestimmten Layer-Status in eine Datei zu exportieren, können Sie die *Export*-Anweisung verwenden. Im ersten Parameter geben Sie den Namen des Layer-Status an, der exportiert werden soll. Im zweiten Parameter den Namen der Datei, in die der Status geschrieben werden soll.

```
StateObj.Export "Status", "C:\Status.txt"
```

Was exportiert werden kann, kann auch wieder eingelesen werden. Dazu dient die *Import*-Anweisung. Da der Name des Status in der Datei festgelegt ist, brauchen Sie im Parameter dieser Anweisung nur den Namen der Datei anzugeben, die importiert werden soll.

```
StateObj.Import "C:\Status.txt"
```

Da die vorhandenen Statusnamen der aktuellen Zeichnung nicht direkt abrufbar sind, haben Sie sich bestimmt schon gefragt, wo AutoCAD die wieder versteckt hat. Die naheliegendste Position ist natürlich die Layer-Auflistung und so ist es auch. AutoCAD bietet für jedes Objekt ein Erweiterungswörterbuch oder auch *Dictionary* (Kapitel 7.12) an, in dem alle zusätzlichen Daten – seien es die von AutoCAD oder Ihre eigenen – abgespeichert werden können. Der Layer-Status wird z.B. unter dem Namen „ACAD_LAYERSTATES“ in diesem *Dictionary* abgelegt. Der Zugriff darauf ist recht einfach. Zunächst wird das Erweiterungswörterbuch der Layer-Auflistung abgerufen und in einer Objektvariablen gespeichert:

```
Dim LayDict As AcadDictionary  
Set LayDict = Layers.GetExtensionDictionary
```

Da es sich bei diesem Objekt um eine Auflistung handelt, in der verschiedene erweiterte Daten und/oder Objektverweise gespeichert sind, können wir die Statusauflistung direkt mit dem Namen „ACAD_LAYERSTATES“ abrufen. Diese kann nun verschiedene erweiterte Daten enthalten, weshalb wir das Wörterbuch nun mit einer *For...Next*-Schleife durchlaufen und so jeden einzelnen Statusnamen ermitteln können.

```
For i = 0 To LayDict("ACAD_LAYERSTATES").Count - 1
    Utility.Prompt LayDict("ACAD_LAYERSTATES").Item(i).Name
Next
```

Um Ihnen das Ganze einmal optisch näher zu bringen, zeigt Abbildung 7.8 ein entsprechendes Beispiel.

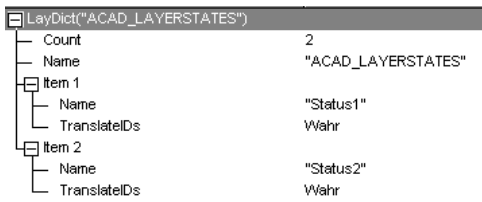


Abbildung 7.8: Statuseintragung im Layers-Dictionary

Eigenschaft	Beschreibung
Mask	Legt die Maske des benannten Layer-Status fest bzw. ruft diese ab.

Tabelle 7.16: Eigenschaften des Layer-Status-Objekts

Prozedur	Beschreibung
Delete	Löscht das Objekt aus der AutoCAD-Zeichnung.
Export	Exportiert den Status in eine ASCII-Datei.
Import	Importiert einen Status aus einer ASCII-Datei.
Rename	Benennt den angegebenen Layer-Status um.
Restore	Stellt den als Parameter festgelegten Status der Layer wieder her.
Save	Speichert die aktuellen Layer-Einstellungen sowie die Attributmaske unter dem angegebenen Statusnamen.
SetDataBase	Ordnet der Objektvariablen eine AutoCAD-Datenbank zu.

Tabelle 7.17: Prozeduren des Layer-Status-Objekts

Anwendungsbeispiel

(Beispielprojekt 07\AcadLayerStateManager.dvb)

Schauen wir uns nun anhand eines Beispielprogramms die Funktionsweise der Klasse *AcadLayerStateManager* an. Es unterteilt sich in zwei verschiedene Programmteile, die jeweils durch eine Startprozedur aufgerufen werden können.

Das erste Programm, mit dem wir uns beschäftigen, ist eine Funktion, die den aktuellen Layer-Status unter einem bestimmten Namen sichern soll. Diese Funktion orientiert sich in Darstellung und Funktionsweise an der Sicherungsfunktion des AutoCAD-Layermanagers.

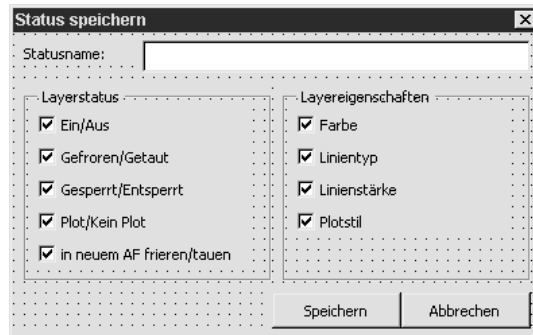


Abbildung 7.9: SaveLayerStateFrm-Formular

Startprozedur

```
Public Sub SaveLayerStatus()
Dim StateObj As AcadLayerStateManager
Dim eMask As AcLayerStateMask
On Local Error Resume Next
SaveLayerStateFrm.Show
```

Nachdem nun alle notwendigen Variablen deklariert und das Formular gestartet wurde, kann der Benutzer seine Einstellungen in diesem Formular vornehmen. Je nachdem, welche Schaltfläche zum Verlassen des Formulars der Benutzer dann auswählt, wird die Variable *OK* mit einem bestimmten Wert belegt. Kontrollieren wir nun diesen Wert, um herauszufinden, ob der Benutzer *SPEICHERN* oder *ABBRECHEN* betätigt hat.

```
If SaveLayerStateFrm.OK = True Then
```

Gibt uns die Variable *OK* den Wert *TRUE* zurück, so hat der Benutzer die Schaltfläche *SPEICHERN* betätigt und wir können mit der Sicherung des Layer-Status fortfahren.

```
Set StateObj = New AcadLayerStateManager
StateObj.SetDatabase ThisDrawing.Database
eMask = acLsNone
```

Nachdem das Objekt *StateObj* initialisiert und die Datenbank des aktuellen Dokuments zugewiesen wurde, kann mit der Zusammenstellung der Statusmaske fortgefahren werden. Dazu kontrollieren wir jede einzelne Kontrollbox auf ihren derzeit eingestellten Wert.

```
With SaveLayerStateFrm
If .CheckBox1.Value = True Then eMask = acLsOn
If .CheckBox2.Value = True Then eMask = eMask + acLsFrozen
If .CheckBox3.Value = True Then eMask = eMask + acLsLocked
If .CheckBox4.Value = True Then eMask = eMask + acLsPlot
If .CheckBox5.Value = True Then eMask = eMask + _
    acLsNewViewport
```

```

If .CheckBox6.Value = True Then eMask = eMask + acLsColor
If .CheckBox7.Value = True Then eMask = eMask + acLsLineType
If .CheckBox8.Value = True Then eMask = eMask + _
    acLsLineWeight
If .CheckBox9.Value = True Then eMask = eMask + _
    acLsPlotStyle
Err.Clear

```

Jetzt muss nur noch die *Save*-Anweisung unter Angabe des Namens (wurde in die Textbox des Formulars eingegeben) und der zuvor zusammengestellten Maske aufgerufen werden.

```

StateObj.Save .TextBox1.Text, eMask

If Err <> 0 Then
    If MsgBox("Schlüssel " & .TextBox1.Text & _
        " bereits vorhanden! Überschreiben?", vbYesNo, _
        "Speichern") = vbYes Then
        StateObj.Delete .TextBox1.Text
        StateObj.Save .TextBox1.Text, eMask
    End If
End If
End With
End If
End Sub

```

Formularcode

Als Nächstes wenden wir uns dem Code des Formulars zu. Zunächst wird die öffentliche Variable *OK* deklariert, auf die durch den Modulcode zugegriffen wird, nachdem das Formular ausgeblendet wurde.

```
Public OK As Boolean
```

Die beiden Schaltflächen erhalten jeweils zwei Codezeilen in ihren *Click*-Ereignisprozeduren. Für die *SPEICHERN*-Schaltfläche wird die Variable *OK* auf *TRUE* gesetzt und dann das Formular ausgeblendet.

```

Private Sub CommandButton1_Click()
    OK = True
    Me.Hide
End Sub

```

Bei Betätigung der Schaltfläche *ABBRECHEN* erhält die Variable *OK* den Wert *FALSE*, bevor das Formular ausgeblendet wird.

```

Private Sub CommandButton2_Click()
    OK = False
    Me.Hide
End Sub

```

Damit wäre das erste Programm zum Layer-Status bereits abgeschlossen und wir können uns dem zweiten und somit letzten Beispiel zu diesem Objekt zuwenden. Wir werden in diesem Beispiel die Funktion *STATUS WIEDERHERSTELLEN* des Layermanagers nachstellen.



Abbildung 7.10: LoadLayerStateFrm-Formular

Startprozedur

Nachdem die Initialisierung der Variable durchgeführt wurde (*StateObj* wurde dieses Mal im Formularcode deklariert, da sämtliche Operationen dort durchgeführt werden), wird die Formularprozedur *initList* aufgerufen. Abschließend brauchen wir nur noch das Formular zu starten, den Rest übernimmt dann der Formularcode.

```
Public Sub LoadLayerStatus()
On Local Error Resume Next
Set LoadLayerStateFrm.StateObj = New AcadLayerStateManager
LoadLayerStateFrm.StateObj.SetDatabase ThisDrawing.Database
LoadLayerStateFrm.initList
LoadLayerStateFrm.Show
End Sub
```

Initialisierungsprozedur

In der folgenden Prozedur, die ja bereits durch die Startprozedur aufgerufen wurde, wird das *Dictionary* der Layer-Auflistung abgerufen, damit in den nächsten Schritten alle vorhandenen Statusnamen ermittelt werden können.

```
Public Sub initList()
Dim LayDict As AcadDictionary
On Local Error Resume Next
Set LayDict = ThisDrawing.Layers.GetExtensionDictionary
```

Anschließend wird die Listbox des Formulars geleert, falls dort noch Einträge von vorherigen Aufrufen vorhanden sein sollten. Die nun folgenden Zeilen rufen die Einträge des *Dictionary* nacheinander ab und fügen die jeweiligen darin enthaltenen Statusnamen zur Listbox hinzu.

```
For i = 0 To LayDict("ACAD_LAYERSTATES").Count - 1
LoadLayerStateFrm.ListBox1.AddItem _
LayDict("ACAD_LAYERSTATES").Item(i).Name
Next
CommandButton1.Enabled = False
CommandButton2.Enabled = False
CommandButton3.Enabled = False
```

```

CommandButton4.Enabled = False
CommandButton6.Enabled = False
End Sub

```

Als Erstes wird im Formularcode die Variable *StateObj* deklariert, welche durch die Prozedur im Modulcode initialisiert wurde.

```
Public StateObj As AcadLayerStateManager
```

Schaltfläche „Wiederherstellen“

Der Code, der ausgeführt wird, sobald der Benutzer die Schaltfläche WIEDERHERSTELLEN gewählt hat, ist recht einfach gehalten. Es wird lediglich die *Restore*-Anweisung des Layer-Status-Objekts aufgerufen und der Name des gewählten Status als Parameter mitgegeben.

```

Private Sub CommandButton1_Click()
StateObj.Restore ListBox1.Text
End Sub

```

Schaltfläche „Bearbeiten“

Sollte der Benutzer diese Schaltfläche gewählt haben, so wird ein Formular eingeblendet, das die Änderung der Statusmaske ermöglicht. Wir nutzen dabei das bereits vorhandene Formular der Statussicherung. Zunächst muss allerdings die aktuelle Maske des angegebenen Status ermittelt werden.

```

Private Sub CommandButton2_Click()
Dim eMask As AcLayerStateMask
On Local Error Resume Next
eMask = StateObj.Mask(ListBox1.Text)

```

Nachdem mithilfe der *Mask*-Eigenschaft die Maske an die Variable *eMask* übergeben wurde, können wir diese nun auswerten und den entsprechenden Wert (*TRUE/FALSE* – Ein/Aus) an das zugehörige Kontrollkästchen übergeben.

```

With SaveLayerStateFrm
.TextBox1.Text = ListBox1.Text
.TextBox1.Locked = True

```

Nehmen wir also den ersten Wert (*acLsOn*), den es zu vergleichen gilt, und kontrollieren, ob dieser in der Maske gesetzt wurde oder nicht. Ist das Bit 1 in der Maske gesetzt, so gibt *eMask And acLsOn* den Wert von *acLsOn* zurück, andernfalls ergibt dieser Vergleich 0. Gleiches gilt für den zweiten Wert (*eMask And acLsFrozen*). Sie müssen sich im Prinzip vor Augen halten, dass es sich bei den Variablen und Konstanten eigentlich um Zahlen handelt. Zum besseren Verständnis ist in Tabelle 7.18 ein Zahlenbeispiel zusammengestellt.

```

.CheckBox1.Value = IIf((eMask And acLsOn) = 1, True, False)
.CheckBox2.Value = IIf((eMask And acLsFrozen) = 2, True, _
False)
.CheckBox3.Value = IIf((eMask And acLsLocked) = 4, True, False)
.CheckBox4.Value = IIf((eMask And acLsPlot) = 8, True, False)
.CheckBox5.Value = IIf((eMask And acLsNewViewport) = 16, _
True, False)
.CheckBox6.Value = IIf((eMask And acLsColor) = 32, True, False)
.CheckBox7.Value = IIf((eMask And acLsLineType) = 64, True, _
False)

```

```
.CheckBox8.Value = IIf((eMask And acLsLineWeight) = 128, _
    True, False)
.CheckBox9.Value = IIf((eMask And acLsPlotStyle) = 256, _
    True, False)
```

Nachdem alle Einstellungen an das Formular übergeben wurden, kann es gestartet werden und auf die entsprechenden Benutzereingaben warten. Sobald der Benutzer dann die Schaltfläche **SPEICHERN** betätigt hat, wird derselbe Code zur Zusammenstellung des Werts der Variablen *eMask* ausgeführt, wie er bereits im vorherigen Programmbeispiel (Status sichern) erläutert wurde.

```
.Show
If .OK = True Then
    eMask = acLsNone
    If .CheckBox1.Value = True Then eMask = acLsOn
    If .CheckBox2.Value = True Then eMask = eMask + acLsFrozen
    If .CheckBox3.Value = True Then eMask = eMask + acLsLocked
    If .CheckBox4.Value = True Then eMask = eMask + acLsPlot
    If .CheckBox5.Value = True Then eMask = eMask + _
        acLsNewViewport
    If .CheckBox6.Value = True Then eMask = eMask + acLsColor
    If .CheckBox7.Value = True Then eMask = eMask + acLsLineType
    If .CheckBox8.Value = True Then eMask = eMask + _
        acLsLineWeight
    If .CheckBox9.Value = True Then eMask = eMask + _
        acLsPlotStyle
```

Der einzige Unterschied besteht in folgender Zeile:

```
StateObj.Mask(.TextBox1.Text) = eMask
```

Im Gegensatz zur Sicherung der Maske durch Erstellung eines neuen Status, wird hier nur die Maske des entsprechenden Status geändert.

```
End If
.TextBox1.Locked = False
End With
End Sub
```

Vorhandene Maske			Vergleichswert	Ergibt	
ein/aus	verwenden	1	<i>acLsOn</i>	= 1	1
frieren/tauen	verwenden	2	<i>acLsFrozen</i>	= 2	2
sperr/en/entsperren	verwenden	4	<i>acLsLocked</i>	= 4	4
Plotten ein/aus	nicht verwenden	0	<i>acLsPlot</i>	= 8	0
In neuem Ansichtsfenster frieren	verwenden	16	<i>acLsNewViewport</i>	= 16	16
Farbe	nicht verwenden	0	<i>acLsColor</i>	= 32	0
Linientyp	verwenden	64	<i>acLsLineType</i>	= 64	64
Linienstärke	nicht verwenden	0	<i>acLsLineWeight</i>	= 128	0
Plotstil	verwenden	256	<i>acLsPlotStyle</i>	= 256	256

Tabelle 7.18: Beispielmaskenvergleich

Schaltfläche „Umbenennen“

Der erste Schritt in dieser Prozedur ist die Auswahl eines neuen Statusnamens. Dazu bedienen wir uns der *InputBox*-Funktion von VB. Den eventuell neuen Namen lassen wir an die Variable *NewName* übergeben.

```
Private Sub CommandButton3_Click()
Dim NewName As String
On Local Error Resume Next
NewName = InputBox("Legen Sie den neuen Namen des Status fest.", _
    "Umbenennen", ListBox1.Text)
```

Hat der Benutzer die *InputBox*-Funktion abgebrochen oder den Vorgabenamen verwendet, so müssen wir darauf reagieren und die Prozedur vorzeitig verlassen.

```
If NewName = "" Or NewName = ListBox1.Text Then Exit Sub
```

Andernfalls kann an dieser Stelle fortgesetzt werden. Was nun folgt, ist die Kontrolle, ob der Statusname bereits vorhanden ist. Ist dies der Fall, so wird ein Hinweis ausgegeben und die Prozedur vorzeitig verlassen.

```
For i = 0 To ListBox1.Count - 1
    If UCase(NewName) = UCase(ListBox1.List(i)) Then
        MsgBox "Statusname ist bereits vorhanden! Der Name muss " & _
            & "eindeutig sein!", vbCritical
        Exit Sub
    End If
Next
```

Bleibt abschließend nur das Umbenennen des Status und die Änderung des Listbox-Eintrags auf den neuen Namen.

```
Err.Clear
StateObj.Rename ListBox1.Text, NewName
If Err = 0 Then ListBox1.List(ListBox1.ListIndex) = NewName
End Sub
```

Schaltfläche „Löschen“

Bevor der Löschvorgang durchgeführt wird, lassen wir uns die Richtigkeit durch den Benutzer bestätigen. Dies geschieht mithilfe eines Meldungsdialogs, der dem Benutzer eine Auswahl mittels der Schaltflächen JA und NEIN ermöglicht.

```
Private Sub CommandButton4_Click()
On Local Error Resume Next
If MsgBox("Status " & ListBox1.Text & " soll gelöscht werden!" & _
    vbCrLf & "Sind Sie sicher?", vbYesNo, "Löschen") = vbYes Then
    Err.Clear
```

Hat dieser nun die Schaltfläche JA betätigt, so wird der aktuell gewählte Statusname aus dem *LayerStateManager* gelöscht

```
StateObj.Delete ListBox1.Text
If Err = 0 Then
```

und anschließend, falls kein Fehler aufgetreten ist, aus dem Listefeld entfernt. Sie könnten auch die komplette Liste der Statusnamen durch Aufruf der Prozedur *initList* erneut einlesen, um sicherzustellen, dass der Statusname wirklich gelöscht wurde.

```
ListBox1.RemoveItem ListBox1.ListIndex
End If
End Sub
```

Schaltfläche „Importieren“

Bleiben noch die Im- und Exportschaltflächen. In diesen Prozeduren kommt das Steuerelement *CommonDialog* zur Anwendung. Bei der Importfunktion wird dabei die Funktion des Steuerelements verwendet, das ein Dialogfenster zur Auswahl eines Dateinamens anbietet. Nachdem einige Voreinstellungen festgelegt wurden, wie die Festlegung eines Dialogtitels sowie eines Auswahlfilters, wird das Dialogfenster über *ShowOpen* gestartet:

```
Private Sub CommandButton5_Click()
On Local Error Resume Next
ComDialog.CancelError = True
ComDialog.DialogTitle = "Layerstatus importieren"
ComDialog.Filter = "Layerstatus (*.las)|*.las|"
Err.Clear
ComDialog.ShowOpen
```

Durch Festlegung der Eigenschaft *CancelError* auf den Wert *TRUE*, wird jedes Mal ein Fehler ausgelöst, sobald der Benutzer den *Open*-Vorgang abbricht. Aus diesem Grund kann die Importfunktion nur fortgesetzt werden, wenn das *Err*-Objekt keinen Fehler zurückgibt. Hat der Benutzer einen gültigen Dateinamen ausgewählt, ist dies der Fall. Da dieser Dateiname einschließlich des Verzeichnisnamens in der Eigenschaft *FileName* abgelegt wurde, können wir diese direkt mit der *Import*-Anweisung verwenden. Nachdem das Einlesen des Status aus der Datei erfolgt ist, muss nur noch die Listbox aktualisiert werden (Anweisung *initList*).

```
If Err = 0 Then
    StateObj.Import ComDialog.FileName
    initList
End If
End Sub
```

Schaltfläche „Exportieren“

Ähnlich wie die Importfunktion funktioniert auch die Exportfunktion. Zunächst werden die Eigenschaften, wie Dialogtitel oder Filter, festgelegt. Der nächste Schritt, das Öffnen des Dialogs, erfolgt aber diesmal nicht durch *ShowOpen*, sondern durch *ShowSave*, da die Datei ja gespeichert werden soll.

```
Private Sub CommandButton6_Click()
On Local Error Resume Next
ComDialog.CancelError = True
ComDialog.DialogTitle = "Layerstatus exportieren"
ComDialog.Filter = "Layerstatus (*.las)|*.las|"
ComDialog.Flags = &H2
ComDialog.ShowSave
```

Nachdem wieder kontrolliert wurde, ob der Benutzer den Vorgang abgebrochen hat, können wir die Exportfunktion ausführen. Als Parameter verwenden wir den Statusnamen, der in der Listbox ausgewählt wurde, und den Dateinamen, den der Benutzer im SPEICHERN-Dialog festgelegt hat.

```
If Err = 0 Then
    StateObj.Export ListBox1.Text, ComDialog.FileName
End If
End Sub
```

Schaltfläche „Schließen“

Durch Benutzung dieser Schaltfläche wird lediglich das Formular ausgeblendet, es ist also keine große Sache mehr.

```
Private Sub CommandButton7_Click()
Me.Hide
End Sub
```

Die nun folgende Ereignisprozedur dient nur der Überwachung der korrekten Einstellung der *Enabled*-Eigenschaft einiger Schaltflächen. Sie wird immer dann ausgelöst, wenn der Benutzer auf einen Eintrag des Listenfelds klickt. Ist dies der Fall, so ist auf jeden Fall ein Element ausgewählt und die Schaltflächen können aktiviert werden.

```
Private Sub ListBox1_Click()
On Local Error Resume Next
CommandButton1.Enabled = True
CommandButton2.Enabled = True
CommandButton3.Enabled = True
CommandButton4.Enabled = True
CommandButton6.Enabled = True
End Sub
```

7.5 Linientypen verwalten

7.5.1 Linientypauflistung verwenden

Wie alle anderen Elemente, z.B. Layer oder Blöcke, werden auch die Linientypen in einer Auflistung verwaltet. Jedes Dokument enthält eine eigene Auflistung von Linientypen, die Sie über das entsprechende Dokument abrufen können.

```
Set ActLTypes = ThisDrawing.LineTypes
```

oder

```
Set ActLTypes = Documents(n).LineTypes
```

Die Linientypauflistung enthält aber nicht nur frei definierbare Linientypen, sondern auch die Definitionen *vonLayer*, *vonBlock* und die Standardlinienart *CONTINUOUS*. Diese drei Linientypen können Sie nicht aus der Auflistung entfernen.

Linientypen hinzufügen/vorhandene abrufen

Das Hinzufügen und Abrufen von Linientypen erfolgt nach der bewährten Methode mittels der Prozeduren *Add*, *Count* und *Item*, wobei die *Item*-Eigenschaft normalerweise keine Anwendung findet, da sie die Standardeigenschaft des Objekts bildet. Soll heißen, Sie brauchen sie nicht explizit anzugeben, sondern können den Index direkt an das Objekt anhängen – *LineTypes(i)* statt *LineTypes.Item(i)*.

Folgende Codezeilen fügen einen neuen Linientyp der Zeichnung und somit auch der Linientypauflistung hinzu. Sie müssen bei der Verwendung der *Add*-Anweisung lediglich darauf achten, dass als Parameter eine Variable des Datentyps *String* erforderlich ist.

```
Dim ActLType As AcadLineType
Dim LTName As String
LTName = InputBox("Geben Sie den Namen des neuen Linientyps ein" _
    , "Neuer Linientyp")
If LTName "" Then Exit Sub
Set ActLType = ThisDrawing.LineTypes.Add(LTName)
```

Um einen vorhandenen Linientyp aus der Auflistung herauszulesen, können Sie folgendermaßen vorgehen:

```
Dim ActLType As AcadLineType
Dim LTName As String
Dim Prompt As String
Prompt = "Geben Sie den Namen des gesuchten neuen Linientyps ein"
LTName = InputBox(Prompt, "Linientyp ermitteln")
If LTName "" Then Exit Sub
Set ActLType = ThisDrawing.LineTypes(LTName)
```

Auch hier ist es wichtig, dass der Parameter als Variable des Datentyps *String* deklariert wird, falls Sie diesen über den Namen auslesen möchten. Andernfalls können Sie auch einen Index als Parameter der *Item*-Funktion verwenden.

```
Dim ActLType As AcadLineType
For i = 0 To ThisDrawing.LineTypes.Count - 1
    Set ActLType = ThisDrawing.LineTypes(i)
Next
```

Neben diesen Standardeigenschaften und -prozeduren besitzt das Auflistungsobjekt noch die *Load*-Anweisung. Diese Anweisung lädt die Definitionen der angegebene Linientypdatei (*.lin) in AutoCAD und fügt die neuen Linientypen der Auflistung hinzu. Sie müssen im ersten Parameter den Namen des Linientyps angeben, der geladen werden soll, was bedeutet, dass Sie den Namen kennen oder die Linientypdatei nach den vorhandenen Namen durchsuchen müssen.

```
FileName = "C:\Linientypen.lin"
ActLTypes.Load "Linientyp1", FileName
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.

Tabelle 7.19: Eigenschaften der Linientypauflistung

Eigenschaft	Beschreibung
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück..

Tabelle 7.19: Eigenschaften der Linientypauflistung (Forts.)

Prozedur	Beschreibung
<i>Add</i>	Diese Anweisung fügt einen Linientyp der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>Load</i>	Lädt die Definitionen der angegebene Linientypdatei (*.lin) in AutoCAD und fügt die neuen Linientypen der Auflistung hinzu.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.20: Prozeduren der Linientypauflistung

7.5.2 Einzelne Linientypen verwenden

Kommen wir nun zu den Möglichkeiten der Bearbeitung eines einzelnen Linientyps. Einzelne Linientypdefinitionen werden in AutoCAD durch die Klasse *AcadLinetype* dargestellt. Diese besitzt wieder eine Reihe von Eigenschaften und Prozeduren, durch deren Verwendung die einzelnen Linientypen angepasst werden können. Bevor Sie diese anwenden können, müssen Sie allerdings erst auf einen bestimmten Linientyp aus der Auflistung zugreifen.

Eine Objektvariable dieser Klasse wird durch folgende Funktionen zurückgegeben:

- *Add*-Anweisung der Linientypauflistung
- *Item*-Funktion der Linientypauflistung
- *ActiveLinetype*-Eigenschaft der Klasse *AcadDocument*

Die *LineType*-Eigenschaft eines Objekts (siehe Kapitel 7.3.1) enthält hingegen nur den Namen des entsprechenden Linientyps, so dass Sie das eigentliche Linientypobjekt erst über die *Item*-Funktion der Linientypauflistung erhalten.

Neben den üblichen Eigenschaften enthält diese Klasse noch die *Description*-Eigenschaft. Diese legt einen beliebigen Beschreibungstext für den Linientyp fest. Der Beschreibungstext eines Linientyps kann bereits in der Linientypdatei vorgeben werden. Dieser steht dann neben dem Namen des Linientyps und könnte schon bei der Auswahl von Linientypen aus einer Datei durch den entsprechenden Programmcode ermittelt werden.

```
*RAND,Rand _ _ _ . _ _ _ . _ _ _ . _ _ _ . _ _ _ .
A, 12.7, -6.35, 12.7, -6.35, 0, -6.35
ActLLType.Description = "mein eigener Linientyp"
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Description</i>	Bestimmt einen beliebigen Beschreibungstext für den Linientyp.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Name</i>	Legt den Namen des Linientyps fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück..

Tabelle 7.21: Eigenschaften des Linientypobjekts

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.22: Prozeduren des Linientypobjekts

Anwendungsbeispiel

(Beispielprojekt: 07\AcadLinetypes.dvb)

Schauen wir uns doch einfach mal ein Beispiel zu diesem Thema an. Das Ziel des nun folgenden Programms wird es sein, die vorhandenen Linientypen der aktuellen Zeichnung in einem Formular aufzulisten und dem Benutzer einige Funktionen zum Laden, Bearbeiten und Löschen anzubieten.

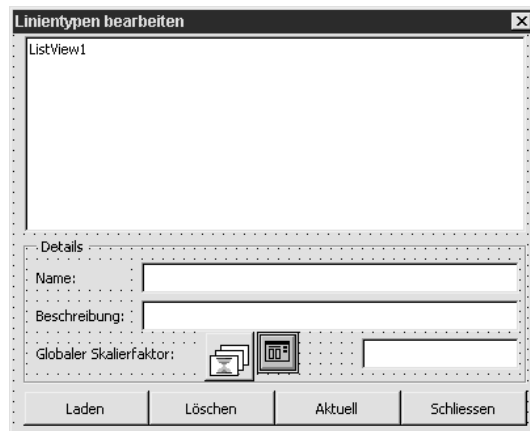


Abbildung 7.11: Formular des Linientypdialogs

Neben diesem Formular enthält das Beispielprojekt ein zweites Formular, das bei der Auswahl zu ladender Linientypen verwendet wird, aber dazu später mehr.

Startprozedur

Als Erstes erzeugen wir eine neue Instanz der Klasse *MyUtil* (Kapitel A.3). Diesmal ist die Objektvariable nicht global, sondern im Formular deklariert, da wir die Funktionen dieser Klasse nur innerhalb dieses Formulars benötigen.

Nachdem die Routine *initLTypeList* zur Initialisierung der Steuerelemente des Formulars aufgerufen wurde, wird das Formular aus Abbildung 7.11 gestartet.

```
Public Sub showLinetypes()  
On Local Error Resume Next  
Set LineTypesFrm.Util = New MyUtil  
LineTypesFrm.initLTypeList  
LineTypesFrm.Show  
End Sub
```

Formularcode

Der folgende Quellcode ist wieder im Formular enthalten und besteht zum größten Teil aus Ereignisprozeduren.

In der Initialisierungsprozedur, die bereits durch die Startprozedur aufgerufen wurde, wird sowohl das *ListView*-Steuerelement mit den Namen und Beschreibungen der vorhandenen Linientypen der aktuellen Zeichnung gefüllt, als auch die *TextBox* mit dem globalen Skalierungsfaktor der Linientypen.

```
Public Sub initLTypeList()
Dim LV As MSComctlLib.ListItem
On Local Error Resume Next
```

Die Füllung des *ListView*-Steuerelements läuft wieder in einer *For...Next*-Schleife ab, wobei bei jedem Durchlauf der Name und die Beschreibung des jeweiligen Linientyps, aufgeteilt auf zwei Spalten, der Liste hinzugefügt werden.

```
ListView1.ListItems.Clear
For i = 0 To ThisDrawing.Linetypes.Count - 1
    Set LV = ListView1.ListItems.Add(, , _
        ThisDrawing.Linetypes(i).Name)
    LV.ListSubItems.Add , , ThisDrawing.Linetypes(i).Description
```

Außerdem wird geprüft, ob es sich bei dem jeweiligen Linientyp um den aktuellen der Zeichnung handelt. Ist dies der Fall, wird ein Symbol vor dem Linientypnamen angezeigt.

```
If ThisDrawing.Linetypes(i).Name = ThisDrawing. _
    ActiveLinetype.Name Then
    LV.SmallIcon = 1
End If
Next
```

Zum Schluss wird noch der globale Skalierfaktor der Linientypen formatiert und in das *TextBox*-Steuerelement eingetragen.

```
TextBox3.Text = Util.RealToString(ThisDrawing. _
    GetVariable("LTSCALE"))
End Sub
```

Wählt der Benutzer einen Eintrag im *ListView*-Steuerelement, so müssen die Eintragungen der Textfelder (Name und Beschreibung) im Bereich *Details* aktualisiert werden. Das *ItemClick*-Ereignis reagiert auf das angesprochene Ereignis, so dass der Code in dieser Prozedur platziert werden kann. Das einzige, was bei dieser Prozedur vorgenommen werden muss, ist die Übergabe des Linientypnamens, der in der Texteigenschaft des *Item*-Objekts enthalten ist, an die *TextBox1* und der Linientypbeschreibung, die mit der Texteigenschaft des Unterelements von *Item* in die *TextBox2* geschrieben wird.

```
Private Sub ListView1_ItemClick(ByVal Item As _
MSComctlLib.ListItem)
On Local Error Resume Next
TextBox1.Text = Item.Text
TextBox2.Text = Item.ListSubItems(1).Text
End Sub
```

Die nächsten drei Prozeduren werden ausgeführt, wenn der Benutzer den Inhalt des jeweils zur Prozedur gehörigen Textfelds ändert. Ändert er z.B. den Namen des Linientyps, kommen folgende Codezeilen zum Einsatz:

```
Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim ActLType As AcadLinetype
On Local Error Resume Next
```

Nach der Deklaration der Variable, kann an eben diese das Linientypobjekt, das dem ausgewählten Element entspricht, übergeben werden:

```
Set ActLType = ThisDrawing.Linetypes(ListView1.SelectedItem.Text)
```

Anschließend versuchen wir, dem Linientyp den neuen Namen zu verpassen. Sollte der Versuch fehlschlagen, löst AutoCAD einen Fehler aus, den Sie gegebenenfalls auswerten können.

```
ActLType.Name = TextBox1.Text
```

In jedem Fall wird der aktuelle Name des Linientyps an die *Text*-Eigenschaft von *ListView1* und *TextBox1* übergeben, um den aktuellen Stand anzeigen zu lassen.

```
ListView1.SelectedItem.Text = ActLType.Name
TextBox1.Text = ActLType.Name
End Sub
```

Gleiches gilt auch für den Beschreibungstext des Linientyps.

```
Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim ActLType As AcadLineType
On Local Error Resume Next
```

Auch hier wird das Linientypobjekt, das zum ausgewählten Namen passt, ausgewählt.

```
Set ActLType = ThisDrawing.Linetypes(ListView1.SelectedItem.Text)
```

Im Anschluss wird die Beschreibung des Linientyps geändert und die aktuellen Werte werden an die *Text*-Eigenschaft des Unterelements von *ListView1* und *TextBox2* übergeben.

```
ActLType.Description = TextBox2.Text
ListView1.SelectedItem.ListSubItems(1).Text = ActLType.Description
TextBox2.Text = ActLType.Description
End Sub
```

Die dritte *BeforeUpdate*-Prozedur soll den eingegebenen Wert der *TextBox3* als globalen Linientypfaktor setzen.

```
Private Sub TextBox3_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim LTFaktor As Double
On Local Error Resume Next
```

Im ersten Schritt wird der Inhalt der *TextBox3* in eine Zahl vom Datentyp *Double* umgewandelt.

```
LTFaktor = Util.DistanceToReal(TextBox3)
```

Anschließend muss kontrolliert werden, ob der eingegebene Wert einer gültigen Zahl entspricht.

```
If LTFaktor <= 0 Then
MsgBox "Linientypfaktor muss ein Wert größer 0 sein", _
vbCritical
Cancel = True
Else
```

Wurde ein korrekter Wert verwendet, können wir diesen an die Systemvariable *LTSCALE* übergeben und anschließend den Inhalt der Textbox neu formatieren, damit der Wert so angezeigt wird, wie es im System vorgesehen ist.

```
ThisDrawing.SetVariable "LTSCALE", LTFaktor
TextBox3 = Util.RealToString(LTFaktor)
End If
End Sub
```

Schaltfläche „Laden“

Die erste Ereignisprozedur wird ausgeführt, wenn der Benutzer die Schaltfläche LADEN betätigt hat. Als Ergebnis dieser Prozedur wird ein Formular eingeblendet, das alle Linientypen einer durch den Benutzer ausgewählten Datei auflistet und zur Auswahl anbietet. Die beiden Funktionen, die innerhalb der Prozedur aufgerufen werden, sind im Anschluss an die folgende Prozedur zu finden.

```
Private Sub CommandButton1_Click()
Dim FData As String
Dim LTName As String
Dim LV As MSComctlLib.ListItem
On Local Error Resume Next
```

Nach der Deklaration der Variablen initialisieren wir wieder einige Eigenschaften des *ComDialog*-Steuerelements und starten anschließend das *Öffnen*-Dialogfenster.

```
ComDialog.DialogTitle = "Linientypen laden"
ComDialog.Filter = "Linientypdateien (*.lin)|*.lin|"
Err.Clear
ComDialog.ShowOpen
If Err = 0 Then
```

Sollte der Benutzer eine Linientypdatei ausgewählt und das Öffnen dieser Datei bestätigt haben, so wird mit den Codezeilen innerhalb des *If...End If*-Blocks fortgesetzt. Dort wird zunächst das *ListView*-Steuerelement des weiter hinten in diesem Kapitel beschriebenen Formulars von vorhandenen Elementen gesäubert und anschließend die ausgewählte Datei zum Lesen geöffnet.

```
With LinetypListFrm
.ListView1.ListItems.Clear
Open ComDialog.FileName For Input As #1
```

In den folgenden Zeilen werden so lange Zeilen eingelesen, bis das Ende der Datei erreicht ist. Erst dann wird sie geschlossen und die Prozedur wird mit den nächsten Codezeilen fortgesetzt.

```
Do While Not EOF(1)
Line Input #1, FData
```

Die Daten befinden sich nun in der Variablen *FData* und können analysiert werden. Wenn Sie einmal eine Linientypdatei öffnen, werden Sie feststellen, dass sich in den Zeilen, die mit einem * beginnen, der Name und die Beschreibung des Linientyps befinden. Also kontrollieren wir die Zeilen auf das Vorhandensein dieses Markers und können dann den entsprechenden Linientyp zum *ListView*-Steuerelement des zweiten Formulars hinzufügen. Da die

Datenzeile ja beides, sowohl den Namen als auch die Beschreibung, des Linientyps enthält, müssen wir diese Informationen herausfiltern. Zu diesem Zweck werden die Funktionen *GetLTName* (Namen herausfiltern) und *GetLTDesc* (Beschreibung herausfiltern) verwendet.

```

If Left(FData, 1) = "*" Then
    Set LV = .ListView1.ListItems.Add(, , _
        GetLTName(FData))
    LV.ListSubItems.Add , , GetLTDesc(FData)
End If
Loop
Close

```

Jetzt heißt es nur noch, das Formular zu starten und die Linientypen der Datei zur Auswahl anzubieten.

```

.Show
If .OK = True Then

```

Wenn der Benutzer im Auswahlformular die Schaltfläche LADEN betätigt hat, enthält die Variable *OK* des Formulars den Wert *TRUE* und die ausgewählten Linientypen können geladen werden. Deshalb werden die einzelnen Elemente des *ListView*-Steuerelements abgerufen und kontrolliert, ob die Eigenschaft *Selected* des *ListView*-Elements auf *TRUE* gesetzt ist. In diesem Fall wurde der Linientyp ausgewählt und er kann geladen werden.

```

For i = 1 To .ListView1.ListItems.Count
    If .ListView1.ListItems(i).Selected = True Then
        LTName = .ListView1.ListItems(i).Text

```

Nachdem der entsprechende Linientypname an die Variable *LTName* übergeben wurde, wird die *Load*-Funktion der Linientypauflistung aktiviert. Im ersten Parameter der *Load*-Anweisung übergeben wir den Linientypnamen, im zweiten den Dateinamen.

```

        ThisDrawing.Linetypes.Load LTName, _
        ComDialog.FileName
    End If
Next

```

Wenn alle Elemente des *ListView*-Steuerelements durchlaufen und alle ausgewählten Linientypen geladen worden sind, muss nur noch die Listenansicht aktualisiert werden.

```

        initLTTypeList
    End If
End With
End If
End Sub

```

Tipp

Wollen Sie einen Linientyp laden, der bereits in der Zeichnung vorhanden ist, so wird ein Fehler ausgelöst und der alte Linientyp bleibt bestehen. Folglich müssten Sie erst den alten Linientyp aus der Zeichnung entfernen und dann den neuen Linientyp laden.

Nun fehlen noch die beiden Funktionen zum Herausfiltern des Namens und der Beschreibung der einzelnen Linientypen. Schauen wir uns einmal eine Datenzeile aus der Datei *acad.lin* an:

```
*RAND,Rand __ . __ . __ . __ . __ . __ .
```


Dort können Sie erkennen, dass Name und Beschreibung durch ein Komma getrennt werden. Also nutzen wir diesen Umstand und lesen den Namen wie folgt aus:

```
Public Function GetLTName(ByVal LTString As String) As String
On Local Error Resume Next
GetLTName = Mid(LTString, 2, InStr(1, LTString, ",") - 1)
End Function
```

Das Ganze für die Beschreibung:

```
Public Function GetLTDesc(ByVal LTString As String) As String
On Local Error Resume Next
GetLTDesc = Mid(LTString, InStr(1, LTString, ",") + 1)
End Function
```

Bei beiden Funktionen liefert die *InStr*-Funktion die genaue Position des Kommas und somit das Ende des Namens und den Anfang der Beschreibung.

Schaltfläche „Löschen“

Die nächste Prozedur soll den bzw. die gewählten Linientypen aus der Zeichnung löschen, wenn der Benutzer die entsprechende Schaltfläche betätigt hat.

```
Private Sub CommandButton2_Click()
Dim LTName As String
Dim AcLT As AcadLineType
On Local Error Resume Next
```

Nach einer Sicherheitsabfrage

```
If MsgBox("Gewählte Linientypen löschen?", vbYesNo, "Löschen") = _
vbYes Then
```

wird das *ListView*-Steuerelement Zeile für Zeile durchlaufen und jedes Element darauf geprüft, ob es ausgewählt wurde.

```
For i = 1 To ListView1.ListItems.Count
LTName = ListView1.ListItems(i).Text
```

Ist dies der Fall, wird der zugehörige Linientyp aus der Linientypauflistung ermittelt und anschließend gelöscht. Achten Sie aber darauf, dass nicht jeder Linientyp gelöscht werden kann. Sollte es dazu kommen, wird ein Fehler ausgelöst und Sie können anhand der Fehlermeldung ermitteln, warum der Linientyp nicht gelöscht werden konnte.

```
Set AcLT = ThisDrawing.Linetypes(LTName)
AcLT.Delete
Next
```

Zum Schluss muss dann nur noch die Liste aktualisiert werden:

```
initLTTypeList
End If
End Sub
```

Schaltfläche „Aktuell“

Bei dieser Prozedur handelt es sich um die Ereignisprozedur der Schaltfläche AKTUELL. Diese soll also den gewählten Linientyp des *ListView*-Steuerelements zum aktuellen Linientyp der Zeichnung machen. Der dazu erforderliche Code ist eigentlich recht einfach gehalten.

```
Private Sub CommandButton3_Click()  
Dim LTName As String  
On Local Error Resume Next
```

Zuerst wird der Name des gewählten Linientyps an die Variable *LTName* übergeben und danach mithilfe dieser Variablen das dazugehörige Linientypobjekt aus der Auflistung ermittelt. Dieses Objekt wird gleichzeitig in die *Document*-Eigenschaft *ActiveLinetype* geschrieben, womit wir schon den aktuellen Linientyp geändert hätten.

```
LTName = ListView1.SelectedItem.Text  
ThisDrawing.ActiveLinetype = ThisDrawing.Linetypes(LTName)
```

Was nun noch folgt, ist lediglich die Aktualisierung der *ListView*-Ansicht. Dabei müssen wir uns den Index des gerade gewählten Linientyps merken, damit er nach der Aktualisierung wieder zum aktiven Linientyp des *ListView*-Steuerelements wird.

```
Idx = ListView1.SelectedItem.Index  
initLTypeList
```

Nachdem die Ansicht neu erstellt wurde, muss das erste Element deselektiert werden, damit der Eintrag nicht unabsichtlich gewählt wird. Anschließend wird der gemerkte Linientyp wieder zum aktuellen Element des *ListView*-Steuerelements.

```
ListView1.ListItems(1).Selected = False  
Set ListView1.SelectedItem = ListView1.ListItems(Idx)  
End Sub
```

In Zusammenhang mit der eben besprochenen Prozedur steht die nächste Prozedur. Diese führt den Code der Schaltfläche AKTUELL aus, sobald der Benutzer einen Doppelklick auf dem *ListView1*-Steuerelement ausgeführt hat.

```
Private Sub ListView1_DblClick()  
CommandButton3_Click  
End Sub
```

Schaltfläche „Schließen“

Nachdem der Benutzer diese Schaltfläche angeklickt hat, müssen Sie nur das Formular mithilfe der *Hide*-Anweisung ausblenden.

```
Private Sub CommandButton4_Click()  
Me.Hide  
End Sub
```

Wenn das Formular das erste Mal gestartet wird, löst dies das Ereignis *Initialize* des Formulars aus. Das können wir nutzen und das *ImageList1*-Steuerelement der *Icon*- und *SmallIcon*-Eigenschaft des *ListView1*-Steuerelements zuordnen. Damit ist das darin enthaltene Symbol für die Anzeige des aktuellen Linientyps für die Einbindung in die Listenansicht verwendbar.

```
Private Sub UserForm_Initialize()
On Local Error Resume Next
Set ListView1.ListItems = ImageList1
Set ListView1.SmallIcons = ImageList1
End Sub
```

Die abschließende Routine wurde bereits am Anfang durch die Startprozedur aufgerufen und soll das *ListView1*-Steuerelement mit den Linientypen der Zeichnung füllen.

```
Public Sub initLTypeList()
Dim LV As MSComctlLib.ListItem
On Local Error Resume Next
```

Nach der Deklaration der Variablen *LV*, die einem einzelnen *ListView*-Element entsprechen wird, können wir zunächst die Liste von vorhandenen Einträgen säubern. Anschließend wird die Linientypauflistung Element für Element durchlaufen.

```
ListView1.ListItems.Clear
For i = 0 To ThisDrawing.Linetypes.Count - 1
```

Für jeden Linientyp wird ein Eintrag im *ListView1*-Steuerelement erzeugt und an die Variable *LV* weitergereicht. Da die Liste eine zweite Spalte enthält, die den Beschreibungstext aufnehmen soll, muss dem *LV*-Objekt ein Subelement hinzugefügt werden.

```
Set LV = ListView1.ListItems.Add(, , _
ThisDrawing.Linetypes(i).Name)
LV.ListSubItems.Add , , ThisDrawing.Linetypes(i).Description
```

Jetzt müssen wir nur noch überprüfen, ob es sich bei dem Linientyp um den aktuellen Linientyp handelt. Ist das so, kann vor dem Linientypnamen das entsprechende Symbol aus dem *ImageList1*-Steuerelement angezeigt werden.

```
If ThisDrawing.Linetypes(i).Name = _
ThisDrawing.ActiveLinetype.Name Then
LV.SmallIcon = 1
End If
Next
```

Zum Schluss setzen wir den Wert der Systemvariablen *LTSCALE* (Globaler Skalierfaktor) formatiert in der *TextBox3* ein.

```
TextBox3.Text = ThisDrawing.Utility.RealToString( _
ThisDrawing.GetVariable("LTSCALE"), acDefaultUnits, 3)
End Sub
```

Damit ist dieses Programm abgeschlossen. Bleibt mir nur noch, einige Anmerkungen zum Thema Linientypen zu machen. Sie haben sich sicher schon gefragt, warum noch keine Informationen zur Abfrage und/oder Bearbeitung des Linienmusters zu finden waren. Nun die Antwort ist ganz einfach: Es ist mit der derzeitigen Version der VBA-Erweiterung noch nicht möglich, das Aussehen der Linientypenmuster direkt zu beeinflussen. Wenn Sie dies vorhaben, müssen Sie auf andere Möglichkeiten zurückgreifen (ObjectARX, Lisp, DXF).

7.6 Layouts

Im Ergebnis der Erstellung einer Zeichnung soll der Benutzer auch die Möglichkeit erhalten, die Zeichnung auf einem Gerät, vorzugsweise einem Plotter, auszugeben. Dazu bietet AutoCAD die Möglichkeit, die einzelnen Blätter so vorzubereiten, dass sie nur noch auf dem Gerät ausgegeben werden müssen.

7.6.1 Layoutverwaltung

Alle Daten einer Zeichnung werden in Layouts abgelegt, wobei eine Zeichnung immer mindestens zwei Layouts besitzt. Bei dem einen handelt es sich um das Layout Modell, in dem die Daten des Modellbereichs gespeichert sind. Sie können dieses Layout weder umbenennen noch löschen. Außerdem ist es nicht möglich, ein weiteres Layout hinzuzufügen, das genau wie der Modellbereich konzipiert ist. Bei allen anderen Layouts handelt es sich um einzelne Papierbereiche, wobei jedes einem Blatt Papier entspricht. Sie können von dieser Art Layouts so viele anlegen, wie Sie benötigen. Alle diese Layouts sind in einer Auflistung zusammengefasst, über die Sie auf jedes einzelne Layout zugreifen können. Diese Auflistung enthält außerdem eine *Add*-Funktion, mit der Sie neue Layouts der Zeichnung hinzufügen können. Die Auflistung ist, genau wie die Layer oder Linientypen, über das Dokumentobjekt abrufbar, wofür Sie die folgenden Codezeilen verwenden könnten.

Erstellung eines Verweises auf die Layoutauflistung der aktiven Zeichnung:

```
Dim AllLayouts As AcadLayouts
Set AllLayouts = Layouts
```

Erstellung eines Verweises auf die Layoutauflistung einer Zeichnung aus der Dokumentauflistung:

```
Dim AllLayouts As AcadLayouts
Set AllLayouts = Documents(n).Layouts
```

Die Klasse *AcadLayouts* stellt natürlich wieder einige Eigenschaften und Funktionen zur Verfügung, mit denen Sie die Layoutauflistung anpassen können. Bevor wir ins Detail gehen, schauen wir uns diese einmal im Einzelnen an.

Layout hinzufügen/vorhandenes Layout abrufen

Die Hauptaufgabe dieser Auflistung besteht natürlich, wie bei allen anderen Auflistungen auch, in der Verwaltung der einzelnen Elemente, in diesem Fall die Layouts der Zeichnung. Dazu gehört natürlich auch die Möglichkeit, neue Layouts in der Zeichnung zu erstellen.

```
Dim NewLayout As AcadLayout
Set NewLayout = Layouts.Add("Layout1")
```

Wenn Sie auf ein bereits vorhandenes Layout zugreifen wollen, benutzen Sie wie gehabt die *Item*-Funktion der Auflistung.

```
Dim CrLayout As AcadLayout
Set CrLayout = Layouts.Item(n)
```

oder

```
Set CrLayout = Layouts(n)
```

Wie bei Layern oder Blöcken können Sie auch mit der *Item*-Funktion der Gruppenauflistung den Gruppennamen verwenden, um auf eine einzelne Gruppe zuzugreifen.

```
Dim LayoutName As String
Dim CrLayout As AcadLayout
LayoutName = "Layout1"
Set CrLayout = Layouts(LayoutName)
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungs-wörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.23: Eigenschaften der Layoutauflistung

Prozedur	Beschreibung
<i>Add</i>	Diese Anweisung fügt ein Layout der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.24: Prozeduren der Layoutauflistung

7.6.2 Einzelne Layouts bearbeiten

Wenden wir uns nun den einzelnen Layoutobjekten zu, über die Sie Zugriff auf die jeweiligen Layoutdaten der Zeichnung erhalten. Mithilfe dieser Objekte können Sie die Layouts entsprechend anpassen. Dazu stellt die Klasse *AcadLayout*, die dem Datentyp eines einzelnen Layout-

objekts entspricht, wieder ein Reihe von Eigenschaften und Funktionen zur Verfügung, die zunächst einmal im Einzelnen beschrieben werden sollen.

In der *Block*-Eigenschaft wird ein Verweis auf die Blockdefinition des Layouts gespeichert. Bei dieser Blockdefinition handelt es sich um besondere (anonyme) Blöcke, die die Daten des Modell- oder Papierbereichs enthalten. Die Verwendung dieses Objekts ist normalerweise nur in Zusammenhang mit den Layouts sinnvoll, die Daten des Papierbereichs enthalten, da das Objekt *PaperSpace* nur den Papierbereich des aktuellen bzw. zuletzt aktivierten Papierbereichslayouts enthält. Wenn Sie beispielsweise die Anzahl aller Daten im Papierbereich ermitteln wollen, könnten Sie nacheinander die Papierlayouts aktivieren und jedes Mal die *Count*-Eigenschaft abfragen.

```
For Each ActLayout In Layouts
    ActiveLayout = ActLayout
    If ActLayout.ModelType = False Then
        Anzahl = Anzahl + PaperSpace.Count
    End If
Next
MsgBox CStr(Anzahl) & " Objekte im Papierbereich gefunden"
```

Diese Vorgehensweise ist aber nicht empfehlenswert, da bei jedem Layoutwechsel der Inhalt des Layouts neu aufgebaut wird, was natürlich einige Zeit in Anspruch nimmt. Besser ist da schon folgende Methode:

```
For I = 0 To Layouts.Count
    If Layouts(i).ModelType = False Then
        Anzahl = Anzahl + Layouts(i).Block.Count
    End If
Next
MsgBox CStr(Anzahl) & " Objekte im Papierbereich gefunden"
```

Bei diesem Beispiel greifen wir auf die Blockdefinition zurück, die denselben Inhalt aufweist, wie das *PaperSpace*-Objekt, wenn das jeweilige Layout aktiviert wird.

Siehe auch *Kapitel 7.10*.

Die *Name*-Eigenschaft speichert die Layoutbezeichnung und kann nur bei Layouts des Papierbereichs geändert werden. Die Änderung der Eigenschaft hat gleichzeitig Auswirkung auf die Beschriftung des Reiters im AutoCAD-Zeichnungsfenster.



Abbildung 7.12: Beschriftung vor der Änderung

```
Layouts("Layout1").Name = "Grundriss"
```



Abbildung 7.13: Beschriftung nach der Änderung

Die Position des Layoutreiters innerhalb der Anordnung im Zeichnungsfenster können Sie über die *TabOrder*-Eigenschaft beeinflussen. Sie können die Position aller Papierbereichslayouts anpassen. Der Reiter für den Modellbereich bleibt davon unberührt, da dieser immer an

erster Stelle in der Anordnung kommen muss. Die *TabOrder*-Eigenschaft hat für das Modell-layout immer den Wert 0. Alle anderen können Sie innerhalb des Bereichs 1 bis Anzahl der Layouts – 1 (*Layouts.Count*) frei positionieren.



Abbildung 7.14: Anordnung vor der Änderung

```
Layouts("Grundriss").TabOrder = 2
```



Abbildung 7.15: Anordnung nach der Änderung

Mit der *ModelType*-Eigenschaft können Sie kontrollieren, ob es sich bei dem Layout um das Modell- oder Papierbereichslayout handelt. Gibt die Eigenschaft den Wert *TRUE* zurück, dann entspricht das Layout dem Layout „Modell“, andernfalls handelt es sich um ein Papierbereichslayout.

Aktualisierung der Informationen über die Plotschnittstelle

Bevor wir uns mit der Verwaltung der Zeichnungsträger und der Einstellung der verschiedenen Plotoptionen auseinander setzen, vorab noch eine wichtige Anmerkung: Bevor Sie gewisse Informationen (Plotgeräte, Plotstiltabellen oder kanonische Zeichnungsträger) über die verfügbaren Plotgeräte abrufen, müssen Sie die *RefreshPlotDeviceInfo*-Anweisung ausführen, da diese die Informationen zu den angesprochenen Listen aktualisiert, um dem aktuellen Systemstatus zu entsprechen. Wenn Sie dies nicht tun, erhalten Sie unter Umständen falsche bzw. gar keine Ergebnisse zu diesen Datenquellen.

```
ActLayout.RefreshPlotDeviceInfo
```

Medien verwalten

Die Verwaltung des Mediums, auf dem die Zeichnung ausgegeben werden soll, erfolgt über verschiedene Eigenschaften und Methoden.

Der erste Schritt besteht in der Zuordnung des Plotausgabegeräts. Die einzelnen Geräte werden in den *pc3*-Dateien verwaltet, woraus folgt, dass jede *pc3*-Datei ein Ausgabegerät darstellt. Um nun dem Layout ein Gerät zuzuweisen, legen Sie den Namen der entsprechenden *pc3*-Datei in der *ConfigName*-Eigenschaft fest. Damit wäre dann auch schon das Ausgabegerät zugewiesen.

Tipp

Wenn Sie den Inhalt dieser Eigenschaft ändern, ändert sich ebenfalls die Liste der Papierformate (*GetCanonicalMediaNames*) und somit auch der Inhalt der Eigenschaft des aktuell eingestellten Papierformats (*CanonicalMediaNames*).

Woher sollen Sie im Vorfeld wissen, welche *pc3*-Dateien im jeweiligen System verfügbar sind? Nun, auch hierfür stellt das Layoutobjekt eine Funktion zur Verfügung. Die *GetPlotDeviceNames*-Funktion gibt ein Array mit den installierten Geräten zurück. Das Array ist nach demselben Prinzip aufgebaut wie in Abbildung 7.16 dargestellt. Jedes Feld enthält einen Gerätenamen, der in AutoCAD verwendet werden kann. Bevor Sie diese Funktion aufrufen, müssen Sie die Geräteliste mit der Anweisung *RefreshPlotDeviceInfo* aktualisieren lassen. Anschließend können Sie die Funktion ausführen lassen und die Elemente des Arrays weiterverwenden.

```

Dim PlotterList As Variant
ActLayout.RefreshPlotDeviceInfo
PlotterList = ActLayout.GetPlotDeviceNames
For i = 0 To UBound(PlotterList)
    MsgBox "Plottername : " & PlotterList(i)
Next

```

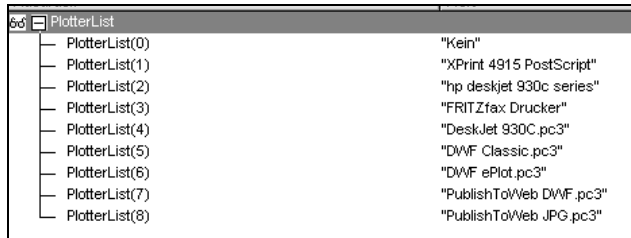


Abbildung 7.16: Beispielausgabe der Gerätenamen

Das festgelegte Papierformat mit seiner kanonischen Bezeichnung ist in der Eigenschaft *CanonicalMediaName* gespeichert. Wenn Sie bei der Seiteneinrichtung für das Layout z.B. folgende Einstellung im Papierformat vornehmen

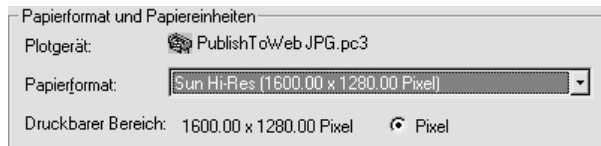


Abbildung 7.17: Auswahl eines Papierformats

dann enthält *CanonicalMediaName* die Zeichenfolge:

```
"Sun_Hi-Res_(1600.00_x_1280.00_Pixels)"
```

Obwohl die Bezeichnung nicht genau mit dem ausgewählten Namen im Listenfeld übereinstimmt, so handelt es sich doch um das aktuelle Format des Listenfelds *Papierformat*. Wenn Sie den Inhalt dieser Eigenschaft ändern, müssen Sie die Zeichnung regenerieren, damit die Änderungen wirksam werden. Wenn Sie versuchen, einen Mediennamen in die Eigenschaft zu schreiben, der nicht definiert ist, löst AutoCAD einen Fehler aus. Eine Liste der kanonischen Papierformate des aktuell eingestellten Ausgabegeräts gibt die Funktion *GetCanonicalMediaNames* zurück. Sie erhalten ein Array mit den entsprechenden Bezeichnungen, das Sie dann weiter auswerten können und beispielsweise einer Listbox in einem Formular hinzufügen. Folgende Codezeilen würden diese Informationen in der AutoCAD-Befehlszeile ausgeben:

```

Layouts(1).RefreshPlotDeviceInfo
MediaNames = Layouts(1).GetCanonicalMediaNames
For I = 0 To UBound(MediaNames)
    Utility.Prompt MediaNames(i)
Next

```


Da die Funktion *GetCanonicalMediaNames* die kanonische Bezeichnung der einzelnen Zeichnungsträger wiedergibt, die nicht mit der lokalen Bezeichnung übereinstimmt, stellt die Klasse *AcadLayout* die Funktion *GetLocaleMediaName* zur Verfügung. Diese gibt die lokalisierte Version der kanonischen Bezeichnung, die Sie im Parameter der Funktion angeben, zurück. In der folgenden Tabelle sind einige Beispiele zusammengestellt, um den Unterschied zwischen beiden Bezeichnungen zu verdeutlichen.

Kanonischer Name	Lokalisierter Name
ANSI_A_(8.50_x_11.00_Inches)	ANSI A (8.50 x 11.00 Zoll)
ARCH_expand_E_(36.00_x_48.00_Inches)	ARCH expand E (36.00 x 48.00 Zoll)
ISO_expand_A0_(841.00_x_1189.00_MM)	ISO expand A0 (841.00 x 1189.00 MM)

Tabelle 7.25: Kanonischer und lokalisierter Name des Zeichnungsformats

Mit dieser Funktion ergänzt, ergibt sich für die vorangegangene Funktion folgender Aufbau. Diese gibt jetzt die lokalisierten Namen der für das aktuell eingestellte Gerät verfügbaren Zeichnungsträger in der AutoCAD-Befehlszeile aus.

```
Layouts(1).RefreshPlotDeviceInfo
MediaNames = Layouts(1).GetCanonicalMediaNames
For I = 0 To UBound(MediaNames)
    Utility.Prompt Layouts(1).GetLocaleMediaName(MediaNames(i))
Next
```

Ploteinstellungen des Layouts anpassen

Die nun folgenden Eigenschaften stehen in direktem Zusammenhang mit der Vorbereitung des Layouts zum Plotten. Sie werden die meisten dieser Eigenschaften auch im Seiteneinrichtungsdialogfenster wiederfinden. Aus diesem Grund wird immer direkt Bezug auf die zur entsprechenden Eigenschaft zugehörige Einstellung in dem Dialogfenster genommen. So können Sie umgehend eine optische Verbindung zur Seiteneinrichtung unter AutoCAD herstellen und wissen dann sofort, was die Änderung der jeweiligen Eigenschaft bewirkt.

Mit der *PaperUnits*-Eigenschaft passen Sie die Einstellung der Einheit an, in der das Bild geplottet werden soll. Im Normalfall werden Sie entweder Zoll oder Millimeter verwenden. Wenn als Ausgabegerät Pixelbilder eingestellt wurden, werden die Einheiten allerdings in Pixel angegeben. Alle drei Möglichkeiten sind in einer Konstantenauflistung zusammengefasst, deren einzelne Konstanten Sie für diese Eigenschaft verwenden können.

Konstante	Beschreibung	Wert
<i>acInches</i>	Papiereinheiten in Zoll	0
<i>acMillimeters</i>	Papiereinheiten in Millimeter	1
<i>acPixels</i>	Papiereinheiten in Pixel	2

Tabelle 7.26: Konstanten der *PaperUnits*-Eigenschaft

```
Layouts("Grundriss").PaperUnits = acMillimeters
```

oder

```
Layouts("Grundriss").PaperUnits = 1
```

Plotoptionen

Die *PlotHidden*-Eigenschaft bestimmt, ob Objekte beim Plotten verdeckt werden sollen oder nicht. Diese Eigenschaft ist identisch mit der Einstellung im Seiteneinrichtungsdialog der Layouts im Abschnitt PLOTOPTIONEN.



Abbildung 7.18: Plotoptionen

Wenn Sie die Option *OBJEKTE AUSBLENDEN* aktivieren, gibt die Eigenschaft den Wert *TRUE* zurück, ansonsten *FALSE*. Auf den umgekehrten Weg trifft natürlich das Gleiche zu. Wenn Sie die Eigenschaft auf *TRUE* setzen, ist die Option aktiviert, andernfalls nicht.

Über die *PlotViewportsFirst*-Eigenschaft steuern Sie, ob der Inhalt der Ansichtsfenster vor den Daten des Papierbereichs geplottet werden soll oder umgekehrt. Diese Eigenschaft steht in direkter Beziehung zur Einstellung *PAPIERBEREICH ZULETZT PLOTTEN* im Seiteneinrichtungsdialog von AutoCAD. Wenn Sie die Eigenschaft *PlotViewportsFirst* auf den Wert *TRUE* setzen, ist die entsprechende Eigenschaft im Einrichtungsdialog mit einem Häkchen versehen, andernfalls nicht.

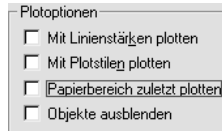


Abbildung 7.19: Plotoptionen

Wenn Sie die Eigenschaft aktivieren, wird der Inhalt der Ansichtsfenster vor den Daten des Papierbereichs geplottet. Andernfalls werden die Daten des Papierbereichs vor den Daten des Ansichtsfensters geplottet.

```
Layouts("Grundriss").PlotViewportsFirst = True
```

Die *PlotWithLineweights*-Eigenschaft entspricht der Einstellung *MIT LINIENSTÄRKEN PLOTTEN* im Einrichtungsdialog von AutoCAD.

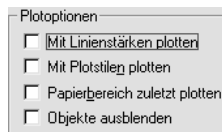


Abbildung 7.20: Plotoptionen

Setzen Sie die Eigenschaft auf den Wert *TRUE*, damit die Objekte mit den jeweils eingestellten Linienstärken geplottet werden, oder auf *FALSE*, wenn die Linienstärken nicht geplottet werden sollen.

```
Layouts("Grundriss").PlotWithLineweights = True
```

Die letzte Eigenschaft, die auch unter den Plotoptionen des Einrichtungdialogs zu finden ist, dient der Aktivierung bzw. Deaktivierung der Plotstile. Wenn Sie die Plotstile aktivieren, indem Sie die *PlotWithPlotStyles*-Eigenschaft auf den Wert *TRUE* setzen, wird auch die Eigenschaft MIT PLOTSTILEN PLOTTEN aktiviert. Enthält die Eigenschaft den Wert *FALSE*, ist auch die Plotoption im Dialogfenster deaktiviert.

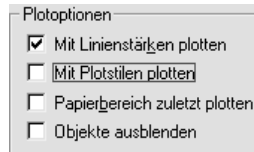


Abbildung 7.21: Plotoptionen

Anders als im Einrichtungdialog wird aber der Wert der Eigenschaft *PlotWithLineweights* nicht automatisch aktiviert, wenn Sie diese Eigenschaft auf *TRUE* setzen. Dies müssten Sie dann zusätzlich vornehmen:

```
Layouts("Grundriss").PlotWithPlotStyles = True
Layouts("Grundriss").PlotWithLineweights = True
```

Plotbereich festlegen

Die Eigenschaft *PlotType* speichert die Einstellung des Plotbereichs, wobei Sie hierfür folgende Konstanten verwenden können:

Konstante	Beschreibung	Wert
<i>acDisplay</i>	Anzeige als Plotbereich verwenden	0
<i>acExtents</i>	Grenzen als Plotbereich verwenden	1
<i>acLimits</i>	Limiten des Papierbereichs verwenden	2
<i>acView</i>	Benannte Ansicht als Plotbereich verwenden	3
<i>acWindow</i>	Fenster als Plotbereich verwenden	4
<i>acLayout</i>	Layout als Plotbereich verwenden	5

Tabelle 7.27: Konstanten der *PlotType*-Eigenschaft

Bei der Einstellung dieser Eigenschaft gilt es noch Folgendes zu beachten. Wenn Sie eine benannte Ansicht als Plotbereich einstellen wollen, muss natürlich auch eine Ansicht vorhanden sein, die Sie mit der Anweisung *ViewToPlot* setzen müssen, bevor Sie diese Eigenschaft auf *acView* einstellen. Die Ansicht, deren Namen Sie hier eintragen, muss in der Zeichnung existieren, da ansonsten ein Fehler ausgelöst wird.

```
ActiveLayout.ViewToPlot = "Ansicht"
```

Eine Auflistung der benannten Ansichten einer Zeichnung erhalten Sie über das Objekt *Views* (Kapitel 7.11), das in jeder Zeichnung abrufbar ist. Sind keine Ansichten verfügbar, kann die *PlotType*-Eigenschaft logischerweise auch nicht auf den Wert *acView* eingestellt werden.

Gleiches gilt für die Auswahl eines Fensterbereichs. Auch hier müssen Sie zunächst das entsprechende Fenster auswählen und mithilfe der Anweisung *SetWindowToPlot* festlegen, bevor Sie die Eigenschaft auf *acWindow* einstellen.

Um noch einen konkreteren Zusammenhang zwischen dieser Eigenschaft und dem Seiteneinrichtungsdialog herzustellen, werden die Konstanten neben den Optionen des Dialogs dargestellt.

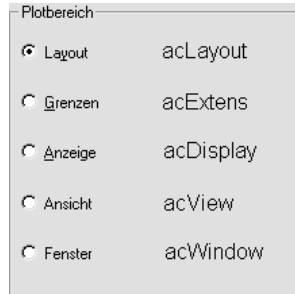


Abbildung 7.22: Plotbereich

Wie bereits erwähnt, müssen Sie für zwei Plotbereiche gesonderte Angaben machen. Bevor Sie die *PlotType*-Eigenschaft auf *acWindow* einstellen, müssen Sie den Plotbereich dieses Fensters anhand von Koordinaten festlegen. Dies geschieht mit der *SetWindowToPlot*-Anweisung. Diese setzt die Koordinaten der linken unteren sowie der rechten oberen Ecke des gewählten Plotbereichs. Sie müssen bei der Angabe der beiden Eckpunkte unbedingt darauf achten, dass diese Funktion nur 2D-Arrays unterstützt (*Dim LowerLeft(0 To 1) As Double*). Wenn Sie also die Eckpunkte über die Funktion *GetPoint* oder *GetCorner* durch den Benutzer auswählen lassen, müssen diese erst in die entsprechende Array-Größe umgewandelt werden. Dafür könnten Sie die *ReDim*-Anweisung verwenden, was folgendes Beispiel verdeutlicht.

```
Dim LeftCorner, RightCorner
On Local Error Resume Next
LeftCorner = Util.GetPoint(, "linke untere Ecke wählen")
If IsEmpty(LeftCorner) = False Then
    RightCorner = Util.GetCorner(LeftCorner, _
        "rechte obere Ecke wählen")
    If IsEmpty(RightCorner) = False Then
        ReDim Preserve LeftCorner(1) As Double
        ReDim Preserve RightCorner(1) As Double
        ActiveLayout.SetWindowToPlot LeftCorner, RightCorner
        ActiveLayout.PlotType = acWindow
    End If
End If
```

Um die aktuell eingestellten Koordinaten des Plotfensters zu ermitteln, benutzen Sie die Anweisung *GetWindowToPlot*. Diese liefert die Koordinaten der linken unteren sowie der rechten oberen Ecke des gewählten Plotbereichs als 2D-Array.

```
ActiveLayout.GetWindowToPlot LowerLeft, UpperRight
```

Plotskalierung festlegen

Um den Plotmaßstab/die Skalierung des Layouts festzulegen, können Sie der Eigenschaft *StandardScale* verschiedene Werte zuweisen. Der Wert dieser Eigenschaft entspricht der Auswahl der Skalierungsliste im Einrichtungsdialog. Wenn Sie den Wert ändern, ändert sich auch die Auswahl der Liste und umgekehrt.

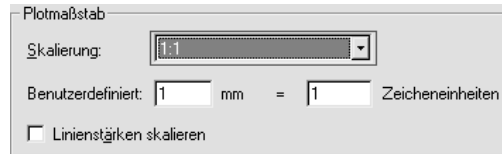


Abbildung 7.23: Standardskalierung

Da es für die Skalierung eine fest vorgegebene Anzahl an Einstellungsmöglichkeiten gibt, wurde eine Konstantenauflistung definiert, aus der Sie Werte verwenden können.

Konstante	Skalierung	Wert	Konstante	Skalierung	Wert
<i>acScaleToFit</i>	Größe angepasst	0	<i>ac1_1</i>	1 : 1	16
<i>ac1_128in_1ft</i>	1/128" = 1'0"	1	<i>ac1_2</i>	1 : 2	17
<i>ac1_64in_1ft</i>	1/64" = 1'0"	2	<i>ac1_4</i>	1 : 4	18
<i>ac1_32in_1ft</i>	1/32" = 1'0"	3	<i>ac1_8</i>	1 : 8	19
<i>ac1_16in_1ft</i>	1/16" = 1'0"	4	<i>ac1_10</i>	1 : 10	20
<i>ac3_32in_1ft</i>	3/32" = 1'0"	5	<i>ac1_16</i>	1 : 16	21
<i>ac1_8in_1ft</i>	1/8" = 1'0"	6	<i>ac1_20</i>	1 : 20	22
<i>ac3_16in_1ft</i>	3/16" = 1'0"	7	<i>ac1_30</i>	1 : 30	23
<i>ac1_4in_1ft</i>	1/4" = 1'0"	8	<i>ac1_40</i>	1 : 40	24
<i>ac3_8in_1ft</i>	3/8" = 1'0"	9	<i>ac1_50</i>	1 : 50	25
<i>ac1_2in_1ft</i>	1/2" = 1'0"	10	<i>ac1_100</i>	1 : 100	26
<i>ac3_4in_1ft</i>	3/4" = 1'0"	11	<i>ac2_1</i>	2 : 1	27
<i>ac1in_1ft</i>	1" = 1'0"	12	<i>ac4_1</i>	4 : 1	28
<i>ac3in_1ft</i>	3" = 1'0"	13	<i>ac8_1</i>	8 : 1	29
<i>ac6in_1ft</i>	6" = 1'0"	14	<i>ac10_1</i>	10 : 1	30
<i>ac1ft_1ft</i>	1'0" = 1'0"	15	<i>ac100_1</i>	100 : 1	31

Tabelle 7.28: Konstantenauflistung *AcPlotScale*

```
ActiveLayout.StandardScale = ac1_100
```

oder

```
ActiveLayout.StandardScale = 26
```

Wenn Sie statt der Standardskalierungswerte eine benutzerdefinierte Skalierung für das Layout verwenden, können Sie mit dieser Prozedur *GetCustomScale* die aktuell eingestellten Skalierungswerte abrufen. Diese Werte werden in den Variablen, die Sie der Prozedur als Parameter mitgeben, zurückgegeben. Die erste Variable (*Nominator*) enthält die Länge in Zoll/Millimeter, die zweite (*Denominator*) die Anzahl der Zeicheneinheiten, denen der erste Wert entsprechen soll.

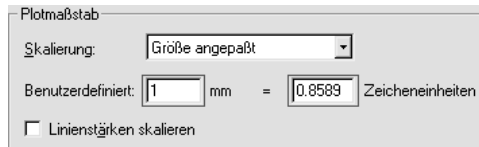


Abbildung 7.24: Benutzerskalierung

```
Dim Nominator As Double, Denominator As Double
ActiveLayout.GetCustomScale Nominator, Denominator
```

Das genaue Gegenstück zu *GetCustomScale* ist *SetCustomScale*, womit klar wäre, dass hiermit die benutzerdefinierte Skalierung gesetzt werden kann. Dementsprechend haben die beiden Parameter die gleiche Bedeutung wie in der vorherigen Prozedur.

```
Dim Nominator As Double, Denominator As Double
Nominator = 1
Denominator = 0.5
ActiveLayout.SetCustomScale Nominator, Denominator
```

Um nun festzulegen, ob eine benutzerdefinierte oder eine der Standardskalierungen verwendet werden soll, passen Sie den Wert der *UseStandardScale*-Eigenschaft entsprechend an. Um die Standardskalierungswerte zu verwenden, setzen Sie diese Eigenschaft auf den Wert *TRUE*, andernfalls auf den Wert *FALSE*.

```
ActiveLayout.UseStandardScale = True
```

Eine weitere Eigenschaft, die mit der Skalierung der Zeichnung zusammenhängt, ist *ScaleLine-weights*. Mit dieser Eigenschaft können Sie festlegen, ob die Linienstärke mit dem Rest der Geometrie skaliert wird, wenn ein Layout gedruckt wird. Diese Eigenschaft finden Sie im Dialog im Abschnitt *PLOTMASSSTAB*.

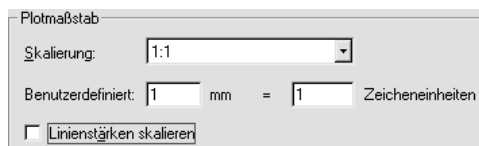


Abbildung 7.25: Linienstärke skalieren

```
ActiveLayout.ScaleLineweights = True
```

Verwenden von Plotstiltabellen

Durch Anpassung der Eigenschaft *ShowPlotStyles* legen Sie fest, ob Plotstile und Plotstilnamen in der Zeichnung angezeigt und geplottet werden sollen. Weisen Sie der Eigenschaft den Wert *TRUE* zu, werden die Plotstile angezeigt und die Option im Einrichtungsdialog ist angehakt. Der Wert *FALSE* deaktiviert die Option.

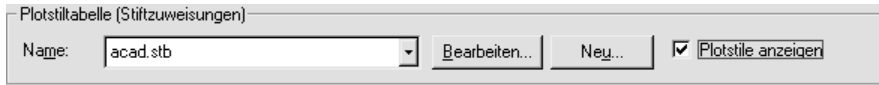


Abbildung 7.26: Plotstile anzeigen

```
ActiveLayout.ShowPlotStyles = True
```

Mit der *StyleSheet*-Eigenschaft können Sie die Plotstiltabelle, die in dem Layout verwendet werden soll, festlegen. Der Name, der in dieser Eigenschaft gespeichert ist, entspricht der Auswahl der Plotstiltabelle im Einrichtungsdialog von AutoCAD.

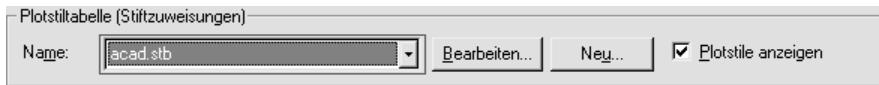


Abbildung 7.27: Plotstiltabelle

Wenn Sie versuchen, eine Plotstiltabelle festzulegen, die nicht existiert, löst AutoCAD einen Fehler aus.

```
ActiveLayout.StyleSheet = "acad.stb"
```

Da im System mehrere Plotstiltabellen verfügbar sein können, stellt das Layoutobjekt für die Ermittlung der Tabellen eine Anweisung zur Verfügung. Die *GetPlotStyleTableNames*-Funktion gibt ein Array mit den installierten Plotstiltabellen zurück. Auch hier müssen Sie zunächst die Anweisung *RefreshPlotDeviceInfo* ausführen, damit die Liste abgerufen werden kann.

```
Dim TableList As Variant
ActLayout.RefreshPlotDeviceInfo
PlotterList = ActLayout.GetPlotStyleTableNames
For i = 0 To UBound(TableList)
    ThisDrawing.Utility.Prompt TableList(i)
Next
```

Plotposition bestimmen

CenterPlot legt fest, ob der Plot auf dem Zeichnungsträger zentriert werden soll oder nicht. Legen Sie für diese Eigenschaft den Wert *TRUE* fest, um den Plot zu zentrieren. Wenn Sie die Eigenschaft *PlotType* auf *acLayout* gesetzt haben, was gleichbedeutend mit der Einstellung Plotbereich → Layout ist, können Sie diese Eigenschaft nicht auf den Wert *TRUE* setzen. In der *PlotOrigin*-Eigenschaft sind die Ursprungskoordinaten des Plotbereichs gespeichert. Diese Koordinaten stellen den Abstand des Plots vom unteren linken Blattrand dar und sind identisch mit den Einstellungen im Abschnitt PLOTABSTAND im Seiteneinrichtungsdialog.

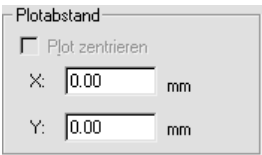


Abbildung 7.28: Plotposition

Die beiden Abstandswerte werden allerdings immer in Millimeter abgelegt, auch wenn Sie die Eigenschaft *PaperUnits* auf Zoll oder Pixel eingestellt haben.

```
Dim Origin(1) As Double
Origin(0) = 10
Origin(1) = 20
ThisDrawing.Layouts("Grundriss").PlotOrigin = Origin
```

Papierausrichtung festlegen

Um die Ausrichtung eines Blatts auf dem Plotter anzupassen, verwenden Sie die Eigenschaft *PlotRotation*. Sie stellt eine Kombination aus den Optionen HOCHFORMAT/QUERFORMAT und AUF DEM KOPF dar. Daraus ergeben sich vier verschiedene Einstellungsmöglichkeiten für diese Eigenschaft, für die jeweils ein Wert in der Konstantenauflistung *acPlotRotation* definiert wurde.

Konstante	Beschreibung	Wert
<i>ac0degrees</i>	Hochformat	0
<i>ac90degrees</i>	Querformat	1
<i>ac180degrees</i>	Hochformat + auf dem Kopf	2
<i>ac270degrees</i>	Querformat + auf dem Kopf	3

Tabelle 7.29: Konstanten der *PlotRotation*-Eigenschaft

```
ActiveLayout.PlotRotation = ac0degrees
```

oder

```
ActiveLayout.PlotRotation = 0
```

Weitere Eigenschaften und Prozeduren

PlotViewportBorders bestimmt, ob die Grenzen des Ansichtsfensters geplottet werden sollen oder nicht. Sie können für diese Eigenschaft zwei Werte festlegen. *TRUE* bestimmt, dass die Ansichtsfenstergrenzen geplottet werden, *FALSE* gibt vor, dass sie nicht geplottet werden.

```
ActiveLayout.PlotViewportBorders = True
```

AutoCAD bietet bekanntlich die Möglichkeit, Seiteneinrichtungen, die Sie an einem Layout vorgenommen haben, als benannte Vorlage zu speichern und so anderen Layouts zur Verfügung zu stellen. Diese Einstellungen werden in der *Plotconfigurations*-Auflistung gespeichert und können somit auch über VBA angesprochen werden. Mit der *CopyFrom*-Anweisung ver-

anlassen Sie AutoCAD, eine bestimmte Plotkonfiguration auf das Layout zu übertragen. Die Ausführung dieser Funktion hat die gleiche Wirkung wie die Auswahl einer benutzerdefinierten Seiteneinrichtung im AutoCAD-Dialog.

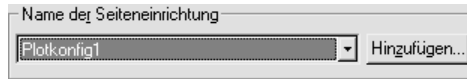


Abbildung 7.29: Seiteneinrichtung übernehmen

Als Parameter müssen Sie einen Verweis auf das entsprechende Konfigurationsobjekt angeben, dessen Einstellungen Sie auf das Layout übertragen wollen.

```
Dim PltCfg As AcadPlotConfiguration
Set PltCfg = PlotConfigurations("Plotkonfig 1")
ActiveLayout.CopyFrom PltCfg
```

GetPaperMargins gibt die Randbreiten des Layouts zurück. Die Variable *LowerLeft* enthält die Breite des linken und des unteren Rands, die Variable *UpperRight* die des rechten und des oberen Rands.

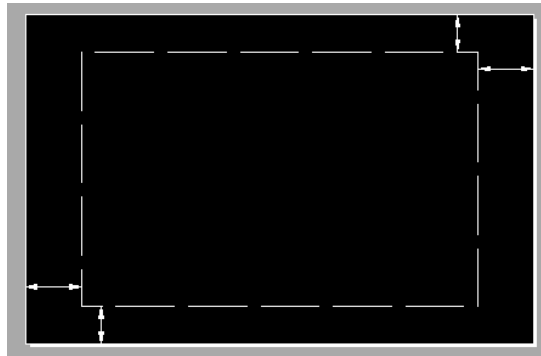


Abbildung 7.30: Randbereiche eines Layouts

```
ActiveLayout.GetPaperMargins LowerLeft, UpperRight
Links = LowerLeft(0)
Unten = LowerLeft(1)
Rechts = UpperRight(0)
Oben = UpperRight(1)
```

GetPaperSize gibt die Breite und die Höhe des konfigurierten Papiers zurück:

```
Dim Width As Double, Height As Double
ActiveLayout.GetPaperSize Width, Height
Prompt = "Aktuelle Breite :" & Util.RealToString(Width) & vbCrLf _
    & "Aktuelle Höhe :" & Util.RealToString(Height)
ThisDrawing.Utility.Prompt Prompt
```

Die Einheit, in der die Papiergröße zurückgegeben wird, entspricht der eingestellten Einheit in der Eigenschaft *PaperUnits*.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Description</i>	Bestimmt einen beliebigen Beschreibungstext für den Linientyp.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Name</i>	Legt den Namen des Layouts fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.30: Weitere Eigenschaften des Layoutobjekts

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.31: Weitere Prozeduren des Layoutobjekts

Anwendungsbeispiel

(Beispielprojekt: 07\AcadLayouts.dvb)

Nach so viel Theorie soll nun wieder ein praktisches Anwendungsbeispiel folgen. In dieser Anwendung werden wir die meisten Eigenschaften und Methoden eines Layouts verwenden, so dass Sie ein Gefühl für die Zusammenhänge zwischen den Eigenschaften bekommen. Das Setzen einer einzelnen Eigenschaft oder der Abruf bestimmter Information über eine Funktion des Layoutobjekts ist ja relativ einfach, aber die Änderung kann durchaus Einfluss auf den Wert einer anderen Eigenschaft haben, so dass Sie auf die entsprechende Änderung reagieren müssen. Zu diesem Zweck werden wir uns ein Programm zusammenbauen, das in etwa dem Seiteneinrichtungsdialog entspricht.

Kommen wir nun zum interessanten Teil, zum Quellcode. Die erste Prozedur, die erzeugt werden muss, ist die Startprozedur des Programms. Diese wird dann über VBARUN oder VBAAUSF gestartet und soll lediglich eine Initialisierungsprozedur ausführen und anschließend das Formular starten.

Abbildung 7.31: Formularentwurf des Layout-Beispielprogramms

Startprozedur

Die Initialisierungsprozedur *iniLayoutDialog* legen wir wieder in den Formularcode, damit der Zugriff auf die Steuerelemente etwas einfacher ist. Das war auch schon alles auf Modulebene, woraufhin wir gleich zum Formularcode weitergehen, der sich dieses Mal etwas umfangreicher gestaltet.

```
Public Sub showLayoutSetup()
Set LayoutFrm.Util = New MyUtility
LayoutFrm.initLayoutDialog
LayoutFrm.Show
End Sub
```

Formularcode

Die ersten beiden Zeilen dimensionieren drei Variablen, die folgende Funktion im Formular übernehmen werden. In der Objektvariablen *SelectedLayout* wird immer ein Verweis auf das ausgewählte Layout erstellt. Die Variable *NoChange* soll vor unbeabsichtigten Änderungen an den Eigenschaften des Layouts schützen. Aber dazu mehr im Verlaufe des Codes. Außerdem wird wieder eine öffentliche Variable *Util* deklariert, der beim Start des Programms eine neue Instanz der Klasse *MyUtility* (siehe Kapitel A.3) zugewiesen wird.

```
Dim SelectedLayout As AcadLayout
Dim NoChange As Boolean
Public Util As MyUtility
```

Initialisierungsprozedur

Die erste Prozedur, die wir dem Formularcode hinzufügen wollen, wird das Formular so vorbereiten, dass alle Combolisten gefüllt sind und ein Layout ausgewählt wird.

```
Public Sub initLayoutDialog()
On Local Error Resume Next
```

Die erste Combobox, die wir mit Einträgen füllen werden, ist die, in der der Benutzer das Layout festlegen kann, dessen Eigenschaften angepasst werden sollen. Um alle Layouts der Zeichnung zu erhalten, müssen wir bekanntlich das Auflistungsobjekt der Layouts verwenden. Um die Liste zu füllen, wird das Auflistungsobjekt vom ersten (0) bis zum letzten Element (*Count* – 1) durchlaufen und der Name des Layouts an der jeweiligen Position *i* zum *ComboBox*-Steuerelement hinzugefügt. Sie könnten auch eine *For Each ... Next*-Schleife statt einer *For...Next* Schleife verwenden, das Ergebnis ist dasselbe.



Abbildung 7.32: Auswahlliste der Layoutnamen

```
LayoutCombo.Clear
For i = 0 To ThisDrawing.Layouts.Count - 1
    LayoutCombo.AddItem ThisDrawing.Layouts(i).Name
Next
```

Die gleiche Methode wird auch bei den Plotkonfigurationen angewendet. Nur dass hier zunächst ein Eintrag den Konfigurationseinträgen vorangestellt wird. Dies ist notwendig, damit später eine Auswahl vorgenommen werden kann, auch wenn nur eine Plotkonfiguration existiert.



Abbildung 7.33: Auswahlliste für die Seiteneinrichtung

```
PlotCfgCombo.Clear
PlotCfgCombo.AddItem "<Seiteneinrichtung wählen>"
```

Bei der Füllung der Combobox ist natürlich zu beachten, dass das *PlotConfigurations*-Objekt verwendet wird, das alle Konfigurationen enthält.

```
For i = 0 To ThisDrawing.PlotConfigurations.Count - 1
    PlotCfgCombo.AddItem ThisDrawing.PlotConfigurations(i).Name
Next
PlotCfgCombo.ListIndex = 0
```

Kommen wir zur nächsten Combobox. Diese wird alle Namen von benannten Ansichten der aktuellen Zeichnung aufnehmen. Die Ansichten sind in der Auflistung *View* gespeichert und werden, genau wie die Layoutnamen oder Plotkonfigurationsnamen zuvor, der Combobox als Einträge hinzugefügt.



Abbildung 7.34: Auswahlliste für benannte Ansichten

```
ViewCombo.Clear
For i = 0 To ThisDrawing.Views.Count - 1
    ViewCombo.AddItem ThisDrawing.Views(i).Name
Next
```

Nachdem die vorhandenen Ansichten der Combobox hinzugefügt wurden, können wir gleich kontrollieren, ob überhaupt Ansichten vorhanden sind, und dementsprechend das Optionsfeld ANSICHT anpassen.

```
If ViewCombo.ListCount = 0 Then
```

Wurden keinen Ansichten gefunden, müssen wir das Optionsfeld und die Combobox deaktivieren, so dass der Benutzer hier nichts mehr auswählen kann.

```
ViewCombo.Enabled = False
PlotType4.Enabled = False
Else
```

Falls doch Ansichten vorhanden sind, muss das Optionsfeld natürlich wieder durch den Benutzer aktivierbar sein, was wir hiermit vornehmen.

```
ViewCombo.Enabled = True
PlotType4.Enabled = True
End If
```

Die nächste Combobox mit Einträgen zu füllen, wird etwas umständlich. Hier wollen wir die einzelnen Standardskalierungen eintragen, die durch die Konstantenauflistung *AcPlotScale* vorgegeben sind. Wir müssen diese ihrem jeweiligen Wert nach der Combobox hinzufügen, damit die Zuweisung des korrekten Werts an die Eigenschaft *StandardScale* etwas einfacher wird.



Abbildung 7.35: Auswahlliste für die Standardskalierung

Vor den eigentlichen Skalierungswerten fügen wir noch den Eintrag „Benutzerdefiniert“ ein. Dieser ist zwar nicht als Konstante definiert und kommt auch nicht als Übergabewert für die Eigenschaft *StandardScale* in Frage, er ist aber notwendig, um die Aktivierung bzw. Deaktivierung der Standardskalierung zu steuern.

```
ScaleCombo.Clear
ScaleCombo.AddItem "Benutzerdefiniert"
ScaleCombo.AddItem "Größe angepasst"
```

Bleiben noch die restlichen Werte, die Sie der Tabelle 7.28 entnehmen können.

```
ScaleCombo.AddItem "1/128'' = 1'0''"
ScaleCombo.AddItem "1/64'' = 1'0''"
ScaleCombo.AddItem "1/32'' = 1'0''"
ScaleCombo.AddItem "1/16'' = 1'0''"
ScaleCombo.AddItem "3/32'' = 1'0''"
ScaleCombo.AddItem "1/8'' = 1'0''"
ScaleCombo.AddItem "3/16'' = 1'0''"
ScaleCombo.AddItem "1/4'' = 1'0''"
ScaleCombo.AddItem "3/8'' = 1'0''"
ScaleCombo.AddItem "1/2'' = 1'0''"
ScaleCombo.AddItem "3/4'' = 1'0''"
ScaleCombo.AddItem "1 '' = 1'0''"
ScaleCombo.AddItem "3'' = 1'0''"
ScaleCombo.AddItem "6'' = 1'0''"
```

```

ScaleCombo.AddItem "1'0'' = 1'0''"
ScaleCombo.AddItem "1 : 1"
ScaleCombo.AddItem "1 : 2"
ScaleCombo.AddItem "1 : 4"
ScaleCombo.AddItem "1 : 8"
ScaleCombo.AddItem "1 : 10"
ScaleCombo.AddItem "1 : 16"
ScaleCombo.AddItem "1 : 20"
ScaleCombo.AddItem "1 : 30"
ScaleCombo.AddItem "1 : 40"
ScaleCombo.AddItem "1 : 50"
ScaleCombo.AddItem "1 : 100"
ScaleCombo.AddItem "2 : 1"
ScaleCombo.AddItem "4 : 1"
ScaleCombo.AddItem "8 : 1"
ScaleCombo.AddItem "10 : 1"
ScaleCombo.AddItem "100 : 1"

```

Nachdem alle Initialisierungen abgeschlossen sind, setzen wir noch den Namen des aktiven Layouts als Auswahl der Layout-Combobox. Das wird dann ein Ereignis auslösen, dessen Prozedur wiederum die Anpassung der einzelnen Steuerelemente übernehmen wird.

```

LayoutCombo.Text = ThisDrawing.ActiveLayout.Name
End Sub

```

Auswahl eines Layoutnamens

Und mit dieser Ereignisprozedur werden wir auch fortfahren. Wie gerade erwähnt, wird sie ausgelöst, wenn die Auswahl der Combobox geändert wird. Dies kann durch den Benutzer oder durch Änderung der Texteigenschaft innerhalb des Programmcodes geschehen.

```

Private Sub LayoutCombo_Click()
Dim MediaList As Variant
On Local Error Resume Next

```

Damit keine unabsichtlichen Änderungen in anderen Ereignisprozeduren vorgenommen werden, solange wir uns innerhalb dieser Prozedur befinden, setzen wir die Variable *NoChange* auf *TRUE*.

```
NoChange = True
```

Jetzt wird ein Verweis auf das ausgewählte Layout anhand des Namens in der Variablen *SelectedLayout* gespeichert. Mit diesem Verweis werden wir innerhalb des Formularcodes dann weiterarbeiten.

```
Set SelectedLayout = ThisDrawing.Layouts(LayoutCombo.Text)
```

Den nun folgenden Code schließen wir in einen *With..End With*-Block ein, um ein wenig Platz zu sparen.

```
With SelectedLayout
```

Werten wir nun die *TabOrder*-Eigenschaft aus. Diese speichert bekanntlich die Position der Registerkarte des Layouts und diese Position tragen wir nun in die entsprechende Textbox ein, damit der Benutzer sie anpassen kann. Da die Position des Modellbereichs festliegt, muss dem

Benutzer die Möglichkeit genommen werden, den Wert zu ändern, wenn das Modelllayout gewählt wurde. Dazu nutzen wir die *Locked*-Eigenschaft und übergeben dieser den Wert der *ModelType*-Eigenschaft. Wurde der Modellbereich gewählt, gibt *ModelType* *TRUE* zurück und sperrt somit die Eingabe im Textfeld. Andernfalls wird die Eingabe wieder freigeschaltet.

```
TabOrder.Text = .TabOrder
TabOrder.Locked = .ModelType
```

Außerdem muss der Text der ersten Option im Abschnitt PLOTBEREICH dahingehend angepasst werden, dass LIMITEN zur Auswahl angeboten wird, wenn der Benutzer den Modellbereich gewählt hat, und in den anderen Fällen LAYOUT.

```
If .ModelType = True Then
    PlotType1.Caption = "Limiten"
Else
    PlotType1.Caption = "Layout"
End If
```

Als Nächstes wird die Combobox, die eine Auswahl eines Papierformats ermöglicht, mit allen Formaten gefüllt, die das aktuelle Gerät bereitstellt.

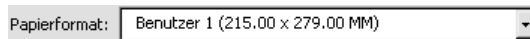


Abbildung 7.36: Auswahlliste der Papierformate

Zunächst müssen die Informationen zu den Zeichnungsträgern aktualisiert werden. Dies erledigt die Anweisung *RefreshPlotDeviceInfo*. Danach kann die Liste der Papierformate abgerufen und in das Array *MediaList* ausgegeben werden. Da diese Papierformate noch nicht der lokalen Bezeichnung entsprechen, muss jedes einzelne Format konvertiert werden, bevor es der Combobox hinzugefügt wird.

```
FormatCombo.Clear
SelectedLayout.RefreshPlotDeviceInfo
MediaList = .GetCanonicalMediaNames
For i = 0 To UBound(MediaList)
    FormatCombo.AddItem .GetLocaleMediaName(MediaList(i))
Next
```

Damit die Combobox das aktuelle Papierformat des Layouts anzeigt, übergeben wir den Inhalt von *CanonicalMediaName* an die *Text*-Eigenschaft der Combobox. Damit auch hier der lokale Name übergeben wird, wird der Wert der *CanonicalMediaName*-Eigenschaft wieder mit der Funktion *GetLocaleMediaName* konvertiert.

```
FormatCombo.Text = .GetLocaleMediaName(.CanonicalMediaName)
PlotterLabel.Caption = .ConfigName
```

Die Einstellung des richtigen Plotbereichs ist relativ einfach. Wir müssen nur die *PlotType*-Eigenschaft in einer *Select Case*-Anweisung vergleichen lassen. Gibt *PlotType* *acLayout* zurück, wird die erste Option aktiviert, bei *acExtents* die zweite usw.

```
Select Case .PlotType
    Case acLayout, acLimits
        PlotType1.Value = True
```

```

Case acExtents
    PlotType2.Value = True
Case acDisplay
    PlotType3.Value = True
Case acView
    PlotType4.Value = True
Case acWindow
    PlotType5.Value = True
End Select

```

Kümmern wir uns nun um die Ausrichtung des Blatts. Die Eigenschaft *PlotRotation*, deren Inhalt wir verwenden werden, kann vier verschiedene Werte liefern, die in zwei Optionsfeldern und einer Checkbox untergebracht werden müssen. Also prüfen wir zunächst, ob es sich bei der Ausrichtung um ein Hochformat handelt. Das ist der Fall, wenn die Eigenschaft den Wert *ac0degrees* oder *ac180degrees* zurückgibt. Also kann in diesen Fällen das Optionsfeld für das Hochformat aktiviert werden, andernfalls das für das Querformat.

```

If .PlotRotation = ac0degrees Or .PlotRotation = _
    ac180degrees Then
    Hochformat.Value = True
Else
    Querformat.Value = True
End If

```

Jetzt muss nur geprüft werden, ob das Blatt auf dem Kopf gedreht werden soll. Das ist immer dann der Fall, wenn die eingestellte Richtung größer als 90° ist, und dementsprechend wird die entsprechende *CheckBox* aktiviert oder eben deaktiviert.

```

If .PlotRotation > ac90degrees Then
    UpSideDown.Value = True
Else
    UpSideDown.Value = False
End If

```

Jetzt müssen noch die Einstellungen für die einzelnen Checkboxes vorgenommen werden, was nicht schwierig ist, da die *Value*-Eigenschaft gleich der entsprechenden Eigenschaft sein muss. Wenn z.B. die Eigenschaft *ScaleLineweights* aktiviert wurde, also den Wert *TRUE* enthält, muss die dazugehörige Checkbox ebenfalls aktiviert werden, also auch den Wert *TRUE* erhalten.

```

ScaleLineweights.Value = .ScaleLineweights
PlotWithLineweights.Value = .PlotWithLineweights
PlotWithPlotStyles.Value = .PlotWithPlotStyles
PlotViewportsFirst.Value = .PlotViewportsFirst
PlotHidden.Value = .PlotHidden

```

Für die Auswahl der Standardskalierung kombinieren wir die Eigenschaften *UseStandardScale* und *StandardScale*. Gibt *UseStandardScale* den Wert *FALSE* zurück, ist die benutzerdefinierte Skalierung aktiviert und der erste Eintrag der Combobox muss ausgewählt werden. Andernfalls setzen wir einfach den Wert, den *StandardScale* liefert, als *ListIndex*, also als ausgewähltes Element der Combobox. Da die Konstanten, die hier verwendet werden, mit 0 beginnen, die entsprechenden Eintragungen aber erst mit dem Index 1 beginnen, muss zu der Konstanten 1 aufaddiert werden.


```

If .UseStandardScale = False Then
    ScaleCombo.ListIndex = 0
Else
    ScaleCombo.ListIndex = .StandardScale + 1
End If

```

Als Letztes wird noch die Prozedur *SetPaperUnits* ausgeführt, die weitere Einstellungen am Dialogfenster vornimmt.

```

SetPaperUnits
End With
NoChange = False
End Sub

```

Einstellung der Papiereinheiten

Da die Einstellung der Papiereinheiten abhängig vom Ausgabegerät ist, müssen wir diese immer wieder überprüfen, wenn eine Änderung vorgenommen wird – sei es an den Optionen für die Einheitenauswahl oder bei der Einstellung der Layouteigenschaften. Da wir nicht in jeder Prozedur, die Auswirkungen auf die Einstellung der Optionsfelder hat, den Quellcode eintragen wollen, fassen wir das Ganze als Prozedur zusammen und rufen an der entsprechenden Stelle nur noch diese Prozedur auf.

```

Public Sub SetPaperUnits()
On Local Error Resume Next
NoChange = True

```

Alles, was diese Prozedur erledigen soll, steht in den folgenden Codezeilen. Mittels einer *Select Case*-Anweisung kann geprüft werden, welche Einheit gewählt wurde, und dementsprechend wird das dazugehörige Optionsfeld ausgewählt. Zusätzlich wird die *Caption*-Eigenschaft der Label angepasst, die die Einheiten anzeigen.

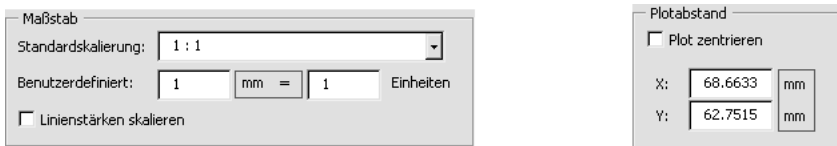


Abbildung 7.37: Einheiteneinstellung in den entsprechenden Dialogabschnitten

```

Select Case SelectedLayout.PaperUnits
Case acInches
    UnitOption1.Value = True
    XLabel.Caption = "Zoll"
    Label7.Caption = "Zoll ="
Case acMillimeters
    UnitOption2.Value = True
    XLabel.Caption = "mm"
    Label7.Caption = "mm ="
Case Else
    UnitOption3.Value = True
    XLabel.Caption = "Pixel"

```

```

Label7.Caption = "Pixel ="
End Select
YLabel.Caption = XLabel.Caption

```

Am Ende der Prozedur wird eine weitere Prozedur ausgeführt, die die restlichen Einstellungen im Dialogfenster vornimmt.

```

RefreshPlotDistance
NoChange = False
End Sub

```

Einstellung der Benutzerskalierung und der Plotposition

Die nun folgende Prozedur stellt die benutzerdefinierten Skalierungen und die Werte für die Plotposition ein und wird von einigen anderen Prozeduren aus aufgerufen.

```

Public Sub RefreshPlotDistance()
Dim Numerator As Double, Denominator As Double
On Local Error Resume Next

```

Als Erstes passen wir die Checkbox für die Zentrierung des Plots dem Wert der Eigenschaft *CenterPlot* an.

```
CenterPlot.Value = SelectedLayout.CenterPlot
```

Jetzt werden die beiden Skalierungswerte ausgelesen und an die entsprechenden Textboxen weitergegeben. Eine Formatierung können wir uns an dieser Stelle sparen, da die Skalierungswerte und damit die Anzahl der verwendeten Nachkommastellen von der Benutzereingabe abhängen.

```

SelectedLayout.GetCustomScale Numerator, Denominator
Numerator.Text = Numerator
Denominator.Text = Denominator

```

Bleibt noch die Position des Plots. Dazu wird der Inhalt der Eigenschaft *PlotOrigin* an eine Variable übergeben und anschließend in die entsprechenden Textfelder geschrieben. Die einzelnen Werte werden in die Systemeinheiten formatiert und mit vier Nachkommastellen versehen. Sie können statt der vier Nachkommastellen auch die Anzahl der Systemeinheiten verwenden, indem Sie die Systemvariable *LUNITS* abfragen.

```

PaperPos = SelectedLayout.PlotOrigin
XPosition.Text = Util.RealToString(PaperPos(0))
YPosition.Text = Util.RealToString(PaperPos(1))
End Sub

```

Ereignisprozeduren

Nachdem die Initialisierungsprozeduren beschrieben worden sind, folgt nun nur noch die Beschreibung der Prozeduren, die auf Aktionen des Benutzers reagieren.

Schaltfläche Fenster

Da der Benutzer den Plotbereich in der Zeichnung auswählen möchte, tun wir ihm den Gefallen und blenden das Dialogfenster aus, um die Auswahl vorzunehmen.

```

Private Sub CommandButton1_Click()
Dim LeftCorner, RightCorner
On Local Error Resume Next
Me.Hide
LeftCorner = Utility.GetPoint(, "linke untere Ecke wählen")
If IsEmpty(LeftCorner) = False Then
    RightCorner = Util.GetCorner(LeftCorner, _
        "rechte obere Ecke wählen")
    If IsEmpty(RightCorner) = False Then

```

Nachdem der Benutzer ein Fenster in der Zeichnung gewählt hat, übergeben wir diese Koordinaten an die Funktion *SetWindowToPlot*, die das gewählte Fenster im Layout registriert. Dabei müssen Sie beachten, dass diese Funktion lediglich 2D-Arrays verlangt, die Funktionen *GetPoint* und *GetCorner* aber 3D-Arrays zurückgeben. Danach kann die Eigenschaft *PlotType* auf *acWindow* umgestellt und außerdem die entsprechende Option aktiviert werden.

```

ReDim Preserve LeftCorner(1) As Double
ReDim Preserve RightCorner(1) As Double
SelectedLayout.SetWindowToPlot LeftCorner, RightCorner
SelectedLayout.PlotType = acWindow
PlotType5.Value = True

```

Da die Änderung dieser Eigenschaften Einfluss auf die Position des Plotbereichs hat, muss die Prozedur *RefreshPlotDistance* aufgerufen werden, damit die Werte der Plotposition im Dialogfenster aktualisiert werden.

```

RefreshPlotDistance
End If
End If

```

Nachdem alles erledigt ist, kann das Dialogfenster wieder gestartet werden.

```

Me.Show
End Sub

```

Auswahl eines Papierformats

Wenn der Benutzer ein neues Papierformat auswählt, müssen einige Einstellungen vorgenommen werden.

```

Private Sub FormatCombo_Click()
Dim MediaList As Variant
On Local Error Resume Next
If NoChange = True Then Exit Sub

```

Nachdem überprüft wurde, dass die Änderung beabsichtigt ist, lesen wir die Zeichnungsträger des aktuellen Geräts aus. Da die Eigenschaft *CanonicalMediaName* ein kanonisches Papierformat verlangt, müssen wir den Umweg über alle Papierformate nehmen. Wir durchlaufen also die Liste der verfügbaren Medien:

```

MediaList = SelectedLayout.GetCanonicalMediaNames
For i = 0 To UBound(MediaList)

```

Wenn eine Übereinstimmung des ausgewählten Formats mit einem konvertierten Mediennamen auftritt, wird dieses Medium an die Eigenschaft *CanonicalMediaName* übergeben. Da eine Änderung der Papiergröße vorgenommen wurde, rufen wir sicherheitshalber die Prozedur *SetPaperUnits* auf, damit die Anpassungen an dem Dialogfenster vorgenommen werden können.

```

If FormatCombo.Text = SelectedLayout.GetLocaleMediaName( _
    MediaList(i)) Then
    SelectedLayout.CanonicalMediaName = MediaList(i)
    SetPaperUnits
    Exit For
End If
Next
End Sub

```

Aktivierung des Hochformats

Wenn der Benutzer das Hochformat für die Ausrichtung des Blatts gewählt hat, müssen zwei Sachen beachtet werden: zum einen die Überprüfung der Option AUF DEM KOPF und zum anderen die Darstellung des korrekten Bilds für die Anzeige der Ausrichtung.

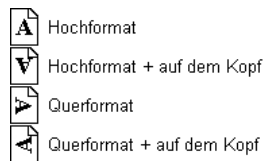


Abbildung 7.38: Symbole für die Darstellung der Papierausrichtung

Wenn der Benutzer die Option HOCHFORMAT aktiviert, kommen nur die ersten beiden Bilder in Frage. Diese sind dann abhängig von der Einstellung der Option AUF DEM KOPF. Dementsprechend wird das Bild zugeordnet.

```

Private Sub Hochformat_Click()
If UpSideDown.Value = True Then
    FormatImage.Picture = HForm2.Picture
Else
    FormatImage.Picture = HForm1.Picture
End If

```

Jetzt wird nur noch die entsprechende Ausrichtung der Eigenschaft *PlotRotation* zugeordnet.

```

If NoChange = False Then
    SelectedLayout.PlotRotation = IIf(UpSideDown.Value = True, _
        ac180degrees, ac0degrees)
End If

```

Da die Änderung der Eigenschaft wiederum die Größe des Plotbereichs beeinflusst, müssen die zugehörigen Felder über die Prozedur *RefreshPlotDistance* angepasst werden.

```

RefreshPlotDistance
End Sub

```

Aktivierung des Querformats

Über diese Prozedur brauchen wir nicht viel Worte zu verlieren, Sie können sich da an dem Ablauf der vorherigen Prozedur orientieren. Diese funktioniert in etwa wie die Einstellung des Hochformats, nur dass die eingestellten Winkel und Bilder andere sind.

```
Private Sub Querformat_Click()
If UpSideDown.Value = True Then
    FormatImage.Picture = QForm2.Picture
Else
    FormatImage.Picture = QForm1.Picture
End If
If NoChange = False Then
    SelectedLayout.PlotRotation = IIf(UpSideDown.Value = True, _
        ac270degrees, ac90degrees)
End If
RefreshPlotDistance
End Sub
```

Änderung der Einstellung auf dem Kopf

Da diese Einstellung bereits bei den Prozeduren für Hoch- und Querformat berücksichtigt wird, brauchen wir nur die Prozedur aufzurufen, die der Einstellung der entsprechenden Option entspricht.

```
Private Sub UpSideDown_Click()
If Hochformat.Value = True Then
    Hochformat_Click
Else
    Querformat_Click
End If
End Sub
```

Änderung der benutzerdefinierten Skalierung

Die nächsten beiden Prozeduren werden ausgelöst, wenn der Benutzer eine Änderung an der benutzerdefinierten Skalierungseinstellung vornimmt.

```
Private Sub Numerat_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Local Error Resume Next
```

Als Erstes werden die beiden Skalierungswerte in Zahlen umgewandelt, um diese anschließend der Anweisung *SetCustomScale* zu übergeben.

```
Numerator = Util.DistanceToReal(Numerat.Text)
Denominator = Util.DistanceToReal(Denominat.Text)
SelectedLayout.SetCustomScale Numerator, Denominator
```

Da jetzt ein benutzerdefinierter Wert eingestellt wurde, muss außerdem dafür gesorgt werden, dass die Auswahl der Standardskalierung angepasst wird.

```
ScaleCombo.ListIndex = 0
End Sub
```

Für das Textfeld, das den zweiten Skalierungswert enthält, könnten wir den Code der vorherigen Prozedur übernehmen oder einfach die vorherige Ereignisprozedur manuell auslösen.

```
Private Sub Denominat_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Numerat_BeforeUpdate Cancel
End Sub
```

Auswahl einer Plotkonfiguration

Nachdem der Benutzer eine Plotkonfiguration in der Combobox ausgewählt hat, wird ein Verweis auf diese Konfiguration in der Objektvariablen *PlotCfg* erstellt und anschließend mit der Anweisung *CopyFrom* an das Layout übertragen.

```
Private Sub PlotCfgCombo_Click()
Dim PlotCfg As AcadPlotConfiguration
On Local Error Resume Next
If NoChange = True And PlotCfgCombo.ListIndex = 0 Then Exit Sub
Set PlotCfg = ThisDrawing.PlotConfigurations(PlotCfgCombo.Text)
SelectedLayout.CopyFrom PlotCfg
```

Da sich nun die komplette Konfiguration des Layouts geändert haben kann, müssen die Inhalte der Steuerelemente ebenfalls aktualisiert werden. Dazu wird einfach die Ereignisprozedur der Combobox ausgeführt, die die Namen der Layouts enthält.

```
LayoutCombo_Click
End Sub
```

Auswahl des Plotbereichstyps

Kommen wir nun zur Einstellung des Plotbereichs. Je nachdem, welche Option der Benutzer gewählt hat, wird der entsprechende Wert an die Eigenschaft *PlotType* übertragen. Dabei muss nur darauf geachtet werden, dass die erste Option entweder die Limiten oder das Layout einstellt.

```
Private Sub PlotType1_Click()
If SelectedLayout.ModelType = True Then
    SelectedLayout.PlotType = acLimits
Else
    SelectedLayout.PlotType = acLayout
End If
RefreshPlotDistance
End Sub
Private Sub PlotType2_Click()
SelectedLayout.PlotType = acExtents
RefreshPlotDistance
End Sub
Private Sub PlotType3_Click()
SelectedLayout.PlotType = acDisplay
RefreshPlotDistance
End Sub
```

Bei der Auswahl einer Ansicht als Plotbereich muss zuerst die gewählte Ansicht eingestellt werden, da sonst die Übergabe des Werts *acView* fehlschlägt.

```
Private Sub PlotType4_Click()
SelectedLayout.ViewToPlot = ViewCombo.Text
```

```

SelectedLayout.PlotType = acView
RefreshPlotDistance
End Sub

```

Hat der Benutzer die Option FENSTER gewählt, muss zunächst die Voraussetzung für diese Option geprüft werden. Dazu werden die Koordinaten des aktuellen Fensters ausgelesen.

```

Private Sub PlotType5_Click()
On Local Error Resume Next
SelectedLayout.GetWindowToPlot LLeft, URight

```

Anschließend wird geprüft, ob die Koordinaten identisch sind. Ist dies der Fall, dann wurde noch kein Fenster ausgewählt, was umgehend nachgeholt wird. Dazu nutzen wir den Code der Schaltfläche FENSTER, die diese Funktion ja bereits ausführt. Andernfalls muss nur der Wert *acWindow* an die Eigenschaft *PlotType* übergeben werden.

```

If LLeft(0) = URight(0) And LLeft(1) = URight(1) Then
    CommandButton1_Click
Else
    SelectedLayout.PlotType = acWindow
End If
RefreshPlotDistance
End Sub

```

Die folgenden Ereignisprozeduren brauchen nicht weiter kommentiert zu werden. Sie schreiben lediglich den Wert der Checkboxes in die dazugehörige Eigenschaft.

```

Private Sub PlotHidden_Click()
SelectedLayout.PlotHidden = PlotHidden.Value
End Sub

Private Sub PlotViewportsFirst_Click()
SelectedLayout.PlotViewportsFirst = PlotViewportsFirst.Value
End Sub

Private Sub PlotWithLineweights_Click()
SelectedLayout.PlotWithLineweights = PlotWithLineweights.Value
End Sub

Private Sub PlotWithPlotStyles_Click()
SelectedLayout.PlotWithPlotStyles = PlotWithPlotStyles.Value
End Sub

Private Sub ScaleLineweights_Click()
SelectedLayout.ScaleLineweights = ScaleLineweights
End Sub

Private Sub CenterPlot_Click()
On Local Error Resume Next
SelectedLayout.CenterPlot = CenterPlot.Value
RefreshPlotDistance
End Sub

```

Auswahl einer Standardskalierung

Bei der Auswahl einer Standardskalierung müssen Sie nur eine Sache beachten. Da der erste Eintrag keiner Konstanten für die Standardskalierung entspricht und nur der Festlegung der benutzerdefinierten Skalierung dient, deaktiviert die Auswahl dieses Eintrags die Benutzung der Standardskalierung (*UseStandardScale*).

```
Private Sub ScaleCombo_Click()
If NoChange = True Then Exit Sub
If ScaleCombo.ListIndex = 0 Then
    SelectedLayout.UseStandardScale = False
Else
```

Für alle anderen Einträge der Combobox muss die Standardskalierung aktiviert und der ausgewählte ListIndex als Skalierungswert gesetzt werden.

```
    SelectedLayout.UseStandardScale = True
    SelectedLayout.StandardScale = ScaleCombo.ListIndex - 1
    RefreshPlotDistance
End If
End Sub
```

Anordnung der Layouts ändern

Im Textfeld TABPOSITION kann der Benutzer eine neue Position für den Layoutreiter festlegen. Folgender Code reagiert auf die Änderung des Eintrags.

```
Private Sub TabOrder_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Local Error Resume Next
```

Nachdem geprüft wurde, dass der Benutzer die Position eines Papierlayouts ändern will, muss nur die neue Position an die *TabOrder*-Eigenschaft des Layoutobjekts übergeben werden. Falls der Benutzer eine ungültige Position eingegeben hat, wird der Wert der Eigenschaft nicht geändert.

```
If SelectedLayout.ModelType = False Then
    SelectedLayout.TabOrder = CLng(TabOrder.Text)
End If
```

Deshalb wird die Position des Layouts danach wieder ausgelesen und in die Textbox geschrieben. Falls der Wert nicht korrekt war, steht nun die Originalposition im Textfeld.

```
TabOrder.Text = SelectedLayout.TabOrder
End Sub
```

Auswahl der Papiereinheiten

Die Auswahl aller drei Papiereinheiten verläuft nach dem gleichen Muster. Zuerst wird der entsprechende Wert an die *PaperUnits*-Eigenschaft übergeben und danach die Anweisung *SetPaperUnits* ausgeführt, deren Funktion bereits beschrieben wurde.

```
Private Sub UnitOption1_Click()
On Local Error Resume Next
SelectedLayout.PaperUnits = acInches
SetPaperUnits
End Sub

Private Sub UnitOption2_Click()
On Local Error Resume Next
SelectedLayout.PaperUnits = acMillimeters
SetPaperUnits
End Sub
```



```
Private Sub UnitOption3_Click()
On Local Error Resume Next
SelectedLayout.PaperUnits = acPixels
SetPaperUnits
End Sub
```

Änderung der Plotposition

Wenn der Benutzer die Koordinaten für die Plotposition ändert, müssen diese an die Eigenschaft *PlotOrigin* übergeben werden.

```
Private Sub XPosition_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
```

Zunächst deklarieren wir ein Array, das die beiden Werte aufnehmen kann.

```
Dim PaperPos(1) As Double
On Local Error Resume Next
```

Da die beiden Werte Zeichenfolgen sind, wandeln wir jeden in einen Streckenwert um und übergeben den X-Wert an das erste und den Y-Wert an das zweite Feld des Arrays.

```
PaperPos(0) = Util.DistanceToReal(XPosition.Text)
PaperPos(1) = Util.DistanceToReal(YPosition.Text)
```

Das so präparierte Array wird zum Schluss der Eigenschaft *PlotOrigins* zugewiesen.

```
SelectedLayout.PlotOrigin = PaperPos
RefreshPlotDistance
End Sub
```

Da der Code zur Änderung der Plotposition bereits komplett in der Ereignisprozedur des *XPosition*-Textfelds vorliegt, können wir diese Prozedur einfach manuell ausführen, damit auch auf die Änderung der Y-Koordinate reagiert werden kann.

```
Private Sub YPosition_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
XPosition_BeforeUpdate Cancel
End Sub
```

Die Schaltfläche „Schließen“

Diese Ereignisprozedur führt lediglich eine Funktion aus: das Ausblenden des Formulars und damit die Beendigung des Programms.

```
Private Sub CommandButton2_Click()
Me.Hide
End Sub
```

7.7 Verwaltung der Textstile

In einer AutoCAD-Zeichnung ist jedem grafisch dargestellten Text, sei es ein einzeliger oder ein MText, ein Stil zugeordnet. AutoCAD bietet die Textstile, um dem Text Schrift, Größe, Winkel, Ausrichtung und andere Eigenschaften zuzuordnen. Wie Sie einen Stil verwenden bzw. ändern oder einen neuen Stil erstellen, das erfahren Sie in diesem Abschnitt.

7.7.1 Verwaltung der Stile in der Textstilauflistung

Die Textstilauflistung beinhaltet alle Textstilobjekte, die in der Zeichnung definiert wurden. Zugriff auf diese Auflistung erhalten Sie, indem Sie das *TextStyles*-Objekt der entsprechenden Zeichnung abrufen. Für das aktuelle Dokument sähe das wie folgt aus:

```
Set AllTextStyles = ThisDrawing.TextStyles
```

Für das n-te Dokument der Zeichnung:

```
Set AllTextStyles = Documents(n).TextStyles
```

Sie brauchen die Auflistung nicht unbedingt an eine Objektvariable zu übergeben, Sie können auch direkt mit *TextStyles* arbeiten. Das Objekt, das der Textauflistung entspricht, ist in jedem Fall vom Datentyp *AcadTextStyle*, der folgende Eigenschaften und Methoden beinhaltet.

Textstile hinzufügen/vorhandene Textstile abrufen

Einzelne Textstile werden, genau wie Layer oder Linientypen, mittels der *Add*-Funktion zur Auflistung hinzugefügt. Sie müssen lediglich den Namen des neuen Textstils festlegen und erhalten als Ergebnis der Funktion ein Objekt des Datentyps *AcadTextStyle*. Über dieses Objekt können Sie dann alle weiteren Eigenschaften, wie Texthöhe oder Font, zuordnen.

```
Dim NewTextStyle As AcadTextStyle  
Set NewTextStyle = ThisDrawing.TextStyles.Add("ISO8")
```

Wir haben mit dieser Codezeile den Textstil „ISO8“ erzeugt und gleichzeitig einen Verweis auf dieses Objekt an die Variable *NewTextStyle* übergeben. Sie könnten nun diese Variable verwenden, um die Eigenschaften des Textstils anzupassen, z.B. die Texthöhe des Textstils.

```
NewTextStyle.Height = 2.5
```

Sollten Sie versuchen, einen Textstil mit einem Namen hinzuzufügen, der bereits durch einen Textstil belegt ist, löst AutoCAD einen Fehler aus.

Um einen vorhandenen Textstil aus der Auflistung auszuwählen, müssen Sie die *Item*-Funktion der Klasse *AcadTextStyle* verwenden:

```
Dim CrTextStyle As AcadTextStyle  
Set CrTextStyle = ThisDrawing.TextStyles.Item(n)
```

oder:

```
Set CrTextStyle = ThisDrawing.TextStyles(n)
```

Sie können natürlich auch beim Abrufen von Textstilen den Namen des Textstils statt des Index als Parameter der *Item*-Funktion verwenden.

```
Dim StyleName As String
Dim CrTStyle As AcadTextStyle
StyleName = "Standard"
Set CrTStyle = ThisDrawing.TextStyles(StyleName)
```

Egal, wie Sie einen Textstil abrufen, als Ergebnis erhalten Sie eine Objektvariable des Typs *AcadTextStyle*, die weiterverarbeitet werden kann.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung sämtlicher Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.32: Eigenschaften der Textstilaufistung

Prozedur	Beschreibung
<i>Add</i>	Diese Anweisung fügt einen Textstil der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.33: Prozeduren der Textstilaufistung

7.7.2 Einzelne Textstile verarbeiten

Da wir nun wissen, wo die einzelnen Textstile zu finden sind und wie sie erstellt bzw. abgerufen werden, bleibt nur noch zu klären, wie man die Eigenschaften einzelner Textstile bearbeitet und was die Eigenschaften und Methoden eines Textstils im Einzelnen bewirken.

Da ein Textstil sowohl als Normalschrift als auch als BigFont-Schrift angezeigt werden kann, können auch beide Stildateien in verschiedenen Eigenschaften gespeichert werden. In der *BigFontFile*-Eigenschaft ist der Name der BigFont-Datei gespeichert, die verwendet wird, falls der Modus für BigFonts aktiviert wurde. In der *fontFile*-Eigenschaft wird der Name der normalen Schriftartendatei verwaltet. Ändern Sie den Inhalt der Eigenschaft, ändert sich auch das Aussehen der Texte, die diesen Textstil in der *StyleName*-Eigenschaft verwenden. Während *fontFile* neben *shx*- auch *ttf*-Dateien akzeptiert, können Sie in der *BigFontFile*-Eigenschaft nur *shx*-Dateien festlegen, die BigFonts darstellen.

Vorgabeeigenschaften festlegen

Mit den folgenden Eigenschaften und Prozeduren können Sie Vorgabewerte definieren, um diese dann den Texten zuzuweisen, die mit dem entsprechenden Textstil erzeugt werden.

Die *Height*-Eigenschaft legt die Vorgabehöhe für neue Texte fest. Logischerweise muss der Wert der Höhe immer eine Zahl ≥ 0 sein. Wenn Sie als Texthöhe 0 eintragen, fordert AutoCAD bei der Erstellung von Texten mit diesem Stil zur Eingabe einer Texthöhe auf.

```
ActTStyle.Height = 2.5
```

Die zuletzt für Texte des entsprechenden Textstils verwendete Höhe ist in der Eigenschaft *LastHeight* gespeichert.

Der Vorgabeneigungswinkel wird in der *ObliqueAngle*-Eigenschaft gespeichert.

In der *Width*-Eigenschaft wird der Vorgabebreitenfaktor des Textstils verwaltet. Wenn Sie einen neuen Wert für diese Eigenschaft festlegen wollen, müssen Sie darauf achten, dass dieser immer größer 0 ist.

```
ActTStyle.Width = 1.1
```

Mit der *TextGenerationFlag*-Eigenschaft kann die Vorgabe für die Texteeigenschaften RÜCKWÄRTS und AUF DEM KOPF festgelegt werden. Um keine der beide zu setzen, wählen Sie den Wert 0. Ansonsten können Sie die Konstanten *acTextFlagBackward* (aktiviert den Modus RÜCKWÄRTS) und *acTextFlagUpsideDown* (aktiviert den Modus AUF DEM KOPF) verwenden. Wenn Sie beide Eigenschaften aktivieren wollen, müssen Sie die Konstanten kombinieren.

```
ActTStyle.TextGenerationFlag = acTextFlagBackward + _
    acTextFlagUpsideDown
```

Über die Anweisung *GetFont* können verschiedene Zusatzinformationen zum Textstil abgerufen werden. Die Anweisung erfordert einige Variablen, in die die Informationen geschrieben werden.

```
TStyle.GetFont TypeFace, Bold, Italic, Charset, PitchAndFamily
```

In der Tabelle 7.34 erhalten Sie eine Übersicht über die Bedeutung und die möglichen Werte, die die einzelnen Parameter enthalten können.

Parameter	Beschreibung																			
TypeFace (String)	Dieser Parameter enthält den Fontnamen des Textstilobjekts.																			
Bold (Boolean)	Zeigt an oder legt fest, dass Texte dieses Textstils fett geschrieben werden sollen. Mögliche Werte des Parameters sind TRUE (fett) oder FALSE (normal).																			
Italic (Boolean)	Zeigt an oder legt fest, dass Texte dieses Textstils kursiv geschrieben werden sollen. Mögliche Werte des Parameters sind TRUE (kursiv) oder FALSE (normal). Der Parameter Kursiv ist unabhängig von der Einstellung ObliqueAngle.																			
Charset (Long)	Mit diesem Parameter wird der Zeichensatz ausgegeben. Für die Zahlenwerte, die Sie dabei erhalten, sind in der Windows-API folgende Konstanten definiert.																			
<table><tr><th>Konstante</th><th>Wert</th><th>Beschreibung</th></tr><tr><td>ANSI_CHARSET</td><td>0</td><td>ANSI-Zeichensatz</td></tr><tr><td>DEFAULT_CHARSET</td><td>1</td><td>Standardzeichensatz der Benutzeroberfläche</td></tr><tr><td>SYMBOL_CHARSET</td><td>2</td><td>Symbolzeichensatz</td></tr><tr><td>SHIFTJIS_CHARSET</td><td>128</td><td>Japanese Microsoft-Standardzeichensatz</td></tr><tr><td>OEM_CHARSET</td><td>255</td><td>Erweiterter Zeichensatz, der normalerweise von DOS-Anwendungen angezeigt wird</td></tr></table>			Konstante	Wert	Beschreibung	ANSI_CHARSET	0	ANSI-Zeichensatz	DEFAULT_CHARSET	1	Standardzeichensatz der Benutzeroberfläche	SYMBOL_CHARSET	2	Symbolzeichensatz	SHIFTJIS_CHARSET	128	Japanese Microsoft-Standardzeichensatz	OEM_CHARSET	255	Erweiterter Zeichensatz, der normalerweise von DOS-Anwendungen angezeigt wird
Konstante	Wert	Beschreibung																		
ANSI_CHARSET	0	ANSI-Zeichensatz																		
DEFAULT_CHARSET	1	Standardzeichensatz der Benutzeroberfläche																		
SYMBOL_CHARSET	2	Symbolzeichensatz																		
SHIFTJIS_CHARSET	128	Japanese Microsoft-Standardzeichensatz																		
OEM_CHARSET	255	Erweiterter Zeichensatz, der normalerweise von DOS-Anwendungen angezeigt wird																		
Public Const ANSI_CHARSET = 0 Public Const DEFAULT_CHARSET = 1 Public Const SYMBOL_CHARSET = 2 Public Const SHIFTJIS_CHARSET = 128 Public Const OEM_CHARSET = 255																				
PitchAndFamily (Long)	Der Wert, den Sie mit diesem Parameter erhalten, enthält gleichzeitig drei verschiedene Informationen, die durch folgende Konstanten definiert sind.																			
Zeichengröße (Pitch) Public Const DEFAULT_PITCH = 0 Public Const FIXED_PITCH = 1 Public Const VARIABLE_PITCH = 2																				
Family Public Const FF_DONTCARE = 0 Public Const FF_ROMAN = 16 Public Const FF_SWISS = 32 Public Const FF_MODERN = 48 Public Const FF_SCRIPT = 64 Public Const FF_DECORATIVE = 80																				
TrueType-Konstante Public Const TMPF_TRUETYPE = &H4																				

Tabelle 7.34: Parameter der *GetFont*- und *SetFont*-Anweisung

Parameter	Beschreibung
<i>PitchAndFamily</i> (Long)	Da in einer Variablen diese drei Konstantenauflistungen enthalten sein können, sind mitunter Kombinationen der einzelnen Konstanten vorhanden. Sie können die Werte dann mit dem AND-Operator vergleichen: If (PitchFam AND FIXED_PITCH) = FIXED_PITCH Then ' Variable enthält den Konstantenwert Fixed_pitch End If oder If (PitchFam AND FF_MODERN) = FF_MODERN Then ' Variable enthält den Konstantenwert FF_MODERN End If

Tabelle 7.34: Parameter der GetFont- und SetFont-Anweisung (Forts.)

Das Gegenstück zu *GetFont* ist die *SetFont*-Anweisung. Genau wie beim Lesen der Definitionsdaten, müssen Sie auch hier einige Variablen zur Übergabe vorbereiten. In diesem Fall werden allerdings die Daten an das Textstilobjekt übergeben, was bedeutet, dass Sie die Variablen vor Aufruf der Funktion mit den entsprechenden Werten füllen müssen. Die Bedeutung und möglichen Werte der einzelnen Parameter können Sie wiederum der Tabelle 7.34 entnehmen. Der Aufruf der Funktion lautet dann wie folgt:

```
Dim TypeFace As String, Bold As Boolean, Italic As Boolean
Dim Charset As Long, PitchAndFamily As Long
TStyle.SetFont TypeFace, Bold, Italic, Charset, PitchAndFamily
```

Die Variablen müssen den verlangten Datentypen der Prozedur entsprechen, weshalb Sie diese vor Aufruf der Funktion deklarieren müssen.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Name</i> (schreibgeschützt)	Legt den Namen des Textstils fest. Im Gegensatz zu anderen Auflistungen können Sie den Namen des Textstils nur bei dessen Erzeugung über die <i>Add</i> -Funktion der Auflistung festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.

Tabelle 7.35: Eigenschaften des Textstilobjekts

Eigenschaft	Beschreibung
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.35: Eigenschaften des Textstilobjekts (Forts.)

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.36: Weitere Prozeduren des Textstilobjekts

Anwendungsbeispiel

(Beispielprojekt: 07\AcadTextStyles.dvb)

Nachdem wir uns mit der Theorie zum Thema Textstile beschäftigt haben, folgt nun ein praktisches Anwendungsbeispiel. Sinn und Zweck des Projekts wird es sein, ein Programm zu erstellen, mit dem sich die Textstile verwalten lassen.

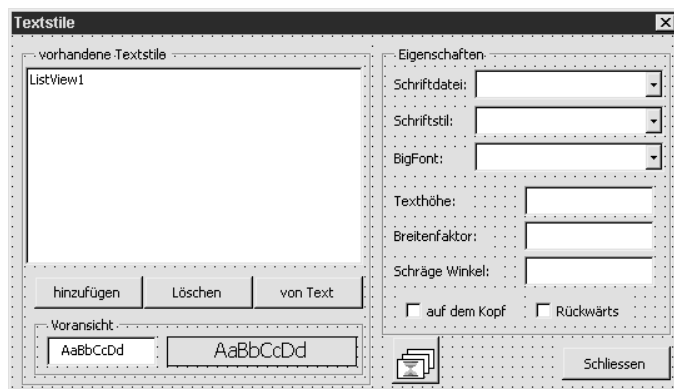


Abbildung 7.39: Formularentwurf des Textstilprojekts

Startprozedur

Beginnen wollen wir zunächst mit der Startprozedur des Projekts, das im Klassenmodul *ThisDrawing* abgelegt wurde. Wie Sie in der Prozedur erkennen können, werden nur verschiedene Formularvariablen und -prozeduren angesprochen bzw. initialisiert, bevor das Formular gestartet wird.

```

Public Sub showTStyles()
Set TextStyleFrm.Util = New MyUtility
TextStyleFrm.initTStyleList
TextStyleFrm.NoChange = False
TextStyleFrm.FontChanged = False
TextStyleFrm.show
End Sub

```

Initialisierungsprozedur

Kommen wir nun zum Hauptteil des Programms, dem Formularcode, der sich zum größten Teil wieder aus Ereignisprozeduren der einzelnen Steuerelemente zusammensetzt. Als Erstes sind dem Formularcode jedoch zwei Variablen hinzugefügt worden:

```

Public NoChange As Boolean
Public FontChanged As Boolean

```

Diese Variablen werden in den verschiedensten Prozeduren auf den aktuellen Inhalt geprüft, um bestimmte Aktion auszuschließen oder auszuführen, aber dazu später mehr. Die erste Prozedur dient der Initialisierung einiger Listfelder und wurde bereits durch die Startprozedur aufgerufen, bevor das Formular gestartet wurde.

```

Public Sub initTStyleList()
Dim TStyle As AcadTextStyle
Dim LV As MSComctlLib.ListItem
Dim CrFonts As ListFonts.Fonts
On Local Error Resume Next

```

Bei der Deklaration der Variablen wird Ihnen bereits der Datentyp *ListFonts.Fonts* aufgefallen sein. Dieser Datentyp entspringt der Laufzeitbibliothek *ListFonts.dll*. Die Entwicklung dieser Bibliothek wurde notwendig, da VBA keine Fontauflistung anbietet. Aber dazu kommen wir gleich. Zunächst wird erst einmal das *ImageList*-Steuerelement, welches zwei Symbole aufgenommen hat, *ListView1* zugeordnet, damit eben diese Symbole später in der Liste angezeigt werden können.

```

Set ListView1.Icons = ImageList1
Set ListView1.SmallIcons = ImageList1
ListView1.ListItems.Clear

```

Nachdem *Listview1* von Elementen aus vorherigen Aufrufen bereinigt wurde, können alle Textstile der Auflistung nacheinander abgerufen und Textstilname, Vorgabehöhe sowie Fontname in die Liste eingetragen werden. Zusätzlich kommen nun die zwei Symbole zum Einsatz. Das erste wird an den Zeilenanfang gesetzt, wenn es sich um einen *TrueTypeFont* handelt, das zweite bei einer *shx*-Datei.

```

For Each TStyle In ThisDrawing.TextStyles
Set LV = ListView1.ListItems.Add(, , TStyle.Name)
LV.ListSubItems.Add , , ThisDrawing.Utility.RealToString( _
TStyle.Height, acDefaultUnits, 3)
LV.ListSubItems.Add , , TStyle.fontFile
If LCase(Right(TStyle.fontFile, 4)) = ".shx" Then
LV.SmallIcon = 2
Else
LV.SmallIcon = 1
End If
Next

```


Die restlichen Eigenschaften werden erst bei Auswahl eines Textstils aus der Liste ermittelt, weshalb dieser Teil hiermit abgeschlossen ist. Der nächste Schritt wäre nun die Füllung der Comboboxen mit den vorhandenen Textstilen des Systems. Da VBA, wie gerade erwähnt, keine Fontauflistung anbietet, müssen wir uns selbst helfen. Die Aufgabe der Zusammenstellung der einzelnen Fontbereiche übernimmt diese Bibliothek. Der erste Schritt besteht in der Erstellung eines neuen Objekts der Klasse *Fonts*, die die benötigten Funktionen enthält. Wichtig hierbei ist die Verwendung des Schlüsselworts *New*.

```
Set CrFonts = New ListFonts.Fonts
```

Da das Objekt jetzt erstellt wurde, können die einzelnen Funktionen aufgerufen werden, zunächst *GetScreenFontsList*. Diese liest alle Schriftarten aus dem Windows-System aus und zusätzlich die *shx*-Fonts, die sich im Font-Verzeichnis von AutoCAD befinden. Ignoriert werden allerdings alle BigFonts von AutoCAD, da diese in einer separaten Combobox aufgeführt werden.

```
lfont = CrFonts.GetScreenFontsList(lfAllFont, _  
    ThisDrawing.Application.Path & "\\Fonts")
```

Nachdem die Funktion ein Array mit den einzelnen Textarten erstellt hat, können diese an die entsprechende Combobox übergeben werden.

```
ComboBox1.Clear  
For i = 0 To UBound(lfont)  
    ComboBox1.AddItem lfont(i)  
Next
```

Füllen wir nun noch die Schriftstil-Combobox mit den einzelnen Möglichkeiten, die einer Schriftart zugeordnet werden können.

```
ComboBox2.Clear  
ComboBox2.AddItem "Fett"  
ComboBox2.AddItem "Fett kursiv"  
ComboBox2.AddItem "Kursiv"  
ComboBox2.AddItem "Normal"
```

Was wir nun noch brauchen, sind die BigFonts, die ebenfalls in einer Combobox abgelegt werden. Hierfür stellt uns das Objekt *CrFonts* eine gesonderte Funktion zur Verfügung. Sie müssen dieser nur das Verzeichnis mitteilen, in dem sich diese Fonts befinden, und als Ergebnis erhalten Sie ein Array, gefüllt mit allen BigFonts des Verzeichnisses. Diese Liste kann dann in die *ComboBox3* übertragen werden.

```
lfont = CrFonts.GetBigFontsList(Application.Path & "\\Fonts")  
ComboBox3.Clear  
For i = 0 To UBound(lfont)  
    ComboBox3.AddItem lfont(i)  
Next  
ComboBox3.ListIndex = 0
```

Abschließend können wir noch den ersten Eintrag des *ListView*-Steuerelements aktivieren, damit bereits eine Auswahl automatisch vorgenommen wird.

```
ListView1.SelectedItem = ListView1.ListItems(1)  
ListView1_ItemClick ListView1.SelectedItem  
End Sub
```

Ereignisprozeduren

Auswahl eines Listview-Eintrags

Diese Prozedur reagiert auf die Auswahl eines Eintrags im *ListView*-Steuerelement durch den Benutzer.

```
Private Sub ListView1_ItemClick(ByVal Item As _
MSComctlLib.ListItem)
Dim TStyle As AcadTextStyle
Dim TypeFace As String, bold As Boolean, italic As Boolean
Dim Charset As Long, pitchFam As Long
On Local Error Resume Next
```

Da es möglich ist, mehrere Zeilen zu markieren, wurde eine Variable eingebaut, die den Status für die Änderungen des Textstils setzen soll. Den Wert dieser Variablen setzen wir zu Beginn der Prozedur auf den Wert *TRUE*. Den Effekt, den dieser Vorgang hat, werden Sie noch im Verlauf der anderen Prozeduren erkennen.

```
NoChange = True
```

Als Nächstes wird ein Verweis auf den gewählten Textstil erstellt. Den Namen des Textstils erhalten wir über die Objektvariable *Item*, die durch die Prozedur übergeben wurde und den gewählten Eintrag enthält. Da nun das Textstilobjekt ermittelt wurde, können die einzelnen Eigenschaften an das jeweilige Steuerelement übergeben werden.

```
Set TStyle = ThisDrawing.TextStyles(Item.Text)
ComboBox1.Text = TStyle.fontFile
TextBox1.Text = ThisDrawing.Utility.RealToString(TStyle.Height, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
TextBox2.Text = ThisDrawing.Utility.RealToString(TStyle.Width, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
TextBox3.Text = ThisDrawing.Utility.AngleToString( _
    TStyle.ObliqueAngle, ThisDrawing.GetVariable("AUNITS"), _
    ThisDrawing.GetVariable("AUPREC"))
CheckBox1.Value = IIf((TStyle.TextGenerationFlag And _
    acTextFlagUpsideDown) = acTextFlagUpsideDown, 1, 0)
CheckBox2.Value = IIf((TStyle.TextGenerationFlag And _
    acTextFlagBackward) = acTextFlagBackward, 1, 0)
```

Da die Eigenschaften *Bold* und *Italic* in der Klasse *TextStyle* nicht definiert wurden, müssen wir auf die Zusatzinformationen zurückgreifen, die uns die Funktion *GetFont* liefert.

```
TStyle.GetFont TypeFace, bold, italic, Charset, pitchFam
```

Die Variablen *Bold* und *Italic* enthalten nun die aktuellen Einstellungen des Textstils und wir können mithilfe dieser Variablen die aktuelle Einstellung der Combobox *Schriftstil* setzen. Dazu wird der Wert der Variablen kontrolliert und der jeweils passende *ListIndex* gesetzt.

```
If bold = True And italic = True Then
    ComboBox2.ListIndex = 1
ElseIf bold = True Then
    ComboBox2.ListIndex = 0
ElseIf italic = True Then
    ComboBox2.ListIndex = 2
```

```
Else
    ComboBox2.ListIndex = 3
End If
```

Jetzt werden noch die Eigenschaften für das Steuerelement angepasst, das der Voransicht des Textstils dient. Dazu müssen die Werte *Bold* und *Italic* des Fontobjekts, das die Schriftdarstellung des Labels steuert, mit den ausgelesenen Werten des Textstils aktualisiert werden. Sie könnten dies direkt über das Fontobjekt bewerkstelligen:

```
Label3.Font.Bold = bold
Label3.Font.Italic = Italic
```

oder die verborgenen Eigenschaften *Fontbold* und *Fontitalic* benutzen:

```
Label3.FONTBOLD = bold
Label3.FONTITALIC = italic
```

Beide Vorgehensweisen würden zum selben Ergebnis führen.

Bleibt abschließend nur noch die Aktivierung bzw. Deaktivierung der Comboboxen für Schriftstil und BigFont. Die *Combobox2* soll sich nur auswählen lassen, wenn es sich bei der Fontdatei nicht um eine *shx*-Datei handelt. Wenn andererseits die Fontdatei eine *shx*-Datei ist, muss sich die *ComboBox3* zur Aufnahme der BigFonts auswählen lassen.

```
If InStr(1, TStyle.fontFile, ".shx") = 0 Then
    ComboBox3.Enabled = False
    ComboBox2.Enabled = True
Else
    ComboBox3.Enabled = True
    ComboBox2.Enabled = False
End If
```

Nachdem alle notwendigen Eigenschaften an die jeweiligen Steuerelemente übertragen wurden, bleibt nur noch, die Variable *NoChange* auf den Wert *FALSE* zu setzen, um wieder Änderungen am Textstil durch den Benutzer zu ermöglichen.

```
NoChange = False
End Sub
```

Änderung der Einstellungen „auf dem Kopf“ und „Rückwärts“

Durch die Änderung des Status der CheckBoxen (angehakt oder nicht angehakt) wird das *Click*-Ereignis der jeweiligen CheckBox ausgelöst. Wir müssen nun auf diese Änderung reagieren und die Auswahl auf die selektierten Textstile übertragen.

```
Private Sub CheckBox1_Click()
    Dim TStyle As AcadTextStyle
    On Local Error Resume Next
    If NoChange = True Then Exit Sub
```

Nachdem geprüft wurde, ob das Ändern des Status der CheckBoxen durch den Benutzer vorgenommen wurde, kann das gesamte *ListView*-Steuerelement Zeile für Zeile durchlaufen werden. Dabei wird mittels der *Selected*-Eigenschaft geprüft, ob die jeweilige Zeile markiert wurde.

```

For i = 1 To ListView1.ListItems.Count
  If ListView1.ListItems(i).Selected = True Then
    Set TStyle = ThisDrawing.TextStyles( _
      ListView1.ListItems(i).Text)

```

Ist das der Fall, wird ein Verweis auf den Textstil, der in dieser Zeile abgebildet ist, in der Variablen *TStyle* erstellt und anschließend die *TextGenerationFlag*-Eigenschaft zur Ermittlung der aktuellen Einstellung der Eigenschaft *Rückwärts* verwendet.

```

  If (TStyle.TextGenerationFlag And acTextFlagBackward) = _
    acTextFlagBackward Then
    TStyle.TextGenerationFlag = acTextFlagBackward
  Else
    TStyle.TextGenerationFlag = 0
  End If

```

Nachdem mit dieser Methode die Einstellung AUF DEM KOPF extrahiert wurde, kann die neue Einstellung an *TextGenerationFlag* übergeben werden.

```

  If CheckBox1.Value = True Then TStyle.TextGenerationFlag = _
    TStyle.TextGenerationFlag + acTextFlagUpsideDown
End If
Next
FontChanged = True
End Sub

```

Für die zweite CheckBox gilt dieselbe Vorgehensweise, allerdings mit dem Unterschied, dass hier RÜCKWÄRTS extrahiert wird, um die aktuelle Einstellung von AUF DEM KOPF zu erhalten und den neuen Wert für RÜCKWÄRTS zu übergeben.

```

Private Sub CheckBox2_Click()
  Dim TStyle As AcadTextStyle
  On Local Error Resume Next
  If NoChange = True Then Exit Sub
  For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
      Set TStyle = ThisDrawing.TextStyles( _
        ListView1.ListItems(i).Text)
      If (TStyle.TextGenerationFlag And acTextFlagUpsideDown) = _
        acTextFlagUpsideDown Then
        TStyle.TextGenerationFlag = acTextFlagUpsideDown
      Else
        TStyle.TextGenerationFlag = acTextFlagUpsideDown
      End If
      If CheckBox2.Value = True Then TStyle.TextGenerationFlag = _
        TStyle.TextGenerationFlag + acTextFlagBackward
    End If
  Next
  FontChanged = True
End Sub

```

Änderung der Fontname-Eigenschaft

Wenn der Benutzer einen Schriftnamen aus der *ComboBox1* auswählt, müssen wir diesen Namen der *Fontname*-Eigenschaft der selektierten Textstile anpassen. Dazu brauchen wir nur den ausgewählten Namen zu übernehmen.

```
Private Sub ComboBox1_Click()
Dim TStyle As AcadTextStyle
On Local Error Resume Next
```

Die erste Aktion, die in dieser Prozedur durchgeführt wird, hat mit der Übernahme des Fontnamens noch nichts zu tun. Vielmehr wird hier nur geprüft, welchen Fonttyp (TrueType oder SHX) der Benutzer gewählt hat, und je nach Auswahl die Schriftstil-Combobox und die Big-Font-Combobox für Benutzereingaben gesperrt bzw. entsperrt.

```
If InStr(1, ComboBox1.Text, ".shx") = 0 Then
    ComboBox3.Enabled = False
    ComboBox2.Enabled = True
Else
    ComboBox2.Enabled = False
    CheckBox3.Enabled = True
End If
```

Zusätzlich übergeben wir den Fontnamen an die *Fontname*-Eigenschaft des *Label*-Steuerelements, die die Voransicht des Textstils übernimmt. Auch hier gibt es wieder die beiden Möglichkeiten

```
Label3.Font.Name = ComboBox1.Text
```

oder

```
Label3.FONTNAME = ComboBox1.Text
```

wobei *Fontname* wieder eine verborgene Eigenschaft darstellt. Was nun noch fehlt, ist das Setzen der *Fontfile*-Eigenschaft der ausgewählten Textstile. Bevor dieser Vorgang ausgeführt wird, erfolgt erneute eine Prüfung, ob die Auswahl des neuen Fontfile durch den Benutzer erfolgte.

```
If NoChange = True Then Exit Sub
```

Ist dies der Fall, wird nach üblicher Vorgehensweise die Liste der Textstile einzeln durchlaufen. Wurde ein Textstil selektiert, wird ein Verweis auf den Textstil in der Objektvariable *TStyle* erstellt, der dann wiederum der ausgewählte Text der Combobox als Fontfile zugeordnet wird.

```
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        Set TStyle = ThisDrawing.TextStyles( _
            ListView1.ListItems(i).Text)
        TStyle.fontFile = ComboBox1.Text
```

Nach der Aktualisierung der Eigenschaft muss das *ListView*-Steuerelement noch angepasst werden. Zunächst ändert sich der Text in der dritten Spalte, die ja die Eigenschaft *FontName* anzeigt. Also übergeben wir den Fontfile-Namen an die *Text*-Eigenschaft des zweiten Unterelements der jeweiligen Zeile. Da der neue Fontfile ja wieder eine TrueType- oder *shx*-Datei sein kann, müssen wir auch dies prüfen und das entsprechende Icon zuordnen, damit die Darstellung wieder stimmt.

```

        ListView1.ListItems(i).ListSubItems(2).Text = _
            TStyle.fontFile
    If LCase(Right(TStyle.fontFile, 4)) = ".shx" Then
        ListView1.ListItems(i).SmallIcon = 2
    Else
        ListView1.ListItems(i).SmallIcon = 1
    End If
End If
Next

```

Jetzt wird wieder die Variable auf *TRUE* gesetzt, die uns einen Hinweis darauf liefert, dass eine Änderung an mindestens einem Textstil vorgenommen worden ist.

```

FontChanged = True
End Sub

```

Änderung des Schriftstils

Als Nächstes befassen wir uns mit der Auswahl eines neuen Schriftstils für die ausgewählten Textstile durch den Benutzer. Dieser hat die Möglichkeit, vier verschiedene Varianten auszuwählen.

```

Private Sub ComboBox2_Click()
    Dim TStyle As AcadTextStyle
    Dim TypeFace As String, bold As Boolean, italic As Boolean
    Dim Charset As Long, pitchFam As Long
    On Local Error Resume Next
    If NoChange = True Then Exit Sub
    For i = 1 To ListView1.ListItems.Count
        If ListView1.ListItems(i).Selected = True Then
            Set TStyle = ThisDrawing.TextStyles( _
                ListView1.ListItems(i).Text)

```

Nach der ganzen Vorarbeit, wie z.B. die Erstellung eines Verweises auf den jeweiligen Textstil, müssen wir prüfen, ob diesem Textstil kein *shx*-Font zugeordnet wurde. Dazu nutzen wir die Eigenschaft *FontFile*. Enthält diese einen Namen, der mit der Endung *shx* endet, gibt die *InStr()*-Funktion in jedem Fall einen Wert größer als 0 zurück.

```

    If InStr(1, TStyle.fontFile, ".shx") = 0 Then

```

Andernfalls können wir mit der Zuordnung der Eigenschaften fortfahren. Zunächst werden die Zusatzinformationen des Textstils ausgelesen und den jeweiligen Variablen zugeordnet. Da wir nur die beiden Variablen *Bold* und *Italic* gegebenenfalls ändern werden, müssen die anderen Eigenschaften zwischengespeichert und am Ende wieder zurückgeschrieben werden.

```

        TStyle.GetFont TypeFace, bold, italic, Charset, pitchFam

```

Was nun folgt, ist die Zuordnung der korrekten Werte für die Variablen *Bold* und *Italic*. Hierzu nutzen wir eine *Select Case*-Anweisung, da ja aus vier verschiedenen Möglichkeiten ausgewählt werden muss. Sie könnten auch eine *If...End If*-Variante anwenden, ich halte *Select Case* aber für praktikabler.

```

        Select Case ComboBox2.ListIndex

```

Das erste Element der Liste wurde gewählt (fett, aber nicht kursiv), weshalb *Bold* den Wert *TRUE* und *Italic* den Wert *FALSE* erhalten.

```
Case 0
    Bold = True
    Italic = False
```

Das zweite Element der Liste wurde gewählt (fett und kursiv), weshalb sowohl *Bold* als auch *Italic* den Wert *TRUE* erhalten.

```
Case 1
    Bold = True
    Italic = True
```

Das erste Element der Liste wurde gewählt (nicht fett, aber kursiv). *Bold* erhält also den Wert *FALSE* und *Italic* den Wert *TRUE*.

```
Case 2
    Italic = True
    Bold = False
```

Wir könnten zwar *Case 3* schreiben, da es aber keine weiteren Einträge in der Liste gibt, ist dies der letzte mögliche Fall. Der Text soll normal geschrieben werden:

```
Case Else
    Italic = False
    Bold = False
End Select
```

Abschließend müssen nur noch die neuen Werte an den Textstil übertragen werden. Nutzen wir also die mit *GetFont* ermittelten Werte und schreiben diese mit *SetFont* in den Textstil zurück.

```
TStyle.SetFont Typeface, bold, italic, Charset, pitchFam
End If
End If
Next
```

Zum Abschluss wieder der Hinweis, dass sich ein Textstil verändert hat.

```
FontChanged = True
End Sub
```

Änderung der BigFont-Datei

Zu diesem Vorgang gibt es nicht allzu viel zu sagen, nur so viel, dass die Eigenschaft in jedem Fall geschrieben werden kann, da sie immer belegt ist. AutoCAD weiß schon, wann sie benutzt oder ignoriert werden muss.

```
Private Sub ComboBox3_Click()
Dim TStyle As AcadTextStyle
On Local Error Resume Next
If NoChange = True Then Exit Sub
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        Set TStyle = ThisDrawing.TextStyles( _
```

```

        ListView1.ListItems(i).Text)
    TStyle.BigFontFile = ComboBox3.Text
End If
Next
FontChanged = True
End Sub

```

Neuen Textstil hinzufügen

Wie ein neuer Textstil hinzugefügt wird, haben Sie ja bereits in diesem Kapitel erfahren. Es folgt nun die praktische Anwendung in der folgenden Ereignisprozedur.

```

Private Sub CommandButton1_Click()
    Dim NewTStyle As String
    Dim TStyle As AcadTextStyle
    On Local Error Resume Next

```

Als Erstes erfolgt die Abfrage des Textstilnamens.

```

NewTStyle = InputBox("Legen Sie den Namen des neuen Textstiles" _
    & " fest", "neuer Textstil")
If NewTStyle = "" Then Exit Sub

```

Hat der Benutzer einen Namen eingegeben, werden wir versuchen, einen Textstil mit eben diesem Namen zu erzeugen.

```

Set TStyle = ThisDrawing.TextStyles.Add(NewTStyle)

```

Durch Prüfung des Objekttyps können wir nun ermitteln, ob dieser Vorgang erfolgreich war.

```

If TypeName(TStyle) = "Nothing" Then
    MsgBox "Textstil konnte nicht erstellt werden", vbCritical
Else

```

Da der Textstil nun in der Zeichnung vorhanden ist, können wir diesem die aktuellen Einstellungen des Formulars übergeben und den Textstil dem *ListView*-Steuerelement hinzufügen.

```

    TStyle.fontFile = ComboBox1.Text
    Set LV = ListView1.ListItems.Add(, , TStyle.Name)
    LV.ListSubItems.Add , , ThisDrawing.Utility.RealToString( _
        TStyle.Height, acDefaultUnits, 3)
    LV.ListSubItems.Add , , TStyle.fontFile
    If LCase(Right(TStyle.fontFile, 4)) = ".shx" Then
        LV.SmallIcon = 2
    Else
        LV.SmallIcon = 1
    End If

```

Außerdem machen wir diesen neuen Eintrag zum aktuellen der Liste, so dass der Textstil in der Liste markiert wird.

```

    ListView1.SelectedItem = LV
End If
End Sub

```


Textstil löschen

Beim Löschen von Textstilen müssen Sie lediglich darauf achten, dass die Liste der Textstile vom letzten bis zum ersten Eintrag durchlaufen wird.

```
Private Sub CommandButton2_Click()
Dim TStyle As AcadTextStyle
On Local Error Resume Next
If MsgBox("Soll die gewählten Textstile gelöscht werden?", _
vbYesNo, "Textsil löschen") = vbYes Then
For i = ListView1.ListItems.Count To 1
If ListView1.ListItems(i).Selected = True Then
Set TStyle = ThisDrawing.TextStyles( _
ListView1.ListItems(i).Text)
Err.Clear
```

Die *Delete*-Anweisung löscht schließlich den Textstil aus der Zeichnung. Wurde die Anweisung erfolgreich ausgeführt, was wir mit dem *Err*-Objekt kontrollieren, kann der entsprechende Eintrag auch aus dem *ListView*-Steuerelement entfernt werden.

```
TStyle.Delete
If Err = 0 Then
ListView1.ListItems.Remove I
End If
End If
Next
End If
End Sub
```

Textstil in der Zeichnung auswählen

Zusätzlich zu den üblichen Funktionen zur Verwaltung der Textstile kann der Benutzer einen Text, egal ob ein- oder mehrzeilig, in der Zeichnung auswählen. Daraufhin wird die Zeile mit dem Textstil, in dem der ausgewählte Text dargestellt wird, im *ListView*-Steuerelement zur aktuellen Zeile gemacht.

```
Private Sub CommandButton3_Click()
Dim Object As Object
On Local Error Resume Next
Me.Hide
```

Nachdem das Formular vorübergehend ausgeblendet wurde, wird der Benutzer aufgefordert, einen Text auszuwählen. Dazu nutzen wir die Funktion *GetEntity*, mit der einzelne Objekte in der Zeichnung ausgewählt werden können.

```
ThisDrawing.Utility.GetEntity Object, PickedPoint, _
"wählen Sie eine Text"
If TypeName(Object) = "IAcadText" Or TypeName(Object) = _
"IAcadMText" Then
```

Wenn die Prüfung des ausgewählten Objekts erfolgreich war, d.h., der Benutzer hat entweder einen einzeiligen oder einen mehrzeiligen Text gewählt, durchlaufen wir die Liste und prüfen die Zeile auf den Namen des Textstils.

```
For i = 1 To ListView1.ListItems.Count
If ListView1.ListItems(i).Text = Object.StyleName Then
```

Stimmt der Textstil des gewählten Textes mit dem Listeneintrag überein, können wir diese Zeile zur aktuellen Zeile des *ListView*-Steuerelements machen.

```

        ListView1.SelectedItem = ListView1.ListItems(i)
    Exit For
End If
Next
End If
Me.show
End Sub

```

Sie können auch die *Key*-Eigenschaft des *ListView*-Steuerelements verwenden, um sich das Durchlaufen der Liste zu ersparen. Wenn Sie einen Wert für diese Eigenschaft festlegen, vorzugsweise den Textstilnamen selbst, können Sie beim Abruf des *ListView*-Eintrags mit dem gesuchten Textstil folgendermaßen vorgehen:

```

ListView1.SelectedItem = ListView1.ListItems(Object.StyleName)

```

Sie können die *Key*-Eigenschaft direkt beim Hinzufügen eines Listeneintrags vergeben, indem Sie den entsprechenden Parameter belegen:

```

Set LV = ListView1.ListItems.Add(, TStyle.Name, TStyle.Name)

```

Änderung der Texthöhe

Bei der Änderung der Texthöhe muss zunächst kontrolliert werden, ob der Benutzer einen gültigen Wert festgelegt hat.

```

Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim TStyle As AcadTextStyle
On Local Error Resume Next

```

Dazu wandeln wir den eingegebenen Text in eine Zahl um und prüfen, ob diese einer positiven Zahl entspricht. Ist dies nicht der Fall, setzen wir den Parameter *Cancel* auf *TRUE*, was das Steuerelement wiederum veranlasst, den Änderungsvorgang abubrechen. Damit der Benutzer die Fehleingabe mitbekommt, wird zusätzlich noch ein Meldungsfenster mit einem Hinweis für den Benutzer eingeblendet.

```

If ThisDrawing.Utility.DistanceToReal(TextBox1.Text, _
acDefaultUnits) < 0 Then
    MsgBox "Wert der Texthöhe zu klein", vbCritical, "Texthöhe"
    Cancel = True
Else

```

Hat der Benutzer hingegen einen korrekten Wert eingegeben, wird wieder die Liste der Textstile einzeln durchlaufen und falls die entsprechende Zeile gewählt wurde, ein Verweis auf den Textstil in die Objektvariable *TStyle* geschrieben.

```

For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        Set TStyle = ThisDrawing.TextStyles( _
            ListView1.ListItems(i).Text)
    End If
Next i

```

Was nun noch erledigt werden muss, ist die Zuweisung der Texthöhe an den jeweiligen Textstil. Dazu wandeln wir den Text in einen numerischen Wert um und übergeben diesen an die *Height*-Eigenschaft des Textstils. Außerdem muss die Eintragung in der zweiten Spalte des *ListView*-Steuerelements aktualisiert werden, was aber auch kein Problem darstellen dürfte.

```
TStyle.Height = ThisDrawing.Utility.DistanceToReal( _
    TextBox1.Text, acDefaultUnits)
ListView1.ListItems(i).ListSubItems(1).Text = _
    ThisDrawing.Utility.RealToString(TStyle.Height, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
End If
Next
```

Da auch im Textfeld ein vernünftig formatierter Wert stehen soll, müssen wir abschließend noch den Eintrag aus der zweiten Spalte des *ListView*-Steuerelements an die *TextBox* übertragen. Wir können dafür den Wert des *SelectedItem*-Objekts verwenden, da dieser ja in jedem Fall selektiert wurde und somit den aktuellen Wert für die Texthöhe beinhaltet.

```
TextBox1.Text = ListView1.SelectedItem.ListSubItems(1).Text
FontChanged = True
End If
End Sub
```

Änderung der Textbreite

Die Änderung der Textbreite entspricht in etwa der Vorgehensweise der vorherigen Ereignisprozedur. Aus diesem Grund können wir uns an dieser Stelle große Erklärungen sparen. Nur so viel: Der Eintrag der *TextBox* wird an die Eigenschaft *Width* des Textstils übergeben.

```
Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim TStyle As AcadTextStyle
On Local Error Resume Next
If Util.DistanceToReal(TextBox2) < 0 Then
    MsgBox "Wert der Textbreite zu klein", vbCritical, "Textbreite"
    Cancel = True
Else
    For i = 1 To ListView1.ListItems.Count
        If ListView1.ListItems(i).Selected = True Then
            Set TStyle = ThisDrawing.TextStyles( _
                ListView1.ListItems(i).Text)
            TStyle.Width = Util.DistanceToReal(TextBox2)
        End If
    Next
    TextBox2 = Util.RealToString(TStyle.Width)
    FontChanged = True
End If
End Sub
```

Änderung des Textneigungswinkels

Gleiches gilt für den Neigungswinkel. Auch dieser entspricht im Ablauf den anderen beiden Prozeduren. Allerdings müssen Sie darauf achten, dass es sich bei dieser Eigenschaft um einen Winkel und nicht um eine Strecke handelt, weshalb bei der Umwandlung auch die Funktionen

AngleToReal und *AngleToString* des *Utility*-Objekts zur Anwendung kommen. Außerdem brauchen wir beim Winkel keine Prüfung auf einen gültigen Wert vorzunehmen, da AutoCAD den eingegebenen Wert in einen Vollkreis zurückrechnet.

```
Private Sub TextBox3_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim TStyle As AcadTextStyle
On Local Error Resume Next
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        Set TStyle = ThisDrawing.TextStyles( _
            ListView1.ListItems(i).Text)
        TStyle.ObliqueAngle = Util.AngleToReal(TextBox3)
    End If
Next
TextBox3 = Util.AngleToString(TStyle.ObliqueAngle)
FontChanged = True
End Sub
```

Änderung des Textes der Voransicht

In dieser Ereignisprozedur wird nur der Text, der für die Voransicht eingegeben werden kann, an die *Caption*-Eigenschaft des *Label*-Steuerelements übertragen.

```
Private Sub TextBox5_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Label3.Caption = TextBox5.Text
End Sub
```

Programm beenden

Nachdem das Formular ausgeblendet wurde, kommt noch die Variable *FontChanged* zum Einsatz. Diese enthält ja bekanntlich die Information darüber, ob sich ein Textstil während der Formuläreinblendung geändert hat oder nicht. Sollte die Variable den Wert *TRUE* enthalten, wird die aktuelle Zeichnung regeneriert, damit die Änderungen sichtbar werden.

```
Private Sub CommandButton4_Click()
On Local Error Resume Next
Me.Hide
If FontChanged = True Then ThisDrawing.Regen acAllViewports
End Sub
```

7.8 Benutzerkoordinatensysteme

Das folgende Kapitel befasst sich eingehend mit der Definition und Verwaltung von Benutzerkoordinatensystemen (BKS) mittels der VBA-Schnittstelle. Da alle Elemente der AutoCAD-Objektbibliothek ausschließlich in Englisch verfasst sind, werden die Klassen, die diese Aufgaben für uns erledigen, auch nicht mit BKS, sondern mit UCS (User Coordinate System) bezeichnet. Die Klasse, mit der Sie die komplette Liste der einzelnen Koordinatensysteme verwalten, trägt die Bezeichnung *AcadUCSs*, ein einzelnes *AcadUCS*. Um auf ein einzelnes Koordinatensystem zugreifen zu können, müssen Sie immer über die Gesamtaufistung *UCSs* gehen.

7.8.1 Koordinatensystemauflistung

Beginnen wir also mit der kompletten Auflistung der Koordinatensysteme und wie Sie von dort aus auf einzelne Koordinatensysteme zugreifen können. Da jede Zeichnung eine eigene Auflistung an Koordinatensystemen enthalten kann, ist es logisch, dass wir über das Dokument an dieses Objekt herankommen. Dazu können Sie entweder auf das aktuelle Dokument zurückgreifen

```
Set ActUCSs = ThisDrawing.UserCoordinateSystems
```

oder über die Dokumentauflistung ein bestimmtes UCS ermitteln:

```
Set ActUCSs = Documents(n).UCSs
```

Über die so erhaltene Auflistung können Sie nun neue Koordinatensysteme zu Ihrer Zeichnung hinzufügen oder ein bereits vorhandenes Koordinatensystem abrufen. Dabei ist die Anzahl an Koordinatensystemen, die die Auflistung aufnehmen kann, unbegrenzt.

Die Klasse *AcadUCSs* besitzt zwar eine *Delete*-Anweisung, es ist aber nicht möglich, diese zu benutzen. Um ein einzelnes Koordinatensystem aus der Auflistung zu löschen, müssten Sie das entsprechende UCS abrufen und darauf dann die *Delete*-Anweisung anwenden.

Im Folgenden erhalten Sie eine Übersicht über alle Eigenschaften und Methoden der Klasse *AcadUCSs*.

Benutzerkoordinatensysteme hinzufügen/vorhandene abrufen

(Beispielprojekt: 07\UCSs.dvb!ThisDrawing.AddUCS / ThisDrawing.ChangeUCS_V1)

Zunächst soll dargestellt werden, wie ein Benutzerkoordinatensystem über die VBA-Schnittstelle erzeugt wird. Zu diesem Zweck stellt Ihnen die Klasse *AcadUCSs* die *Add*-Anweisung zur Verfügung. Bevor Sie diese Anweisung ausführen können, müssen allerdings eine ganze Reihe von Parametern festgelegt werden. Als Erstes wird der Ursprungspunkt bestimmt, der über die *GetPoint*-Funktion der *Utility*-Klasse abgefragt werden könnte. Dann müssen noch die X- und Y-Vektoren festgelegt werden und zusätzlich der Name des neuen Koordinatensystems, der allerdings keine Probleme bereiten sollte. Das Ganze lässt sich anhand eines Beispiels besser veranschaulichen.

Als Erstes ermitteln wir den Ursprungspunkt des Koordinatensystems. Dazu kommt die *GetPoint*-Funktion des *Utility*-Objekts zum Einsatz, die den Benutzer auffordert, einen beliebigen Punkt in der Zeichnung auszuwählen. Sie könnten auch ein Zeichnungsobjekt abfragen (*GetEntity*) und dann einen Stützpunkt dieses Elements (Einfügepunkt, Linienendpunkt, Polylinienstützpunkt usw.) als Basispunkt verwenden.

```
Public Sub AddUCS()  
Dim OrgCoords(2) As Double  
Dim XCoords(2) As Double  
Dim YCoords(2) As Double  
Dim varPoint As Variant  
Dim UCSName As String  
On Local Error Resume Next  
varPoint = Utility.GetPoint(, "Ursprungspunkt des BKS wählen")
```

Jetzt kontrollieren wir, ob der gewählte Punkt auch tatsächlich Koordinaten enthält. Um diese Abfrage einmal ein wenig zu variieren, verwenden wir nicht die *TypeName()*-Funktion, sondern die *IsEmpty()*-Abfrage. Diese Funktion liefert für den Fall, dass die Variable *varPoint* keine Werte enthält (Benutzer hat keinen Punkt gewählt), den Wert *TRUE* zurück. Die Prozedur wird dann vorzeitig beendet. Ansonsten können Sie mit dem Code fortfahren und die ermittelten Koordinaten an das Array *OrgCoords* übergeben.

```
If IsEmpty(varPoint) = True Then Exit Sub
OrgCoords(0) = varPoint(0)
OrgCoords(1) = varPoint(1)
OrgCoords(2) = varPoint(2)
```

Dieses Array kommt nun umgehend zum Einsatz, um einen Richtungsvektor zu ermitteln.

Zuvor bedarf es jedoch noch einiger Anmerkungen zum Thema Vektoren. Die Werte, die im Richtungsarray enthalten sind, stellen keine Koordinaten in dem Sinne dar. Vielmehr handelt es sich um ein „internes“ Koordinatensystem, das die entsprechenden Richtungen definieren soll. Schauen wir uns hierzu die Abbildung 7.40 an.

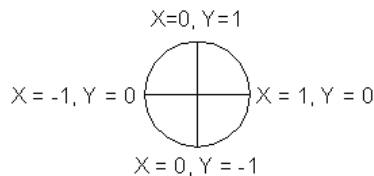


Abbildung 7.40: Definition der Vektormatrix

Das Zentrum dieser Matrix bestimmt immer den entsprechenden Ausgangspunkt, in unserem Fall also den Basis- oder Ursprungspunkt des UCS. Das Koordinatensystem wiederum bezieht sich auf das WKS unabhängig von den Einheiteneinstellungen für Winkel, da ja immer Koordinaten angegeben werden. Liegen also die Linien in Richtung des ersten Quadranten, so erhalten die Richtungsvektoren die Werte $X=1$ und $Y=0$. Zeigt die Linie in Richtung des zweiten Quadranten, also senkrecht nach oben, so erhalten die Vektoren die Werte $X=0$ und $Y=1$. Sie müssen also nur die Koordinaten ermitteln, die der aktuelle Richtungspunkt auf dem gedachten Kreis definiert.

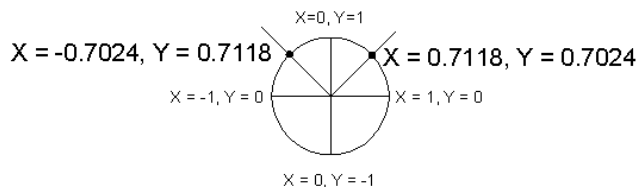


Abbildung 7.41: Vektordefinitionen auf dem gedachten Kreis

Die so ermittelten Koordinaten legen den jeweiligen Vektor fest. Da X- und Y-Achse immer rechtwinklig aufeinander stehen, müssen wir folgerichtig nur einen Vektor ermitteln und dem anderen den rechten Winkel abziehen oder hinzufügen. Möchten Sie das Koordinatensystem

räumlich verdrehen, müssen Sie auch die Z-Achse berücksichtigen, die ebenfalls von 1 bis -1 definiert ist. Die gesamte Vektorendefinition würde sich dann nicht als Kreis, sondern als Kugel darstellen (siehe auch *Kapitel 8.1.2*).

Aber weiter im Code. Als Erstes wird ein Punkt auf der X-Achse des neuen Koordinatensystems abgefragt. Die Abfrage entspricht im Wesentlichen der Abfrage des Ursprungspunkts. Lediglich der Parameter *Point* wird dieses Mal vergeben, um ein Gummiband vom Ausgangspunkt zur besseren Orientierung zeichnen zu lassen.

```
varPoint = Empty
varPoint = Utility.GetPoint(OrgCoords, "X-Richtung wählen:")
If IsEmpty(varPoint) = True Then Exit Sub
XCoords(0) = varPoint(0)
XCoords(1) = varPoint(1)
XCoords(2) = varPoint(2)
```

Der gleiche Vorgang spielt sich für die Y-Richtung ab. Auch hier wird der Ursprungspunkt verwendet, um das Gummiband anzeigen zu lassen.

```
varPoint = Empty
varPoint = Utility.GetPoint(OrgCoords, "X-Richtung wählen:")
If IsEmpty(varPoint) = True Then Exit Sub
YCoords(0) = varPoint(0)
YCoords(1) = varPoint(1)
YCoords(2) = varPoint(2)
```

Fehlt zum Abschluss nur noch der Name des neuen BKS. Dieser wird über die Funktion *Input-box* ermittelt.

```
Prompt = "Geben Sie den Namen des neuen Koordinatensystems an"
UCSName = InputBox(Prompt, "Neues BKS")
If UCSName = "" Then Exit Sub
```

Hat der Benutzer einen Namen angegeben, so kann das neue Koordinatensystem der Auflistung hinzugefügt werden. Wir brauchen beim Hinzufügen die Richtungsvektoren nicht in der vorab beschriebenen Matrix zu übergeben, vielmehr können die originalen Koordinaten verwendet werden. AutoCAD weiß schon, was damit gemeint ist.

```
UserCoordinateSystems.Add OrgCoords, XCoords, YCoords, UCSName
```

oder

```
Set ActUCS = UserCoordinateSystems.Add(OrgCoords, XCoords, _
    YCoords, UCSName)
End Sub
```

Sollten Sie bei der Erzeugung das BKS an eine Objektvariable übergeben, so können Sie es anschließend weiterverwenden, um z.B. dieses BKS zum aktuellen BKS zu machen.

```
ThisDrawing.ActiveUCS = ActUCS
```

oder auch

```
ThisDrawing.ActiveUCS = UserCoordinateSystems.Add(OrgCoords, _
    XCoords, YCoords, UCSName)
```

Beachten Sie dabei, dass bei der Übergabe eines BKS als aktives Koordinatensystem das Schlüsselwort *Set* nicht verwendet wird.

Nun noch ein kleines Beispiel zur Ermittlung aller definierten Koordinatensysteme mithilfe der Eigenschaft *Item* und *Count*. Wir werden dabei eine Liste dieser BKS in einem Formular erstellen und es dem Benutzer ermöglichen, durch Auswahl eines dieser BKS zum aktuellen zu machen. Dazu benötigen wir natürlich wieder ein Formular. Diesem Formular fügen Sie eine *ListBox* hinzu und zwei Schaltflächen: eine zum Schließen des Formulars und eine, um dem Benutzer die Auswahl des aktuellen BKS zu ermöglichen.



Abbildung 7.42: BKS-Formular

Das gesamte Programm erfordert nicht allzu viele Codezeilen, wir werden es im weiteren Verlauf aber noch etwas aufpeppen.

Zuerst wieder der Modulcode. Dort bereinigen wir die *ListBox* des Formulars von eventuell noch vorhandenen Einträgen und fügen dann die Namen der Benutzerkoordinatensysteme des aktuellen Dokuments hinzu. Anschließend wird das Formular aufgerufen.

```
Public Sub ChangeBKS()
On Local Error Resume Next
BKSListFrm.ListBox1.Clear
For i = 0 To UserCoordinateSystems.Count - 1
    BKSListFrm.ListBox1.AddItem UserCoordinateSystems(i).Name
Next
BKSListFrm.Show
End Sub
```

Der Formularcode enthält nur drei Prozeduren, die folgende Anweisungen ausführen sollen:

Die erste Prozedur wird ausgeführt, wenn der Benutzer die Schaltfläche AKTUELL betätigt hat. Sie setzt das ausgewählte Koordinatensystem als das aktuelle der Zeichnung. Da es sich bei der Eigenschaft *ActiveUCS* um eine Objektvariable vom Typ *AcadUCS* handelt, müssen wir zunächst das komplette BKS anhand seines Namens aus der Koordinatensystemauflistung herausuchen lassen und dieses dann an *ActiveUCS* weiterreichen. Danach wird nur noch das Formular ausgeblendet.

```
Private Sub CommandButton1_Click()
ThisDrawing.ActiveUCS = ThisDrawing.UserCoordinateSystems( _
    ListBox1.Text)
Me.hide
End Sub
```


Der Code der zweiten Schaltfläche blendet nur das Formular aus.

```
Private Sub CommandButton2_Click()
Me.Hide
End Sub
```

Den folgenden Ereigniscode könnten Sie auch weglassen. Ich habe ihn nur hinzugefügt, um es dem Benutzer zu ermöglichen, das BKS über einen Doppelklick auf einen *ListBox*-Eintrag auszuwählen. Er braucht also nur die Funktion *CommandButton1_Click* aufzurufen, da das Ergebnis ja dasselbe sein soll, wie wenn der Benutzer die Schaltfläche AKTUELL angeklickt hätte.

```
Private Sub ListBox1_DbClick(ByVal Cancel As _
MSForms.ReturnBoolean)
CommandButton1_Click
End Sub
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.37: Eigenschaften der UCS-Auflistung

Prozedur	Beschreibung
<i>Add</i>	Diese Anweisung fügt ein BKS der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.38: Prozeduren der UCS-Auflistung

7.8.2 Einzelne Koordinatensysteme verwenden

Beschäftigen wir uns nun mit einem einzelnen Benutzerkoordinatensystem. Sie erhalten solch ein BKS, indem Sie auf die vorab beschriebene Auflistung zugreifen oder das aktuelle BKS einer Zeichnung abrufen.

```
ActUCS = ThisDrawing.ActiveUCS
```

oder

```
Set ActUCS = ThisDrawing.UserCoordinateSystems(Name)
```

Dieses BKS können Sie dann mithilfe einiger Eigenschaften und Methoden anpassen. Der ausschlaggebende Punkt bei Benutzerkoordinatensystemen ist natürlich die Definition seiner Richtungen und des Ursprungspunkts.

Die Ursprungskoordinaten des UCS werden als 3D-Array in der Eigenschaft *Origin* gespeichert. Diese Koordinaten werden wie bei allen anderen Objekten auch in WKS angegeben.

```
Dim OrgCoords(2) As Double
OrgCoords(0) = 0 'X
OrgCoords(1) = 0 'Y
OrgCoords(2) = 0 'Z
ActUCS.Origin = OrgCoords
```

Die X- und Y-Richtung des Koordinatensystems werden als Vektoren in den Eigenschaften *XVector* und *YVector* abgelegt. Der jeweilige Richtungsvektor zeigt dabei die Ausrichtung relativ zum WKS an, unabhängig von den EinheitenEinstellungen des Systems. Die jeweiligen Vektoren liegen in einem Wertebereich von -1 bis 1.

Systemvariable *UCSXDIR* / *UCSYDIR*:

```
Dim XCoords(2) As Double
Dim YCoords(2) As Double
XCoords(0) = 1 'X
XCoords(1) = 0 'Y
XCoords(2) = 0 'Z
ActUCS.XVector = XCoords
YCoords(0) = 0 'X
YCoords(1) = 1 'Y
YCoords(2) = 0 'Z
ActUCS.YVector = YCoords
```

Neben den Eigenschaften *XVector* und *YVector* bietet das Objekt noch die Funktion *GetUCSMatrix* an. Diese Anweisung ruft die Transformationsmatrix, bestehend aus Daten des BKS-Koordinatensystems, als Array ab. Die Matrix besteht aus 4x4 Feldern, wobei die Matrix folgendermaßen aufgebaut ist:

Y-Vektor(0)	Y-Vektor(1)	Y-Vektor(2)	Ursprungspunkt(0)
X-Vektor(0)	X-Vektor(1)	X-Vektor(2)	Ursprungspunkt(1)
Z-Vektor(0)	Z-Vektor(1)	Z-Vektor(2)	Ursprungspunkt(2)
0	0	0	1

Tabelle 7.39: Matrix der *GetUCSMatrix*-Anweisung

Siehe auch *Kapitel 7.1.2.*

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Name</i>	Legt den Namen des BKS fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.40: Eigenschaften des UCS-Objekts

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.41: Weitere Prozeduren des UCS-Objekts

Anwendungsbeispiel

(Beispielprojekt 07\AcadUCS.dvb!ThisDrawing.ChangeUCS_V2)

Passen wir unser zuvor besprochenes Programm noch um einige zusätzliche Funktionen an. Zuerst werden wir das Aussehen des Formulars etwas ändern. Entfernen Sie die *ListBox* und fügen Sie an deren Position ein *ListView*-Steuerelement hinzu. Außerdem benötigen wir ein zweites *ListView*-Steuerelement, welches in einem Rahmenelement positioniert wird. Dann wird noch eine dritte Schaltfläche benötigt, die anschließend eine Löschfunktion ausführen wird. Da wir nun ein *ListView*-Steuerelement benutzen, kann das aktuelle BKS durch ein Symbol dargestellt werden. Dieses Symbol wird in einem *ImageList*-Steuerelement verankert, was ebenfalls dem Formular hinzugefügt werden muss. Sie können sich bei der Erstellung des Formulars an der Abbildung 7.43 orientieren.



Abbildung 7.43: Geändertes BKS-Formular

Nachdem das Programm das erste Mal gestartet wurde, müssen die *ListView*-Steuerelemente initialisiert werden, das heißt, dem ersten soll *ImageList* als Icon-Vorlage zugeordnet und dem zweiten einige Zeilen hinzugefügt werden. Nutzen wir also die Ereignisprozedur *Initialize* des Formulars, die ja nur aufgerufen wird, wenn das Formular geladen wird.

```
Private Sub UserForm_Initialize()  
Dim LV As MSComctlLib.ListItem  
On Local Error Resume Next
```

Nach der Deklaration der Objektvariable *LV* (wird innerhalb des Codes verwendet) wird mittels folgender Zeilen das *ImageList1*-Steuerelement der *Icon*- und *SmallIcon*-Eigenschaft des *ListView1*-Steuerelements zugeordnet:

```
Set ListView1.Icons = ImageList1  
Set ListView1.SmallIcons = ImageList1
```

Danach fügen wir einige Einträge *ListView2* hinzu und setzen für die Unterelemente schon mal Vorgabewerte, damit die Liste ordnungsgemäß gefüllt ist. Dadurch entsteht eine Listendarstellung, wie in Abbildung 7.44 zu sehen.

	X	Y	Z
Ursprung:	0	0	0
X-Achse:	0	0	0
Y-Achse:	0	0	0
Z-Achse:	0	0	0

Abbildung 7.44: In der *Initialize*-Prozedur erzeugte Liste

Erste Zeile:

```
Set LV = ListView2.ListItems.Add(, , "Ursprung:")  
With LV.ListSubItems  
    .Add , , Util.RealToString(0)  
    .Add , , Util.RealToString(0)  
    .Add , , Util.RealToString(0)  
End With
```

Zweite Zeile in *ListView2*:

```
Set LV = ListView2.ListItems.Add( , , "X-Achse:")
With LV.ListSubItems
    .Add , , Util.RealToString(0)
    .Add , , Util.RealToString(0)
    .Add , , Util.RealToString(0)
End With
```

Dritte Zeile in *ListView2*:

```
Set LV = ListView2.ListItems.Add( , , "Y-Achse:")
With LV.ListSubItems
    .Add , , Util.RealToString(0)
    .Add , , Util.RealToString(0)
    .Add , , Util.RealToString(0)
End With
```

Vierte Zeile in *ListView2*:

```
Set LV = ListView2.ListItems.Add( , , "Z-Achse:")
With LV.ListSubItems
    .Add , , Util.RealToString(0)
    .Add , , Util.RealToString(0)
    .Add , , Util.RealToString(0)
End With
End Sub
```

Die nächste Codezeile kam so ungefähr bereits im vorherigen Programm vor. Die einzige Änderung, die wir noch vornehmen mussten, ist die Zuweisung des BKS-Namens. Der Name wird hier nicht als *ListBox*-Element abgerufen, sondern mit der *Text*-Eigenschaft des ausgewählten Elements von *ListView1* → *ListView1.SelectedItem.Text*.

```
Private Sub CommandButton1_Click()
ThisDrawing.ActiveUCS = ThisDrawing.UserCoordinateSystems( _
    ListView1.SelectedItem.Text)
Me.hide
End Sub
```

Der Code der SCHLIESSEN-Schaltfläche hat sich hingegen nicht geändert. Es wird immer noch das Formular geschlossen.

```
Private Sub CommandButton2_Click()
Me.hide
End Sub
```

Als Nächstes erstellen wir den Ereigniscode für die zusätzliche Schaltfläche (LÖSCHEN). In ihm wird dem Benutzer zuerst eine Sicherheitsabfrage angeboten, in der er auswählen kann, ob das BKS wirklich gelöscht werden soll oder nicht.

```
Private Sub CommandButton3_Click()
Dim ActUCS As AcadUCS
On Local Error Resume Next
If MsgBox("Gewähltes BKS löschen", vbYesNo, "Löschen") = vbYes _
    Then
```

Hat der Benutzer den Löschvorgang schließlich bestätigt, kann das BKS, das dem ausgewählten Namen entspricht, ausgelesen und dann gelöscht werden. Danach rufen wir die Prozedur *initBKSList* auf, um die *ListView*-Ansicht zu aktualisieren.

```
Set ActUCS = ThisDrawing.UserCoordinateSystems( _
    ListView1.SelectedItem.Text)
ActUCS.Delete
initBKSList
End If
End Sub
```

In der folgenden Ereignisprozedur nutzen wir die Möglichkeit des *ListView*-Steuerelements, einen Listeneintrag manuell zu ändern. Das Ereignis, auf das wir dann reagieren, wird ausgelöst, nachdem die Namensänderung vollzogen wurde. Dabei wird der neue Name als Parameter *NewString* mitgeliefert, so dass der Namenstausch des BKS einfach zu vollziehen ist.

```
Private Sub ListView1_AfterLabelEdit(Cancel As Integer, _
    NewString As String)
Dim ActUCS As AcadUCS
On Local Error Resume Next
Set ActUCS = ThisDrawing.UserCoordinateSystems( _
    ListView1.SelectedItem.Text)
ActUCS.Name = NewString
initBKSList
End Sub
```

Das folgende Ereignis führt den Code der Schaltfläche AKTUELL aus, sobald der Benutzer einen Doppelklick auf dem *ListView1*-Steuerelement ausgeführt hat.

```
Private Sub ListView1_DbClick()
CommandButton1_Click
End Sub
```

Ziel der nächsten Prozedur wird es sein, bei der Auswahl eines *ListView1*-Elements, egal welcher Zeile, die Eigenschaften (Ursprung und Vektoren) des ausgewählten BKS zu ermitteln und anzeigen zu lassen. Als Erstes werden wir uns das gewählte BKS aus der BKS-Auflistung ausgeben lassen. Der Name des gesuchten BKS steht in der Texteigenschaft des übergebenen Items (entspricht der gewählten Zeile).

```
Private Sub ListView1_ItemClick(ByVal Item As _
    MSComctlLib.ListItem)
Dim ActUCS As AcadUCS
On Local Error Resume Next
Set ActUCS = ThisDrawing.UserCoordinateSystems(Item.Text)
```

Auslesen und Zuordnen der Ursprungskoordinaten

Hierbei wird die Eigenschaft *Origin* des BKS verwendet. Diese gibt die Koordinaten als 3D-Array zurück, so dass in den nächsten Zeilen jeweils ein formatierter Text der X-, Y- und Z-Koordinate an die erste, zweite und dritte Zusatzspalte der ersten Zeile des *ListView2*-Steuerelements zugeordnet werden kann.

```
varPoint = ActUCS.Origin
For i = 0 To 2
    With ListView2.ListItems(1)
```

```

        .ListSubItems(i + 1).Text = Util.RealToString(varPoint(i))
    End With
Next

```

Auslesen und Zuordnen der X-Vektoren

Auch hier wird der Inhalt der entsprechenden Eigenschaft (*XVector*) an die jeweilige Spalte formatiert übergeben.

```

varPoint = ActUCS.XVector
For i = 0 To 2
    With ListView2.ListItems(2)
        .ListSubItems(i + 1).Text = Util.RealToString(varPoint(i))
    End With
NextDas

```

Auslesen und Zuordnen der Y-Vektoren

Genau derselbe Vorgang wie zuvor. Nur die Eigenschaft ist eine andere (*YVector*).

```

varPoint = ActUCS.YVector
For i = 0 To 2
    With ListView2.ListItems(3)
        .ListSubItems(i + 1).Text = Util.RealToString(varPoint(i))
    End With
Next

```

Auslesen und Zuordnen der Z-Vektoren

Da der Z-Vektor nicht direkt abrufbar ist, müssen wir die Funktion *GetUCSMatrix* zur Ermittlung verwenden. Diese Funktion gibt eine komplette Matrix mit allen Vektoren zurück. Wir benötigen aber lediglich die Daten des zweiten Arrays, also *Matrix(2)*.

```

varPoint = ActUCS.GetUCSMatrix
For i = 0 To 2
    With ListView2.ListItems(4)
        .ListSubItems(i + 1).Text = Util.RealToString(varPoint(2, _
            i))
    End With
Next

```

Der folgenden Code war so ähnlich bereits in der Startprozedur des vorherigen Programms vorhanden. Wir haben die Funktionsweise der Anweisungen in den Formularcode verlagert, da er durch andere Prozeduren bei Bedarf aufgerufen werden muss. Auch unterscheidet sich die Syntax in einigen Punkten. Zunächst wird die *ListItems*-Auflistung des *ListView1*-Objekts geleert, falls dort noch Elemente enthalten waren.

```

Public Sub initBKSList()
    Dim LV As MSComctlLib.ListView
    On Local Error Resume Next
    ListView1.ListItems.Clear

```

Jetzt wird die UCS-Auflistung durchlaufen und jedes Benutzerkoordinatensystem nacheinander abgerufen. Jeder BKS-Name wird nun dem *ListView*-Steuerelement hinzugefügt.

```
For i = 0 To ThisDrawing.UserCoordinateSystems.Count - 1
    Set LV = ListView1.ListItems.Add(, , _
        ThisDrawing.UserCoordinateSystems(i).Name)
```

Zusätzlich können wir noch ein Symbol dem Element hinzufügen, für den Fall, dass es sich um das aktuelle BKS handelt. Als Ergebnis dieser Prozedur müssten Sie eine Liste, ähnlich Abbildung 7.45, erhalten.

```
    If ThisDrawing.ActiveUCS.Name = _
        ThisDrawing.UserCoordinateSystems(i).Name Then
        LV.SmallIcon = 1
    End If
Next
End Sub
```

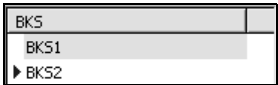


Abbildung 7.45: Markierung des aktuellen BKS mit einem Symbol

7.9 Anwendungen in AutoCAD registrieren

Mit den im folgenden Kapitel beschriebenen Klassen können Sie Ihre Anwendungen in AutoCAD-Zeichnungen „registrieren“ lassen. Das heißt, es wird ein Eintrag mit dem Namen der Anwendung erstellt, der dann fest in der Zeichnung verankert ist. Wozu aber diesen Aufwand betreiben? Um die Anwendung durch AutoCAD identifizieren zu können, muss der Name der Anwendung, die in AutoCAD benutzt wird, auch in AutoCAD angemeldet werden.

Anwendungsnamen werden mit den erweiterten Daten jedes Zeichnungsobjekts gespeichert, das diese Anwendung verwendet. Jeder Fall der erweiterten Daten, die in einer Zeichnung bezogen werden, muss seine Anwendung haben, die in der Zeichnung registriert wird.

7.9.1 Die Anwendungsauflistung

In der Anwendungsauflistung, die durch die Klasse *AcadRegisteredApplications* interpretiert wird, sind alle in der Zeichnung registrierten Anwendungen enthalten. Sie können dieser Auflistung neue Anwendungsnamen hinzufügen, um sie zu registrieren, oder bereits eingetragene Anwendungsnamen abrufen, um sie zu ändern oder anderweitig zu verarbeiten.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.

Tabelle 7.42: Eigenschaften der Anwendungsauflistung

Eigenschaft	Beschreibung
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.42: Eigenschaften der Anwendungsauflistung (Forts.)

Prozedur	Beschreibung
<i>Add</i>	Diese Anweisung fügt einen Anwendungsamen der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.43: Prozeduren der Anwendungsauflistung

7.9.2 Einzelne Einträge der Anwendungsauflistung verarbeiten

Sie können auf einzelne Einträge der Anwendungsauflistung zugreifen, indem Sie das bei der *Add*-Methode zurückgegebene Objekt weiterverarbeiten oder eine bereits registrierte Anwendung aus der Auflistung abrufen. In jedem Fall wird ein Objekt vom Typ *AcadRegisteredApplication* erzeugt.

Sie sollten bei der Entfernung von registrierten Anwendungen vorsichtig vorgehen, da diese meist mit definierten Objekten zusammenhängen.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.

Tabelle 7.44: Eigenschaften des UCS-Objekts

Eigenschaft	Beschreibung
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Name</i>	Legt den Namen des BKS fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.44: Eigenschaften des UCS-Objekts (Forts.)

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.45: Weitere Prozeduren des UCS-Objekts

Anwendungsbeispiel

(Beispielprojekt 07\AcadRegisteredApplications.dvb)

Im nun folgenden Beispiel sind die Funktionen beider in diesem Kapitel besprochenen Klassen zusammengefasst. Dieses Programm soll alle Einträge der Anwendungsauflistung in einem Formular anzeigen und gegebenenfalls neue Anwendungen registrieren oder aus der Auflistung entfernen.

Zunächst erstellen Sie ein neues Projekt und fügen diesem ein Formular hinzu. Auf diesem Formular positionieren Sie ein *ListBox*-Steuerelement und drei Schaltflächen, deren *Caption*-Eigenschaften Sie in HINZUFÜGEN (*CommandButton1*), ENTFERNEN (*CommandButton2*) und SCHLIESSEN (*CommandButton3*) ändern müssen. Wenn Sie alles erledigt haben, sollte ein Formular entstehen, das der Abbildung 7.46 ähnelt.

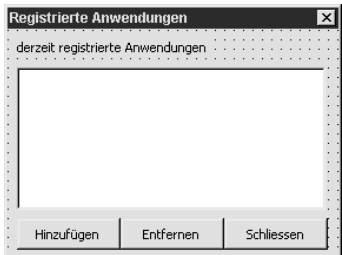


Abbildung 7.46: Anwendungsformular

Dem Formular müssen nun wieder einige Codezeilen hinzugefügt werden, um auf die Eingaben des Benutzers zu reagieren.

Mit der ersten Ereignisprozedur reagieren wir auf die Anweisung des Benutzers, eine Anwendung zur Auflistung hinzufügen zu wollen. Der erste Schritt wird dabei die Abfrage des Applikationsnamens sein. Dieser Name wird über die Funktion *InputBox* abgefragt:

```
Private Sub CommandButton1_Click()
Dim AppName As String
On Local Error Resume Next
AppName = InputBox("Geben Sie den Anwendungsnamen ein:", _
    "Neue Anwendung")
```

Hat der Benutzer einen Namen festgelegt und die Eingabebox mit OK bestätigt, wird die Anwendung registriert. Andernfalls wird die Prozedur vorzeitig verlassen.

```
If AppName = "" Then Exit Sub
ThisDrawing.RegisteredApplications.Add AppName
```

Was nun noch bleibt, ist die Aktualisierung der *ListBox*, so dass die gerade registrierte Anwendung mit enthalten ist.

```
ListBox1.Clear
For i = 0 To ThisDrawing.RegisteredApplications.Count - 1
    ListBox1.AddItem ThisDrawing.RegisteredApplications(i).Name
Next
End Sub
```

Die zweite Prozedur ist eine Reaktion auf die Benutzung der ENTFERNEN-Schaltfläche. Hier wird zunächst eine Sicherheitsabfrage durchgeführt. Nach Bestätigung durch den Benutzer wird der Anwendungsname, der in der *ListBox* selektiert wurde, aus der Auflistung entfernt.

```
Private Sub CommandButton2_Click()
Dim AppName As String
On Local Error Resume Next
If MsgBox("Anwendung " & ListBox1.Text & " entfernen?", vbYesNo, _
    "Anwendung entfernen") = vbYes Then
    ThisDrawing.RegisteredApplications(ListBox1.Text).Delete
```

Anschließend wird wiederum der Inhalt der *ListBox* aktualisiert.

```
ListBox1.Clear
For i = 0 To ThisDrawing.RegisteredApplications.Count - 1
    ListBox1.AddItem ThisDrawing.RegisteredApplications(i).Name
Next
End If
End Sub
```

Zum Schluss wird noch auf die Schaltfläche SCHLIESSEN reagiert und das Formular ausgeblendet.

```
Private Sub CommandButton3_Click()
Me.Hide
End Sub
```

Dem Klassenmodul *ThisDrawing* fügen Sie folgende Startprozedur hinzu, die die ListBox des Formulars mit den Anwendungsnamen füllt und anschließend das Formular startet.

```
Public Sub showApplications()
On Local Error Resume Next
RegAppFrm.ListBox1.Clear
For i = 0 To RegisteredApplications.Count - 1
    RegAppFrm.ListBox1.AddItem RegisteredApplications(i).Name
Next
RegAppFrm.Show
End Sub
```

7.10 Blockdefinitionen verwalten

Blockdefinitionen stellen ja bekanntlich eine Art Zeichnung innerhalb einer Zeichnung dar. Verwaltet werden diese Blöcke unter VBA in der *Blockauflistung*. Wenn Sie also die Blöcke einer Zeichnung manipulieren wollen, müssen Sie zunächst auf diese Auflistung zugreifen und einen Verweis auf den entsprechenden Block ermitteln, was zumeist über dessen Namen geschehen wird.

Die Blockdefinitionen selbst unterteilen sich in drei Kategorien. Zur ersten gehören ganz normale Blockdefinitionen, die der Benutzer mit den AutoCAD-Befehlen `_insert` oder `einfüge` in die Zeichnung einfügen kann. Die zweite Kategorie bilden unbenannte (anonyme) Blockdefinitionen. Diese werden durch ein `*` vor dem Namen gekennzeichnet. Zu diesen Blockdefinitionen gehören unter anderem der Modell- (**ModelSpace*) und der Papierbereich (**PaperSpace*), aber auch Definitionen von Schraffuren oder Assoziativbemaßungen. Auf diese Blöcke kann der Benutzer nicht direkt über AutoCAD zugreifen. Die dritte Kategorie bilden Blockdefinitionen externer Referenzen. Diese sind daran zu erkennen, dass dem Namen des Blocks der Name der Referenz vorangestellt wurde und beide Namen durch das `|`-Zeichen getrennt werden. Wurde beispielsweise eine externe Referenz mit dem Namen „ExtRef1“ in die Zeichnung eingefügt und enthält diese die Blockdefinitionen „Block1“ und „Block2“, dann werden Sie folgende Definitionen in der Blockauflistung vorfinden:

```
ExtRef1
ExtRef1|Block1
ExtRef1|Block2
```

7.10.1 Blockauflistung

In der Blockauflistung sind alle Blockdefinitionen der Zeichnung gespeichert – so auch der Modellbereich und die verschiedenen Layouts, die ja ebenfalls einer Blockdefinition entsprechen. Aus diesem Grund ergibt sich die Schlussfolgerung, dass die Objekte *ModelSpace* und *PaperSpace*, die ja diese Bereiche darstellen, eigentlich Blockdefinitionen sind. Somit ist auch klar, dass das Blockdefinitionsobjekt über dieselben Eigenschaften und Methoden verfügt wie die Objekte *ModelSpace* und *PaperSpace*. Aber dazu später mehr. Zugriff auf die Blockauflistung erhalten Sie über das jeweilige Dokumentobjekt.

```
Set AllBlocks = Blocks
```

bzw.

```
Set AllBlocks = ThisDrawing.Blocks
```

für das aktuelle Dokument oder

```
Set AllBlocks = Documents(n).Blocks
```

für die Blockauflistung einer beliebigen Zeichnung, die in der aktuellen AutoCAD-Sitzung geladen wurde.

Blockdefinition erstellen/vorhandene Blockdefinition abrufen

Wie bei allen anderen Auflistungen auch, enthält das Blockauflistungsobjekt ebenfalls eine *Add*-Funktion, mit der einzelne Blockdefinitionen erzeugt werden können. Im Gegensatz zu anderen *Add*-Anweisungen müssen Sie bei der Blockauflistung neben dem Namen der neuen Blockdefinition einen weiteren Parameter festlegen. Bei diesem handelt es sich um den Basispunkt der Blockdefinition, also um die Koordinaten, auf die sich der Einfügepunkt der Blockreferenzen (siehe *Kapitel 8.3.29*) bezieht. Dieser Basispunkt wird anschließend in der Eigenschaft *Origin* gespeichert, die bereits vom Dokumentobjekt her bekannt sein dürfte.

```
Dim Origin(2) As Double
Set NewBlock = Blocks.Add(Origin, "neuer Block")
```

Leider hat man es versäumt, eine *Load*-Funktion wie beispielsweise bei den Linientypen in die Blockauflistung zu integrieren, so dass Sie einen etwas anderen Weg gehen müssen, um eine vorhandene Zeichnung als Blockdefinition in das Dokument einzubinden. In diesem Fall müssen Sie auf die *InsertBlock*-Funktion des Modellbereichsobjekts zurückgreifen und zunächst eine Blockreferenz unter Angabe des Zeichnungsnamens erzeugen. Diese können Sie anschließend gleich wieder löschen, da sie ihren Zweck erfüllt hat – das Erzeugen der Blockdefinition in der Blockauflistung. Genauso sieht es aus, wenn Sie eine vorhandene Blockdefinition durch die Daten einer vorhandenen Zeichnung ersetzen wollen. Auch hier müssen Sie den Umweg über die *Insert*-Funktion einschlagen, da diese auch die Daten eines Blocks durch die Daten einer Zeichnung ersetzt, wenn dieser bereits in der Blockauflistung definiert wurde. Ausschlaggebend ist hier immer der Name der Zeichnung (exklusive der Verzeichnisangabe und der Erweiterung *dwg* bzw. *dxf*), die eingelesen wird.

```
Dim InsPoint
DwgName = "C:\Zeichnungen\Zeichnung1.dwg"
InsPoint = GetVariable("VIEWCTR")
Set BlkRef = ModelSpace.InsertBlock(InsPoint, DwgName, 1, 1, 1, 0)
BlkRef.Delete
```

Als Koordinaten des Einfügepunkts des einzulesenden Blocks sollten Sie den aktuellen Zentrumspunkt des Zeichnungsausschnitts verwenden, damit die aktuellen Zeichnungsgrenzen nicht unbeabsichtigt verändert werden.

Nachdem Sie eine Blockdefinition erstellt haben, auf welche Weise auch immer, können Sie ganz normal über die *Item*-Funktion der Blockauflistung auf einzelne Blockdefinitionen zugreifen. Dabei können Sie entweder die genaue Position der jeweiligen Definition angeben:

```
Set ActBlockRef = Blocks(i)
```

oder den Namen des gesuchten Blocks verwenden, was meist der Fall sein dürfte:

```
Set ActBlockRef = Blocks("Block1")
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.46: Eigenschaften der Blockauflistung

Prozedur	Beschreibung
<i>Add</i>	Diese Anweisung fügt eine Blockdefinition der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.47: Prozeduren der Blockauflistung

7.10.2 Einzelne Blockdefinitionen verarbeiten

Einzelne Blockdefinitionen bilden die Grundlage der in der jeweiligen Zeichnung grafisch dargestellten Blockreferenzen. Da, wie bereits erwähnt, auch Modell- und Papierbereich zu diesen Blockdefinitionen zählen, können Sie die Blöcke genauso behandeln, wie wenn es sich um die Objekte *ModelSpace* bzw. *PaperSpace* handelt. Das bedeutet, dass Sie den Blöcken grafische Objekte hinzufügen oder auch aus diesen wieder entfernen können.

Objekte der Blockdefinition hinzufügen

Zu diesem Zweck stellt die Klasse *AcadBlock*, in der eine einzelne Blockdefinition abgebildet wird, die verschiedensten *Add*-Funktionen zur Verfügung, deren genaue Beschreibung Sie im *Kapitel 6.4.2* nachlesen können.

Prozedur	Beschreibung
<i>Add3DFace</i>	Erstellt ein 3D-Flächenobjekt.
<i>Add3DMesh</i>	Erstellt ein Freiform-3D-Netz.
<i>Add3DPoly</i>	Erstellt eine 3D-Polylinie.
<i>AddArc</i>	Erstellt einen Bogen.
<i>AddAttribute</i>	Erstellt eine Attributdefinition.
<i>AddCircle</i>	Erstellt einen Kreis.
<i>AddCustomObject</i>	Erstellt ein benutzerdefiniertes Objekt.
<i>AddEllipse</i>	Erstellt eine Ellipse.
<i>AddHatch</i>	Erstellt ein Schraffurobjekt.
<i>AddLightWeightPolyline</i>	Erstellt ein optimierte Polylinie.
<i>AddLine</i>	Erstellt eine Linie.
<i>AddMInsertBlock</i>	Fügt eine Mehrfachblockreferenz ein.
<i>AddMLine</i>	Erstellt eine Multilinie.
<i>AddMText</i>	Erstellt einen Absatztext.
<i>AddPoint</i>	Erzeugt ein Punktobjekt.
<i>AddPolyfaceMesh</i>	Erstellt ein Vielflächenetz.
<i>AddPolyline</i>	Erstellt ein 2D-Polyline.
<i>AddRaster</i>	Fügt ein Pixelbild anhand einer vorhandenen Bilddatei ein.
<i>AddRay</i>	Erstellt einen Strahl.
<i>AddRegion</i>	Erstellt eine Region.
<i>AddShape</i>	Fügt ein Symbol ein.
<i>AddSolid</i>	Erstellt ein 2D-Volumenkörperpolygon.
<i>AddSpline</i>	Erstellt einen Spline.
<i>AddText</i>	Erstellt einen einzeiligen Text.
<i>AddTrace</i>	Erstellt ein Bandobjekt.
<i>AddXline</i>	Erstellt eine Konstruktionslinie.
<i>AttachExternalReference</i>	Bindet eine externe Referenz in die Blockreferenz ein.
<i>InsertBlock</i>	Fügt eine Blockreferenz der Blockdefinition hinzu.

Tabelle 7.48: Methoden zur Erstellung von grafischen Objekten in Blockdefinitionen

3D-Volumenkörper	
<i>AddBox</i>	Erstellt einen quaderförmigen 3D-Volumenkörper.
<i>AddCone</i>	Erstellt einen kegelförmigen 3D-Volumenkörper.
<i>AddCylinder</i>	Erstellt einen zylinderförmigen 3D-Volumenkörper.
<i>AddEllipticalCone</i>	Erstellt einen kegelförmigen 3D-Volumenkörper mit elliptischer Grundfläche.
<i>AddEllipticalCylinder</i>	Erstellt einen zylinderförmigen 3D-Volumenkörper mit elliptischer Grundfläche.
<i>AddExtrudedSolid</i>	Extrudiert einen 3D-Volumenkörper anhand von Profil, Höhe und Verjüngungswinkel.
<i>AddExtrudedSolidAlongPath</i>	Extrudiert einen 3D-Volumenkörper anhand des Profils und eines Extrusionspfads.
<i>AddRevolvedSolid</i>	Erstellt einen Rotationsvolumenkörper.
<i>AddSphere</i>	Erstellt einen kugelförmigen Volumenkörper.
<i>AddTorus</i>	Erstellt einen ringförmigen Volumenkörper.
<i>AddWedge</i>	Erstellt einen keilförmigen Volumenkörper.

Bemaßungsobjekte	
<i>AddDim3PointAngular</i>	Erstellt eine Winkelbemaßung für einen Bogen, zwei Linien oder einen Kreis.
<i>AddDimAligned</i>	Erstellt ein ausgerichtetes Bemaßungsobjekt.
<i>AddDimAngular</i>	Erstellt eine Winkelbemaßung für einen Bogen, zwei Linien oder einen Kreis.
<i>AddDimDiametric</i>	Erstellt eine Durchmesserbemaßung.
<i>AddDimOrdinate</i>	Erstellt eine Koordinatenbemaßung.
<i>AddDimRadial</i>	Erstellt eine Radiusbemaßung.
<i>AddDimRotated</i>	Erstellt eine gedrehte lineare Bemaßung.
<i>AddLeader</i>	Erstellt eine Führungslinie
<i>AddTolerance</i>	Erstellt ein Toleranzobjekt.

Tabelle 7.48: Methoden zur Erstellung von grafischen Objekten in Blockdefinitionen (Forts.)

Das Hinzufügen der einzelnen Objekte mittels der *Add*-Funktionen dürfte vom Prinzip her klar sein. Das einzige, worauf Sie hierbei achten müssen, ist die Position des Ursprungspunkts der Blockdefinition. Nehmen wir einfach ein Beispiel. Stellen Sie sich vor, Sie wollen es dem Benutzer ermöglichen, Linien einem Block hinzuzufügen, indem er einfach eine Blockreferenz in der Zeichnung wählt und dann über die Auswahl zweier Punkte eine Linie erzeugen soll. Zunächst müsste er also die Blockreferenz in der Zeichnung wählen, damit Sie daraus die Blockdefinition ermitteln.

```
Utility.GetEntity Object, PickedPoint, "Blockreferenz wählen:"  
Set ActBlkDef = Blocks(Object.Name)
```


Anschließend lassen wir die beide Endpunkte der hinzuzufügenden Linie auswählen. Da der Benutzer diese in Bezug zur gewählten Blockreferenz bestimmen, also in der Nähe der durch die Blockreferenz dargestellten Objekte anwählen wird.

```
Point1 = Utility.GetPoint(, "Startpunkt:")
Point2 = Utility.GetPoint(Point1, "Endpunkt:")
```

Damit die neue Linie nun nicht irgendwo in der Blockdefinition erscheint, was ja der Fall ist, da durch die Angabe der gewählten Koordinaten die Linie an eben diesen Koordinaten und nicht an der, vom Benutzer angedachten Position erscheint, sind noch zwei Dinge bei der Bestimmung der Koordinaten zu berücksichtigen. Zum einen der Einfügepunkt der Blockreferenz und zum anderen der Basispunkt der Blockdefinition.

```
InsPoint = Object.InsertionPoint
Origin = ActBlkDef.Origin
Point1(0) = Point1(0) + Origin(0) - InsPoint(0)
Point1(1) = Point1(1) + Origin(1) - InsPoint(1)
Point2(0) = Point2(0) + Origin(0) - InsPoint(0)
Point2(1) = Point2(1) + Origin(1) - InsPoint(1)
```

Nachdem die Koordinaten der gewählten Punkte nun auf die gewählte Blockreferenz angepasst wurden, können wir die Linie der Blockdefinition hinzufügen.

```
ActBlkDef.AddLine Point1, Point2
Object.Update
```

Was ist aber mit Objekten, die bereits in der Zeichnung existieren und nun der gewählten Blockreferenz hinzugefügt werden sollen? Nun hier hilft die *CopyObjects*-Anweisung des *Document*-Objekts weiter. Mit dieser Funktion können ja beliebige AutoCAD-Objekte dupliziert werden. Dazu müssen wir zunächst ein Array mit den Objekten zusammenstellen, die in die Blockdefinition eingefügt werden sollen. Dazu erstellen wir einen Auswahl Satz (*Kapitel 8.2.1*), den der Benutzer über die *SelectOnScreen*-Anweisung mit den gewünschten Objekten füllen kann.

```
Set sSet = SelectionSets.Add("Auswahl")
sSet.SelectOnScreen
```

Anschließend wird das Array *ObjList* neu dimensioniert und die Objektverweise aus dem Auswahl Satz werden den einzelnen Feldern des Arrays zugeordnet.

```
ReDim ObjList(sSet.Count - 1) As Object
For i = 0 To sSet.Count - 1
    Set ObjList(i) = sSet.Item(i)
Next
```

Jetzt erfolgt der Aufruf der *CopyObjects*-Anweisung. Als Parameter wird der Anweisung das Array der zu kopierenden Objekte, der Verweis auf das neue Eigentümerobjekt, was ja die Blockdefinition sein soll, und die Variable *IdPairs* übergeben. Diese wird nach dem Kopiervorgang ein Array mit Objekten der Klasse *AcadIdPair* enthalten. Die einzelnen Objekte enthalten Informationen zu den erzeugten Kopien der Ausgangsobjekte, unter anderem die *ObjectID* des Objekts. Diese wird nun dazu verwendet, einen Verweis auf das erstellte Objekt mithilfe der *ObjectIdToObject*-Funktion zu ermitteln, falls die *Primary*-Eigenschaft den Wert *TRUE* zurückgibt. Ist dies der Fall, handelt es sich um ein grafisches Objekt, das direkt in das Zielobjekt kopiert wurde, also kein Eigentum eines anderen kopierten Objekts ist. Wenn Sie beispiels-

weise Blockreferenzen kopieren, werden logischerweise auch alle Objekte mitkopiert, die in der Blockdefinition dieser Referenz enthalten sind. Das können außer den grafischen Elementen auch die Layer, Linientypen, Textstile, Ansichten usw. sein (siehe auch *Kapitel 8.3.33*).

```
CopyObjects ObjList, ActBlkDef, IdPairs
FromPoint = Object.InsertionPoint
For i = 0 To UBound(IdPairs)
    If IdPairs(i).IsPrimary = True Then
```

Wenn es sich also um solch ein Objekt handelt, müssen auch hier die Koordinaten entsprechend der aktuellen Position der Ausgangsblockreferenz angepasst werden. Da vorher nicht bekannt ist, um welchen Objekttyp es sich handelt, und wir auch jedes grafische Objekt unterstützen wollen, müssen die einzelnen Objekte mit der *Move*-Anweisung auf ihre neue Position verschoben werden.

```
        Set Obj = ObjectIdToObject(IdPairs(i).Value)
        Obj.Move FromPoint, ToPoint
    End If
Next
```

Diese Methoden können Sie auf alle Kategorien einer Blockdefinition anwenden, auch auf externe Referenzen. In der Blockauflistung werden alle Definitionen gleich behandelt.

Objekte aus der Blockdefinition abrufen

Objekte, die Eigentum einer Blockdefinition sind, können direkt mit der *Item*-Funktion des *Block*-Objekts abgerufen werden. Diese Methode eignet sich besonders bei der Ermittlung der Attributdefinitionen eines Blocks. Sie brauchen in diesem Fall nur die Objekte der Blockdefinition nacheinander abzurufen und dann zu kontrollieren, ob es sich bei dem Objekt um das besagte Attribut handelt.

```
Dim Object As Object
Dim ActBlkDef As AcadBlock
On Error Resume Next
Utility.GetEntity Object, PickedPoint, "Blockreferenz wählen:"
Set ActBlkDef = Blocks(Object.Name)
I = 0
For Each BlkObj In ActBlkDef
    If TypeName(BlkObj) = "IAcadAttribute" Then
        I = I + 1
    End If
Next
MsgBox "Der gewählte Block enthält " & CStr(I) & " Attribute"
```

Wenn Sie es dem Benutzer ermöglichen wollen, einzelne Objekte einer Blockdefinition selbst zu wählen, dann kommt die *GetSubEntity*-Funktion der Klasse *Utility* zum Einsatz. Mit dieser Funktion können Sie das gewählte Unterobjekt der angeklickten Blockreferenz ermitteln lassen. Als Ergebnis erhalten Sie einen Verweis auf diese Objekte.

```
Dim Object As Object
Utility.GetSubEntity Object, PickedPoint, transMatrix, _
    ContextData, "Blockreferenz wählen"
MsgBox "Type des gewählten Objekts: " & Object.EntityName
```

Siehe auch *Kapitel 5.3.16*.

Spezielle Eigenschaften externer Referenzen

Wenn es sich bei der gewählten Blockdefinition um eine externe Referenz handelt, können Sie zusätzlich zu den normalen Eigenschaften dieses Objekts noch auf die *XRefDatabase*-Eigenschaft zugreifen. Diese gibt Ihnen das Datenbankobjekt der externen Referenz zurück. Bevor Sie diese abrufen, können Sie mit der *IsXRef*-Eigenschaft prüfen, ob es sich bei dem angegebenen Block auch wirklich um eine externe Referenz handelt. Gibt diese den Wert *TRUE* zurück, können Sie die Eigenschaft *XRefDatabase* verwenden, andernfalls ist diese nicht initialisiert. Über dieses Objekt erhalten Sie nun Zugriff auf die Auflistungsobjekte der externen Referenz (*Layers*, *Blocks*, *LineTypes* usw.).

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Gibt die Anzahl der in der Blockdefinition enthaltenen Objekte zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>IsLayout</i> (schreibgeschützt)	Bestimmt, ob der angegebene Block ein Layout-Block ist.
<i>IsXRef</i> (schreibgeschützt)	Bestimmt, ob der angegebene Block ein XRef-Block ist.
<i>Name</i>	Legt den Namen des BKS fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>Origin</i>	Bestimmt die Ursprungskoordinaten der Blockdefinition.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>XRefDatabase</i> (schreibgeschützt)	Ruft ein Datenbankobjekt ab, das den Inhalt des XRef-Blocks definiert.

Tabelle 7.49: Weitere Eigenschaften des Block-Objekts

Prozedur	Beschreibung
Delete	Löscht das Objekt aus der AutoCAD-Zeichnung.
GetExtensionDictionary	Ruft das erweiterte Wörterbuch des Objekts ab.
GetXData	Ruft die erweiterten Daten des Objekts ab.
SetXData	Speichert die erweiterten Daten des Objekts.

Tabelle 7.50: Weitere Prozeduren des Block-Objekts

Anwendungsbeispiel

(Beispielprojekt: 07\AcadBlocks.dvb)

Im folgenden Anwendungsbeispiel befassen wir uns mit einem Programm, mit dem der Benutzer die Blöcke der aktuellen Zeichnung verwalten kann. Ziel der Anwendung wird es sein, die Blockdefinition der Zeichnung in einem *ListView*-Steuerelement darzustellen. Sobald der Benutzer einen Eintrag aus dieser Liste ausgewählt hat, soll er die Möglichkeit erhalten, die eventuell vorhandenen Attributdefinitionen des Blocks zu bearbeiten oder zu löschen, neue Attribute hinzuzufügen und einen Beschreibungstext für den Block festzulegen. Außerdem soll er Blöcke importieren oder vorhandene Blöcke löschen können.

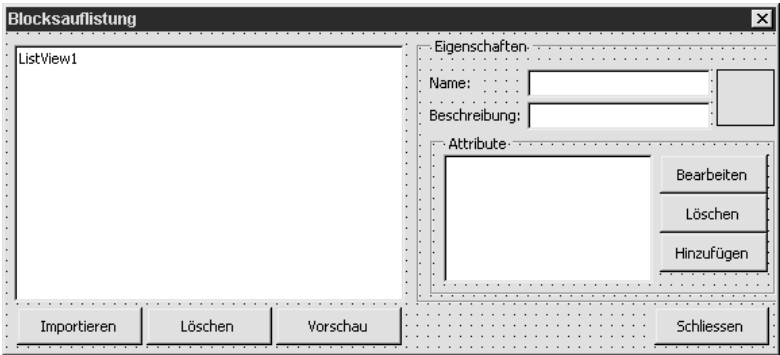


Abbildung 7.47: Blockverwaltungsformular

In das Projekt wurden außerdem zwei zusätzliche Bibliotheken eingebunden. Die erste stellt dem Benutzer einen Farbauswahldialog zur Verfügung. Die zweite ist die Klasse *MyUtility*, die in Kapitel A.3 genauer beschrieben ist.

Startprozedur

In der Startprozedur des Projekts wird zunächst eine neue Instanz der Klasse *MyUtility* erstellt. Anschließend wird die Initialisierungsprozedur *initBlocksFrm* des Formulars und danach das Formular selbst aufgerufen.

```
Public Sub showBlocks()  
Set Util = New MyUtility  
BlocksFrm.initBlocksFrm  
BlocksFrm.Show  
Unload BlocksFrm  
End Sub
```

Initialisierungsprozedur

In der Initialisierungsprozedur des Formulars *BlocksFrm* werden die Namen und Beschreibungen der einzelnen Blockdefinitionen dem *ListView*-Steuerelement hinzugefügt. Für die Beschreibungstexte werden wir uns der erweiterten Daten des jeweiligen Blocks bedienen.

```
Public Sub initBlocksFrm()
Dim LV As MSComctlLib.ListView
Dim xTypeOut As Variant, xDataOut As Variant
Dim BlkDef As AcadBlock
On Error Resume Next
ListView1.ListItems.Clear
For Each BlkDef In ThisDrawing.Blocks
    If Left(BlkDef.Name, 1) <> "*" And InStr(1, BlkDef.Name, "|") _
        = 0 And BlkDef.IsXRef = False Then
```

Handelt es sich bei der Blockdefinition nicht um eine externe Referenz, eine Blockdefinition einer externen Referenz oder einen anonymen Block, kann der Name des Blocks in die erste Spalte des *ListView*-Steuerelements eingetragen werden.

```
Set LV = ListView1.ListItems.Add(, , BlkDef.Name)
```

Den Beschreibungstext des Blocks werden wir in den erweiterten Daten des Blocks unter dem Anwendungsnamen „BlockDef“ speichern. Sie können natürlich auch einen anderen Namen verwenden. Sie müssen in dem Fall dann nur den Namen beim Lesen und Schreiben der erweiterten Daten anpassen. Außerdem müssen Sie den Namen auch nicht unbedingt in der *Applications*-Auflistung anmelden, da es sich hier ja nur um eine Beispielanwendung handelt. Erst wenn Sie komplexere Anwendungen entwickeln, in denen Sie erweiterte Daten verwenden, sollten Sie dies tun. Wir lesen also die erweiterten Daten, die unter dem Anwendungsnamen „BlockDef“ gespeichert wurden.

```
BlkDef.GetXData "BlockDef", xTypeOut, xDataOut
If IsEmpty(xTypeOut) = False Then
```

Konnten welche gefunden, fügen wir den Eintrag des zweiten Felds des Arrays *xDataOut* als zweite Spalte dem neuen *ListView*-Element hinzu.

```
LV.ListSubItems.Add , , xDataOut(1)
Else
```

Andernfalls wird nur die Spalte selbst erzeugt, damit bei späteren Änderungen der Beschreibung auch in dieser Spalte ein Text dargestellt werden kann.

```
LV.ListSubItems.Add
End If
End If
Next
CreateBlockSymbol Application.Preferences.Files.TempFilePath
End Sub
```

Symbole der Blockdefinitionen erzeugen

Mithilfe dieser Prozedur werden wir nun von den einzelnen Blockdefinitionen Symboldateien erzeugen, die dann zur Vorschau der Blockdefinitionen verwendet werden. Dies ist zwar nicht die eleganteste, aber ohne Verwendung von zusätzlichen Bibliotheken die einfachste und

schnellste Methode, Abbildungen der Blockdefinitionen zu erhalten. Als Dateityp der Bilddateien verwenden wir *wmf*. Zum einen lässt sich dieser Bildtyp erheblich schneller erstellen als beispielsweise *bmp*-Dateien und zum anderen unterstützt AutoCAD beim Exportieren in dieses Bildformat die Verwendung von Auswahlätzen.

Der Prozedur können insgesamt zwei Parameter übergeben werden, wobei der erste zwingend erforderlich und der zweite optional ist. Im ersten Parameter soll das Verzeichnis angegeben werden, in dem die Bilddateien der einzelnen Blöcke gespeichert werden sollen (z.B. das temporäre AutoCAD-Verzeichnis). Über die Angabe des zweiten Parameters teilt sich dann die Prozedur in zwei Bereiche auf.

```
Public Function CreateBlockSymbol(PathName As String, Optional _
BlockName)
Dim sSet As AcadSelectionSet
Dim pSelSet(0) As AcadEntity
Dim Pict As IPicture
Const PI = 3.14159265358979
On Error Resume Next
```

Bevor der eigentliche Exportvorgang startet, sind noch einige Vorbereitungen zu treffen. So werden die letzte Position und die Größe des aktuellen Ausschnitts gespeichert und ein neuer Auswahlatz wird erzeugt.

```
LastCtr = ThisDrawing.GetVariable("VIEWCTR")
LastSize = ThisDrawing.GetVariable("VIEWSIZE")
Set sSet = ThisDrawing.SelectionSets.Add("Auswahl")
```

An dieser Stelle teilt sich dann die Prozedur. Wurde der zweite Parameter nicht angegeben, werden alle vorhandenen Blockdefinitionen in einer *For Each...Next*-Schleife nacheinander abgearbeitet. Andernfalls wird nur die im zweiten Parameter angegebene Blockdefinition exportiert. Der Exportvorgang an sich ist bei beiden Varianten derselbe.

```
If IsMissing(BlockName) = True Then
For Each BlkDef In ThisDrawing.Blocks
If Left(BlkDef.Name, 1) <> "*" And BlkDef.IsXRef = False _
Then
```

Nachdem sichergestellt wurde, dass es sich bei dem Block nicht um eine externe Referenz, einen Block einer externen Referenz oder einen anonymen Block handelt, wird eine Blockreferenz mit dem Namen des Blocks in der Mitte des aktuellen Ausschnitts erzeugt. Der Verweis auf diesen Block wird im Array *pSelSet* gespeichert, das anschließend dem Auswahlatz hinzugefügt wird.

```
Set pSelSet(0) = ThisDrawing.ModelSpace.InsertBlock( _
ThisDrawing.GetVariable("VIEWCTR"), BlkDef.Name, 1, _
1, 1, 2 * PI - ThisDrawing.GetVariable("VIEWTWIST"))
sSet.AddItem pSelSet
```

Jetzt müssen wir noch auf einen Ausschnitt zoomen, damit der Block in seiner gesamten Größe sichtbar wird und wir somit ein vollständiges Bild des Blocks beim Export erhalten. Dazu werden die beiden Punkte ermittelt, die ein umgrenzendes Rechteck um die Blockreferenz bilden. Auf diese Koordinaten wird der aktuelle Ausschnitt dann gezoomt.

```
pSelSet(0).GetBoundingBox Min, Max
Application.ZoomWindow Min, Max
```

Jetzt kann die Blockreferenz in eine *wmf*-Datei exportiert werden. Als Dateiname wird einfach der angegebene Pfad zuzüglich des Namens der Blockdefinition festgelegt, so dass Sie beispielsweise folgende Dateien erhalten: *C:\Temp\Block1.wmf* oder *C:\Temp\Block2.wmf* usw.

Da die Dateierweiterung im zweiten Parameter der *Export*-Anweisung festgelegt wurde, braucht diese nicht extra beim Namen festgelegt zu werden. AutoCAD hängt das Kürzel automatisch an die Datei an. Nachdem der Export des Blocks abgeschlossen wurde, muss nur noch die erstellte Blockreferenz gelöscht und der Auswahl Satz geleert werden. Danach wird der Vorgang mit dem nächsten Block wiederholt.

```

        ThisDrawing.Export PathName & BlkDef.Name, "WMF", sSet
        pSelSet(0).Delete
        sSet.Clear
    End If
Next
Else

```

Für einen einzeln angegebenen Block läuft derselbe Vorgang wie in der Schleife ab, nur dass hier eben nur ein bestimmter Block exportiert wird.

```

Set BlkDef = ThisDrawing.Blocks(CStr(BlockName))
If Left(BlkDef.Name, 1) <> "*" And BlkDef.IsXRef = False Then
    Set pSelSet(0) = ThisDrawing.ModelSpace.InsertBlock( _
        ThisDrawing.GetVariable("VIEWCTR"), BlkDef.Name, 1, 1, _
        1, 2 * PI - ThisDrawing.GetVariable("VIEWTWIST"))
    sSet.AddItem pSelSet
    pSelSet(0).GetBoundingBox Min, Max
    Application.ZoomWindow Min, Max
    ThisDrawing.Export PathName & BlkDef.Name, "WMF", sSet
    pSelSet(0).Delete
    sSet.Clear
End If
End If

```

Zum Schluss müssen wir den Ausgangspunkt, der beim Aufruf der Prozedur bestanden hat, wieder herstellen. Dazu wird zunächst der erzeugte Auswahl Satz wieder gelöscht und die Zeichnung auf den ursprünglichen Ausschnitt gezoomt. Die Koordinaten wurden ja in den Variablen *LastCtr* (Zentrum des Ausschnitts) und *LastSize* (Höhe des Ausschnitts) zwischengespeichert.

```

ThisDrawing.SelectionSets("Auswahl").Delete
Application.ZoomCenter LastCtr, LastSize
End Function

```

Auswahl einer Blockdefinition

Sobald der Benutzer einen Eintrag in dem *ListView*-Steuerelement auswählt, werden wir darauf mit folgendem Code reagieren.

```

Private Sub ListView1_ItemClick(ByVal Item As _
    MSComctlLib.ListItem)
On Error Resume Next

```

Als Erstes werden der Name und die Beschreibung der gewählten Blockdefinition in die entsprechenden Textfelder übertragen. Das ist nicht sehr schwierig, da die erste und zweite Spalte des gewählten *ListView*-Elements (Item) diese Informationen bereits enthalten.

```
TextBox1 = Item.Text
TextBox2 = Item.ListSubItems(1).Text
```

Anschließend wird das Bild der Blockdefinition (wurde ja bereits in der Initialisierungsprozedur von allen Blockdefinitionen erzeugt) mit der *LoadPicture*-Eigenschaft in die *Picture*-Eigenschaft des *Image*-Steuerelements geladen.

```
Image1.Picture = LoadPicture(Application.Preferences.Files. _
    TempFilePath & Item.Text & ".wmf")
```

Jetzt müssen wir nur noch die Attributdefinitionen des gewählten Blocks ermitteln. Als Erstes wird eine neue Instanz eines Auflistungsobjekts in der Variablen *mCol* erzeugt. In dieser Auflistung werden wir anschließend die Verweise auf die einzelnen Attributdefinitionen speichern, damit die Ermittlung des ausgewählten Attributs etwas einfacher zu handhaben ist.

```
ListBox1.Clear
Set mCol = New Collection
```

Nun erstellen wir einen Verweis auf den gewählten Block und durchlaufen diesen mittels einer *For Each...Next*-Schleife. Bei jedem Durchlauf wird ein Verweis auf das jeweils nächste grafische Objekt der Blockdefinition abgerufen und in der Variablen *BlkObj* gespeichert. Jetzt brauchen wir nur zu prüfen, ob es sich bei dem jeweiligen Objekt um eine Attributdefinition handelt. Ist dies der Fall, wird der Inhalt der *TagString*-Eigenschaft (Attributbezeichnung) der *ListBox* und der Verweis zur Auflistung *mCol* hinzugefügt.

```
Set ActBlkDef = ThisDrawing.Blocks(Item.Text)
For Each BlkObj In ActBlkDef
    If TypeName(BlkObj) = "IAcadAttribute" Then
        ListBox1.AddItem BlkObj.TagString
        mCol.Add BlkObj
    End If
Next
End Sub
```

Änderung des Blocknamens

Bei der Änderung des Namens der Blockdefinition (durch Änderung des Inhalts der *TextBox1*) gilt es nur eine wichtige Sache zu beachten. Wenn der Block umbenannt wird, müssen wir ebenfalls den Namen der Bilddatei ändern, damit die Zuordnung zu den *Image*-Steuerelementen nicht verloren geht.

```
Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _
    MSForms.ReturnBoolean)
    Dim BlkDef As AcadBlock
    On Error Resume Next
```

Als Erstes speichern wir wieder einen Verweis auf die gewählte Blockdefinition (*Listview.SelectedItem*) in einer Objektvariablen. Anschließend wird der alte Name des Blocks an die Variable *LastName* übergeben.

```
Set BlkDef = ThisDrawing.Blocks(ListView1.SelectedItem.Text)
LastName = BlkDef.Name
```


Jetzt brauchen wir nur den eingegebenen Namen aus der *TextBox* in die *Name*-Eigenschaft des Blocks zu schreiben. Könnte der Block nicht umbenannt werden, also der Name des Blocks stimmt nicht mit der Zeichenfolge aus der *TextBox* überein, wird eine Fehlermeldung ausgegeben, die einen Hinweis darauf liefern könnte, warum dies der Fall ist. Außerdem wird der Vorgang mit dem Setzen des *Cancel*-Parameters auf *TRUE* abgebrochen.

```
BlkDef.Name = TextBox1
If UCase(BlkDef.Name) <> UCase(TextBox1) Then
    MsgBox Err.Description
    Cancel = True
Else
```

Wurde hingegen der Blockname geändert, muss auch der Eintrag im *ListView*-Steuerelement geändert und der Name der entsprechenden *wmf*-Datei geändert werden. Da unter Standard-VBA kein *Rename* oder Ähnliches zur Verfügung steht, wird die alte Datei einfach mittels der *FileCopy*-Anweisung in eine Datei mit dem neuen Namen kopiert und anschließend wird die Ursprungsdatei gelöscht (*Kill*-Anweisung).

```
    ListView1.SelectedItem.Text = BlkDef.Name
    FileCopy Application.Preferences.Files.TempFilePath & _
        LastName & ".wmf", Application.Preferences.Files. _
        TempFilePath & BlkDef.Name & ".wmf"
    Kill Application.Preferences.Files.TempFilePath & _
        LastName & ".wmf"
End If
End Sub
```

Änderung des Beschreibungstextes

Die Änderung des Beschreibungstextes der ausgewählten Blockdefinitionen wird durch Änderung der Zeichenfolge im *TextBox2*-Steuerelement ausgelöst, genauer gesagt durch die Ereignisprozedur *BeforeUpdate*. Dieser Beschreibungstext wird in den erweiterten Daten der jeweiligen Blockdefinition mithilfe der Anweisung *SetXData* geschrieben. Zunächst werden hierfür die beiden Arrays *xType* und *xData* deklariert. Im ersten Feld der Arrays werden der Anwendungsname und der dazugehörige Typ festgelegt, im zweiten der Beschreibungstext und der Typ 1000, wodurch dieser Datensatz als Zeichenfolge interpretiert wird.

```
Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim xType(1) As Integer, xData(1) As Variant
Dim BlkDef As AcadBlock
On Error Resume Next
xType(0) = 1001
xData(0) = "BlockDef"
xType(1) = 1000
xData(1) = TextBox2
```

Anschließend wird das *ListView*-Steuerelement Eintrag für Eintrag durchlaufen und kontrolliert, ob das jeweils angesprochene durch den Benutzer selektiert wurde.

```
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
```

Ist dies der Fall, wird die dazugehörige Blockdefinition ermittelt und die erweiterten Daten werden über die Anweisung *SetXData* in das Objekt geschrieben.

```

Set BlkDef = ThisDrawing.Blocks(ListView1.ListItems(i).Text)
BlkDef.SetXData xType, xData
BlkDef.GetXData "BlockDef", xTypeOut, xDataOut
If IsEmpty(xTypeOut) = False Then
    ListView1.ListItems(i).ListSubItems(1).Text = xDataOut(1)
Else
    ListView1.ListItems(i).ListSubItems(1).Text = ""
End If
End If
Next
TextBox2 = ListView1.SelectedItem.ListSubItems(1).Text
End Sub

```

Blockdefinition importieren

Über die Schaltfläche IMPORTIEREN wird es dem Benutzer ermöglicht, eine Zeichnung als Blockdefinition in AutoCAD einzulesen. Wie das funktioniert, wurde ja zuvor bereits kurz erläutert. Als AuswahlDialog nutzen wir wieder den ÖFFNEN-Standarddialog des Windows-Steuerelements *CommonDialog*. Die notwendigen Einstellungen, wie Titel, Filter und *Cancel-Error*-Eigenschaft wurden bereits zur Entwurfszeit festgelegt, so dass hier nur der ÖFFNEN-Dialog mit der *ShowOpen*-Anweisung aufgerufen werden muss.

```

Private Sub CommandButton1_Click()
On Error Resume Next
ComDialog.ShowOpen
If Err = 0 Then

```

Hat der Benutzer eine Zeichnungsdatei ausgewählt, wird mittels der *InsertBlock*-Anweisung eine neue Blockreferenz mit dem Inhalt der *FileName*-Eigenschaft erzeugt.

```

Set NewBlkRef = ThisDrawing.ModelSpace.InsertBlock( _
    ThisDrawing.GetVariable("VIEWCTR"), ComDialog.FileName, 1, _
    1, 1, 0)
If TypeName(NewBlkRef) = "Nothing" Then Exit Sub

```

War die Erstellung der Referenz erfolgreich, müssen wir zunächst ein neues Symbol des Blocks erzeugen und anschließend prüfen, ob es sich um eine neue Blockdefinition handelt oder ob eine vorhandene neu definiert wurde.

```

CreateBlockSymbol Application.Preferences.Files.TempFilePath, _
    NewBlkRef.Name
If BlockIsInList(NewBlkRef.Name) = False Then

```

Handelt es sich um eine neue Blockdefinition, wird diese dem *ListView*-Steuerelement hinzugefügt.

```

Set BlkDef = ThisDrawing.Blocks(NewBlkRef.Name)
Set LV = ListView1.ListItems.Add(, , BlkDef.Name)
BlkDef.GetXData "BlockDef", xTypeOut, xDataOut
If IsEmpty(xTypeOut) = False Then
    LV.ListSubItems.Add(, , xDataOut(1))
Else

```

```

        LV.ListSubItems.Add
    End If
    CreateBlockSymbol Application.Preferences.Files. _
        TempFilePath, BlkDef.Name
End If
NewBlkRef.Delete
End If
End Sub

```

Blockdefinitionen löschen

Das Löschen einzelner Blockdefinitionen aus der Auflistung stellt Sie vor keine großen Probleme. Im ersten Schritt wird das *ListView*-Steuerelement Zeile für Zeile durchlaufen und falls ein Eintrag selektiert wurde, was die *Selected*-Eigenschaft wiedergibt, ein Verweis auf die Definition ermittelt. Anschließend wird versucht, die Blockdefinition zu löschen. Ob das gelungen ist, zeigt der Aufruf der *initBlockFrm*-Anweisung. Diese liest nach Beendigung des Durchlaufs alle Blockdefinition erneut in das *ListView*-Steuerelement ein.

```

Private Sub CommandButton4_Click()
On Error Resume Next
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        Set BlkDef = ThisDrawing.Blocks(ListView1.ListItems(i).Text)
        BlkDef.Delete
    End If
Next
initBlocksFrm
End Sub

```

Blockvorschaudialog

Im Blockvorschaudialog erhält der Benutzer eine optische Übersicht über die Blockdefinitionen der Zeichnung. Da dies sehr flexibel geschehen muss, wir wissen ja im Voraus nicht, wie viele Blockdefinitionen die Zeichnung enthält, wird der Inhalt des Formulars erst zur Laufzeit erstellt.

Pro Bilddatei (wurden ja bereits beim Start des Programms erstellt) benötigen wir ein *Image*- und ein *Label*-Steuerelement. Im *Image*-Steuerelement wird die Bilddatei des jeweiligen Blocks abgebildet und im Label entweder der Name des Blocks oder, falls vorhanden, der Beschreibungstext. Damit die Bilder nicht genau am Rand des Rahmens, in dem sie dargestellt werden, positioniert werden, legen wir als Ausgangsposition jeweils einen Wert von 10 Pixel für den Abstand vom oberen und vom linken Rahmenrand fest.

```

Private Sub CommandButton8_Click()
Dim NewImage As MSForms.Image
Dim NewLabel As MSForms.Label
On Error Resume Next
LastTop = 10
LastLeft = 10

```

Jetzt brauchen wir nur alle Elemente des *ListView*-Steuerelements zu durchlaufen und für jeden Eintrag ein *Image*- und ein *Label*-Steuerelement über die *Controls*-Auflistung des Rahmens zu laden (siehe Kapitel 3.4.2).

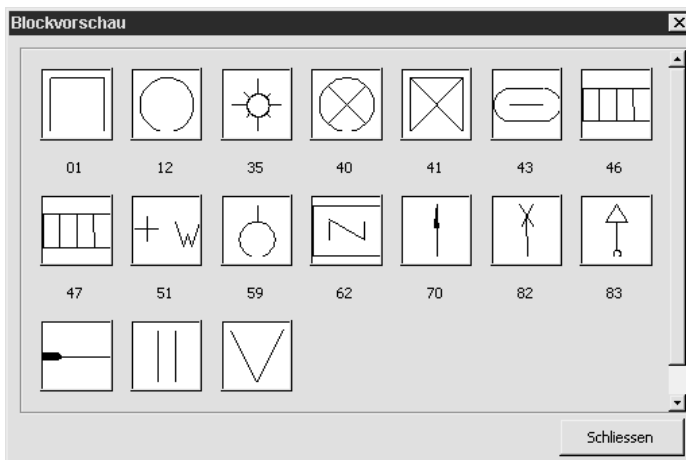


Abbildung 7.48: Beispiel einer Blockvorschau

```
For i = 1 To ListView1.ListItems.Count
```

Für die *Image*-Steuerelemente wird die Programm-ID „Forms.Image.1“ verwendet und mittels der *Add*-Anweisung erzeugt. Der Verweis auf das Steuerelement wird in der Variablen *NewImage* gespeichert. Mithilfe dieser Objektvariablen passen wir nun einige Eigenschaften des neuen Steuerelements an (Breite, Höhe, Bildarstellungsmodus und Hintergrundfarbe), wobei als Hintergrundfarbe die des AutoCAD-Zeichnungsfensters verwendet wird. Diese muss ja mit der Farbe der Bilddateien übereinstimmen.

```
Set NewImage = PreviewFrm.Frame1.Controls.Add("Forms.Image.1")
NewImage.Height = 40
NewImage.Width = 40
NewImage.PictureSizeMode = fmPictureSizeModeZoom
NewImage.BackColor = Application.Preferences.Display. _
    GraphicsWinModelBackgrndColor
```

Anschließend wird die entsprechende Bilddatei in die *Picture*-Eigenschaft des Steuerelements geladen und die neue Position von *Image* festgelegt.

```
Set NewImage.Picture = LoadPicture(Application.Preferences. _
    Files.TempFilePath & ListView1.ListItems(i).Text & ".wmf")
NewImage.Left = LastLeft
NewImage.top = LastTop
```

Für das Label gilt Ähnliches. Zunächst wird ein neues Label der *Controls*-Auflistung des Rahmens hinzugefügt. Dann wird es unterhalb des *Image* positioniert und die entsprechende Beschriftung wird in die *Caption*-Eigenschaft geschrieben.

```
Set NewLabel = PreviewFrm.Frame1.Controls.Add("Forms.Label.1")
NewLabel.Left = LastLeft
NewLabel.top = LastTop + NewImage.Height + 10
NewLabel.Caption = IIf(ListView1.ListItems(i).ListSubItems( _
    1).Text = "", ListView1.ListItems(i).Text, _
    ListView1.ListItems(i).ListSubItems(1).Text)
```

Jetzt folgen noch einige Formatierungen am Label (Ausrichtung der Beschriftung sowie die Festlegung der Höhe und Breite).

```
NewLabel.TextAlign = fmTextAlignCenter
NewLabel.Height = 25
NewLabel.Width = 40
```

Damit die Bilder nicht im unsichtbaren Bereich des Rahmens verschwinden, falls der sichtbare Bereich nicht ausreichen sollte, muss die *ScrollHeight*-Eigenschaft des Rahmens entsprechend angepasst werden.

```
PreviewFrm.Frame1.ScrollHeight = LastTop + NewImage.Height + 30
```

Zum Schluss folgt dann nur noch die Bestimmung der Position des nächsten *Image*-Steuerelements. Sollte die Breite des Rahmens für ein neues Steuerelement nicht mehr ausreichen, wird in der nächsten Spalte fortgesetzt.

```
If LastLeft + 50 > PreviewFrm.Frame1.Width - 20 Then
    LastTop = LastTop + NewImage.Height + 30
    LastLeft = 10
Else
```

Andernfalls wird dieses Steuerelement rechts neben dem vorangegangenen Steuerelement positioniert.

```
    LastLeft = LastLeft + 50
    LastTop = NewImage.top
End If
Next
```

Nach dem Abschluss der Vorbereitungen wird das Formular, das diese Abbildungen anzeigen soll, eingeblendet. Für diese Formular ergibt sich eine Darstellung wie in Abbildung 7.48, natürlich mit den Bildern der Blöcke der entsprechenden Blockauflistung.

```
PreviewFrm.Show
End Sub
```

7.11 Benannte Ansichten

AutoCAD bietet dem Benutzer die Möglichkeit, einen Zeichnungsausschnitt zu benennen, um diesen dann zu einem späteren Zeitpunkt wiederherzustellen. AutoCAD speichert verschiedene Ansichten für Modellbereich und Papierbereich. Diese werden in der *Views*-Auflistung des *Document*-Objekts verwaltet und können mit den für Auflistungen üblichen Eigenschaften und Methoden verwaltet werden. Zugriff auf die Blockauflistung erhalten sie über das jeweilige Dokumentobjekt.

```
Set AllViews = Views
```

bzw.

```
Set AllViews = ThisDrawing.Views
```

für das aktuelle Dokument oder

```
Set AllViews = Application.Documents(n).Views
```

für die *Views*-Auflistung einer beliebigen Zeichnung, die in der aktuellen AutoCAD-Sitzung geladen wurde.

Benannte Ansichten erstellen/vorhandene Ansichten abrufen

Um der Auflistung und damit der Zeichnung eine benannte Ansicht hinzuzufügen, können Sie wie gehabt die *Add*-Anweisung des Auflistungsobjekts verwenden. Als Parameter brauchen Sie hierfür nur den Namen der Ansicht festzulegen, wobei dieser wieder eindeutig sein muss, also in der *Views*-Auflistung noch nicht vorhanden sein darf.

```
sValue = InputBox("Legen Sie den Namen der neuen Ansicht fest.", _  
    "neuen Ansicht")  
If sValue = "" Then Exit Sub  
Views.Add sValue
```

Damit wäre zwar die Ansicht der Auflistung hinzugefügt, allerdings müssen Sie nun einige Einstellungen an der Ansicht vornehmen, damit diese auch Sinn macht. Dazu können Sie entweder den Verweis auf die Ansicht, die durch die *Add*-Funktion zurückgegeben wird, verwenden oder mit der *Item*-Funktion eine benannte Ansicht nach der üblichen Vorgehensweise abrufen.

```
Set ActView = Views(i)
```

Oder den Namen des gesuchten Blocks verwenden, was meist der Fall sein dürfte.

```
Set ActView = Views("Ansicht 1")
```

In jedem Fall erhalten Sie als Ergebnis einen Verweis auf ein benanntes Ansichtsfenster, das in VBA durch die Klasse *AcadView* dargestellt wird.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.

Tabelle 7.51: Eigenschaften der *Views*-Auflistung

Eigenschaft	Beschreibung
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.51: Eigenschaften der Views-Auflistung (Forts.)

Prozedur	Beschreibung
<i>Add</i>	Diese Anweisung fügt eine benannte Ansicht der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.52: Prozeduren der Views-Auflistung

Einzelne benannte Ansichten verwalten

Nachdem nun die Ansicht hinzugefügt wurde, müssen noch einige Einstellungen an diesem Objekt vorgenommen werden, damit die benannte Ansicht auch ihren Zweck erfüllt, nämlich die Wiederherstellung eines Zeichnungsausschnitts. Dazu stellt die Klasse *AcadView* einige Eigenschaften zur Verfügung, die zunächst einmal beschrieben werden sollen. Leider hat man bei der Umsetzung dieses Objekts in die Typenbibliothek nicht alle Eigenschaften und Methoden eines benannten Ausschnitts berücksichtigt, so dass Sie häufig auf die *SendCommand*-Anweisung zurückgreifen müssen.

Außerdem gilt es, zwischen zwei Ausrichtungen des Ausschnitts zu unterscheiden: zum einen die 2D-Ansicht, bei der der Mittelpunkt die Position des Ausschnitts bestimmte, zum anderen die perspektivische Ansicht, für die sich dieses Objekt eher eignet, da für diese Darstellung alle notwendigen Eigenschaften vorhanden sind.

In der *Center*-Eigenschaft wird der Mittelpunkt der Ansicht gespeichert. Im Gegensatz zu anderen Koordinatenarrays benötigt *Center* allerdings nur zweidimensionale Koordinaten (X und Y). Sie müssen also darauf achten, dass ein eventuelles 3D-Array zu einem 2D-Array zurückdimensioniert wird, bevor Sie es dieser Eigenschaft zuweisen.

```
Dim Point
Point = Utility.GetPoint(, "Zentrumspunkt der Ansicht:")
ReDim Preserve Point(1) As Double
ActView.Center = Point
```

Die perspektivische Ausrichtung des Ansichtsfensters wird durch einen Richtungsvektor, der in der *Direction*-Eigenschaft gespeichert ist, und den Zielpunkt der Eigenschaft *Target* bestimmt. Beide Eigenschaften zusammen legen diese perspektivische Ausrichtung des Ausschnitts fest. Sie müssen sich einfach eine Linie vorstellen, die vom Zielpunkt mit den Vektorangaben der Eigenschaft *Direction* gezogen wird. Diese Richtung bestimmt dann die Blickrichtung des Ausschnitts in Richtung des Zielpunkts.

```
Dim Point
Point = ActView.Target
Target = Util.GetPoint(Point, "neues Ziel wählen:")
If TypeName(Target) = "Double()" Then
    ActView.Target = Target
End If
Direction = Util.GetPoint(Target, "Ausrichtung wählen:")
If TypeName(Direction) = "Double()" Then
    Direction(0) = Direction(0) - Target(0)
    Direction(1) = Direction(1) - Target(1)
    Direction(2) = Direction(2) - Target(2)
    ActView.Direction = Direction
End If
```

Die Höhe und die Breite des Ausschnitts werden in den Eigenschaften *Height* und *Width* gespeichert.

Um eine gespeicherte Ansicht wiederherzustellen, müssen Sie den Umweg über ein Ansichtsfensterobjekt gehen. In erster Linie könnte dies das aktuelle Ansichtsfenster der Zeichnung sein. Sie können aber auch jedes andere Ansichtsfenster verwenden, nur dass der Ausschnitt dann eben in diesem wiederhergestellt wird. Zu diesem Zweck stellt das entsprechende *View-Port*-Objekt die Anweisung *SetView* zur Verfügung. Da diese Anweisung das komplette *View*-Objekt und nicht nur dessen Namen verlangt, müssen Sie zunächst einen Verweis auf den Ausschnitt ermitteln und diesen dann der *SetView*-Anweisung übergeben (siehe auch *Kapitel 7.12*). Um beispielsweise den Ausschnitt mit dem Namen „Ansicht1“ im aktuellen Ansichtsfenster wiederherzustellen, könnten Sie folgende Codezeile verwenden.

```
ActiveViewport.SetView Views("Ansicht1")
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Center</i>	Legt den Zentrums punkt der benannten Ansicht fest.
<i>Direction</i>	Legt die Ausrichtung für eine 3D-Visualisierung der Zeichnung fest.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Height</i>	Bestimmt die Höhe der benannten Ansicht.
<i>Name</i>	Legt den Namen der benannten Ansicht fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.

Tabelle 7.53: Eigenschaften des View-Objekts

Eigenschaft	Beschreibung
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>Target</i>	Bestimmt den Zielpunkt für die benannte Ansicht.
<i>Width</i>	Bestimmt die Breite der benannten Ansicht.

Tabelle 7.53: Eigenschaften des View-Objekts (Forts.)

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.54: Prozeduren des View-Objekts

Anwendungsbeispiel

(Beispielprojekt: 07\AcadView.dvb)

Im nun folgenden Anwendungsbeispiel werden wir uns in erster Linie mit der perspektivischen Ansicht befassen. Die gesamte Verwaltung läuft wieder über ein Formular, in dem zunächst die vorhandenen Ansichten in einem *ListView*-Steuerelement aufgelistet werden. Klickt dann der Benutzer eine bestimmte Ansicht in dieser Liste an, werden in den einzelnen Steuerelementen die Informationen zu der gewählten Ansicht angezeigt. Der Benutzer bekommt nun die Möglichkeit, einzelne Daten dieser Ansicht anzupassen.

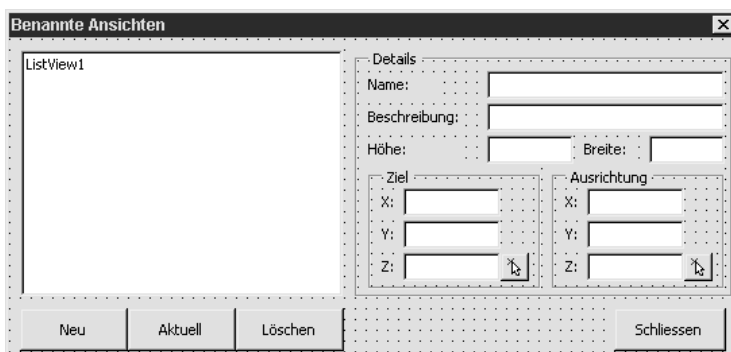


Abbildung 7.49: Formular des Anwendungsbeispiels

Startprozedur

Die Startprozedur dieses kleinen Anwendungsprogramms ist wieder nicht allzu umfangreich. Zunächst wird eine neue Instanz der Klasse *MyUtility* erzeugt, dann die Initialisierungsprozedur des Formulars ausgeführt und schließlich das Formular gestartet.

```
Public Sub showViews()
On Error Resume Next
Set Util = New MyUtility
ViewsFrm.InitViewsFrm
ViewsFrm.Show
End Sub
```

Initialisierungsprozedur

In der Initialisierungsprozedur werden alle vorhandenen Ansichten dem *ListView*-Steuerelement hinzugefügt. Außerdem werden die dazugehörigen Beschreibungstexte, falls vorhanden, aus den erweiterten Daten der jeweiligen Ansicht ermittelt und in der zweiten Spalte eingetragen.

```
Public Sub InitViewsFrm()
Dim ActView As AcadView
Dim xData, xType
Dim LV As MSComctlLib.ListView
On Error Resume Next
Set ListView1.SmallIcons = ImageList1
ListView1.ListItems.Clear
For Each ActView In ThisDrawing.Views
Set LV = ListView1.ListItems.Add(, , ActView.Name)
ActView.GetXData "Description", xType, xData
If IsEmpty(xData) = False Then
LV.ListSubItems.Add , , xData(1)
Else
LV.ListSubItems.Add
End If
Next
Set ListView1.SelectedItem = ListView1.ListItems(1)
ListView1_ItemClick ListView1.SelectedItem
End Sub
```

Auswahl einer Ansicht

Wurde eine Ansicht im *ListView*-Steuerelement gewählt, sei es durch den Benutzer oder durch Aufruf der Prozedur aus dem Code heraus, wird mit dem Namen der gewählten Ansicht (wurde ja der Text-Eigenschaft des *ListView*-Elements hinzugefügt) das Ansichtsobjekt ermittelt.

```
Public Sub ListView1_ItemClick(ByVal Item As MSComctlLib.ListViewItem)
On Error Resume Next
Set ActView = ThisDrawing.Views(Item.Text)
```

Anschließend werden nun die einzelnen Eigenschaften dieses Objekts an die entsprechenden Steuerelemente übertragen.

```
VNameTxt = ActView.Name 'Name der Ansicht
Description = Item.ListSubItems(1).Text 'beschreibung der Ansicht
HeightTxt = Util.RealToString(ActView.Height) 'Höhe
```

```

WidthTxt = Util.RealToString(ActView.Width) 'Breite
Target = ActView.Target 'Zielpunktkoordinaten lesen
TargetTxt1 = Util.RealToString(Target(0)) 'X des Zielpunkts
TargetTxt2 = Util.RealToString(Target(1)) 'Y des Zielpunkts
TargetTxt3 = Util.RealToString(Target(2)) 'Z des Zielpunkts
Direction = ActView.Direction 'Ausrichtungsvektoren lesen
DirectionTxt1 = Util.RealToString(Direction(0)) 'X der Ausrichtung
DirectionTxt2 = Util.RealToString(Direction(1)) 'Y der Ausrichtung
DirectionTxt3 = Util.RealToString(Direction(2)) 'Z der Ausrichtung
End Sub

```

Schaltfläche „Neu“

Über diese Schaltfläche kann der Benutzer eine neue Ansicht der aktuellen Zeichnung hinzufügen. Als Erstes müssen Sie hierfür den Namen dieser Ansicht durch den Benutzer festlegen lassen.

```

Private Sub CommandButton2_Click()
Dim ActView As AcadView, ViewName As String
Dim Center
On Error Resume Next
ViewName = InputBox("Ausschnitt speichern unter", _
    "neuer Ausschnitt")
If ViewName = "" Then Exit Sub

```

Hat dieser eine Zeichenfolge eingegeben, wird mithilfe der *Add*-Anweisung eine neue Ansicht erzeugt und der Verweis auf diese Ansicht in der Variablen *ActView* gespeichert. Falls die Erstellung fehlgeschlagen ist (der Name der Ansicht war z.B. schon vergeben), wird eine Fehlermeldung ausgegeben, die die Ursache des Scheiterns beschreibt, und die Funktion vorzeitig verlassen.

```

Set ActView = ThisDrawing.Views.Add(ViewName)
If TypeName(ActView) = "Nothing" Then
    If Err<>0 Then MsgBox Err.Description
    Exit Sub
End If

```

Sollte allerdings alles funktioniert haben, ordnen wir als Zentrumspunkt dieser Ansicht das Zentrum des aktuellen Ansichtsfensters zu. Sie könnten auch die *Direction*- und *Target*-Eigenschaften des Ansichtsfensters verwenden:

```

Center = ThisDrawing.GetVariable("VIEWCTR")
ReDim Preserve Center(1) As Double
ActView.Center = Center
ActView.Height = ThisDrawing.GetVariable("VIEWSIZE")

```

Abschließend braucht dann nur noch der Name der neuen Ansicht dem *ListView*-Steuerelement hinzugefügt zu werden.

```

Set LV = ListView1.ListItems.Add(, , ActView.Name)
Set ListView1.SelectedItem = LV
ListView1_ItemClick LV
End Sub

```

Ansicht umbenennen

Das Umbenennen der gewählten Ansicht erfolgt durch die Änderung des Eintrags im entsprechenden Textfeld durch den Benutzer. In der dazugehörigen Ereignisprozedur wird dann nur diese Zeichenfolge an die *Name*-Eigenschaft der gewählten Ansicht übergeben.

```
Private Sub VNameTxt_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
ActView.Name = VNameTxt
```

Jetzt müssen wir nur das Ergebnis der Umbenennung auswerten. Hat alles geklappt, was wir daran erkennen, dass der Inhalt der *Name*-Eigenschaft mit dem des Textfelds übereinstimmt, übertragen wir diesen Namen ebenfalls an die *Text*-Eigenschaft des ausgewählten *ListView*-Eintrags.

```
If ActView.Name = VNameTxt Then
    ListView1.SelectedItem.Text = ActView.Name
Else
    MsgBox Err.Description
End If
End Sub
```

Beschreibungstext ändern

Die Änderung des Beschreibungstextes der gewählten Ansicht wird, wie im vorherigen Abschnitt, in den erweiterten Daten abgelegt. Zunächst erfolgt hierfür wieder die Deklaration der beiden Arrays *xType* und *xData*. Im ersten Feld der Arrays werden der Anwendungsname und der dazugehörige Typ festgelegt, im zweiten der Beschreibungstext und der Typ 1000, wodurch dieser Datensatz als Zeichenfolge interpretiert wird.

```
Private Sub Description_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim xType(1) As Integer, xData(1) As Variant
On Error Resume Next
xType(0) = 1001
xData(0) = "Description"
xType(1) = 1000
xData(1) = Description
```

Anschließend werden diese Arrays mittels der *SetXData*-Anweisung in die erweiterten Daten des Objekts geschrieben.

```
ActView.SetXData xType, xData
```

Jetzt brauchen wir nur die erweiterten Daten wieder auszulesen und der zweiten Spalte des gewählten *ListView*-Eintrags zuzuordnen. Damit schlagen wir gleich zwei Fliegen mit einer Klappe. Zum einen wurde geprüft, ob alles geklappt hat, und zum anderen wird gleich die entsprechende Spalte des gewählten Elements aktualisiert.

```
ActView.GetXData "Description", xTypeOut, xDataOut
If IsEmpty(xTypeOut) = False Then
    ListView1.SelectedItem.ListSubItems(1).Text = xDataOut(1)
Else
    ListView1.SelectedItem.ListSubItems(1).Text = ""
```

```
End If
ListView1_ItemClick ListView1.SelectedItem
End Sub
```

Höhe und Breite der Ansicht ändern

Um die Höhe und/oder Breite der gewählten Ansicht zu ändern, muss der Benutzer nur die entsprechenden Werte in den dafür zuständigen Textfeldern anpassen. In den jeweiligen Ereignisprozeduren dieser Textfelder werden dann die Höhe und/oder Breite der Ansicht angepasst und die Inhalte der Textfelder aktualisiert.

```
Private Sub HeightTxt_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
ActView.Height = Util.DistanceToReal(HeightTxt)
HeightTxt = Util.RealToString(ActView.Height)
End Sub

Private Sub WidthTxt_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
ActView.Width = Util.DistanceToReal(WidthTxt)
WidthTxt = Util.RealToString(ActView.Width)
End Sub
```

Zielpunkt in der Zeichnung wählen

Um einen Zielpunkt in der Zeichnung auszuwählen, muss der Benutzer nur auf das Symbol neben den Textfeldern klicken, die die Koordinaten des aktuellen Zielpunkts der gewählten Ansicht anzeigen. In der *Click*-Ereignisprozedur wird dann der nachfolgend beschriebene Code ausgeführt.

Nach Prüfung der *SelectedItem*-Eigenschaft (zeigt an, ob ein Eintrag gewählt wurde), blenden wir das Formular vorübergehend aus, damit der Benutzer einen neuen Zielpunkt in der Zeichnung wählen kann.

```
Private Sub DirectionBtn_Click()
If TypeName(ListView1.SelectedItem) = "Nothing" Then Exit Sub
Me.Hide
```

Damit der Benutzer einen Bezug zum ursprünglichen Zielpunkt erhält, übergeben wir diese Koordinaten an die Variable *Point*. Anschließend kann der Benutzer dann den Punkt mittels der *GetPoint*-Funktion in der Zeichnung auswählen.

```
Point = ActView.Target
Target = Util.GetPoint(Point, "neues Ziel wählen:")
If TypeName(Target) = "Double()" Then
```

Hat er einen gültigen Punkt gewählt, setzen wir diesen in der *Target*-Eigenschaft ein und rufen die *ListLiew1_ItemClick*-Prozedur auf (hat die gleiche Wirkung, wie wenn der Benutzer den Eintrag nochmals angeklickt hätte). Daraufhin werden die Einträge in den Textfeldern aktualisiert.

```
    ActView.Target = Target
    ListView1_ItemClick ListView1.SelectedItem
End If
Me.Show
End Sub
```

Ausrichtung in der Zeichnung wählen

Die Wahl der Ausrichtung durch den Benutzer in der Zeichnung läuft im Wesentlichen genauso ab wie die Auswahl des Zielpunkts, nur dass nach der Auswahl eines Punkts nicht die direkt gewählten Koordinaten verwendet werden.

```
Private Sub TargetBtn_Click()
Dim Point
If TypeName(ListView1.SelectedItem) = "Nothing" Then Exit Sub
Me.Hide
Point = ActView.Target
Direction = Util.GetPoint(Point, "Ausrichtung wählen:")
If TypeName(Direction) = "Double()" Then
```

Hier müssen wir darauf achten, dass es sich bei der *Direction*-Eigenschaft um Vektorangaben handelt, aus denen die Richtung bestimmt wird.

```
Direction(0) = Direction(0) - Point(0)
Direction(1) = Direction(1) - Point(1)
Direction(2) = Direction(2) - Point(2)
ActView.Direction = Direction
ListView1_ItemClick ListView1.SelectedItem
End If
Me.Show
End Sub
```

Schaltfläche „Aktuell“

Die Wiederherstellung einer Ansicht kann der Benutzer durch Auswahl einer Ansicht im *ListView*-Steuerelement und anschließender Betätigung der Schaltfläche AKTUELL auslösen. Dabei wird zunächst das Formular ausgeblendet und die gewählte Ansicht für das aktuelle Ansichtsfenster der Zeichnung mittels der *SetView*-Anweisung wiederhergestellt.

```
Private Sub CommandButton3_Click()
On Error Resume Next
Me.Hide
ActiveViewport.SetView Views(ListView1.SelectedItem.Text)
```

Um danach diese Ansicht als wiederhergestellt zu markieren, setzen wir das Symbol des *ImageList*-Steuerelements in die *SmallIcon*-Eigenschaft des ausgewählten Elements. Zuvor müssen allerdings die Symbole der anderen Listenelemente zurückgesetzt werden, damit nicht plötzlich zwei Ansichten als wiederhergestellt angezeigt werden.

```
For i = 1 To ListView1.ListItems.Count
ListView1.ListItems(i).SmallIcon = 0
Next
ListView1.SelectedItem.SmallIcon = 1
Me.Show
End Sub
```

Schaltfläche „Löschen“

Über diese Schaltfläche kann der Benutzer die im *ListView*-Steuerelement ausgewählten Ansichten löschen. Der Vorgang an sich ist nicht schwierig. Hier werden nur alle Elemente des *ListView*-Steuerelements nacheinander durchlaufen und falls eines selektiert wurde (*Selected* =

True), wird mit der *Text*-Eigenschaft des gewählten Eintrags ein Verweis auf die Ansicht abgerufen und im gleichen Atemzug versucht, diese Ansicht mit der *Delete*-Anweisung zu entfernen. Zum Schluss wird dann nur die Liste durch Aufruf der *initViewsFrm*-Prozedur aktualisiert.

```
Private Sub CommandButton4_Click()
On Error Resume Next
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        ThisDrawing.Views(ListView1.ListItems(i).Text).Delete
    End If
Next
InitViewsFrm
End Sub
```

7.12 Benannte Ansichtsfenster

Mithilfe der Ansichtsfenster können Sie einen Bereich der Zeichnung in einem Layout darstellen. Diese Ansichtsfenster unterteilen sich in feste Ansichtsfenster für den Modell- und die verschiebbaren Ansichtsfenster für den Papierbereich. Die festen Ansichtsfenster für den Modellbereich werden in der *Viewports*-Auflistung verwaltet, die für den Papierbereich als einzelne Objekte im *PaperSpace*-Objekt. Zunächst befassen wir uns mit den festen Ansichtsfenstern. Wenn eine Zeichnung erzeugt wurde, wird erst nur eines angezeigt. Dieses entspricht der *ActiveViewport*-Eigenschaft des *Document*-Objekts und entspricht einem einzelnen Ansichtsfenster. Nun können diese festen Ansichtsfenster auch mehrfach aufgeteilt werden, so dass sich im Modelllayout verschiedene Ansichten gleichzeitig darstellen lassen. Um zu verstehen, wie AutoCAD diese Ansichtsfenster verwaltet, wird zunächst die *Viewports*-Auflistung beschrieben.

7.12.1 Ansichtsfenster verwalten

In der *Viewports*-Auflistung werden alle festen Ansichtsfenster der entsprechenden Zeichnung verwaltet. Im Gegensatz zu anderen Auflistungen besitzt diese Auflistung einige Besonderheiten, durch die die Darstellung mehrerer Ansichtsfenster im Modellbereich erst möglich wird. Die erste Besonderheit ist das Ansichtsfenster mit dem Namen „*Active“. Dieses stellt das aktuell eingestellte Ansichtsfenster dar. Eine andere Besonderheit der *Viewports*-Auflistung ist, dass hier mehrere Ansichtsfenster gleichen Namens verwaltet werden. Das kommt daher, dass mehrere feste Ansichtsfenster im Modellbereich dargestellt werden können. Für jedes dieser Ansichtsfenster wird nun ein Eintrag in der *Viewports*-Auflistung angelegt. Da Ansichtsfenster, die gleichzeitig dargestellt werden, denselben Namen besitzen, kommt es eben vor, dass in der *Viewports*-Auflistung mehrere Eintragungen gleichen Namens gespeichert werden.

Zugriff auf die *Viewports*-Auflistung erhalten Sie über das entsprechende *Document*-Objekt:

```
Set AllVPorts = Viewports
```

bzw.

```
Set AllVPorts = ThisDrawing.Viewports
```

für das aktuelle Dokument oder

```
Set AllVPs = Application.Documents(n).Viewports
```

für die *Viewports*-Auflistung einer beliebigen Zeichnung, die in der aktuellen AutoCAD-Sitzung geladen wurde.

Ansichtsfensterkonfigurationen verwalten

Die *Viewports*-Auflistung können Sie nun genauso behandeln, wie jede andere Auflistung auch. Sie können weitere Ansichtsfensterkonfigurationen der Auflistung mit der *Add*-Anweisung hinzufügen oder über die *Item*-Funktion eine vorhandene abrufen. Wenn Sie ein Ansichtsfenster hinzufügen wollen, von dem bereits eines oder mehrere in der Auflistung existieren, erhalten Sie zwar einen Verweis auf das Ansichtsfenster, dieser gilt dann aber dem ersten Ansichtsfenster dieses Namens, das in der Auflistung gefunden wird.

```
sValue = InputBox("Namen der neuen Ansichtsfensterkonfiguration", _  
    "neues Ansichtsfenster")  
If sValue = "" Then Exit Sub  
Viewports.Add sValue
```

Genauso verhält es sich bei der *Item*-Funktion. Auch diese gibt den Verweis auf das erste gefundene Ansichtsfenster an, wenn Sie als Index den Namen angeben.

```
Set ActVP = Viewports("*Active")
```

Also sollten Sie bei der Verwaltung der *Viewports*-Auflistung die direkte Positionsangabe verwenden:

```
Set ActVP = Viewports(i)
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück..

Tabelle 7.55: Eigenschaften der *Viewports*-Auflistung

Prozedur	Beschreibung
<i>Add</i>	Fügt eine Ansichtsfensterkonfiguration der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.56: Prozeduren der Viewports-Auflistung

Einzelne Ansichtsfensterkonfigurationen

Die erste Frage lautet nun: Wie kann man mehrere Ansichtsfenster pro Konfiguration erzeugen? Hierfür stellt das Ansichtsfensterobjekt, das in VBA durch die Klasse *AcadViewports* abgebildet wird, eine Anweisung zur Verfügung. Durch Ausführung der *Split*-Anweisung wird das angesprochene Ansichtsfenster in mehrere aufgeteilt. In wieviele und in welcher Konfiguration bestimmt der Parameter, den Sie der *Split*-Anweisung übergeben. Allerdings können Sie *Split* nicht auf das *ActiveViewport*-Objekt anwenden. Um das aktuelle Ansichtsfenster zu teilen, müssten Sie über die *ActiveViewport*-Eigenschaft dieses Ansichtsfenster in der *Viewports*-Auflistung ermitteln (z. B. über die *Name*-Eigenschaft), dann dieses Ansichtsfenster teilen und in die *ActiveViewport*-Eigenschaft des *Document*-Objekts zurückschreiben.

```
Viewports(ActiveViewport.Name).Split acViewport2Horizontal
ActiveViewport = Viewports("Viewport1")
```

Konstante	Beschreibung	Wert
<i>acViewport2Horizontal</i>	Teilt das Ansichtsfenster in zwei horizontale Ansichtsfenster.	0
<i>acViewport2Vertical</i>	Teilt das Ansichtsfenster in zwei vertikale Ansichtsfenster.	1
<i>acViewport3Left</i>	Teilt das Ansichtsfenster in drei Ansichtsfenster, wobei links neben einem durchgehenden zwei geteilte angeordnet werden.	2
<i>acViewport3Right</i>	Teilt das Ansichtsfenster in drei Ansichtsfenster, wobei rechts neben einem durchgehenden zwei geteilte angeordnet werden.	3
<i>acViewport3Horizontal</i>	Teilt das Ansichtsfenster in drei horizontale Ansichtsfenster.	4
<i>acViewport3Vertical</i>	Teilt das Ansichtsfenster in drei vertikale Ansichtsfenster.	5
<i>acViewport3Above</i>	Teilt das Ansichtsfenster in drei Ansichtsfenster, wobei unter einem durchgehenden zwei geteilte angeordnet werden.	6
<i>acViewport3Below</i>	Teilt das Ansichtsfenster in drei Ansichtsfenster, wobei über einem durchgehenden zwei geteilte angeordnet werden.	7
<i>acViewport4</i>	Teilt das Ansichtsfenster in vier Ansichtsfenster.	8

Tabelle 7.57: Konstanten der Split-Anweisung

Da ja immer nur das angesprochene Ansichtsfenster geteilt wird, die vorhandenen aber erhalten bleiben, sollten Sie darauf achten, dass Sie den Überblick über die einzelnen Ansichtsfenster der jeweiligen Konfiguration nicht verlieren. Das heißt, dass Sie eventuell erst bestimmte Ansichtsfenster mit der *Delete*-Anweisung löschen, bevor Sie mit der *Split*-Anweisung neue

erzeugen. Schauen wir uns das Vorgehen an einem Beispiel an. Angenommen, Sie haben im Modellbereich zunächst ein geteiltes Fenster, also zwei Ansichtsfenster, die unter dem Namen „Viewport1“ gespeichert sind.

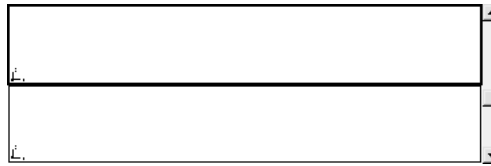


Abbildung 7.50: Zwei Ansichtsfenster

Wenn Sie nun folgende Anweisung ausführen

```
ActiveViewport = Viewports(".*Active")
Viewports("Viewport1").Split acViewport3Vertical
ActiveViewport = Viewports("Viewport1")
```

ergibt sich folgende Darstellung für diese Konfiguration:

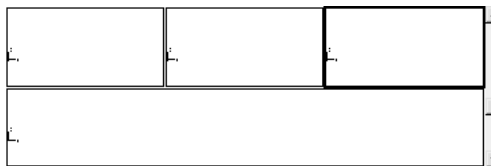


Abbildung 7.51: aktives Ansichtsfenster mit Split geteilt

Wenn Sie nun wiederum dieselbe Anweisung ausführen, ergibt sich jetzt folgende Darstellung:

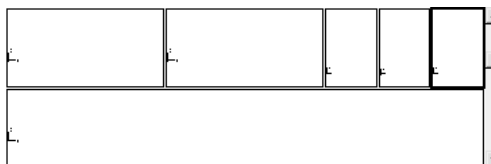


Abbildung 7.52: Aktives Ansichtsfenster erneut geteilt

Diesen Vorgang können Sie fortsetzen, bis Sie die Konfiguration der Ansichtsfenster zusammengestellt haben. Sie müssen lediglich darauf achten, auf welches Ansichtsfenster Sie die *Split*-Anweisung anwenden.

Nun können Sie aber nicht nur einzelne Ansichtsfenster in neue Bereiche aufteilen, sondern auch die bereits vorhandenen anpassen. Dazu stellt das *Viewport*-Objekt verschiedene Eigenschaften zur Verfügung. Einige dieser Eigenschaften kennen Sie bereits von den benannten Ansichten, die im vorherigen Abschnitt bereits abgehandelt wurden. Zu diesen Eigenschaften gehören *Center*, *Direction*, *Target*, *Height* und *Width*. Sie haben dieselbe Funktion wie ihre gleichnamigen Eigenschaften bei den benannten Ansichten, was eine weitere Ausführung an dieser Stelle überflüssig machen dürfte.

Neben diesen Eigenschaften verfügt ein Ansichtsfenster aber noch über weitere Eigenschaften, die dessen Darstellung erheblich beeinflussen. Mit *ArcSmoothness* beispielsweise können Sie die Glättung von Kreisen, Bögen und Ellipsen bestimmen. Der Glättungswert liegt dabei im Bereich von 1 bis 20000. Je höher der Wert, umso länger benötigt AutoCAD für die Regenerierung.

Mit der *GridOn*-Eigenschaft können Sie die Anzeige des Rasters für das Ansichtsfenster steuern. Ist *GridOn* auf *TRUE* gesetzt, wird das Raster angezeigt, andernfalls nicht. Den aktuellen Abstand der einzelnen Raster können Sie mit der *GetGridSpacing*-Anweisung abrufen und mit *SetGridSpacing* festlegen. In beiden Anweisungen bestimmen die Parameter *XSpacing* und *YSpacing* den jeweiligen Abstand in X- bzw. Y-Richtung. Außerdem lässt sich mit der *SnapOn*-Eigenschaft der Fangmodus aktivieren (*TRUE*) bzw. deaktivieren (*FALSE*). Der Ursprung des Fangmodus ist in der *SnapBasePoint*-Eigenschaft als 2D-Array gespeichert. *SnapRotationAngle* bestimmt den Fangdrehwinkel für das Ansichtsfenster relativ zum aktuellen BKS. Den Abstand des Fangrasters können Sie mit der *GetSnapSpacing*-Anweisung abrufen und mit *SetSnapSpacing* festlegen. Auch hier bestimmen die Parameter *XSpacing* und *YSpacing* den jeweiligen Abstand in X- bzw. Y-Richtung. Mit der *OrthoOn*-Eigenschaft kann der Ortho-Modus festgelegt werden. Ist die Eigenschaft auf *TRUE* gesetzt, ist dieser eingeschaltet, *FALSE* schaltet ihn aus. Die Anzeige des BKS-Symbols kann ebenfalls für jedes Ansichtsfenster gesondert gesteuert werden. *UCSIconOn* legt fest, ob das BKS-Symbol angezeigt werden soll (*TRUE*) oder nicht (*FALSE*). Mit der *UCSIconAtOrigin* wird festgelegt, ob das Symbol im Ursprung des jeweiligen Koordinatensystems dargestellt werden soll (falls es eingeschaltet ist). Ist die Eigenschaft auf *FALSE* eingestellt, wird das Symbol immer in der linken unteren Ecke des Ansichtsfensters angezeigt.

Falls im Modellbereich das Ansichtsfenster in mehrere aufgeteilt wurde, können Sie mit den Eigenschaften *LowerLeftCorner* und *UpperRightCorner* die genaue Position der einzelnen Ansichtsfenster abrufen. Beide Eigenschaften geben jeweils ein 2D-Array zurück, das diese Position innerhalb der Konfiguration anzeigt. Der Bereich, über den sich der Inhalt dieser Eigenschaften erstreckt, liegt zwischen 0 und 1. Liegt das Ansichtsfenster z.B. in der linken unteren Ecke, gibt *LowerLeftCorner* die Werte 0 und 0 zurück. Am besten lässt sich die Wirkungsweise dieser Eigenschaften anhand eines Beispiels darstellen. Stellen Sie sich vor, Sie haben eine Konfiguration von vier Ansichtsfenstern, wobei eines vertikal durchgezogen und die anderen drei links neben diesem Ansichtsfenster angeordnet sind.

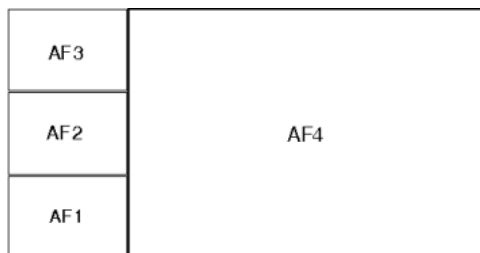


Abbildung 7.53: Ansichtsfensteranordnung

Da die ersten drei Ansichtsfenster alle links angeordnet sind, enthält die Y-Koordinate der *LowerLeftCorner*-Eigenschaft den Wert 0. Die X-Koordinate dieser Eigenschaft ergibt sich nun aus der Aufteilung dieser drei Fenster. Das erste liegt unten und hat den Wert 0, das

zweite um ein Drittel höher also 0.33 und das dritte um zwei Drittel, also 0.66. Daraus ergeben sich nun folgende Werte für die *LowerLeftCorner*-Eigenschaft der links angeordneten Ansichtsfenster:

- AF1 *LowerLeftCorner* = (0, 0)
- AF2 *LowerLeftCorner* = (0, 0.33)
- AF3 *LowerLeftCorner* = (0, 0.66)

Die linke untere Ecke des großen Ansichtsfensters beginnt bei den Koordinaten 0.25 und 0, da diese drei Viertel des Bereichs in der Breite ausfüllt. Daraus ergeben sich auch die Werte der Eigenschaft *UpperRightCorner* für die drei kleinen Ansichtsfenster.

- AF1 *UpperRightCorner* = (0.25, 0.33)
- AF2 *UpperRightCorner* = (0.25, 0.66)
- AF3 *UpperRightCorner* = (0.25, 1.00)

Der Wert der *UpperRightCorner*-Eigenschaft des großen Ansichtsfensters dürfte hingegen klar sein. Da dieses in der äußeren oberen Ecke endet, besitzt diese die Werte 1 und 1.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Center</i>	Legt den Zentrumspunkt der Ansicht fest.
<i>Direction</i>	Bestimmt die Ausrichtung für eine 3D-Visualisierung der Zeichnung.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>Height</i>	Bestimmt die Höhe des Ansichtsfensters.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Name</i>	Legt den Namen der benannten Ansicht fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>Width</i>	Bestimmt die Breite des Ansichtsfensters.
<i>Target</i>	Legt den Zielpunkt für das Ansichtsfenster fest.

Tabelle 7.58: Weitere Eigenschaften des Viewport-Objekts

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetView</i>	Stellt einen benannten Ausschnitt für das Ansichtsfenster wieder her.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.59: Weitere Prozeduren des Viewport-Objekts

Anwendungsbeispiel

(Beispielprojekt: 07\AcadViewports.dvb)

Im ersten Beispiel wollen wir uns mit der Teilung der Ansichtsfenster beschäftigen. Dazu lassen wir zunächst alle benannten Ansichtsfenster in einem Formular anzeigen. Falls mehrere Ansichtsfenster unter einem Namen gespeichert werden, sollen diese zu einem Eintrag zusammengefasst und dann die jeweiligen Fenster untergeordnet werden.

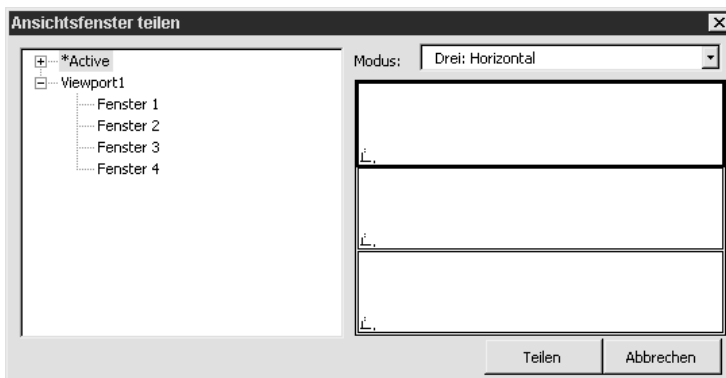


Abbildung 7.54: Formular des ersten Anwendungsbeispiels

Den jeweiligen Modus kann der Benutzer in einer Combobox auswählen, wobei diese dann optisch in einem *Image*-Steuerelement dargestellt wird.

Startprozedur

In der Startprozedur des Projekts wird wie gehabt nur der Inhalt des Formulars initialisiert und anschließend das Formular gestartet.

```
Public Sub SplitViewport()
On Error Resume Next
SplitFrm.initSplitFrm
SplitFrm.Show
End Sub
```

Initialisierungsprozedur

Die Initialisierung des Formulars läuft in zwei Bereichen ab. In der ersten Prozedur, die durch die Startprozedur aufgerufen wurde, wird der Inhalt der *TreeView*-Liste erzeugt.

```
Public Sub initSplitFrm()
Dim VP As AcadViewport
Dim xNode As MSComctlLib.Node
On Error Resume Next
TreeView1.Nodes.Clear
```

Dabei wird die *Viewports*-Auflistung Element für Element durchlaufen und je nachdem, ob es sich um das jeweils erste Ansichtsfenster mit dem entsprechenden Namen handelt ein neuer Haupteintrag mit eben dem Namen des Ansichtsfensters erzeugt. Andernfalls wird es als Untereintrag zum vorangegangenen Eintrag hinzugefügt.

```
For Each VP In ThisDrawing.Viewports
  If xNode.Text <> VP.Name Then
    Set xNode = TreeView1.Nodes.Add(, , VP.Name)
    TreeView1.Nodes.Add xNode, tvwChild, , "Fenster 1"
    x = 2
  Else
    TreeView1.Nodes.Add xNode, tvwChild, , "Fenster " & CStr(x)
    x = x + 1
  End If
Next
End Sub
```

Die zweite Initialisierungsroutine wird nur nach dem Laden des Formulars ausgeführt und füllt die Combobox mit den entsprechenden Auswahleinträgen.

```
Private Sub UserForm_Initialize()
ComboBox1.AddItem "Keine Teilung"
ComboBox1.AddItem "Zwei: Horizontal"
ComboBox1.AddItem "Zwei: Vertikal"
ComboBox1.AddItem "Drei: Links"
ComboBox1.AddItem "Drei: Rechts"
ComboBox1.AddItem "Drei: Horizontal"
ComboBox1.AddItem "Drei: Vertikal"
ComboBox1.AddItem "Drei: Oberhalb"
ComboBox1.AddItem "Drei: Unterhalb"
ComboBox1.AddItem "Vier"
ComboBox1.ListIndex = 0
End Sub
```

Auswahl eines Teilungsmodus

Um dem Benutzer seine Auswahl des jeweiligen Teilungsmodus optisch näher zu bringen, wurden die verschiedenen Modi mit einem Bild hinterlegt. Diese Bilder sind in dem *ImageList*-Steuerelement abgelegt und werden nach der Auswahl durch den Benutzer über die *Picture*-Eigenschaft des *Image*-Steuerelements zugeordnet.

```
Private Sub ComboBox1_Click()
Image1.Picture = ImageList1.ListImages(ComboBox1.ListIndex + _
1).Picture
End Sub
```

Ansichtsfenster teilen

Die Teilung des Ansichtsfensters kann der Benutzer über die Schaltfläche TEILEN auslösen. Dabei müssen wir zunächst zwischen zwei Auswahlmöglichkeiten durch den Benutzer unterscheiden.

```
Private Sub CommandButton1_Click()
Dim ActVPName As String
Dim ActVP As AcadViewport
On Error Resume Next
If TypeName(TreeView1.SelectedItem.Parent) = "Nothing" Then
```

Die erste Möglichkeit: Der Benutzer hat einen Haupteintrag ausgewählt. In diesem Fall merken wir uns den Namen des gewählten Ansichtsfensters und löschen dann alle Ansichtsfenster dieses Namens aus der *Viewports*-Auflistung. Anschließend wird wieder ein neues Ansichtsfenster mit diesem Namen erzeugt und der entsprechende Teilungsmodus auf die *Split*-Anweisung angewendet.

```
    ActVPName = ThisDrawing.Viewports( _
        TreeView1.SelectedItem.Text).Name
    For i = ThisDrawing.Viewports.Count - 1 To 0 Step -1
        If ThisDrawing.Viewports(i).Name = ActVPName Then
            ThisDrawing.Viewports(i).Delete
        End If
    Next
    Set ActVP = ThisDrawing.Viewports.Add(ActVPName)
    ActVP.Split ComboBox1.ListIndex - 1
Else
```

Die zweite Möglichkeit: Der Benutzer wählt direkt ein Ansichtsfenster (Untereintrag) aus. In diesem Fall erstellen wir einen Verweis auf dieses Fenster und wenden danach ebenfalls die *Split*-Anweisung an.

```
    Set ActVP = ThisDrawing.Viewports( _
        TreeView1.SelectedItem.Parent.Text)
    x = 0
    For i = 0 To ThisDrawing.Viewports.Count - 1
        If ThisDrawing.Viewports(i).Name = ActVP.Name Then
            x = x + 1
            If x = CLng(Mid(TreeView1.SelectedItem.Text, 8)) Then
                ActVP.Split ComboBox1.ListIndex - 1
                Exit For
            End If
        End If
    Next
End If
Me.Hide
End Sub
```

Im zweiten Anwendungsbeispiel befassen wir uns mit den Anpassungen der Eigenschaften eines Ansichtsfensters. Die Start- und Initialisierungsprozeduren entsprechen in etwa denen des vorherigen Beispielprogramms, weshalb ich mir an dieser Stelle großartige Erklärungen sparen kann.

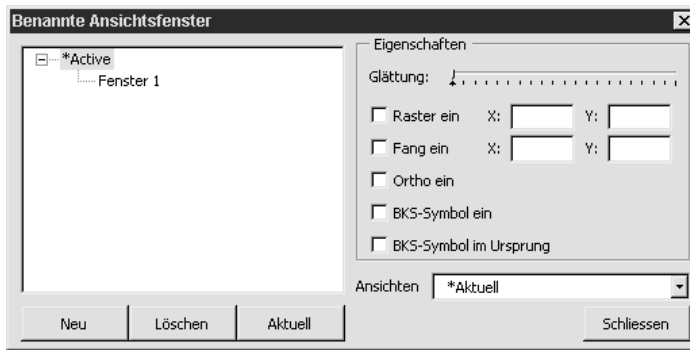


Abbildung 7.55: Formular des zweiten Anwendungsbeispiels

Da die Zuordnung der Eigenschaftswerte so ziemlich klar sein dürfte, wollen wir uns hier nur mit den wichtigsten Prozeduren dieses Projekts beschäftigen.

Auswahl eines Ansichtsfensters durch den Benutzer

Nachdem der Benutzer einen Eintrag im *TreeView*-Steuerelement gewählt hat wird die Ereignisprozedur *NodeClick* ausgelöst. Auf dieses Ereignis reagieren wir mit folgendem Code.

```
Private Sub TreeView1_NodeClick(ByVal Node As MSComctlLib.Node)
    Dim ActVP As AcadViewport
    Dim XSpacing As Double, YSpacing As Double
    On Error Resume Next
    NoChange = True
```

Zunächst wird geprüft, welchen Eintrag der Benutzer gewählt hat. Handelt es sich um den obersten Eintrag, gibt die *Parent*-Eigenschaft des gewählten *Node*-Objektes den Wert *Nothing* zurück.

```
If TypeName(Node.Parent) = "Nothing" Then
```

In diesem Fall hat der Benutzer das aktive Ansichtsfenster selektiert und wir können den Verweis auf das jeweilige Ansichtsfenster direkt über die *Text*-Eigenschaft abrufen und in der Variablen *ActVP* speichern.

```
    Set ActVP = ThisDrawing.Viewports(Node.Text)
Else
```

Hat der Benutzer hingegen einen der untergeordneten Fenster ausgewählt, müssen wir zunächst die *Viewports*-Auflistung durchlaufen und dabei kontrollieren, ob es sich bei dem dem gewählten Fenster übergeordneten Objekt um das gesuchte Ansichtsfenster handelt.

```
    x = 0
    For i = 0 To ThisDrawing.Viewports.Count - 1
        If ThisDrawing.Viewports(i).Name = Node.Parent.Text Then
```

Dabei wird bei jedem Auffinden des entsprechenden Ansichtsfensters der Zähler *X* um 1 erhöht.

```
        x = x + 1
        If x = CLng(Mid(Node.Text, 8)) Then
```


Stimmt nun der Zähler mit der Nummer des Fensters überein, wurde das gewünschte Ansichtsfenster gefunden und der Verweis auf dieses Fenster kann zur weiteren Verarbeitung in der Variablen *ActVP* gespeichert werden.

```

        Set ActVP = ThisDrawing.Viewports(Node.Parent.Text)
    Exit For
End If
Next
End If

```

Danach brauchen wir nur noch die Werte der Eigenschaften des gewählten Ansichtsfensters an die jeweiligen Steuerelemente zu übertragen

```

Slider1.Value = ActVP.ArcSmoothness
CheckBox1.Value = ActVP.GridOn
CheckBox2.Value = ActVP.SnapOn
CheckBox3.Value = ActVP.OrthoOn
CheckBox4.Value = ActVP.UCSTIconOn
CheckBox5.Value = ActVP.UCSTIconAtOrigin
ActVP.GetGridSpacing XSpacing, YSpacing
TextBox1 = Util.RealToString(XSpacing)
TextBox2 = Util.RealToString(YSpacing)
ActVP.GetSnapSpacing XSpacing, YSpacing
TextBox3 = Util.RealToString(XSpacing)
TextBox4 = Util.RealToString(YSpacing)
NoChange = False
End Sub

```

Änderung einer Eigenschaft

Wenn der Benutzer irgendeine Eigenschaft ändert, z.B. durch Anklicken der entsprechenden Checkbox oder durch Änderung eines Abstandswerts in der Textbox, wird durch die ausgelöste Ereignisprozedur die *SaveVPPProperties*-Anweisung aufgerufen. In dieser Prozedur wird zunächst das gewählte Ansichtsfenster ermittelt und dann der Verweis auf das Ansichtsfenster mit *SetVPPProperties* aufgerufen.

```

Private Sub SaveVPPProperties()
Dim ActVP As AcadViewport, ActVPName As String
Dim XSpacing As Double, YSpacing As Double
On Error Resume Next
If NoChange = True Then Exit Sub
If TypeName(TreeView1.SelectedItem.Parent) = "Nothing" Then
    ActVPName = TreeView1.SelectedItem.Text
    For Each ActVP In ThisDrawing.Viewports
        If UCase(ActVP.Name) = UCase(ActVPName) Then
            SetVPPProperties ActVP
        End If
    Next
Else
    Set ActVP = ThisDrawing.Viewports( _
        TreeView1.SelectedItem.Parent.Text)
    x = 0
    For i = 0 To ThisDrawing.Viewports.Count - 1

```

```

    If ThisDrawing.Viewports(i).Name = ActVP.Name Then
        x = x + 1
        If x = CLng(Mid(TreeView1.SelectedItem.Text, 8)) Then
            SetVPPProperties ActVP
            Exit For
        End If
    End If
Next
End If
End Sub

```

Diese Prozedur passt dann die Eigenschaften des gewählten Ansichtsfensterobjekts durch Übergabe der Werte der Steuerelemente an.

```

Private Sub SetVPPProperties(ActVP As AcadViewport)
    Dim XSpacing As Double, YSpacing As Double
    On Error Resume Next
    ActVP.ArcSmoothness = Slider1.Value
    ActVP.GridOn = CheckBox1.Value
    ActVP.SnapOn = CheckBox2.Value
    ActVP.OrthoOn = CheckBox3.Value
    ActVP.UCSIconOn = CheckBox4.Value
    ActVP.UCSIconAtOrigin = CheckBox5.Value
    XSpacing = Util.DistanceToReal(TextBox1)
    YSpacing = Util.DistanceToReal(TextBox2)
    ActVP.SetGridSpacing XSpacing, YSpacing
    XSpacing = Util.DistanceToReal(TextBox3)
    YSpacing = Util.DistanceToReal(TextBox4)
    ActVP.SetSnapSpacing XSpacing, YSpacing
End Sub

```

7.12.2 Ansichtsfenster im Papierbereich

Neben den festen Ansichtsfenstern des Modellbereichs besteht die Möglichkeit, verschiebbare Ansichtsfenster für den Papierbereich zu erstellen. Mit diesem Typ von Ansichtsfenster kann der Benutzer einen Ausschnitt des Modellbereichs im Papierbereich darstellen. Diese Ansichtsfenster werden allerdings nicht wie die festen Fenster in einer Auflistung verwaltet, sondern als grafisches Element (wie Linien, Polylinien usw.) dem jeweiligen Papierlayout hinzugefügt. Ein einzelnes verschiebbares Ansichtsfenster wird in VBA durch die Klasse *AcadPViewport* abgebildet. Um einen Verweis auf ein verschiebbares Ansichtsfenster zu erhalten, können Sie entweder mittels der *AddPViewport*-Funktion des *PaperSpace*-Objekts ein neues Ansichtsfenster im Papierbereich erstellen (siehe Kapitel 6.4.2) oder ein vorhandenes Ansichtsfenster über die *Item*-Funktion des *PaperSpace*-Objekts oder eine Benutzerauswahl (*GetEntity*, *SelectionSet*) ermitteln.

Nachdem Sie nun einen Verweis auf ein bestimmtes Ansichtsfenster ermittelt haben, können Sie wieder eine ganze Reihe von Eigenschaften und Methoden auf dieses Objekt anwenden, um das Ansichtsfenster anzupassen.

Da es sich bei diesem Objekt ebenfalls um ein Ansichtsfenster handelt, besitzt es auch einige Eigenschaften, die Sie bereits von den festen Ansichtsfenstern her kennen. Dazu gehören beispielsweise *ArcSmoothness*, die den Glättungswert festlegt, oder auch *GridOn*, die bestimmt, ob das Raster ein- oder ausgeschaltet ist.

Die *UCSPerViewport*-Eigenschaft legt fest, ob das BKS mit dem Ansichtsfenster gespeichert werden soll (*TRUE*) oder nicht (*FALSE*). Die Höhe und Breite des Ansichtsfensters ergeben sich aus der *Height*- und der *Width*-Eigenschaft. Dabei handelt es sich um die direkte Höhe und Breite des Ansichtsfensterobjekts. Die *ViewportOn*-Eigenschaft bestimmt, ob das Ansichtsfenster aktiviert oder deaktiviert ist. Wurde es deaktiviert (*FALSE*), dann wird der Inhalt des Ansichtsfensters nicht angezeigt, andernfalls schon. Da Sie diese Eigenschaft nicht direkt beeinflussen können, müssen Sie diesen Status mit der *Display*-Anweisung umschalten. Dazu geben Sie nur den entsprechenden Status als Parameter der Anweisung mit.

ActivePViewport.Display True

Mit der *DisplayLocked*-Eigenschaft können Sie die Skalierfaktorsperrung des Ansichtsfensters aktivieren (*TRUE*) oder deaktivieren (*FALSE*). Durch diese Sperrung wird die für das ausgewählte Ansichtsfenster festgelegte Skalierung gesperrt. Wenn die Sperre aktiviert wurde, können Sie die Objekte in dem Ansichtsfenster bearbeiten, ohne dass sich die Ansichtsfensterskalierung ändert. Der Skalierungsfaktor für das Ansichtsfenster wird durch die Eigenschaften *CustomScale* und *StandardScale* bestimmt. Für diese Eigenschaften gelten ähnliche Einstellungsmöglichkeiten wie für die Layouts. Mit der *StandardScale*-Eigenschaft legen Sie entweder einen Vorgabefaktor durch Setzen der entsprechenden Konstante fest oder Sie vergeben für diese Eigenschaft die Konstante *acVpCustomScale*. Dadurch wird das Ansichtsfenster auf eine benutzerdefinierte Skalierung festgelegt.

Konstante	Skalierung	Wert	Konstante	Skalierung	Wert
<i>acVpScaleToFit</i>	Größe angepasst	0	<i>acVp100_1</i>	100 : 1	17
<i>acVpCustomScale</i>	Benutzerdefiniert	1	<i>acVp1_128in_1ft</i>	1/128"=1'	18
<i>acVp1_1</i>	1 : 1	2	<i>acVp1_64in_1ft</i>	1/64"=1'	19
<i>acVp1_2</i>	1 : 2	3	<i>acVp1_32in_1ft</i>	1/32"=1'	20
<i>acVp1_4</i>	1 : 4	4	<i>acVp1_16in_1ft</i>	1/16"=1'	21
<i>acVp1_8</i>	1 : 8	5	<i>acVp3_32in_1ft</i>	3/32"=1'	22
<i>acVp1_10</i>	1 : 10	6	<i>acVp1_8in_1ft</i>	1/8"=1'	23
<i>acVp1_16</i>	1 : 16	7	<i>acVp3_16in_1ft</i>	3/16"=1'	24
<i>acVp1_20</i>	1 : 20	8	<i>acVp1_4in_1ft</i>	1/4"=1'	25
<i>acVp1_30</i>	1 : 30	9	<i>acVp3_8in_1ft</i>	3/8"=1'	26
<i>acVp1_40</i>	1 : 40	10	<i>acVp1_2in_1ft</i>	1/2"=1'	27
<i>acVp1_50</i>	1 : 50	11	<i>acVp3_4in_1ft</i>	3/4"=1'	28
<i>acVp1_100</i>	1 : 100	12	<i>acVp1in_1ft</i>	1"=1'	29
<i>acVp2_1</i>	2 : 1	13	<i>acVp3in_1ft</i>	3"=1'	30
<i>acVp4_1</i>	4 : 1	14	<i>acVp6in_1ft</i>	6"=1'	31
<i>acVp8_1</i>	8 : 1	15	<i>acVp1ft_1ft</i>	1'=1'	32
<i>acVp10_1</i>	10 : 1	16			

Tabelle 7.60: Konstantenauflistung *AcViewportScale*

Egal, welches Skalierungsformat Sie in der Eigenschaft *StandardScale* einstellen, den entsprechenden Wert dieser Skalierung finden Sie in der *CustomScale*-Eigenschaft wieder. Nehmen wir einmal an, Sie legen den Standardskalierungswert auf *acVp1_50* fest. In diesem Fall erhält die Eigenschaft *CustomScale* den Wert 0.02, was ja einem Verhältnis von 1:50 entspricht. Genauso reagiert auf eine Änderung des Werts der *CustomScale*-Eigenschaft die *StandardScale*-Eigenschaft. Existiert für die eingegebene Skalierung eine Konstante, wird diese automatisch in *StandardScale* eingetragen. Entspricht der eingegebene Wert keiner Konstanten, wird *AcVpCustomScale* eingestellt.

Die *TwistAngle*-Eigenschaft bestimmt den Drehwinkel, um den der Inhalt des Ansichtsfensters gedreht wird. Diese Eigenschaft bezieht sich nur auf den Inhalt, nicht aber auf die Drehung des Ansichtsfensters selbst, da diese nicht gedreht dargestellt werden können. Zu vergleichen ist diese Eigenschaft mit der Systemvariablen *VIEWTWIST*, die den Ansichtsdrehwinkel für das aktuelle Ansichtsfenster speichert.

Wenn Sie im Ansichtsfenster eine perspektivische Ansicht der Zeichnung darstellen, können Sie mit der *LensLength*-Eigenschaft die Brennweite (in Millimeter) dieser Ansicht festlegen.

Die *Clipped*-Eigenschaft verwenden Sie, wenn das Ansichtsfenster zugeschnitten wurde. In diesem Fall bestimmt diese Eigenschaft, ob nur der zugeschnittene Bereich (*TRUE*) oder das gesamte Ansichtsfenster (*FALSE*) angezeigt werden soll. Besitzt das Ansichtsfenster hingegen keine zugeschnittene Umgrenzung, kann der Inhalt dieser Eigenschaft auch nicht angepasst werden. Mit *RemoveHiddenLines* legen Sie fest, ob Objekte, die durch andere überlagert werden, beim Plotten verdeckt, also nicht dargestellt werden sollen. Die Wirkung dieser Eigenschaft macht sich allerdings erst beim Plotten bemerkbar. Ist diese Eigenschaft auf den Wert *TRUE* festgelegt, werden die verdeckten Linien nicht mitgeplottet.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>ArcSmoothness</i>	Bestimmt die Glättung von Kreisen, Bögen und Ellipsen.
<i>Center</i>	Legt den Zentrums punkt der Ansicht fest.
<i>Color</i>	Legt die Farbe des Volumenkörpers fest oder gibt diese zurück.
<i>Direction</i>	Bestimmt die Ausrichtung für eine 3D-Visualisierung der Zeichnung.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>GridOn</i>	Bestimmt den Status des Ansichtsfensterrasters
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungs-wörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Ansichtsfensters zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Ansichtsfensters an oder legt diesen fest.

Tabelle 7.61: Weitere Eigenschaften des PViewport-Objekts

Eigenschaft	Beschreibung
<i>Linetype</i>	Zeigt den Linientyp des Ansichtsfensters an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Ansichtsfensters an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuellen Linienstärke für das Ansichtsfenster an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstil-Namen für das Ansichtsfenster zurück oder legt diesen fest.
<i>SnapBasePoint</i>	Legt den Fangbasispunkt für das Ansichtsfenster fest.
<i>SnapOn</i>	Bestimmt den Status des Fangs.
<i>SnapRotationAngle</i>	Legt den Fangdrehwinkel für das Ansichtsfenster relativ zum aktuellen BKS fest.
<i>Target</i>	Legt den Zielpunkt für das Ansichtsfenster fest.
<i>UCSIconAtOrigin</i>	Bestimmt, ob das BKS-Symbol am Ursprung angezeigt wird.
<i>UCSIconOn</i>	Bestimmt, ob das BKS-Symbol aktiviert ist.
<i>Visible</i>	Steuert die Sichtbarkeit des Ansichtsfensters

Tabelle 7.61: Weitere Eigenschaften des *PViewport*-Objekts (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Ansichtsfensters ab.
<i>GetGridSpacing</i>	Legt den Rasterabstand für das Ansichtsfenster fest.
<i>GetSnapSpacing</i>	Legt den Fangwert für das Ansichtsfenster fest.
<i>GetXData</i>	Ruft die erweiterten Daten des Ansichtsfensters ab.

Tabelle 7.62: Weitere Prozeduren des *PViewport*-Objekts

Prozedur	Beschreibung
<i>Highlight</i>	Hebt den Volumenkörper hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Ansichtsfensters mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt ein Ansichtsfenster als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt ein Ansichtsfenster als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt das Ansichtsfenster von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht das Ansichtsfenster um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht das Ansichtsfenster um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Ansichtsfenster um einen bestimmten Faktor.
<i>SetGridSpacing</i>	Bestimmt den Rasterabstand für das Ansichtsfenster.
<i>SetSnapSpacing</i>	Bestimmt den Fangwert für das Ansichtsfenster.
<i>SetXData</i>	Speichert die erweiterten Daten des Ansichtsfensters.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Ansichtsfenster an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Ansichtsfensters im AutoCAD-Zeichnungsfenster.

Tabelle 7.62: Weitere Prozeduren des PViewport-Objekts (Forts.)

Anwendungsbeispiel

(Beispielprojekt: 07\AcadPViewport.dvb)

Im Anwendungsbeispiel zu diesem Objekt werden verschiebbare Ansichtsfenster erzeugt, die durch Rechtecke im Modellbereich festgelegt wurden. Dazu muss der Benutzer nur ein solches Rechteck auswählen und das Programm erzeugt dann ein Ansichtsfenster, das den Bereich innerhalb des Rechtecks darstellt. Das Programm an sich ist nicht sehr umfangreich und läuft zum größten Teil in der Startprozedur ab. Nur die Auswahl des Layouts, in dem das Ansichtsfenster erzeugt werden soll, und die Eingabe des Maßstabs erfolgen in einem Formular.

```
Public Sub createPViewport()
Dim NVP As AcadPViewport
Dim Center(2) As Double
Dim LP, RP
Dim Width As Double, Height As Double
Dim FrameObj As AcadEntity
Const PI = 3.14159265358979
On Local Error Resume Next
Set Util = New MyUtility
```

Nachdem zunächst einige Variablen deklariert wurden, schalten wir auf den Modellbereich zurück, falls sich die Zeichnung noch in einem Papierlayout befinden sollte. Danach kann dann der Benutzer ein Rechteck im Modellbereich auswählen, dessen Inhalt in dem Ansichtsfenster dargestellt werden soll.

```

ActiveLayout = Layouts("Model")
Util.GetEntity FrameObj, PickedPoint, "Rahmen wählen:"
If TypeName(FrameObj) = "Nothing" Then Exit Sub

```

Anschließend wird das Formular zur Auswahl des Layouts und zur Eingabe des Maßstabs eingeblendet.

```

With PViewportFrm
    .InitForm
    .Show
    If .OK = False Then Exit Sub

```

Hat der Benutzer nun alle Vorbereitungen abgeschlossen, beginnt der eigentliche Umsetzungsteil des Programms. Zunächst wird auf das im Formular ausgewählte Layout umgeschaltet. Danach werden die Eckpunkte des Rechtecks ausgelesen und mit diesen Koordinaten die Breite und Höhe des Ansichtsfensters ermittelt. An diese Längenangaben wird gleichzeitig der festgelegte Maßstab angebracht. Damit könnte der Benutzer dann das Ansichtsfenster mit einer Skalierung von 1:1 plotten. Wenn Sie lieber den Skalierungsfaktor des Layouts verwenden, brauchen Sie nur die Skalierung der Breite und Höhe entfernen und können den Maßstab mit der Anweisung *SetCustomScale* an das Layout übertragen.

```

ActiveLayout = Layouts(.ComboBox1.Text)
LP = FrameObj.Coordinate(0)
RP = FrameObj.Coordinate(1)
Width = Util.CalcDistance(LP, RP) * _
    Util.DistanceToReal(.TextBox1)
LP = FrameObj.Coordinate(2)
Height = Util.CalcDistance(LP, RP) * _
    Util.DistanceToReal(.TextBox1)

```

Jetzt braucht nur noch das Ansichtsfenster im Papierlayout erzeugt zu werden. Dafür benötigen wir neben den Angaben von Höhe und Breite noch den Mittelpunkt des Ansichtsfensters. Dazu verwenden wir den Basispunkt des Layoutblocks. Sie könnten diesen Punkt auch durch den Benutzer mittels der *GetPoint*-Funktion des *Utility*-Objekts ermitteln lassen.

```

MSpace = False
varPoint = ActiveLayout.Block.Origin
Center(0) = varPoint(0) / 2
Center(1) = varPoint(1) / 2

```

Auf jeden Fall wird anschließend das Ansichtsfenster mit den entsprechenden Angaben über die *AddPViewport*-Anweisung dem Papierbereich hinzugefügt.

```

Set NVP = PaperSpace.AddPViewport(Center, Width, Height)

```

Nun muss an dieses Fenster noch der Drehwinkel des gewählten Rechtecks übertragen werden. Dazu wird erst die Ausrichtung vom ersten und zweiten Stützpunkt des Rechtecks bestimmt und an die *TwistAngle*-Eigenschaft des Ansichtsfensters übertragen. Damit ist der Inhalt des Fensters nach dem gewählten Rechteck ausgerichtet.

```

NVP.TwistAngle = (2 * PI) - Util.AngleFromXAxis( _
    FrameObj.Coordinate(0), FrameObj.Coordinate(1))

```

Jetzt muss nur noch der Bereich des gewählten Rechtecks auf das Ansichtsfenster gezoomt werden, damit auch nur dieser Bereich im Ansichtsfenster sichtbar ist. Zu diesem Zweck werden die Stützpunkte der linken unteren und der rechten oberen Ecke des Rechtecks ermittelt.

Danach wird das Ansichtsfenster zum aktuellen Fenster gemacht. Dieses wird auf den Modellbereich umgeschaltet und auf den Bereich des Rechtecks gezoomt.

```
NVP.ViewportOn = True
LP = FrameObj.Coordinate(0)
RP = FrameObj.Coordinate(2)
ActivePViewport = NVP
MSpace = True
ReDim Preserve LP(2) As Double
ReDim Preserve RP(2) As Double
Application.ZoomWindow LP, RP
MSpace = False
ActiveLayout.PlotType = acExtents
Regen acActiveViewport
End With
End Sub
```

7.13 Bemaßungsstile

Bemaßungsstile bestimmen die Darstellung von Bemaßungsobjekten, wobei jedem ein einzelner Stil über dessen *StyleName*-Eigenschaft zugewiesen werden kann. Die einzelnen Bemaßungsstile einer Zeichnung werden in der *DimStyles*-Auflistung des *Document*-Objekts verwaltet. Im Grunde genommen können Sie mit den Bemaßungsstilobjekten, die Sie über diese Auflistung erhalten, nur ermitteln, welche Stile in der Zeichnung verfügbar sind. Die Einstellungen an den einzelnen Bemaßungsstilen müssen Sie allerdings über die Systemvariablen von AutoCAD vornehmen, da diese nicht im Stil-Objekt verfügbar sind.

7.13.1 Bemaßungsstilauflistung

In der *DimStyles*-Auflistung werden wie gesagt die einzelnen Bemaßungsstile der jeweiligen Zeichnung verwaltet. Der Zugriff auf die Stil-Objekte erfolgt auf dieselbe Art und Weise, wie bei allen anderen Auflistungen auch. Mit der *Add*-Anweisung können Sie der Zeichnung einen neuen Bemaßungsstil hinzufügen

```
Set NewStyle = DimStyles.Add("neuer Stil")
```

und mittels der *Item*-Funktion einen Verweis auf einen vorhandenen Stil abrufen.

```
Set CrStyle = DimStyles("neuer Stil")
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.

Tabelle 7.63: Eigenschaften der *DimStyles*-Auflistung

Eigenschaft	Beschreibung
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.63: Eigenschaften der DimStyles-Auflistung (Forts.)

Prozedur	Beschreibung
<i>Add</i>	Diese Anweisung fügt ein Bemaßungsstil der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.64: Prozeduren der DimStyles-Auflistung

7.13.2 Einzelne Bemaßungsstile verarbeiten

Die Objektvariable, in der der Verweis auf einen Bemaßungsstil gespeichert ist, entspricht der Klasse *AcadDimStyle*. Diese Klasse stellt ein einzelnes Bemaßungsstil-Objekt in VBA dar. Diese Klasse besitzt neben den Standardeigenschaften und Methoden nur eine einzige interessante Prozedur. Mit der *CopyFrom*-Anweisung können Sie die Stileigenschaften eines Objekts auf den aufrufenden Bemaßungsstil übertragen. Das Objekt, von dem diese Daten übernommen werden sollen, legen Sie im Parameter der Anweisung fest. Sie können hierfür drei unterschiedliche Objekte verwenden.

- Übertragung der Bemaßungsstileigenschaften von einem grafischen Bemaßungsobjekt. Sie können hierfür jedes Bemaßungsobjekt verwenden, das über eine *StyleName*-Eigenschaft verfügt, die ein Bemaßungsstil festlegt.
- Übertragung der Bemaßungsstileigenschaften von einem anderen Bemaßungsstilobjekt. Dabei spielt es keine Rolle, ob das Bemaßungsobjekt, von dem die Daten übertragen werden sollen, aus der gleichen Zeichnung kommt oder in einer anderen geöffneten Zeichnung gespeichert ist.

- Übertragung der Bemaßungsstileigenschaften von einem Zeichnungsdokument. Hierfür müssen Sie im Parameter der *CopyFrom*-Anweisung das *Document*-Objekt angeben. Daraufhin werden die Daten des aktuellen Bemaßungsstils (*ActiveDimStyle*) des Dokuments auf den aufrufenden Bemaßungsstil übertragen.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Name</i>	Legt den Namen des Bemaßungsstils fest
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.65: Eigenschaften des Bemaßungsstil-Objekts

Prozedur	Beschreibung
<i>CopyFrom</i>	Kopiert die Bemaßungsstildaten von einem Quellobjekt.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.66: Prozeduren des Bemaßungsstil-Objekts

7.14 Erweiterte Daten (Erweiterungswörterbuch)

Erweiterungswörterbücher dienen im Allgemeinen zum Abspeichern zusätzlicher Informationen in dem Dokument, sei es über bestimmte Objekte oder rein informative Informationen. So werden z.B. die Gruppenu Auflistung, die Layouts der Zeichnung, Multiliniestile und Plotstile als Erweiterungswörterbuch gespeichert. Bei einigen dieser Objekte müssen Sie nicht über dieses Objekt auf die entsprechende Auflistung zugreifen, weil durch AutoCAD-VBA bereits eine Klasse definiert wurde (*AcadGroups*, *AcadLayouts*). Sie sollten die Erweiterungswörterbücher allerdings dazu nutzen, eigene erweiterte Daten zu verwalten. Wie dies im Einzelnen gemacht wird, werden wir uns im folgenden Kapitel genauer anschauen.

7.14.1 Wörterbuchsammlung

Alle Wörterbücher einer Zeichnung werden zentral in einer Auflistung gesammelt und können über diese abgerufen werden. Die Auflistung aller Wörterbücher ist im *Document*-Objekt jeder Zeichnung gespeichert und kann somit über dieses abgerufen werden. Für die aktuelle Zeichnung wäre dies:

```
Set AllDicts = Dictionaries
```

Für die Auflistung eines beliebigen geöffneten Dokuments der AutoCAD-Sitzung:

```
Set AllDicts = Documents(n).Dictionaries
```

Die so erzeugte Objektvariable ist vom Datentyp *AcadDictionaries* und enthält einen Verweis auf die komplette Liste der Erweiterungswörterbücher.

Wörterbücher erstellen

Wie jede andere Auflistung stellt auch das *Dictionaries*-Objekt die *Add*-Funktion zur Verfügung, um ein neues Wörterbuch zu erstellen. Da dem Wörterbuch ebenfalls ein eindeutiger Name zugeordnet werden muss, wird dieser wieder im Parameter verlangt.

```
Dim NewDict As AcadDictionary  
Set NewDict = ThisDrawing.Dictionaries.Add("MyDict")  
Oder  
ThisDrawing.Dictionaries.Add "MyDictionary"
```

Da der Name eines Wörterbuchs eindeutig sein muss, müssen Sie sicher gehen, dass der Name noch nicht vergeben wurde. Ansonsten wird ein Fehler ausgelöst und das vorgesehene Wörterbuch nicht erstellt.

Da Wörterbücher permanent in der Zeichnung gespeichert werden, ist es auch möglich, eine Vorlagendatei mit Ihrem persönlichen Wörterbuch auszustatten. Es muss dann nicht mehr erstellt werden, wenn der Benutzer eine Zeichnung nach der entsprechenden Vorlage erstellt.

Vorhandene Wörterbücher über die Auflistung abrufen

Für den Zugriff auf einzelne Wörterbücher hält AutoCAD zwei Varianten bereit. Bei der ersten erfolgt der Zugriff auf die gleiche Art, wie Sie es bereits von anderen Auflistungen her gewohnt sind. Auch hier bedienen wir uns der *Item*-Funktion, die das Wörterbuch anhand des Index oder des Namens abrufen.

```
Dim CrDict As AcadDictionary  
Set CrDict = ThisDrawing.Dictionaries.Item(n)
```

oder:

```
Set CrDict = ThisDrawing.Dictionaries(n)
```

Sinnvoller ist natürlich der Abruf über den Wörterbuchnamen, da Sie ja normalerweise wissen, auf welche Daten Sie zugreifen wollen:

```
Const MyDictName = "MyDictionary"  
Dim CrDict As AcadDictionary  
Set CrDict = ThisDrawing.Dictionaries(MyDictName)
```

Wie Sie bereits festgestellt haben dürften, habe ich bei diesem Aufruf keine Variable, sondern eine Konstante verwendet, um den Namen des Wörterbuchs zu definieren. Dies soll aber keine Vorschrift, sondern vielmehr ein Hinweis darauf sein, welche Vorgehensweise sinnvoll im Umgang mit Wörterbüchern sein könnte. Ich gehen einmal davon aus, dass das oder die Wörterbücher, mit denen Sie arbeiten werden, immer auf derselben Struktur basiert und sich damit auch nicht der Name ändert, was sehr unvorteilhaft bei der Entwicklungsarbeit wäre. Aus diesem Grund erscheint es doch recht sinnvoll, eine globale Konstante zu erstellen, die den Namen des Wörterbuchs enthält.

Zugriff auf das Wörterbuch über ein Objekt

Da jedem einzelnen AutoCAD-Objekt ein Erweiterungswörterbuch zugeordnet werden kann, hält auch jedes Objekt die *GetExtensionDictionary*-Funktion bereit, um das jeweilige Wörterbuch abzurufen. Da diese Funktion in jedem Fall ein Objekt zurückgibt, können Sie mit der Funktion *HasExtensionDictionary* vorab prüfen, ob dem Objekt wirklich ein Wörterbuch zugeordnet wurde. Gibt diese Funktion als Ergebnis der Prüfung den Wert *TRUE* zurück, ist ein Wörterbuch zugeordnet, andernfalls erhalten Sie den Wert *FALSE*.

```
Dim ObjDict As AcadDictionary
If Object.HasExtensionDictionary = True Then
    Set ObjDict = Object.GetExtensionDictionary
End If
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.67: Eigenschaften der Wörterbuch-Auflistung

Prozedur	Beschreibung
<i>Add</i>	Fügt eine Ansichtsfensterkonfiguration der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.68: Prozeduren der Wörterbuch-Auflistung

7.14.2 Einzelne Wörterbücher verwalten

Bei der Verwaltung von Daten in einem Erweiterungswörterbuch spielt in erster Linie der *XRecord* eine Rolle. Dieser dient zum Abrufen und Speichern von erweiterten Daten und wird in VBA durch die Klasse *AcadXRecord* dargestellt. Um einen solchen Datensatz dem Wörterbuch hinzuzufügen, müssen Sie lediglich die *AddXRecord*-Anweisung unter Angabe eines Schlüsselworts, unter dem der Datensatz wiedergefunden werden soll, ausführen. Als Ergebnis erhalten Sie einen Verweis auf den Datensatz, den Sie dann in einer Objektvariablen speichern, um diesem schließlich erweiterte Daten hinzuzufügen oder Daten abzurufen.

```
Set NewRecord = CrDict.AddXRecord("MyRecord")
```

Haben Sie dem Wörterbuch einmal einen *XRecord* hinzugefügt, können Sie diesen unter Angabe des Schlüsselworts, das Sie beim Hinzufügen verwendet haben, mittels der *Item*-Funktion abrufen.

Das *XRecord*-Objekt stellt neben den Standardfunktionen eines AutoCAD-Objekts im Wesentlichen zwei Funktionen zur Verfügung, mit denen Sie die erweiterten Daten des Datensatzes abrufen bzw. schreiben können.

Die *SetXRecordData*-Anweisung speichert die in den Parametern angegebenen Daten im Datensatz. Diese sind vom Konzept her ähnlich den *XDaten* einzelner Objekte, nur dass hier die Standard-Gruppencodes von AutoCAD zur Anwendung kommen. Das bedeutet, dass Sie hier alle Codes unterhalb der *XDaten* (1000+) verwenden können.

Gruppencodes	Beschreibung
0-9	Zeichenfolge. (Mit der Einführung der erweiterten Symbolnamen in AutoCAD 2000 wurde die Beschränkung auf 255 Zeichen aufgehoben. Es gibt keine explizite Beschränkung hinsichtlich der Anzahl der Byte pro Zeile, in der Regel sollten die Zeilen jedoch nicht mehr als 2049 Byte an Daten enthalten.)
10-39	3D-Punkte mit doppelter Genauigkeit
40-59	Reelle Zahl
60-79	16-Bit-Ganzzahlwert
90-99	32-Bit-Ganzzahlwert
100	Zeichenfolge (maximal 255 Zeichen; bei Unicode-Zeichenfolgen entsprechend weniger)

Tabelle 7.69: DXF-Gruppencodes

Gruppencodes	Beschreibung
102	Zeichenfolge (maximal 255 Zeichen; bei Unicode-Zeichenfolgen entsprechend weniger)
105	Zeichenfolge für hexadezimalen Referenzwert
110-119	Gleitkommawert mit doppelter Genauigkeit
120-129	Gleitkommawert mit doppelter Genauigkeit
130-139	Gleitkommawert mit doppelter Genauigkeit
140-149	Skalarer Gleitkommawert mit doppelter Genauigkeit
170-179	16-Bit-Ganzzahlwert
280-289	16-Bit-Ganzzahlwert
300-309	Beliebige Textzeichenfolge
310-319	Zeichenfolge für Hexadezimalwert eines Binärblocks
320-329	Zeichenfolge für hexadezimalen Referenzwert
330-369	Zeichenfolge für hexadezimale Objekt-IDs
370-379	16-Bit-Ganzzahlwert
380-389	16-Bit-Ganzzahlwert
390-399	Zeichenfolge für hexadezimalen Referenzwert
400-409	16-Bit-Ganzzahlwert
410-419	Zeichenfolge
999	Kommentar (Zeichenfolge)

Tabelle 7.69: DXF-Gruppencodes (Forts.)

Um nun einen Datensatz mit erweiterten Daten zu schreiben, müssen Sie zunächst ein Array mit der Größe der Anzahl der Daten deklarieren, das die Codenummer aufnimmt und ein Array, das die Daten selbst aufnimmt. Da der Gruppencode immer ein Integer-Wert ist, wird dieses Array auch als *Integer* deklariert. Das Daten-Array allerdings als *Variant*, da die einzelnen Felder ja Daten unterschiedlichen Datentyps aufnehmen können, was ja immer vom jeweiligen Gruppencode abhängig ist. Um beispielsweise den Datensatz mit einem Layer-Namen und einem Kommentartext zu füllen, müssten Sie folgende Codezeilen verwenden.

```
Dim xDataType(1) As Integer
Dim xDataValue(1) As Variant
xDataType(0) = 8 'Gruppencode für Layer
xDataValue(0) = "0" 'Layer 0
xDataType(1) = 999 'Gruppencode für Kommentartext
xDataValue(1) = "Kommentar" 'und die Zeichenfolge
NewRecord.SetXRecordData xDataType, xDataValue
```

Um den Datensatz wieder abzurufen, verwenden Sie die *GetXRecordData*-Anweisung. Diese verlangt als Parameter ebenfalls eine Variable, in der die Gruppencodes zurückgeschrieben werden, und eine zur Aufnahme der Daten. Beide Variablen müssen vom Datentyp *Variant* sein, da im Voraus ja nichts über die Anzahl und den Inhalt der zurückgegebenen Daten bekannt ist.

```
Dim xTypeOut
Dim xDataOut
NewRecord.GetXRecordData xDataType, xDataValue
```

Eine zweite Möglichkeit, Daten dem erweiterten Wörterbuch hinzuzufügen, besteht in der *AddObject*-Anweisung. Diese fügt ein komplettes Objekt dem Wörterbuch hinzu. Dazu müssen Sie zunächst die Anwendung geladen haben, mit der dieses Objekt erzeugt wird, und den Klassennamen des Objekts kennen.

```
Application.LoadArx "MyApp.arx"
Set Obj = CrDict.AddObject("Object1", "CAskdObject")
```

Die *GetName*-Funktion ruft den Namen ab, unter dem das im Parameter der Funktion angegebene Objekt im Erweiterungswörterbuch gespeichert wurde. Bei dem Objekt kann es sich entweder um ein *XRecord* oder um ein benutzerdefiniertes Objekt handeln, das dem Erweiterungswörterbuch mit der *AddObject*-Funktion hinzugefügt wurde.

```
ObjName = CrDict.GetName(Obj)
```

Die *GetObject*-Funktion ist das Gegenstück zu *GetName*. Diese liefert das Objekt, das in dem Erweiterungswörterbuch unter dem angegebenen Namen gespeichert wurde.

```
Set Obj = CrDict.GetObject("Object1")
```

Mithilfe der *Remove*-Anweisung können Sie ein benanntes AutoCAD-Objekt aus dem Erweiterungswörterbuch entfernen. Die *Replace*-Anweisung ersetzt ein Element des Erweiterungswörterbuchs durch ein anderes. Einen Eintrag des Wörterbuchs können Sie mit der *Rename*-Anweisung umbenennen. Dazu müssen Sie im ersten Parameter den Namen des vorhandenen Eintrags angeben und im zweiten den Namen, den dieser Eintrag erhalten soll.

```
CrDict.Rename "MyRecord", "Record1"
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Einträge des Erweiterungswörterbuchs.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Name</i>	Legt den Namen der benannten Ansicht fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.

Tabelle 7.70: Weitere Eigenschaften des *XRecord*-Objekts

Eigenschaft	Beschreibung
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 7.70: Weitere Eigenschaften des XRecord-Objekts (Forts.)

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft einen Wörterbucheintrag an einer angegebenen Position oder über das Schlüsselwort ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 7.71: Weitere Prozeduren des XRecord-Objekts

Anwendungsbeispiel

(Beispielprojekt: 07\AcadDictionaries.dvb)

Das folgende Anwendungsbeispiel befasst sich mit den *XRecords* eines Wörterbuchs. Ziel der Anwendung ist es, dem Benutzer ein Wörterbuch zur Verfügung zu stellen, in dem er die verschiedensten Datensätze verwalten kann. Da es sich hier nur um ein Beispiel handelt, werden bei den Gruppencodes keine allzu großen Unterschiede gemacht. Wir werden lediglich darauf achten müssen, dass der Benutzer zwischen Texten, Ganzzahlen, reellen Zahlen und 3D-Punkten unterscheiden kann.

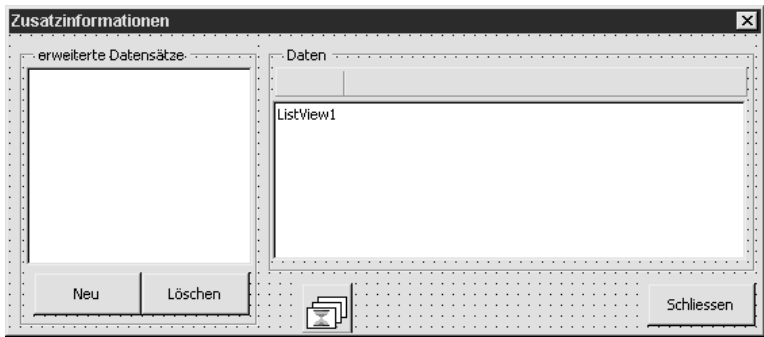


Abbildung 7.56: Hauptformular des Projekts

Als Erstes wird eine Objektvariable im Klassenmodul *ThisDrawing* deklariert, die den Verweis auf das Wörterbuch speichern soll.

```
Public MyDict As AcadDictionary
```


Auf diesen Verweis wird dann im Verlauf des Projekts immer zugegriffen, wenn Daten im Wörterbuch zu speichern oder aus diesem abzurufen sind.

Startprozedur

In der Startprozedur wird zunächst geprüft, ob das Wörterbuch „MyDictionary“ bereits existiert. Sie können statt „MyDictionary“ auch einen anderen Namen verwenden. Sie müssten dann nur die entsprechenden Anpassungen an der Startprozedur vornehmen. Im restlichen Programm wird nur noch mit dem Verweis in der Variablen *MyDict* gearbeitet, so dass der Name des Wörterbuchs dort keine Rolle mehr spielt.

Wurde also das Wörterbuch nicht gefunden, muss es zunächst mit der *Add*-Anweisung erstellt werden.

```
Public Sub showDictionaries()
On Local Error Resume Next
Set Util = New MyUtility
If TypeName(Dictionaries("MyDictionary")) = "Nothing" Then
    Dictionaries.Add "MyDictionary"
End If
```

Danach wird der Verweis auf das Wörterbuch in der Variablen *MyDict* gespeichert. Anschließend werden die einzelnen Datensätze aus dem Wörterbuch gelesen und deren Namen der Listbox des Hauptformulars hinzugefügt. Ist das erledigt, wird nur noch das Formular gestartet.

```
Set MyDict = Dictionaries("MyDictionary")
DictFrm.ListBox1.Clear
For Each Xrec In MyDict
    DictFrm.ListBox1.AddItem Xrec.Name
Next
DictFrm.Show
End Sub
```

Datensatz hinzufügen

Um dem Wörterbuch einen neuen *XRecord* hinzuzufügen, fragen wir zunächst den Namen des neuen Datensatzes ab. Hat der Benutzer einen eingegeben, müssen wir nur die *AddXRecord*-Funktion unter Angabe des eingegebenen Namens ausführen.

```
Private Sub CommandButton1_Click()
    On Error Resume Next
    sValue = InputBox("Name des neuen Datensatzes", "neuer Datensatz")
    If sValue = "" Then Exit Sub
    ThisDrawing.MyDict.AddXRecord sValue
```

Konnte der Datensatz nicht erzeugt werden, löst AutoCAD einen Fehler aus und die Beschreibung des Fehlers wird über ein Meldungsfeld angezeigt. Andernfalls kann der Name des neuen Datensatzes der Listbox hinzugefügt werden.

```
If Err <> 0 Then
    MsgBox Err.Description
Else
    ListBox1.AddItem sValue
End If
End Sub
```

Datensatz löschen

Das Löschen eines einzelnen *XRecord* aus dem Wörterbuch ist relativ einfach. Sie müssen diesen nur über seinen Namen, der ja durch den Benutzer im *ListBox*-Steuerelement gewählt wurde, aus dem Wörterbuch abrufen und die *Delete*-Anweisung auf den *XRecord* anwenden. Anschließend muss nur der Eintrag aus dem Listenfeld entfernt werden.

```
Private Sub CommandButton2_Click()
On Error Resume Next
ThisDrawing.MyDict.Item(ListBox1.Text).Delete
ListBox1.RemoveItem ListBox1.ListIndex
End Sub
```

Auswahl eines Datensatzes

Um die einzelnen Daten eines *XRecord*, den der Benutzer ja über die Auswahl des Namens im *ListBox*-Steuerelement wählen kann, anzuzeigen, müssen zunächst die in diesem Datensatz gespeicherten Daten ermittelt werden.

```
Private Sub ListBox1_Click()
Dim ActRec As AcadXRecord
Dim XRecordType, XRecordData
On Error Resume Next
```

Dazu wird zunächst der Verweis auf den gewählten Datensatz in der Variablen *ActRec* gespeichert. Über diese Variable rufen wir die *GetXRecordData*-Anweisung auf. Sind bereits Daten im Datensatz gespeichert, werden die Variablen *XRecordType* und *XRecordData* mit diesen Daten gefüllt.

```
Set ActRec = ThisDrawing.MyDict.Item(ListBox1.Text)
ActRec.GetXRecordData XRecordType, XRecordData
```

Anschließend werden die Daten dieses Datensatzes, falls vorhanden, im *ListView*-Steuerelement dargestellt.

```
ListView1.ListItems.Clear
If TypeName(XRecordType) <> "Empty" Then
    For i = 0 To UBound(XRecordType)
```

Die Gruppencodes der einzelnen Datenzeilen kommen in die erste Spalte

```
Set LV = ListView1.ListItems.Add(, , CStr(XRecordType(i)))
```

und die Daten selbst in die zweite Spalte. Dabei müssen wir nur die unterschiedlichen Datentypen beachten, die ein Feld des Arrays enthalten kann. Ausschlaggebend hierfür sind die Gruppencodes der Datenzeile.

```
Select Case XRecordType(i)
Case 0 To 9, 100, 102, 105, 300 To 369, 410 To 419, 999
    LV.ListSubItems.Add , , XRecordData(i)
Case 10 To 39
    LV.ListSubItems.Add , , _
        Util.RealToString(XRecordData(i)(0)) & " " & _
        Util.RealToString(XRecordData(i)(1)) & " " & _
        Util.RealToString(XRecordData(i)(2))
Case 60 To 79, 90 To 99, 170 To 179, 280 To 289, _
```

```

        370 To 389, 400 To 409
        LV.ListSubItems.Add , , CStr(XRecordData(i))
    Case Else
        LV.ListSubItems.Add , , _
            Util.RealToString(XRecordData(i))
    End Select
Next
End If

```

Datensatz verarbeiten

Nun soll der Benutzer natürlich auch die Möglichkeit erhalten, die einzelnen Datensätze zu verarbeiten, also Datenzeilen hinzuzufügen, zu ändern oder zu löschen. Die Auswahl hierfür kann er über die drei Schaltflächen des *Toolbar*-Steuerelements treffen. Je nachdem, welche Schaltfläche er gewählt hat, müssen unterschiedliche Funktionen ausgeführt werden. Da das Betätigen einer einzelnen Schaltfläche immer dasselbe Ereignis auslöst, wurden die Schaltflächen mit einem Schlüssel versehen. Da die Ereignisprozedur das Schaltflächenobjekt, das das Ereignis ausgelöst hat, im Parameter *Button* liefert, können wir nun mittels einer *Select Case*-Anweisung diesen Schlüssel abfragen, woraufhin *Select Case* die entsprechende Verzweigung ausführt.

```

Private Sub Toolbar1_ButtonClick(ByVal Button As _
MSComctlLib.Button)
    Dim LV As MSComctlLib.ListItem
    On Error Resume Next
    If ListBox1.ListIndex = -1 Then Exit Sub
    Select Case Button.Key

```

Um eine Datenzeile hinzuzufügen, muss zunächst das Formular gestartet werden, in dem der Benutzer die entsprechenden Daten für die neue Datenzeile festlegt.

```

        Case "Add"
            XRecFrm.Show
            If XRecFrm.OK = True Then

```

Hat der Benutzer die Daten im Formular festgelegt, wird ein neuer Eintrag mit diesen Daten im *ListView*-Steuerelement erzeugt. In die erste Spalte kommt wieder der gewählte Gruppen-code. Die zweite erhält die Daten, die der Benutzer eingegeben hat.

```

Set LV = ListView1.ListItems.Add(, , _
    XRecFrm.ComboBox1.Text)
Select Case CInt(XRecFrm.ComboBox1.Text)
    Case 0 To 9, 100, 102, 105, 300 To 369, 410 To 419, _
        999
        LV.ListSubItems.Add , , XRecFrm.TextBox
    Case 10 To 39
        LV.ListSubItems.Add , , XRecFrm.XBox & " " & _
            XRecFrm.YBox & " " & XRecFrm.ZBox
    Case 60 To 79, 90 To 99, 170 To 179, 280 To 289, _
        370 To 389, 400 To 409
        LV.ListSubItems.Add , , XRecFrm.IntBox
    Case Else
        LV.ListSubItems.Add , , XRecFrm.RealBox
    End Select

```

Nachdem dies erledigt ist, braucht der Datensatz nur noch gespeichert zu werden, was die Prozedur *saveXRecord* erledigt, die weiter unten beschrieben ist.

```
        saveXRecord
    End If
```

Das Löschen einzelner Datenzeilen ist relativ einfach, da der Hauptteil wieder durch die *saveXRecord*-Anweisung ausgeführt wird. Nachdem also die gewählten Zeilen aus dem *List-View*-Steuerelement entfernt wurden, wird nur *saveXRecord* aufgerufen.

```
Case "Delete"
    For i = ListView1.ListItems.Count To 1 Step -1
        If ListView1.ListItems(i).Selected = True Then
            ListView1.ListItems.Remove i
        End If
    Next
    saveXRecord
```

Die Bearbeitung einer einzelnen Datenzeile läuft ähnlich ab wie das Hinzufügen, nur dass hier zunächst die vorhandenen Daten im Formular zur *XRecFrm* dargestellt werden müssen.

```
Case "Edit"
    XRecFrm.ComboBox1.Text = ListView1.SelectedItem.Text
    Select Case CInt(ListView1.SelectedItem.Text)
        Case 0 To 9, 100, 102, 105, 300 To 369, 410 To 419, 999
            XRecFrm.TextBox = ListView1.SelectedItem. _
                ListSubItems(1).Text
        Case 10 To 39
            X = ListView1.SelectedItem.ListSubItems(1).Text
            XRecFrm.XBox = Left(X, InStr(1, X, " ") - 1)
            X = Mid(X, InStr(1, X, " ") + 1)
            XRecFrm.YBox = Left(X, InStr(1, X, " ") - 1)
            X = Mid(X, InStr(1, X, " ") + 1)
            XRecFrm.ZBox = X
        Case 60 To 79, 90 To 99, 170 To 179, 280 To 289, _
            370 To 389, 400 To 409
            XRecFrm.IntBox = ListView1.SelectedItem. _
                ListSubItems(1).Text
        Case Else
            XRecFrm.RealBox = ListView1.SelectedItem. _
                ListSubItems(1).Text
    End Select
```

Danach wird wieder das Formular gestartet und je nach Auswahl des Benutzers werden die im Formular eingegebenen Daten des Benutzers an die aktuelle Zeile des *List-View*-Steuerelements übertragen.

```
XRecFrm.Show
If XRecFrm.OK = True Then
    Set LV = ListView1.SelectedItem
    ListView1.SelectedItem.Text = XRecFrm.ComboBox1.Text
    Select Case CInt(XRecFrm.ComboBox1.Text)
        Case 0 To 9, 100, 102, 105, 300 To 369, 410 To 419, _
            999
            LV.ListSubItems(1).Text = XRecFrm.TextBox
```

```

Case 10 To 39
    LV.ListSubItems(1).Text = XRecFrm.XBox & " " & _
        XRecFrm.YBox & " " & XRecFrm.ZBox
Case 60 To 79, 90 To 99, 170 To 179, 280 To 289, _
    370 To 389, 400 To 409
    LV.ListSubItems(1).Text = XRecFrm.IntBox
Case Else
    LV.ListSubItems(1).Text = XRecFrm.RealBox
End Select
saveXRecord
End If
End Select
End Sub

```

Datensatz speichern

Kommen wir nun zur Speicherung eines einzelnen Datensatzes (*XRecord*). Die folgende Prozedur wurde ja mehrfach in der vorangegangenen Ereignisprozedur verwendet. Sie dient schlicht und einfach dazu, die Daten aus dem *ListView*-Steuerelement zu den entsprechenden Arrays zusammenzufassen und mit der *SetXRecordData*-Anweisung im ausgewählten Datensatz zu speichern.

```

Public Sub saveXRecord()
Dim XDataType, XDataValue
Dim Point(2) As Double
On Error Resume Next

```

Als Erstes muss geprüft werden, ob überhaupt Daten im *ListView*-Steuerelement eingegeben wurden. Ist dies nicht der Fall, was die *Count*-Eigenschaft wiedergibt, können wir die Daten nicht mit der *SetXRecordData*-Anweisung schreiben, da ja keine Daten vorhanden sind. In diesem Fall muss der komplette Datensatz gelöscht und danach als neuer und damit leerer Datensatz mit demselben Namen wieder zum Wörterbuch hinzugefügt werden.

```

If ListView1.ListItems.Count = 0 Then
    ThisDrawing.MyDict.Item(ListBox1.Text).Delete
    ThisDrawing.MyDict.AddXRecord ListBox1.Text
Else

```

Die Zusammenstellung des *XRecord* ist wieder eine recht einfache Sache. Zunächst deklarieren wir die Arrays *XDataType* und *XDataValue* neu. Diese Arrays werden dann den Gruppencode (*XDataType*) und die Daten (*XDataValue*) aufnehmen.

```

ReDim XDataType(ListView1.ListItems.Count - 1) As Integer
ReDim XDataValue(ListView1.ListItems.Count - 1) As Variant

```

Anschließend wird das *ListView*-Steuerelement Zeile für Zeile durchlaufen, wobei jede Zeile eine eigene Datenzeile im *XRecord* darstellen wird. Da der Gruppencode in der ersten Spalte der jeweiligen Zeile steht, kann dieser direkt an das entsprechende Feld des Arrays *XDataType* übergeben werden. Bei den Daten, die in der zweiten Spalte abgelegt wurden, müssen wir nach dem jeweiligen Datentyp, der durch den Gruppencode bestimmt wird, unterscheiden.

```

For i = 1 To ListView1.ListItems.Count
    XDataType(i - 1) = CInt(ListView1.ListItems(i).Text)
    Select Case XDataType(i - 1)

```

Textwerte:

```
Case 0 To 9, 100, 102, 105, 300 To 369, 410 To 419, 999
  XDataValue(i - 1) = ListView1.ListItems(i). _
    ListSubItems(1).Text
```

Koordinaten:

```
Case 10 To 39
  X = ListView1.ListItems(i).ListSubItems(1).Text
  Point(0) = Util.DistanceToReal(Left(X, InStr(1, X, " " - 1))
    " " - 1))
  X = Mid(X, InStr(1, X, " " + 1)
  Point(1) = Util.DistanceToReal(Left(X, InStr(1, X, " " - 1))
    " " - 1))
  X = Mid(X, InStr(1, X, " " + 1)
  Point(2) = Util.DistanceToReal(CStr(X))
  XDataValue(i - 1) = Point
```

Lange Ganzzahlen:

```
Case 60 To 79
  XDataValue(i - 1) = CLng(ListView1.ListItems(i). _
    ListSubItems(1).Text)
```

Ganzzahlen:

```
Case 90 To 99, 170 To 179, 280 To 289, 370 To 389, _
  400 To 409
  XDataValue(i - 1) = CInt(ListView1.ListItems(i). _
    ListSubItems(1).Text)
```

Fließkommazahlen:

```
Case Else
  XDataValue(i - 1) = Util.DistanceToReal( _
    ListView1.ListItems(i).ListSubItems(1).Text)
End Select
Next
```

Mit den so zusammengestellten Arrays wird dann nur noch die *SetXRecordData*-Anweisung des gewählten *XRecord* (wurde ja im *ListBox*-Steuerelement gewählt) ausgeführt. Damit ist der Datensatz gespeichert, wobei dessen alte Daten, falls vorhanden, ersetzt wurden.

```
ThisDrawing.MyDict.Item(ListBox1.Text).SetXRecordData _
  XDataType, XDataValue
End If
End Sub
```

8 Die Zeichnungsobjekte von AutoCAD

Wenn grafische Zeichnungselemente in AutoCAD mithilfe der VBA-Schnittstelle verarbeitet werden, erstellt AutoCAD einen Verweis auf das entsprechende Element in einer Objektvariablen. Dabei entsprechen diese Objekte einer bestimmten Klasse der AutoCAD-Typenbibliothek. Im folgenden Kapitel erfahren Sie alles über die einzelnen Klassen, die die entsprechenden Zeichnungsobjekte wie Linien, Polylinien, Texte usw. darstellen.

Kapitelübersicht

8.1 Allgemeines

8.2 Zugriff auf vorhandene AutoCAD-Objekte

8.3 AutoCAD-Objekte verarbeiten

8.1 Allgemeines

Alle grafischen Elemente werden in einem Container gesammelt. Dies kann der Modell- bzw. Papierbereich oder eine Blockdefinition sein, egal ob in der Zeichnung enthalten oder als externe Referenz eingebunden. Sie können somit Zugriff auf ein Element erhalten, indem Sie es über die *Item*-Anweisung der jeweiligen Auflistung abrufen.

```
Dim CrEnt As AcadEntity  
Set CrEnt = ModelSpace.Item(n)
```

Das Element *CrEnt* muss dabei als Datentyp *AcadEntity* deklariert werden, da der Modell- bzw. Papierbereich oder einzelne Blöcke ja die verschiedensten Objekte enthalten können. Dieser Datentyp bietet zunächst nur die Eigenschaften und Methoden, die alle AutoCAD-Elemente besitzen. Nachdem Sie einen Verweis auf das Objekt an die Objektvariable übergeben haben, können Sie dann allerdings auf alle objektspezifischen Eigenschaften und Methoden zugreifen. Die Eigenschaften werden zwar während der Entwicklung nicht im Drop-down-Listenfeld (Liste der Bezeichner) angezeigt, sie sind aber durchaus vorhanden. Sie können dies ganz einfach kontrollieren, indem Sie die Objektvariable *CrEnt* in das Überwachungsfenster eintragen. Das Gleiche gilt übrigens auch für den Datentyp *Object*, der ja gelegentlich zum Einsatz kommt (*GetEntity*).

Eine zweite Möglichkeit, ein Objekt zu erhalten, ist das Hinzufügen zu einem der oben genannten Bereiche. Benutzen Sie hierzu die entsprechende *Add*-Methode (siehe *Kapitel 6*) und lassen Sie sich das Ergebnis als Objektvariable zurückgeben.

```
Dim CrEnt As AcadPoint  
Dim InsPoint(2) As Double  
Set CrEnt = ModelSpace.AddPoint(InsPoint)
```

Da Sie mit der entsprechenden Anweisung (`AddPoint`, `AddLine` ...) ein bestimmtes Objekt zu AutoCAD hinzufügen, wird auch immer der entsprechende Datentyp an eine Objektvariable zurückgeliefert (`AddPoint` gibt `AcadPoint` zurück).

8.1.1 Arbeiten mit Koordinaten

Jedes grafische AutoCAD-Objekt verfügt über eine oder mehrere Eigenschaften, die dessen Position bestimmen. Bei einigen Objekten reicht die Angabe einzelner Punkte, andere wiederum (z.B. Polylinien) benötigen ganze Koordinatenlisten, die die einzelnen Stützpunkte definieren. Generell werden zwei- oder dreidimensionale Koordinatenpaare im WKS (Weltkoordinatensystem) verwendet, um einen Punkt festzulegen. Der erste Schritt bei der Festlegung eines Punkts besteht darin, die einzelnen Koordinatenwerte des entsprechenden Punkts zu bestimmen. Bei zweidimensionalen Punkten wären dies die X- und die Y-Koordinate, bei dreidimensionalen Punkten kommt zusätzliche die Höhe (Z-Koordinate) hinzu. Diese Werte werden dann zu einem Array zusammengefasst, wobei das erste Feld den X-, das zweite den Y- und das dritte, wenn erforderlich, den Z-Wert angibt. Für einfache Punkte ergeben sich daraus entweder 2D- oder 3D-Arrays, wobei jedes mit dem Index 0 beginnt.

Deklaration eines Arrays für einen zweidimensionalen Punkt:

```
Dim InsPoint(0 To 1) As Double
```

Oder:

```
Dim InsPoint(1) As Double
```

Deklaration eines Arrays für einen dreidimensionalen Punkt:

```
Dim InsPoint(0 To 2) As Double
```

Oder:

```
Dim InsPoint(2) As Double
```

Nachdem Sie die Variable für den Punkt deklariert haben, können Sie den einzelnen Feldern die entsprechenden Werte zuweisen:

```
InsPoint(0) = 1000.1234 'X-Koordinate  
InsPoint(1) = 300.3231 'Y-Koordinate  
InsPoint(2) = 10.133 'Z-Koordinate
```

Das so erzeugte Array könnte nun den Objekteigenschaften zugewiesen werden, die nur einfache Punkte verwalten, wie beispielsweise *InsertionPoint* einer Blockreferenz, die den Einfügepunkt des Blocks bestimmt, oder *StartPoint* und *EndPoint* einer Linie, zur Festlegung der Endpunkte.

Etwas komplizierter wird es, wenn das Objekt seine Stützpunkte über eine Eigenschaft verwaltet, meist *Coordinates*. Auch hier gilt die grundlegende Punktdefinition. Zuerst werden die einzelnen Koordinaten bestimmt, bei zweidimensionalen X und Y, bei dreidimensionalen zusätzlich Z. Anschließend werden die einzelnen Stützpunkte zu einem einzelnen Array zusammengefasst. Bei der Dimensionierung des Arrays müssen Sie dann nur die Anzahl der Stützpunkte mit der Anzahl der Koordinaten eines einzelnen Stützpunkts multiplizieren. Bei zweidimensionalen Stützpunkten wäre dies 2 für X und Y.


```
Dim PointsList((StpCount * 2) - 1) As Double
```

Bei dreidimensionalen 3 für die zusätzliche Z-Koordinate.

```
Dim PointsList((StpCount * 3) - 1) As Double
```

Da das Array immer mit dem Index 0 beginnt, müssen Sie von der Koordinatenanzahl immer den Wert 1 abziehen, damit Sie wieder konform mit der Feldanzahl laufen. Dem so dimensionierten Array können nun die einzelnen Koordinaten zugewiesen werden.

```
PointsList(0) = 1000.1234 'X-Koordinate des 1. Stützpunkts
PointsList(1) = 300.3231 'Y-Koordinate des 1. Stützpunkts
PointsList(2) = 10.133 'Z-Koordinate des 1. Stützpunkts
PointsList(3) = 900.3453 'X-Koordinate des 2. Stützpunkts
PointsList(4) = 350.1437 'Y-Koordinate des 2. Stützpunkts
PointsList(5) = 10.224 'Z-Koordinate des 2. Stützpunkts
```

usw.

Diese Array können Sie der *Coordinates*-Eigenschaft des entsprechenden Objekts übergeben:

```
Object.Coordinates = PointsList
```

Tipp

Wenn Sie die Koordinatenliste für ein Objekt erzeugen, müssen Sie unbedingt darauf achten, dass Sie die richtige Koordinatenanzahl für die einzelnen Stützpunkte verwenden. Einige Objekte erfordern nur die Angabe der X- und Y-Koordinate.

Das Setzen der Koordinate war recht einfach zu bewerkstelligen. Das Auslesen bzw. die Auswertung der ausgelesenen Koordinaten funktioniert auf die gleiche Weise. Die Eigenschaften, die die Position des Objekts definieren, geben ein Array mit den Koordinaten der Stützpunkte bzw. des Stützpunkts zurück. Zunächst müssen Sie eine Variable vom Datentyp *Variant* definieren. Dieser Variablen übergeben Sie dann den Wert der entsprechenden Eigenschaft, die die Koordinaten des Objekts enthält.

Für ein Objekt mit mehreren Stützpunkten (z.B. Polylinie):

```
Dim InsPoint As Variant
InsPoint = Object.Coordinates
```

Für ein Objekt mit einem Einfügepunkt (z.B. Blockreferenz):

```
Dim InsPoint As Variant
InsPoint = Object.InsertionPoint
```

Für ein Objekt, bei dem die Stützpunkte auf mehrere Eigenschaften verteilt wurden (z.B. Linien):

```
Dim Point1 As Variant
Dim Point2 As Variant
Point1 = Object.StartPoint
Point2 = Object.EndPoint
```

Egal, welche Art von Punkt die Eigenschaft zurückgibt, Sie erhalten immer ein Array, das die einzelnen Koordinaten enthält. Die Berechnung der Feldgröße können Sie entweder mit der *UBound()*-Funktion bestimmen, was die gängige Methode wäre, oder Sie kennen die genaue

Anzahl der in dem Array verwendeten Koordinaten (zwei- bzw. dreidimensional). Dann könnten Sie die Stützpunktanzahl direkt berechnen. Auch hier kommt die *UBound()*-Funktion zum Einsatz, denn die Obergrenze des zurückgegebenen Arrays müssten Sie in jedem Fall bestimmen.

Bestimmung der Anzahl der Stützpunkte mit jeweils drei Koordinaten (X, Y, Z):

```
PointsList = Object.Coordinates
StpCount = (UBound(PointsList) + 1) / 3
```

Bestimmung der Anzahl der Stützpunkte mit jeweils zwei Koordinaten (X, Y):

```
PointsList = Object.Coordinates
StpCount = (UBound(PointsList) + 1) / 2
```

Nach der eben beschriebenen Methode müssen Sie allerdings sicher sein, dass die Koordinatenliste entweder Z-Koordinaten enthält oder nicht. Das Ganze können Sie aber auch flexibler gestalten, indem Sie die Anzahl der Koordinaten eines einzelnen Stützpunkts mit der *UBound()*-Funktion ermitteln. Da die meisten Objekte, die die *Coordinates*-Eigenschaft besitzen, auch über eine *Coordinate*-Eigenschaft verfügen, mit der einzelne Stützpunkte abgerufen werden können, lässt sich dies ganz einfach bewerkstelligen. Diese Eigenschaft erfordert hierfür nur die Angabe des Index des gesuchten Stützpunkts. Nehmen wir hierfür einfach den ersten Stützpunkt der Koordinatenliste, da dieser ja immer vorhanden sein muss.

```
StpPoint = Object.Coordinate(0)
```

Die Variable *StpPoint* enthält nun ein Array mit den Koordinaten des ersten Stützpunkts. Wie viele Koordinaten enthalten sind, können Sie nun wieder mit der *UBound()*-Funktion überprüfen.

```
CoordCount = UBound(StpPoint) + 1
```

Da *UBound()* die Obergrenze des Arrays zurückgibt und nicht die Anzahl der Felder, müssen Sie dem Ergebnis noch den Wert 1 hinzuaddieren, um auf das korrekte Ergebnis zu kommen. Jetzt können Sie die Anzahl der Stützpunkte ermitteln, ohne zu wissen, ob diese mit oder ohne Z-Koordinate bestückt sind.

```
PointsList = Object.Coordinates
StpCount = (UBound(PointsList) + 1) / CoordCount
```

Aber die *Coordinate*-Eigenschaft dient nicht nur dem Abrufen oder Setzen einzelner Stützpunkte. Sie können hiermit auch direkt auf einzelne Koordinaten zugreifen, was mit der *Coordinates*-Eigenschaft nicht zu bewerkstelligen ist, da diese die Angabe eines Index nicht gestattet. Dazu müssen Sie zunächst den entsprechenden Stützpunkt ansprechen und zusätzlich das Feld, in dem sich die gewünschte Koordinate befindet. Um die Z-Koordinate des ersten Stützpunkts auf den Wert 0 zu setzen, geben Sie folgende Codezeile an:

```
Object.Coordinate(0)(2) = 0
```

Die X-Koordinate des zweiten Stützpunkts lässt sich mit folgender Zeile abrufen:

```
Point2_X = Object.Coordinate(1)(0)
```

Egal, wie Sie die einzelnen Stützpunkte abrufen oder anpassen, Sie müssen nur Folgendes beachten. Wenn Sie die *Coordinates*-Eigenschaft verwenden, müssen Sie diese immer erst in ein Array schreiben lassen, anschließend das Array verarbeiten und dieses dann zurückschrei-

ben. Eine direkte Eingabe mit *Coordinates(0) = 0* ist nicht möglich. Die direkte Änderung einer Koordinate ist nur mit der Eigenschaft *Coordinate* unter Angabe des Stützpunkt-Index und der entsprechenden Koordinate möglich.

Anpassen der Stützpunkte

Einige Objekte bieten die Möglichkeit, die Anzahl der Stützpunkte flexibel zu gestalten. Zwar besitzen diese dann auch Funktionen, um beispielsweise Stützpunkte hinzuzufügen. Allerdings erweisen sich diese Funktionen zumeist als unzureichend, wenn es darum geht, die Stützpunkte an beliebigen Stellen einzufügen oder die Positionen von Stützpunkten zu tauschen. Im Folgenden nun einige Prozeduren, die in den verschiedensten Fälle verwendet werden könnten.

Stützpunkt an beliebiger Position einfügen

(Projekt: 08\Coordinates.dvb!Modules.InsertVertex)

Mit der ersten Funktion kann ein beliebiger Punkt (2D oder 3D) in der Koordinatenliste, die im Parameter *Coordinates* übergeben wird, an einer beliebigen Stelle hinzugefügt werden. Diese Position wird durch den Parameter *Index* bestimmt und entspricht der Positionsangabe der *Coordinate*-Eigenschaft (0 – N). Mit dem dritten Parameter liefern Sie die Koordinaten des Stützpunkts, der in die Koordinatenliste eingefügt werden soll. Mit dem letzten Parameter kann zusätzlich festgelegt werden, ob es sich um eine 2D-Koordinatenliste handelt.

```
Public Function InsertVertex(Coordinates, Index As Integer, _
NewVertex, Optional Is2D As Boolean) As Variant
Dim PointsList
On Local Error Resume Next
If Is2D = True Then
    CoordCount = 2
Else
    CoordCount = 3
End If
If Index > (UBound(Coordinates) + 1) / CoordCount Then
    ReDim Preserve Coordinates(UBound(Coordinates) + CoordCount) _
        As Double
    y = CoordCount - 1
    For x = 0 To CoordCount - 1
        Coordinates(UBound(Coordinates) - y) = NewVertex(x)
        y = y - 1
    Next
    InsertVertex = Coordinates
Else
    ReDim PointsList(UBound(Coordinates) + CoordCount) As Double
    If Index < 0 Then Index = 0
    For i = 0 To UBound(Coordinates)
        PointsList(i + CoordCount) = Coordinates(i)
    Next
    For i = 0 To Index * CoordCount - 1
        PointsList(i) = Coordinates(i)
    Next
    For i = 0 To UBound(NewVertex)
        PointsList(Index * CoordCount + i) = NewVertex(i)
    Next
```

```

    InsertVertex = PointsList
End If
End Function

```

Mit folgender Anweisung könnte z.B. einer Polylinie (Variable *Object*) ein zusätzlicher Stützpunkt zwischen dem dritten und vierten Punkt eingefügt werden.

```

LastPt = Object.Coordinate(2)
NVertex = Utility.GetPoint(LastPt, "Stützpunkt wählen")
Object.Coordinates = InsertVertex(Object.Coordinates, 3, NVertex)

```

Stützpunkte tauschen

(Projekt: 08\Coordinates.dvb!Modules.ChangeVertex)

Mit dieser Funktion können Sie die Position zweier Stützpunkte vertauschen. Im ersten Parameter übergeben Sie die Koordinatenliste, die die beiden Stützpunkte enthält, die vertauscht werden sollen. Die Position dieser Punkte legen Sie dann in den Parametern *FirstIndex* und *SecondIndex* fest. Die Funktion wird dann den Punkt, der im ersten Index angegeben wurde, an die Position des zweiten Index verschieben und umgekehrt.

```

Public Function ChangeVertex(Coordinates, FirstIndex As Integer, _
    SecondIndex As Integer, Optional Is2D As Boolean) As Variant
    Dim Point1(2) As Double
    Dim Point2(2) As Double
    Dim CoordCount As Long
    On Local Error Resume Next
    If Is2D = True Then
        CoordCount = 2
    Else
        CoordCount = 3
    End If
    Point1(0) = Coordinates((FirstIndex + 1) * CoordCount - _
        CoordCount)
    Point1(1) = Coordinates((FirstIndex + 1) * CoordCount - _
        CoordCount + 1)
    Point1(2) = Coordinates((FirstIndex + 1) * CoordCount - _
        CoordCount + 2)
    Point2(0) = Coordinates((SecondIndex + 1) * CoordCount - _
        CoordCount)
    Point2(1) = Coordinates((SecondIndex + 1) * CoordCount - _
        CoordCount + 1)
    Point2(2) = Coordinates((SecondIndex + 1) * CoordCount - _
        CoordCount + 2)
    Coordinates((SecondIndex + 1) * CoordCount - CoordCount) = _
        Point1(0)
    Coordinates((SecondIndex + 1) * CoordCount - CoordCount + 1) = _
        Point1(1)
    If Is2D = False Then
        Coordinates((SecondIndex + 1) * CoordCount - CoordCount + 2) = _
            Point1(2)
    End If
    Coordinates((FirstIndex + 1) * CoordCount - CoordCount) = _
        Point2(0)
    Coordinates((FirstIndex + 1) * CoordCount - CoordCount + 1) = _

```

```

    Point2(1)
If Is2D = False Then
    Coordinates((FirstIndex + 1) * CoordCount - CoordCount + 2) _
        = Point1(2)
End If
ChangeVertex = Coordinates
End Function

```

Um z.B. die Positionen des dritten und fünften Stützpunkts zu tauschen, könnte die Funktion wie folgt verwendet werden:

```
Object.Coordinates = RemoveVertex(Object.Coordinates, 2, 4)
```

Stützpunkte entfernen

(Projekt: 08\Coordinates.dvb!Modules.RemoveVertex)

Die nächste Funktion könnte verwendet werden, um einen beliebigen Stützpunkt aus der Koordinatenliste zu entfernen. Neben der Koordinatenliste müssen Sie nur den Index des Stützpunkts angeben, der gelöscht werden soll.

```

Public Function RemoveVertex(Coordinates, Index As Integer, _
Optional Is2D As Boolean) As Variant
Dim PointsList
On Local Error Resume Next
If Is2D = True Then
    CoordCount = 2
Else
    CoordCount = 3
End If
If Index > (UBound(Coordinates) + 1) / CoordCount Then
    RemoveVertex = Coordinates
Else
    If Index < 0 Then Index = 0
    PointsList = Coordinates
    ReDim Preserve PointsList(UBound(Coordinates) - CoordCount) _
        As Double
    For i = Index * CoordCount + CoordCount To UBound(Coordinates)
        PointsList(i - CoordCount) = Coordinates(i)
    Next
    RemoveVertex = PointsList
End If
End Function

```

Um z.B. den dritten Stützpunkt einer 3D-Polylinie zu entfernen, könnte die Funktion folgendermaßen verwendet werden:

```
Object.Coordinates = RemoveVertex(Object.Coordinates, 2)
```

Bei einer optimierten Polylinie legen Sie einfach den Parameter *Is2D* fest:

```
Object.Coordinates = RemoveVertex(Object.Coordinates, 2, True)
```

Stützpunktliste umkehren

(Projekt: 08\Coordinates.dvb!Modules.ReverseCoordinates)

Die folgende Funktion kehrt die Richtung der Stützpunkte, die im Parameter *Coordinates* übergeben wird, um und gibt als Ergebnis ein Array mit den Stützpunkten in umgekehrter Reihenfolge zurück.

```
Public Function ReverseCoordinates(Coordinates, Optional Is2D As _
Boolean) As Variant
Dim PointsList
On Local Error Resume Next
If Is2D = True Then
    CoordCount = 2
Else
    CoordCount = 3
End If
ReDim PointsList(UBound(Coordinates)) As Double
x = 0
For i = UBound(Coordinates) To 0 Step -CoordCount
    PointsList(x) = Coordinates(i - 2)
    x = x + 1
    PointsList(x) = Coordinates(i - 1)
    x = x + 1
    If Is2D = False Then
        PointsList(x) = Coordinates(i)
        x = x + 1
    End If
Next
ReverseCoordinates = PointsList
End Function
```

Aufgerufen werden könnte diese Funktion folgendermaßen:

```
Object.Coordinates = ReverseCoordinates(Object.Coordinates)
```

8.1.2 Arbeiten mit Vektoren

Einige AutoCAD-Objekte verwenden Vektoren zur Definition von räumlichen Richtungsangaben. Diese Vektorangaben sind im Prinzip nichts anderes als BKS-Angaben, ausgehend von einem Basispunkt, der zumeist den Einfügepunkt des Objekts bestimmt. Abgebildet wird der Vektor, genau wie eine Koordinate, in einem 3D-Array, wobei jedes Feld einen Vektor (X, Y, Z) aufnimmt. Die Definition der einzelnen Vektoren ist recht simpel.

Das Zentrum dieser Matrix bestimmt immer den entsprechenden Ausgangspunkt. Liegt also die Linie in Richtung des ersten Quadranten, so erhalten die Richtungsvektoren die Werte X=1 und Y=0. Zeigt die Linie in Richtung des zweiten Quadranten, also senkrecht nach oben, so erhalten die Vektoren die Werte X=0 und Y=1. Sie müssen also nur die Koordinaten ermitteln, die der aktuelle Richtungspunkt auf dem gedachten Kreis definiert.

Die so ermittelten Koordinaten legen den jeweiligen Vektor fest. Da X- und Y-Achse immer rechtwinklig zueinander stehen, muss folgerichtig nur ein Vektor ermittelt und der andere über den rechten Winkel berechnet werden.

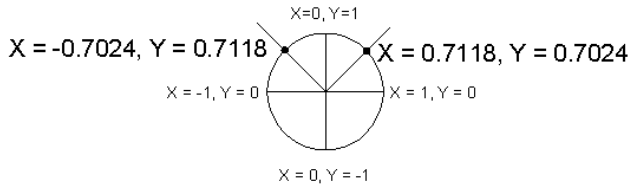


Abbildung 8.1: Vektordefinitionen auf dem gedachten Kreis

Möchten Sie das Koordinatensystem räumlich verdrehen, wird es schon etwas komplizierter. Sie müssen dann auch die Z-Achse berücksichtigen, die ebenfalls von 1 bis -1 definiert ist. Die gesamte Vektorendefinition würde sich dann nicht als Kreis, sondern als Kugel darstellen. Die Interpretation der Koordinaten bliebe allerdings die Gleiche. Ausgehend vom Ursprungspunkt bestimmt die Vektorkoordinate die Richtung der Z-Achse des entsprechenden Objekts.

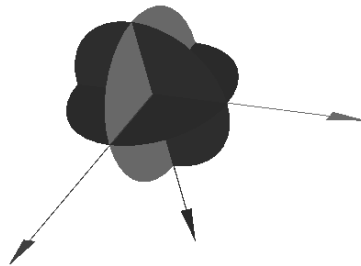


Abbildung 8.2: Definition der Z-Achsen verschiedener Kreise

Die am häufigsten verwendete Vektoreigenschaft ist der *Normal*-Vektor. Diese Eigenschaft bestimmt die räumliche Ausrichtung des Objekts innerhalb des Koordinatensystems der Zeichnung.

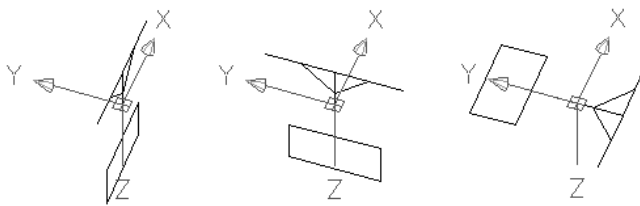


Abbildung 8.3: Verschiedene räumliche Normal-Vektorendefinitionen

8.1.3 Arbeiten mit Winkeln

Winkel werden in AutoCAD grundsätzlich in Radiant (Bogenmaß) verarbeitet. Für diejenigen, die sich mit dieser Winkleinheit nicht auskennen: Ein Vollkreis entspricht einem Wert von $2 * \pi$, also $\approx 6,2832$. Wenn Sie in einer anderen Winkleinheit arbeiten, müssen Sie den Winkelwert entsprechend umrechnen. Hierfür können Sie die Funktionen *AngleToString* bzw. *AngleToReal* der

Klasse *AcadUtility* oder die der erweiterten Klasse *MyUtility* (siehe *Kapitel A.3*) verwenden. Generell sollten Sie sich allerdings angewöhnen, die Winkel in Radiant zu verarbeiten.

in Neugrad	in Altgrad	in Radiant
0 Gon	0 °	0
100 Gon	90 °	0,5 * PI
200 Gon	180 °	PI
300 Gon	270 °	1,5 * PI
400 Gon	360 °	2 * PI

Tabelle 8.1: Beziehungen zwischen den Winkeleinheiten

Da Winkeleingaben in AutoCAD immer abhängig von den Werten der Systemvariablen *ANGBASE* (definiert den Null-Richtungswinkel) und *ANGDIR* (in oder gegen den Uhrzeigersinn) sind, muss dies bei der Festlegung der Parameter berücksichtigt werden. Die eine Möglichkeit wäre, die Winkel mit der Funktion *GetOrientation* abzufragen, da hier der Inhalt der Systemvariable berücksichtigt wird. Die andere wäre, den Winkel mit der Funktion *GetAngle* zu ermitteln. Allerdings müssen Sie bei Verwendung dieser Funktion die Variablen *ANGDIR* und *ANGBASE* im Nachhinein berücksichtigen.

Siehe auch *Kapitel 6.3.6*.

8.2 Zugriff auf vorhandene AutoCAD-Objekte

Ziel bei der Programmierung mit AutoCAD-VBA ist es, mit AutoCAD-Objekten zu kommunizieren, sprich Objekte zu erstellen, zu manipulieren oder wieder zu löschen. Wie Sie grafische Objekte in AutoCAD erzeugen können, haben wir ja bereits im vorherigen Kapitel besprochen. In diesem Kapitel werde ich Ihnen nun die verschiedensten Methoden näher bringen und eine Verbindung zu einzelnen AutoCAD-Objekten herstellen.

8.2.1 Grafische AutoCAD-Objekte wählen

Um auf bereits vorhandene AutoCAD-Objekte in der Zeichnung zugreifen zu können, stellt die VBA-Schnittstelle von AutoCAD verschiedene Funktionen zur Verfügung. Zum einen greifen Sie direkt auf ein Objekt zu, indem Sie den Inhalt der Eigenschaften *Handle* oder *ObjectId* benutzen oder den Modell- bzw. Papierbereich durchlaufen. Eine andere Möglichkeit wäre die Auswahl der Objekte durch den Benutzer. Im folgenden Abschnitt werden wir uns ausführlich mit den einzelnen Möglichkeiten der Objektauswahl beschäftigen.

Direkte Objektauswahl

Für die Auswahl eines Objekts durch direktes Ansprechen, sei es durch anhand der Adresse oder über das Durchsuchen des Objektcontainers (Modellbereich, Papierbereich, Block), stehen verschiedene Funktionen und Prozeduren zur Verfügung. Zum einen haben Sie die Möglichkeit, das *Handle* oder die *ObjectId* des AutoCAD-Elements zu verwenden. Die Adressen, sowohl *Handle* als auch *ObjectId*, stellen dabei einen eindeutigen Identifier des jeweiligen

Objekts dar. Ein Unterschied zwischen beiden Adresstypen besteht darin, dass das *Handle* bei der Erzeugung des Objekts vergeben wird und sich nicht mehr ändert. Im Gegensatz dazu wird die *ObjectID* bei jedem Öffnungsvorgang der Zeichnung neu gesetzt. Ein weiterer Unterschied ist der Datentyp, mit dem *Handle* (String) und *ObjectID* (Long) verwaltet werden. Das Zeichnungsobjekt stellt zwei Funktionen zur Verfügung, mit denen Sie einen Verweis auf ein bestimmtes Objekt mittels der Referenz herstellen.

```
Dim CrObj As Object
Set CrObj = ThisDrawing.HandleToObject(SelHandle)
```

Die Variable *SelHandle* übergibt dabei das entsprechende *Handle* des Objekts an diese Funktion. Haben Sie eine gültige Referenz übergeben, erstellt *HandleToObject* als Ergebnis einen Verweis auf das Objekt in der Objektvariablen, in unserem Fall *CrObj*. Rein theoretisch müsste diese Rückgabeveriable vom Typ *Object* oder zumindest *AcadEntity* sein, damit alle Datentypen von AutoCAD abgedeckt werden können. Wenn Sie aber sicher sind, dass das Ergebnis dieser Operation einem bestimmten Datentyp entspricht, können Sie diesen im Voraus so deklarieren.

```
Dim CrTxt As AcadText
Set CrTxt = ThisDrawing.HandleToObject(TxtHandle)
```

In diesem Fall müssten Sie also sicher sein, dass als Ergebnis dieser Operation ein Textobjekt zurückgegeben wird. Andernfalls wird die Ausführung einen Fehler auslösen. Vor Ausführung dieser Funktion gilt es außerdem zu beachten, dass es sich bei dem übergebenen *Handle* um eine gültige Referenz handelt. In einigen Fällen könnte es zu ungewollten Reaktionen von AutoCAD führen, bis hin zum Absturz, wenn Sie kein korrektes *Handle* an diese Funktion übergeben.

Für die Ermittlung eines Objekts aus seiner *ObjectID* stellt AutoCAD-VBA die Funktion *ObjectIdToObject* zur Verfügung. Diese Funktion hat eine ähnliche Syntax wie *HandleToObject* und gibt als Ergebnis ebenfalls einen Verweis auf das Objekt, das dieser *ObjectID* entspricht, zurück.

```
Dim CrObj As Object
Set CrObj = ThisDrawing.ObjectIdToObject(SelObjectId)
```

Im Gegensatz zur Funktion *HandleToObject* erwartet diese Funktion als Übergabeparameter allerdings eine Variable vom Datentyp *Long*. Ansonsten gelten allerdings dieselben Regeln wie bei der Ausführung der Funktion *HandleToObject*.

Nun noch einen kurzen Hinweis, wie Sie an das *Handle* oder die *ObjectID* eines bestimmten Objekts herankommen. Da jedes Objekt, ob sichtbar wie Linien, Kreise oder Polylinien, oder unsichtbar, wie Layer, Linientyp oder Textstil, eine *Handle*- und eine *ObjectID*-Eigenschaft besitzen, müssen Sie nur die entsprechende Referenz des Objekts abrufen:

```
TxtHandle = TxtObject.Handle
```

oder

```
TxtObjectId = TxtObject.ObjectId
```

TxtObject steht dabei für eine Objektvariable, die bereits auf ein Objekt verweist, z.B. einen Text.

Egal, welche Methode zur direkten Objektwahl Sie verwenden, als Ergebnis erhalten Sie immer die Referenz auf das entsprechende Objekt. Sollte eine Funktion, im Speziellen seien dabei *ObjectARX*-Funktionen genannt, nicht explizit die *ObjectID* als Parameter verlangen, ist der Zugriff auf ein Objekt über dessen *Handle* die zu bevorzugende Variante.

Eine andere Möglichkeit der direkten Objektauswahl ist das Durchsuchen der Objektcontainer. Mit Objektcontainer ist dabei der Bereich von AutoCAD gemeint, der andere Elemente beinhalten kann, beispielsweise der Modell- oder der Papierbereich. Alle diese Container enthalten Funktionen und Eigenschaften, mit denen Sie auf einzelne Objekte zugreifen können. Ich möchte mich in diesem Abschnitt nur auf diese Bereiche beziehen, obwohl auch Layer- oder Linientypauflistungen Objekte enthalten, da in diesen Bereichen die sichtbaren Objekte wie Linien, Polylinien usw. verwaltet werden. Im folgenden Beispiel ist kurz beschrieben, wie Sie auf diese Elemente zugreifen können.

```
For Each Objs in ThisDrawing.ModelSpace
    ThisDrawing.Utility.Prompt TypeName(Objs)
Next
```

In diesem kurzen Beispiel wird die Auflistung des Modellbereichs Objekt für Objekt durchlaufen und bei jedem Durchlauf das aktuelle Objekt an die Objektvariable *Objs* übergeben. Anschließend wird jeder Objekttyp in der AutoCAD-Befehlszeile mithilfe der *Utility*-Funktion *Prompt* ausgegeben.

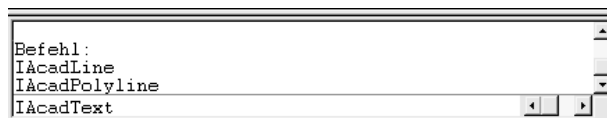


Abbildung 8.4: Ausgabe des Beispielcodes

Sie könnten mit dieser Vorgehensweise alle Elemente, die im Modellbereich abgelegt wurden, nacheinander abrufen und an die entsprechende Objektvariable weitergeben. Sie könnten die Daten des Modellbereichs aber auch mit der *Item*-Funktion der Klasse *ModelSpace* abrufen. Diese Funktion gibt Ihnen die Daten des Modellbereichs an einem, als Parameter mitgegebenen, Index (n-tes Objekt des Containers) zurück.

```
For I = 0 To ThisDrawing.ModelSpace.Count - 1
    Set Objs = ThisDrawing.ModelSpace.Item(i)
    ThisDrawing.Utility.Prompt TypeName(Objs)
Next
```

Die *Item*-Funktion wird auch dann ausgeführt, wenn Sie das Schlüsselwort *Item* nicht benutzen und den Index-Parameter direkt mit der Klasse *ModelSpace* verwenden.

```
Set Objs = ThisDrawing.ModelSpace(i)
```

Mit der *Count*-Eigenschaft rufen Sie die Anzahl der Elemente, die der Modellbereich enthält, ab. Da das erste Element mit dem Index 0 und nicht 1 abgerufen wird, liegt der Index des letzten Elements bei *Count* - 1.

Objektauswahl durch den Benutzer

Eine wichtige Aufgabe bei der Kommunikation mit AutoCAD-Objekten besteht darin, es dem Benutzer zu ermöglichen, bestimmte Objekte im Zeichnungsfenster auswählen zu können. Auch für diesen Zweck stellt AutoCAD-VBA einige Klassen und Funktionen zur Verfügung.

Auswahl einzelner AutoCAD-Objekte

Für die Auswahl einzelner Objekte im AutoCAD-Zeichnungsfenster hält VBA die Funktion *GetEntity* in der Klasse *Utility* bereit. Mithilfe dieser Funktion ist es dem Benutzer möglich, ein einzelnes beliebiges Objekt auszuwählen, das dann im weiteren Programmverlauf manipuliert werden kann. Aber schauen wir uns das Ganze einmal anhand eines kleinen Beispiels an. Ziel dieser kleinen Prozedur ist es, dem Benutzer die Auswahl eines beliebigen Objekts zu ermöglichen und danach den Objektnamen (*AcDbLine*, *AcDbText* ...) des ausgewählten Objekts in einem Dialogfenster anzuzeigen. Die Prozedur beginnt wie immer mit der Deklaration der benötigten Variablen.

```
Dim Object As Object
Dim Prompt As String
Dim PickedPoint As Variant
On Local Error Resume Next
```

Diese drei Variablen sollen als Parameter für die Funktion *GetEntity* dienen, wobei in den Variablen *Object* und *PickedPoint* die Ergebnisse gespeichert werden. Die Variable *Prompt* soll nur einen Text in die AutoCAD-Befehlszeile schreiben, den Sie selbst festlegen können. Sie brauchen diese Variable nicht unbedingt zu verwenden, sondern können den Text auch direkt als Parameter festlegen. Um einen ungewollten Abbruch der Prozedur zu verhindern, sollten Sie zusätzlich die Zeile

```
On Local Error Resume Next
```

einfügen. Folgende Befehlszeile führt dann die Benutzerabfrage in der AutoCAD-Befehlszeile aus. Sobald der Benutzer ein Objekt gewählt hat oder die Abfrage abbricht, wird die Programmausführung in der nächsten Zeile fortgesetzt.

```
Prompt = "Wählen Sie ein Objekt:"
Utility.GetEntity Object, PickedPoint, Prompt
```

oder

```
Utility.GetEntity Object, PickedPoint, "Wählen Sie ein Objekt:"
```

Hat der Benutzer ein Objekt gewählt, wird dieses Objekt in der Variablen *Object* gespeichert, so dass Sie das gewählte Objekt im weiteren Programmablauf weiterverarbeiten können. Zusätzlich zu diesem Objekt wird auch der Auswahlpunkt als dreidimensionale Koordinate zurückgegeben. Dieser Auswahlpunkt wird in dem Array *PickedPoint* gespeichert. Sie haben also nicht nur die Möglichkeit, das Objekt auszuwerten, sondern können auch die Position, an der dieses Objekt gewählt wurde, weiterverarbeiten. Dies ist besonders dann sinnvoll, wenn Sie z.B. prüfen wollen, auf welcher Seite eine Linie oder Polylinie angeklickt wurde, oder um das gewählte Polyliniensegment einer Polylinie zu ermitteln.

Zum Abschluss des kleinen Beispiels wird mittels der Funktion *TypeName()* ermittelt, ob überhaupt ein Objekt durch den Benutzer gewählt wurde. Ist dies nicht der Fall, gibt die Funktion als Ergebnis „Nothing“ zurück.

```
If TypeName(Object) <> "Nothing" Then
    MsgBox Object.ObjectName
End If
```

Für die Auswahl von einzelnen AutoCAD-Objekten durch den Benutzer gibt es aber noch eine weitere Funktion: *GetSubEntity*. Diese Funktion besitzt eine ähnliche Syntax wie *GetEntity*. Im Gegensatz zu *GetEntity* können Sie hiermit auch Unterelemente z.B. einer Blockreferenz ermitteln.

Auswahl mehrerer AutoCAD-Objekte

Für die Auswahl einer ganzen Reihe von Objekten stellt AutoCAD-VBA die Klasse *AcadSelectionSet* zur Verfügung. Mithilfe dieser Klasse ist es möglich, dem Benutzer eine Vielzahl verschiedenster Methoden zur Auswahl von Objekten zur Verfügung zu stellen. Dies beschränkt sich nicht nur auf die Auswahl im Zeichnungsfenster. Ganz im Gegenteil, Sie können mithilfe der Funktionen dieser Klasse auch eine Objektauswahl im Hintergrund durchführen, das heißt, ohne dass der Benutzer irgendein Auswahlfenster definiert. Außerdem können Sie Filter definieren, ähnlich der Filterfunktion von AutoCAD, so dass ganz bestimmte Objekte in AutoCAD selektiert werden. Schauen wir uns das Ganze aber mal im Einzelnen an. Bevor Sie eine Auswahl, in welcher Form auch immer, durchführen, muss der entsprechende Auswahl Satz (*SelectionSet*), ich werde diesen ab jetzt so bezeichnen, definiert bzw. initialisiert werden. Dazu deklarieren Sie eine Variable des Typs *AcadSelectionSet*.

```
Dim AcSSet As AcadSelectionSet
```

Es gibt in AutoCAD verschiedene Möglichkeiten, auf einen Auswahl Satz zuzugreifen. Zum einen steht Ihnen eine Auflistung (*SelectionSets*) zur Verfügung, welche alle definierten Auswahl Sätze beinhaltet. Die Auflistungen der einzelnen Zeichnungen können Sie über die Dokumentklasse erreichen:

```
ThisDrawing.SelectionSets
```

oder

```
Documents(n).SelectionSets
```

Um auf einen bereits vorhandenen Auswahl Satz zurückgreifen zu können oder einen neuen zu erstellen, müssen Sie über diese Auflistung auf den jeweiligen Auswahl Satz zugreifen. Dazu dienen die folgenden Codezeilen:

```
If TypeName(ThisDrawing.SelectionSets("Auswahl")) = "Nothing" Then
    ThisDrawing.SelectionSets.Add "Auswahl"
End If
```

Mit der ersten Codezeile wird geprüft, ob bereits ein Auswahl Satz mit dem Namen „Auswahl“ existiert. Ist dies nicht der Fall, wird mit der folgenden Zeile, die sich innerhalb der *If...End If*-Anweisung befindet, ein neuer Auswahl Satz mit dem Namen „Auswahl“ zu der Auflistung hinzugefügt. Sie können übrigens bis zu 128 Auswahl Sätze pro Zeichnung definieren, aber das nur mal am Rande. Da der Auswahl Satz namens „Auswahl“ nun existiert, können wir einen Verweis auf diesen mittels der Codezeile

```
Set AcSSet = ThisDrawing.SelectionSets("Auswahl")
```

erstellen. Um ganz sicher zu gehen, dass alles geklappt hat, können Sie den nachfolgenden Code, der die eigentliche Auswahl der AutoCAD-Objekte durchführt, durch folgende *If...End If*-Anweisung einschließen.

```
If TypeName(AcSSet) = "IAcadSelectionSet" Then
.....
End If
```

Gibt die Funktion *TypeName()* als Ergebnis der Prüfung die Zeichenfolge „IAcadSelectionSet“ zurück, so der Klassenname des Auswahlsatzes, war die Initialisierung erfolgreich und die eigentliche Auswahl von Objekten kann durchgeführt werden.

Neben der Deklaration eines Auswahlsatzes gibt es auch noch zwei weitere Methoden des Zugriffs auf Auswahlsätze. Wenn Sie im Objektkatalog die Dokumentklasse auswählen, finden Sie dort die Einträge *ActiveSelectionSet* und *PickfirstSelectionSet*. Diese speziellen Auswahlsätze haben folgende Bedeutung:

- **ActiveSelectionSet:** Diese Objektvariable ruft den gerade aktiven Auswahlsatz ab. Die Variable ist nicht initialisiert, wenn die Auflistung aller Auswahlsätze (*SelectionSets*) keine Auswahlsätze enthält. Zugriff auf diesen Auswahlsatz erhalten Sie über folgenden Code:

```
Set AcSSet = ThisDrawing.ActiveSelectionSet
If TypeName(AcSSet) <> "Nothing" Then
...
End If
```

- **PickfirstSelectionSet:** Dies ist eine Objektvariable, die einen besonderen Auswahlsatz enthält. Dieser Auswahlsatz ist nicht Bestandteil der Auswahlsatzauflistung und enthält all die Objekte, die mit dem Zeigegerät markiert wurden, ohne dass ein Befehl ausgeführt wurde. Wie Sie die Objekte markieren, können Sie in der Online-Hilfe von AutoCAD nachlesen. Obwohl dieser Auswahlsatz ein besonderes Objekt ist, so gelten auch für dieses Objekt die gleichen Eigenschaften und Prozeduren wie beim „normalen“ Auswahlsatz. Im Gegensatz zum *ActiveSelectionSet* oder auch zum Auswahlsatz aus der Auflistung der Auswahlsätze ist der *PickfirstSelectionSet* immer initialisiert und wird unter dem Namen „PICKFIRST“ gespeichert. Wurden keine Objekte gewählt, gibt die *Count*-Eigenschaft des Auswahlsatzes 0 zurück.

```
If ThisDrawing.PickfirstSelectionSet.Count > 0 Then
...
End If
```

Bevor wir uns nun den einzelnen Auswahlmethoden zuwenden, wechseln Sie einmal in den Objektkatalog und selektieren dort die Klasse *AcadSelectionSet*.

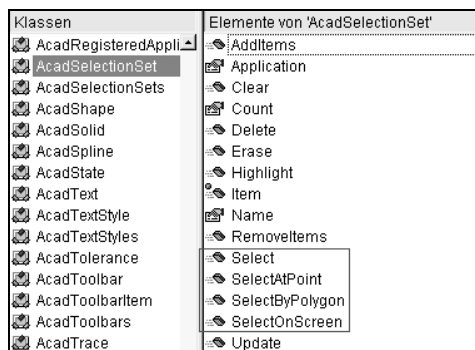


Abbildung 8.5: Auswahlsatzelemente

Wie Sie in Abbildung 8.5 erkennen können, habe ich die einzelnen Auswahlmethoden des Auswahlsatzes rot markiert. Es gibt außer diesen Methoden zwar noch die *AddItems*-Anweisung, aber darauf komme ich später noch einmal zurück. Beschäftigen wir uns zunächst mit den einzelnen *Select*-Anweisungen.

Auswahl mithilfe der *Select*-Methode

Mithilfe der *Select*-Anweisung können Sie Objekte aus der Datenbank herausfiltern, ohne dass der Benutzer eine Auswahl im Zeichnungsfenster vornimmt. Anhand der Parameter, die Sie der *Select*-Anweisung übergeben, können Sie noch einige Feinjustierungen zur Objektwahl vornehmen. Der erste Parameter dient dabei der Festlegung des Modus für die Auswahl der Objekte. Folgende Konstanten können verwendet werden:

■ *AcSelectionSetWindow*

Legt fest, dass ein rechteckiges Auswahlfenster auf die Objektauswahl angewendet werden soll. Dieses Auswahlfenster wählt dann alle Objekte, die sich komplett innerhalb des Auswahlbereichs befinden. Die Koordinaten, also die linke untere und die rechte obere Ecke, werden an die Parameter *Point1* und *Point2* übergeben. In diesem Fall müssen dann diese Punkte jeweils als dreidimensionales Array deklariert werden.

```
Dim Point1(2) As Double
```

```
Dim Point2(2) As Double
```

Um diese Punktarrays mit Koordinaten zu füllen, können Sie entweder den Benutzer auffordern, ein Fenster unter Verwendung der *Utility*-Funktionen *GetPoint* und *GetCorner* auszuwählen:

```
VarPoint = ThisDrawing.Utility.GetPoint(, "Linke untere Ecke")
Point1(0) = VarPoint(0)
Point1(1) = VarPoint(1)
VarPoint = ThisDrawing.Utility.GetCorner(Point1, "Rechte obere Ecke")
Point2(0) = VarPoint(0)
Point2(1) = VarPoint(1)
```

oder die Koordinaten den Arrays direkt zuweisen:

```
Point1(0) = 0
Point1(1) = 0
Point2(0) = 100
Point2(1) = 100
```

Nachdem alle Parametervariablen initialisiert wurden, können Sie die Auswahl der Objekte mit folgender Codezeile ausführen:

```
AcSSet.Select acSelectionSetWindow, Point1, Point2
```

Den letzten beiden Parametern, die diese Funktion anbietet, brauchen Sie im Moment noch keine Beachtung zu schenken. Diese speziellen Parameter werden noch gesondert besprochen.

■ *AcSelectionSetCrossing*

Dieser Modus hat eine ähnliche Bedeutung wie der vorhergehende. Der Unterschied besteht lediglich darin, dass alle Objekte gewählt werden, die sich vollständig oder teilweise innerhalb des Auswahlbereichs befinden. Sie können für die Definition der einzelnen

Parameter genauso vorgehen, wie bereits beim vorherigen Modus beschrieben. Für den Aufruf der Objektauswahl müssen Sie lediglich den Modusparameter ändern.

```
AcSSet.Select acSelectionSetCrossing, Point1, Point2
```

■ *AcSelectionSetPrevious*

Mit diesem Modus legen Sie fest, dass die *Select*-Methode den zuletzt verwendeten Auswahlset wiederherstellen und an diese Auswahlsetvariable übergeben soll. Dies funktioniert aber nur dann, wenn Sie nicht zwischen Papier- und Modellbereich hin- und herschalten. Andernfalls wird dieser Modus ignoriert.

```
AcSSet.Select acSelectionSetPrevious
```

■ *AcSelectionSetLast*

Mithilfe dieses Modus können Sie das zuletzt erstellte Element auswählen lassen. Wenn Sie diesen Modus verwenden, werden die Parameter, die die Auswahlpunkte definieren, ignoriert.

```
AcSSet.Select acSelectionSetLast
```

■ *AcSelectionSetAll*

Dieser Modus weist die *Select*-Methode an, alle Objekte der AutoCAD-Datenbank auszuwählen. Auch in diesem Fall werden die Punktparameter ignoriert, da ja alle Objekte ausgewählt werden.

```
AcSSet.Select acSelectionSetAll
```

Auswahl mithilfe der *SelectAtPoint*-Methode

Wie der Name dieser Methode bereits verrät, können Sie mit dieser Anweisung Objekte wählen, die sich an einem bestimmten Auswahlpunkt befinden. Da Sie über diese Funktion ja bereits den Modus für die Objektauswahl festgelegt haben, enthält die *SelectAtPoint*-Anweisung nur noch den Punktparameter, der die entsprechende Koordinate festlegt. Die beiden letzten Parameter können Sie für den Moment ebenfalls ignorieren. Diese werden gesondert besprochen.

```
Dim Point(2) As Double
VarPoint = ThisDrawing.Utility.GetPoint(, "Punkt wählen")
Point(0) = VarPoint(0)
Point(1) = VarPoint(1)
Point(2) = VarPoint(2)
AcSSet.SelectAtPoint Point
```

Auswahl mithilfe der *SelectByPolygon*-Methode

Diese *Select*-Methode ermöglicht die Auswahl von Objekten, die innerhalb einer bestimmten Auswahlregion liegen oder von einem Auswahlpolygon (Zaun) geschnitten werden. Die Koordinaten für diese Auswahlmethode werden dabei, anders als bei der Direktwahl durch den Benutzer, als Parameter mitgeliefert. Genau wie bei der *Select*-Anweisung, wird für den ersten Parameter eine der folgende Konstanten benötigt, die die Art der Objektauswahl festlegen.

■ *AcSelectionSetFence*

Wenn Sie diesen Modus aktivieren, werden durch die *SelectByPolygon*-Anweisung alle Objekte selektiert, die durch die im zweiten Parameter mitgegebene Koordinatenliste gekreuzt werden. Die Obergrenze dieses Arrays von Koordinaten ergibt sich aus der Anzahl an Stützpunkten * 3 (X, Y, Z) – 1.

```
Dim PointList(11) As Double
PointList(0) = 0: PointList(1) = 0
PointList(3) = 100: PointList(4) = 0
PointList(6) = 100: PointList(7) = 100
PointList(9) = 0: PointList(10) = 100
AcSSet.SelectByPolygon acSelectionSetFence, PointList
```

■ *AcSelectionSetWindowPolygon*

Mithilfe dieses Modus wird die Funktion angewiesen, alle Objekte innerhalb der gewählten Umgrenzung zum Auswahlset hinzuzufügen. Die Koordinaten dieser Umgrenzung werden ebenfalls als zweiter Parameter der Funktion mitgeteilt. Um eine geschlossene Umgrenzung herzustellen, verbindet die Funktion den ersten und den letzten Punkt der Koordinatenliste.

```
Dim PointList(11) As Double
PointList(0) = 0: PointList(1) = 0
PointList(3) = 100: PointList(4) = 0
PointList(6) = 100: PointList(7) = 100
PointList(9) = 0: PointList(10) = 100
AcSSet.SelectByPolygon AcSelectionSetWindowPolygon, PointList
```

■ *AcSelectionSetCrossingPolygon*

Genau wie der vorherige Modus, wählt auch dieser die Objekte innerhalb der festgelegten Umgrenzung, fügt aber zusätzlich auch die Objekte hinzu, die nur teilweise in dem Auswahlbereich liegen. Die Koordinaten der vorgesehenen Umgrenzungslinie werden auch bei diesem Modus als zweiter Parameter übergeben. Bei der Zusammenstellung der Umgrenzungslinie müssen Sie allerdings darauf achten, dass sich die einzelnen Linien nicht kreuzen.

```
Dim PointList(11) As Double
PointList(0) = 0: PointList(1) = 0
PointList(3) = 100: PointList(4) = 0
PointList(6) = 100: PointList(7) = 100
PointList(9) = 0: PointList(10) = 100
AcSSet.SelectByPolygon AcSelectionSetCrossingPolygon, PointList
```

Auswahl mithilfe der SelectOnScreen-Methode

Neben den *Select*-Anweisungen, bei denen die Objekte nicht direkt im Zeichnungsfenster ausgewählt werden, stellt AutoCAD-VBA auch eine Funktion zur Verfügung, mit der der Benutzer eine direkte Auswahl im Zeichnungsfenster vornehmen kann. Dabei handelt es sich um die *SelectOnScreen*-Anweisung. Die Ausführung dieser Anweisung hat in der AutoCAD-Befehlszeile die gleiche Funktionalität wie bei jedem anderen AutoCAD-Befehl, der mehrere Objekte gleichzeitig auswählen lässt. Der Benutzer kann Objekte aktivieren, indem er diese einzeln mit dem Mauszeiger auswählt, ein Auswahlfenster um sie zeichnet, die Koordinaten eingibt oder eine der im Folgenden aufgelisteten Objektwahlmethoden verwendet.

Fenster/Letztes/Kreuzen/QUADER/Alle/Zaun/FPolygon/KPolygon/Gruppe/Hinzufügen/Entfernen/
Mehrfach/Vorher/Zurück/Auto/Einzel

Da der Benutzer die Aufgabe der Auswahl des Selektionsmodus übernimmt, werden für diese Funktion außer des Filters keine weiteren Parameter benötigt. Aus diesem Grund werde ich die bisher verschobene Beschreibung der Filterparameter anhand von Beispielen genauer erklären. Sollten Sie keinen Filter verwenden wollen, können Sie diese Funktion folgendermaßen aktivieren:

```
AcSSet.SelectOnScreen
```


Mit Filter wird die Funktion dagegen folgendermaßen aufgerufen:

```
AcSSet.SelectOnScreen FilterType, FilterData
```

Um die Filtervariablen verwenden zu können, müssen sie wie folgt deklariert werden:

```
Dim FilterType(n) As Integer
Dim FilterData(n) As Variant
```

Der Zähler *n* setzt die Obergrenze der beiden Arrays *Type* und *Data*. Da das Array, wie bereits bei anderen Funktionen, mit Index 0 beginnt, liegt die Obergrenze bei „Anzahl der Filter – 1“. Sie können die beiden Arrays natürlich auch dynamisch dimensionieren, um z.B. einen Filterdialog für den Anwender zu erstellen, indem Sie die Deklaration folgendermaßen vornehmen:

```
ReDim FilterType(n) As Integer
ReDim FilterData(n) As Variant
```

Welchem Zweck dienen nun aber diese beiden Parameter, die ja offensichtlich eng miteinander verknüpft sind? Nun, mit dem ersten Parameter, in diesem Fall also die Variable *FilterType*, legen Sie den Datentyp bzw. DXF-Gruppencode des Filterarguments fest, im zweiten Parameter (*FilterData*) den eigentlichen zu filternden Wert. Angenommen, Sie wollen alle Blockreferenzen der aktuellen Zeichnung auswählen. Zu diesem Zweck müssten Sie die Filtervariablen folgendermaßen füllen:

```
Dim FilterType(0) As Integer
Dim FilterData(0) As Variant
FilterType(0) = 0
FilterData(0) = "INSERT"
AcSSet.SelectOnScreen FilterType, FilterData
```

Der erste Parameter (*FilterType*) wird mit dem DXF-Gruppencode 0 angegeben, also einem Hinweis darauf, dass die Objekte eines bestimmten Elementtyps gefiltert werden sollen. Welcher Elementtyp gemeint ist, legt der zweite Parameter fest. Da sich der Objektfilter an den Definitionen der DXF-Schnittstelle orientiert, wird als Name des Elementtyps nicht *Blockreferenz* sondern *INSERT* verwendet. Um das Ganze besser zu veranschaulichen, folgt ein weiteres Beispiel. Als Nächstes sollen nicht die Blockreferenzen, sondern alle Elemente mit Namen „01“ gefiltert werden.

```
Dim FilterType(0) As Integer
Dim FilterData(0) As Variant
FilterType(0) = 2
FilterData(0) = "01"
AcSSet.SelectOnScreen FilterType, FilterData
```

In solch einem Fall könnten aber nicht nur Blöcke mit dem Namen „01“ zurückgegeben werden, sondern auch Linien, Texte usw. Wollen Sie nur Blockreferenzen mit dem Namen „01“ selektieren, so muss der Auswahlfilter weiter eingeschränkt werden.

```
Dim FilterType(1) As Integer
Dim FilterData(1) As Variant
FilterType(0) = 0
FilterData(0) = "INSERT"
FilterType(1) = 2
FilterData(1) = "01"
AcSSet.SelectOnScreen FilterType, FilterData
```

Diese Filterlisten könnten Sie weiter aufbauen, indem Sie die Variablen entsprechend dimensionieren und jeweils DXF-Gruppencode und Wert den entsprechenden Variablen zuweisen. Beachten müssen Sie dabei, dass die einzelnen DXF-Gruppencodes als Werte Daten unterschiedlicher Typen verlangen. Die Tabelle 8.2 bietet eine kurze Übersicht über einige DXF-Gruppencodes, die Verwendung finden könnten. Im Anhang des Buchs finden Sie eine Auflistung aller DXF-Gruppencodes.

Gruppencode	Beschreibung	Datentyp
0	Textzeichenfolge für den Elementtyp	String
2	Name (Attributbezeichnung, Block ...)	String
5	Elementreferenz (Handle)	String
10	X-Wert (Linienanfangspunkt, Mittelpunkt, Einfügapunkt...)	Double
20	Y-Wert (Linienanfangspunkt, Mittelpunkt, Einfügapunkt...)	Double
30	Z-Wert (Linienanfangspunkt, Mittelpunkt, Einfügapunkt...)	Double
48	Linientypfaktor	Double
60	Sichtbarkeit (0: sichtbar, 1: unsichtbar)	Integer
62	Farbnummer	Integer

Tabelle 8.2: Auflistung einiger DXF-Gruppencodes

Eine weitere wichtige Funktion, die der Filter bieten sollte, ist die Möglichkeit, bestimmte Werte auszuschließen. Zu diesem Zweck können Sie mithilfe des DXF-Gruppencodes -4 einen Vergleichsoperator festlegen. In der Tabelle 8.3 sind alle möglichen Vergleichsoperatoren aufgelistet.

Operator	Beschreibung
*	alles verwenden
=	gleich
!=	ungleich
/=	ungleich
<>	ungleich
<	kleiner
<=	kleiner gleich
>	größer
>=	größer gleich
&	bitweise UND (nur für integer Gruppencode)
&=	bitweise maskiertes gleich (nur für integer Gruppencode)

Tabelle 8.3: Vergleichsoperatoren

Alle diese Operatoren können Sie einsetzen, um bestimmte Vergleiche in den nachfolgenden Filtervariablen vorzunehmen. Ein Beispiel: Wenn Sie z.B. alle Kreise einer Zeichnung herausfiltern möchten, deren Radius größer 5.0 m ist, muss der entsprechende Code folgendermaßen aussehen:

```

Dim FilterType(2) As Integer
Dim FilterData(2) As Variant
FilterType(0) = 0
FilterData(0) = "CIRCLE"
FilterType(1) = -4
FilterData(1) = ">"
FilterType(2) = 40
FilterData(2) = 5
AcSSet.SelectOnScreen FilterType, FilterData

```

Ein weiteres Beispiel: Aus der Zeichnung sollen alle Kreise herausgefiltert werden, deren Radius größer 5, aber kleiner 10 ist. Der Code dieser Aufgabenstellung:

```

Dim FilterType(4) As Integer
Dim FilterData(4) As Variant
Dim AcSSet As AcadSelectionSet
If TypeName(ThisDrawing.SelectionSets("Auswahl")) = "Nothing" Then
    ThisDrawing.SelectionSets.Add "Auswahl"
End If
Set AcSSet = ThisDrawing.SelectionSets("Auswahl")
FilterType(0) = 0
FilterData(0) = "CIRCLE"
FilterType(1) = -4
FilterData(1) = ">"
FilterType(2) = 40
FilterData(2) = 5
FilterType(3) = -4
FilterData(3) = "<"
FilterType(4) = 40
FilterData(4) = 10
AcSSet.SelectOnScreen FilterType, FilterData

```

Neben der Möglichkeit, durch Verwendung von Vergleichsoperatoren bestimmte Objekte aus- oder einzuschließen, können Sie auch Gruppierungsoperatoren (*AND*, *OR*, *XOR* und *NOT*) benutzen. Mithilfe dieser Gruppierungsoperatoren werden Beziehungen zwischen diesen Objekten hergestellt, die als Ergebnis eine Auswahl von Objekten liefern, die bestimmten Parametern entsprechen. Die Anzahl der eingeschlossenen Operanden ist dabei von der jeweiligen Operation abhängig. In der Tabelle 8.4 ist eine Auflistung der einzelnen Gruppierungsoperatoren aufgeführt. Bis auf einen, sind alle anderen Operatoren relativ leicht verständlich. Bei diesem einen Operator handelt es sich um die *XOR*-Gruppierung. Ich möchte zu diesem Operator noch einige zusätzliche Anmerkungen machen, bevor wir uns Beispiele zu diesem Thema anschauen. Mit dem *XOR*-Operator soll die Filterfunktion eine logische Exklusion zwischen den zwei Ausdrücken vornehmen. Wenn genau ein Ausdruck wahr ist, so ist das Ergebnis ebenfalls wahr. Ist einer der beiden Ausdrücke allerdings falsch, so ist das Ergebnis ebenfalls falsch. Ist keiner der Ausdrücke falsch, so wird das Ergebnis gemäß der Tabelle 8.5 festgelegt.

Start	Beinhaltet	Ende	Beschreibung
<AND	Einen oder mehrere Operanden	AND>	Legt fest, dass das gewählte Objekt allen Operanden entsprechen muss, um ausgewählt zu werden.
<OR	Einen oder mehrere Operanden	OR>	Legt fest, dass das gewählte Objekt einem der Operanden entsprechen muss, um ausgewählt zu werden.
<XOR	Zwei Operanden	XOR>	Dient zum Durchführen einer logischen Exklusion zwischen zwei Objekten.
<NOT	Einen Operanden	NOT>	Schließt alle Objekte aus, die dem Operanden entsprechen.

Tabelle 8.4: Gruppierungsoperatoren

Erster Operator	Zweiter Operator	Ergebnis
TRUE	TRUE	Nicht gefiltert
TRUE	FALSE	Gefiltert
FALSE	TRUE	Gefiltert
FALSE	FALSE	Nicht gefiltert

Tabelle 8.5: Mögliche XOR-Ergebnisse

Ziel des nun folgenden Beispiels soll es sein, sowohl 2D-Polylinien als auch LWPolylinien aus der Zeichnung auszuwählen. Zu diesem Zweck müssen diese beiden Objekttypen in einer OR-Gruppierung eingeschlossen werden. Das Ganze sieht dann folgendermaßen aus:

```
Dim FilterType(3) As Integer
Dim FilterData(3) As Variant
FilterType(0) = -4
FilterData(0) = "<OR"
FilterType(1) = 0
FilterData(1) = "POLYLINE"
FilterType(2) = 0
FilterData(2) = "LWPOLYLINE"
FilterType(3) = -4
FilterData(3) = "OR>"
AcSSet.SelectOnScreen FilterType, FilterData
```

Die einzelnen Gruppierungen können auch ineinander verschachtelt werden. Erweitern wir das Beispiel so, dass nur Polylinien und LWPolylinien gefiltert werden, die auf Layer 0 und Layer 1 liegen:

```
Dim FilterType(9) As Integer
Dim FilterData(9) As Variant
FilterType(0) = -4
FilterData(0) = "<OR"
FilterType(1) = 0
FilterData(1) = "POLYLINE"
FilterType(2) = 0
FilterData(2) = "LWPOLYLINE"
```

```

FilterType(3) = -4
FilterData(3) = "OR>"
FilterType(4) = -4
FilterData(4) = "<AND"
FilterType(5) = -4
FilterData(5) = "<OR"
FilterType(6) = 8
FilterData(6) = "0"
FilterType(7) = 8
FilterData(7) = "1"
FilterType(8) = -4
FilterData(8) = "OR>"
FilterType(9) = -4
FilterData(9) = "AND>"
AcSSet.SelectOnScreen FilterType, FilterData

```

Nachdem Sie einen Überblick über die einzelnen Kombinationsmöglichkeiten des Filters erhalten haben, möchte ich noch eine Variante erwähnen, die Sie anstelle der *AND/OR*-Operatoren verwenden könnten. Um zum Beispiel Objekte aus der Zeichnung herauszufiltern, die auf den Layern „Wasserleitung“ und „Stromleitung“ liegen, könnten Sie dies mithilfe der *AND/OR*-Gruppierung erledigen:

```

Dim FilterType(3) As Integer
Dim FilterData(3) As Variant
FilterType(0) = -4
FilterData(0) = "<OR"
FilterType(1) = 8
FilterData(1) = "Wasserleitung"
FilterType(2) = 8
FilterData(2) = "Stromleitung"
FilterType(3) = -4
FilterData(3) = "OR>"
AcSSet.SelectOnScreen FilterType, FilterData

```

Sie können alternativ auch die so genannten Wildcards, wie sie ja bereits aus DOS-Zeiten bekannt sind, zur Auswahl der Objekte verwenden. Da die beiden Layer einen gemeinsamen Textteil („leitung“) enthalten, kann der Rest des gewünschten Layer-Namens mit Platzhaltern aufgefüllt werden, um zum gleichen Ergebnis zu kommen.

```

Dim FilterType(0) As Integer
Dim FilterData(0) As Variant
FilterType(0) = 8
FilterData(0) = "*leitung"
AcSSet.SelectOnScreen FilterType, FilterData

```

Welche Wildcards Sie für die Filterung verwenden können und welche Wirkung diese bei der Filterung erzielen, können Sie der folgenden Tabelle entnehmen.

Wildcard	Beschreibung
#	Platzhalter für eine beliebige Ziffer (0-9).
@	Platzhalter für ein beliebiges alphanumerisches Zeichen.
.	Platzhalter für ein beliebiges nicht alphanumerisches Zeichen.

Tabelle 8.6: Wildcard-Zeichen

Wildcard	Beschreibung
*	Kein oder mehrere Zeichen sind bis zur folgenden oder ab der vorhergehenden Zeichenfolge gültig.
?	Dient als Platzhalter eines einzelnen Zeichens.
~	Filtert alle Objekte, die nicht der Zeichenfolge entsprechen, die dem Tilde-Zeichen folgt.
[]	Ergibt eine Übereinstimmung, wenn die Zeichenfolge ein beliebiges Zeichen, das in der Zeichenliste gefunden wird, enthält.
[~]	Ergibt eine Übereinstimmung, wenn in der Zeichenfolge ein beliebiges Zeichen, außer den Zeichen in der Zeichenliste, gefunden wird.
-	
,	Trennt zwei Muster voneinander
`	Dieses Zeichen legt fest, dass das nachfolgende Zeichen nicht als Wildcard interpretiert werden soll.

Tabelle 8.6: Wildcard-Zeichen (Forts.)

Hier noch eine Anmerkung zum besseren Verständnis zur Benutzung des umgekehrten Quote-Zeichens ('). Dieses Zeichen kann z.B. verwendet werden, um anonyme Blöcke herauszufiltern. Da anonyme Blöcke einen * vor ihrem Namen verwenden, muss dem Muster dieses Zeichen vorangestellt werden, damit der * als Zeichen und nicht als Platzhalter interpretiert werden kann:

```
Dim FilterType(0) As Integer
Dim FilterData(0) As Variant
FilterType(0) = 0
FilterData(0) = "INSERT"
FilterType(1) = 2
FilterData(1) = "`*U1"
AcSSet.SelectOnScreen FilterType, FilterData
```

Zusätzlich zu den *Select*-Anweisungen, bietet AutoCAD-VBA weitere Funktionen zur Manipulation der Auswahlsätze. Im Folgenden werden diese Funktionen aufgelistet und näher beschrieben.

- **AddItems:** Diese Anweisung fügt dem Auswahlatz ein oder mehrere Objekte hinzu. Diese Funktion erwartet als Parameter eine Liste der Objekte, die zum Auswahlatz hinzugefügt werden sollen. Diese Liste wird als Array deklariert, wobei jedes einzelne Element der Liste als *AcadEntity* behandelt wird. Im folgenden Beispiel werden eine Linie und eine Polylinie dem Auswahlatz hinzugefügt. Die Linie könnte dabei auf verschiedene Weise ermittelt worden sein, worauf ich in diesem Abschnitt aber nicht weiter eingehen möchte. Wie Sie eine Linie oder Polylinie erzeugen, erfahren Sie in den entsprechenden Kapiteln.

```
Dim ssObjs(1) As AcadEntity
Set ssObjs(0) = SelLine
Set ssObjs(1) = SelPoly
AcSSet.AddItem ssObjs
```

- **Clear:** Wenn Sie diese Anweisung auf den entsprechenden Auswahlatz ausführen, werden alle Elemente aus dem Auswahlatz entfernt. Der Auswahlatz bleibt erhalten, enthält aber keine ausgewählten Objekte mehr.

- **Count:** Diese Eigenschaft gibt die Anzahl der im Auswahlset enthaltenen Objekte zurück.
- **Delete:** Mit dieser Anweisung können Sie den Auswahlset aus der Auflistung aller Auswahlsets löschen. Im Gegensatz zu den anderen Auflistungen, wie Layer oder Textstile, können Sie einen Auswahlset nicht in der Auflistung löschen, sondern müssen dann auf diese Anweisung zurückgreifen.
- **Erase:** Im Gegensatz zur *Delete*-Anweisung entfernt diese Funktion die Objekte nicht nur aus dem Auswahlset, sondern sie löscht sie auch aus der Zeichnung.
- **Highlight:** Mit dieser Anweisung können Sie den Markierungsstatus für alle Objekte des angegebenen Auswahlsets festlegen. Wenn Sie als Parameter *TRUE* mitgeben, werden alle Objekte hervorgehoben, mit *FALSE* wird diese Markierung wieder aufgehoben

```
AcSSet.Highlight True
```

- **Item:** Diese Methode ruft ein bestimmtes Element an einer bestimmten Stelle im Auswahlset ab und gibt dieses als Ergebnis an eine Variable weiter.

```
Dim ssObj As AcadEntity
Set ssObj = AcSSet.Item(0)
```

- **Name:** Mithilfe dieser Eigenschaft können Sie den Namen des Auswahlsets abrufen. Da diese Eigenschaft schreibgeschützt ist, können Sie den Namen nur auslesen, aber nicht ändern.
- **RemoveItems:** Diese Methode bildet das Gegenstück zur Anweisung *AddItems* und soll dementsprechend die Objekte, die als Parameter übergeben wurden, aus dem Auswahlset entfernen. Der Codeaufbau dieser Funktion ist weitgehend identisch mit der *AddItems*-Anweisung:

```
Dim ssObjs(1) As AcadEntity
Set ssObjs(0) = SelLine
Set ssObjs(1) = SelPoly
AcSSet.RemoveItems ssObjs
```

- **Update:** Mit dieser Anweisung können Sie den Auswahlset aktualisieren.

Zusammenfassend gilt festzuhalten, dass die Verwendung von Auswahlsets in zwei Schritten abläuft. Zuerst wird der Auswahlset erzeugt und im zweiten Schritt werden Objekte gewählt, sei es durch den Benutzer oder per Code.

8.3 AutoCAD-Objekte verarbeiten

8.3.1 Grundlegende Eigenschaften und Methoden grafischer Objekte

Es gibt gewisse Eigenschaften und Methoden, die in jedem grafischen AutoCAD-Objekt enthalten sind. Dazu gehören z.B. die farbliche Gestaltung des Elements oder die Layer-Eigenschaft. Im folgenden Abschnitt sind alle diese Eigenschaften und Methoden aufgeführt, die jedes grafische Objekt besitzt. Egal von welchem Datentyp ein AutoCAD-Element ist, Sie können auf jeden Fall diese Eigenschaften und Methoden auf das Objekt anwenden.

Farbe von AutoCAD-Objekten

Sie können allen grafischen Objekten Farben zuweisen. Die Farben werden in AutoCAD zwar mit ihrem Namen angezeigt, zugewiesen werden aber Nummern im Bereich von 0 bis 256. Dies entspricht 255 Farben (1–255) sowie den zusätzlichen Werten für die Farbe des Layers, auf dem das Objekt abgelegt wurde, und der Farbe einer Blockdefinition. Einige Farben sind in VBA bereits Konstanten zugeordnet worden, die Sie statt der Nummer verwenden können. Die restlichen Farben müssen anhand ihrer Nummer zugewiesen werden.

Konstante	Beschreibung	Wert
<i>acByBlock</i>	Farbe Block	0
<i>acByLayer</i>	Farbe Layer	256
<i>acRed</i>	Farbe Rot	1
<i>acYellow</i>	Farbe Gelb	2
<i>acGreen</i>	Farbe Grün	3
<i>acCyan</i>	Farbe Cyan	4
<i>acBlue</i>	Farbe Blau	5
<i>acMagenta</i>	Farbe Magenta	6
<i>acWhite</i>	Farbe Weiß/Schwarz, je nach AutoCAD-Hintergrundfarbe	7

Tabelle 8.7: Farbkonstanten

Zuweisung der Farbe mittels einer Konstanten:

```
Object.Color = acBlue
```

oder über die Farbnummer:

```
Object.Color = 5
```

Auf der Buch-CD finden Sie übrigens eine Bibliothek, die ein Formular enthält, das dem Farbauswahldialog von AutoCAD nachempfunden wurde. Sie können dieses Formular in Ihren VBA-Projekten verwenden, um dem Benutzer eine erweiterte Farbauswahl anzubieten.

Hyperlinks

Hyperlinks sind Verbindungen von Objekten zu Dokumenten, z.B. anderen Zeichnungen oder Textdokumenten. Da es sich bei dieser Eigenschaft um ein komplexes Thema handelt, habe ich den Hyperlinks das *Kapitel 7.3* gewidmet.

Layer

Jedes Zeichnungselement ist bekanntlich auf einem Layer abgelegt. Sie können den Layer dieser Objekte jederzeit ändern, indem Sie den entsprechenden Layer in die Layer-Eigenschaft des Objekts eintragen.

```
Object.Layer = "0"
```

Wie Sie anhand der Codezeile sehen können, wird dabei nicht das gesamte Layer-Objekt, sondern nur der Name des entsprechenden Layers verwendet. Falls Sie den Namen des Layers ändern, erfolgt eine Anpassung der Layer-Eigenschaft automatisch durch AutoCAD.

Siehe auch *Kapitel 8.4.1*.

Linientypen

Die Linientypeigenschaft bestimmt die linienhafte Darstellung des Objekts und wird in der Eigenschaft *Linetype* abgespeichert. Sie können dieser Eigenschaft alle Linientypen der Zeichnung, in der das Objekt erzeugt wurde, zuordnen. Ausgenommen davon sind natürlich Linientypen aus einer externen Referenz. Der Linientyp wird zwar zugeordnet, was aber unter Umständen zu Fehlern führen könnte. Deshalb müssen Sie darauf achten, dass solche Operationen durch Ihre Programme ausgeschlossen werden. Die Zuordnung des Linientyps erfolgt, wie bei den Layern, nicht durch das komplette Objekt, sondern nur über den Namen des entsprechenden Linientyps.

```
Object.Linetype = "Continuous"
```

Wenn Sie die Linientypen „vonLayer“ oder „vonBlock“ zuordnen wollen, müssen Sie die englischen Bezeichnungen „byLayer“ oder „byBlock“ verwenden, damit AutoCAD keinen Fehler auslöst.

Siehe auch *Kapitel 7.5.1*.

Linientypfaktor

Wenn Sie den Linientypfaktor eines Objekts anpassen möchten, können Sie auf die Objekteigenschaft *LinetypeScale* zurückgreifen. In dieser wird der aktuelle Wert des Linientypfaktors eines einzelnen Objekts verwaltet. Als Wert müssen Sie eine positive reelle Zahl verwenden, andernfalls wird ein Fehler ausgelöst.

```
Object.LinetypeScale = 1.5
```

Linienstärke

Die Linienstärke wird über die Eigenschaft *Lineweight* jedem grafischen Objekt zugeordnet. Der Wert, den Sie dort festlegen können, entspricht der Stärke der Linie. Für eine Strichstärke von z. B. 0.25 mm müssen Sie den Wert 25 oder für 0,13 mm den Wert 13 angeben. In VBA ist bereits eine Konstantenauflistung definiert worden, die alle möglichen Linienstärken definiert:

Konstante	Wert	Englisch (Zoll)	Deutsch (mm)
<i>AcLnWtByLwDefault</i>	-3	Default	Vorgabe
<i>AcLnWtByBlock</i>	-2	byBlock	vonBlock
<i>AcLnWtByLayer</i>	-1	byLayer	vonLayer
<i>acLnWt000</i>	0	0.000"	0,00 mm
<i>acLnWt005</i>	5	0.002"	0,05 mm
<i>acLnWt009</i>	9	0.004"	0,09 mm
<i>acLnWt013</i>	13	0.005"	0,13 mm
<i>acLnWt015</i>	15	0.006"	0,15 mm
<i>acLnWt018</i>	18	0.007"	0,18 mm
<i>acLnWt020</i>	20	0.008"	0,20 mm
<i>acLnWt025</i>	25	0.010"	0,25 mm

Tabelle 8.8: Konstanten der Linienstärke

Konstante	Wert	Englisch (Zoll)	Deutsch (mm)
acLnWt030	30	0.012"	0,30 mm
acLnWt035	35	0.014"	0,35 mm
acLnWt040	40	0.016"	0,40 mm
acLnWt050	50	0.020"	0,50 mm
acLnWt053	53	0.021"	0,53 mm
acLnWt060	60	0.024"	0,60 mm
acLnWt070	70	0.028"	0,70 mm
acLnWt080	80	0.031"	0,80 mm
acLnWt090	90	0.035"	0,90 mm
acLnWt100	100	0.039"	1,00 mm
acLnWt106	106	0.042"	1,06 mm
acLnWt120	120	0.047"	1,20 mm
acLnWt140	140	0.055"	1,40 mm
acLnWt158	158	0.062"	1,58 mm
acLnWt200	200	0.079"	2,00 mm
acLnWt211	211	0.083"	2,11 mm

Tabelle 8.8: Konstanten der Linienstärke (Forts.)

Sie können nun die Zuordnung der Linienstärke über den eigentlichen numerischen Wert vornehmen:

```
Object.Lineweight = 90
```

oder eine der Konstanten aus Tabelle 8.8 verwenden:

```
Objekt.Lineweight = acLnWt090
```

Plotstilname

In dieser Eigenschaft ist der Name des Plotstils, der dem grafischen Objekt zugeordnet ist, abgespeichert. Auch hier gilt, verwenden Sie die Plotstile „vonLayer“ oder „vonBlock“, müssen Sie bei der Zuordnung die englische Bezeichnung nehmen.

```
Object.PlotstyleName = "ByLayer"
```

Sichtbarkeit grafischer Objekte steuern

Die Sichtbarkeit grafischer Objekte können Sie mithilfe der Eigenschaft *Visible* steuern. Die Verwendung dieser Eigenschaft ist recht einfach. Wenn Sie dieser Eigenschaft den Wert *FALSE* zuweisen, ist das Objekt im Zeichnungsfenster nicht mehr sichtbar und somit auch nicht durch den Benutzer auswählbar. Sollten Sie das Objekt unsichtbar machen, müssen Sie es dem Benutzer ermöglichen, auf das Objekt zuzugreifen. Sie können hierzu den Handle des Objekts in einer Variablen speichern und später den Handle wieder in ein Objekt umwandeln (*HandleToObject*).

Objekt unsichtbar machen:

```
Object.Visible = False
```

Objekt wieder sichtbar machen:

```
Object.Visible = True
```

Eine polare Anordnung erstellen

Diese Funktion *ArrayPolar* erstellt eine polare Anordnung eines Ursprungsobjekts. Mit den Parametern *Winkel* und *Anzahl* bestimmen Sie die Position der neuen Objekte. Der Winkel wird dabei durch die Anzahl der Objekte – 1 geteilt. Die neu erstellten Objekte sind von der gleichen Klasse wie das Ursprungsobjekt. Als Ergebnis der Funktion erhalten Sie ein Array mit den Objektvariablen der durch diese Funktion entstandenen Objekte. Entscheidend für die erste Richtung und Strecke ist der Parameter *Zentrumspunkt* und das Objekt, das ausgewählt wurde. Aus diesen beiden Vorgaben berechnet AutoCAD eine Ausgangsgerade. Der jeweilige Bezugspunkt dieser Geraden wird dabei wie folgt bestimmt.

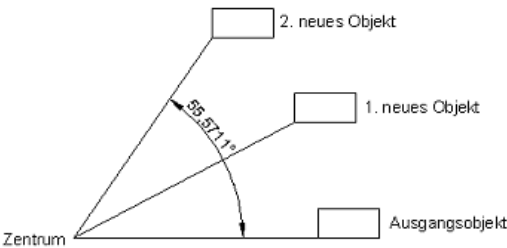


Abbildung 8.6: Polare Anordnung

Objekt	Verwendeter Punkt
<i>AcadCircle, AcadArc</i>	Das Zentrum des Objekts wird als Bezugspunkt verwendet.
<i>AcadBlockReference, AcadShape</i>	Der Einfügepunkt des Objekts wird als Bezugspunkt verwendet.
<i>AcadText</i>	Der Startpunkt des Textes wird als Bezugspunkt verwendet.
<i>AcadLine, AcadTrace</i>	Einer der Endpunkte des Objekts wird als Bezugspunkt verwendet.

Tabelle 8.9: Bezugspunktdefinition verschiedener Objekte für polare Anordnungen

```
Public Sub CreateArrayPolar()  
Dim VarPoint As Variant  
Dim Object As Object  
Dim PickedPoint As Variant  
Dim FillAngle As Double  
Dim ObjCount As Integer  
Dim Center(2) As Double  
Dim Prompt As String  
On Local Error Resume Next  
Prompt = "Wählen Sie ein Objekt:"
```

```

ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Prompt = "Wählen Sie das Zentrum der polaren Anordnung:"
    VarPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Prompt = "Geben Sie den Winkel zwischen den Objekten an:"
    FillAngle = ThisDrawing.Utility.GetOrientation(, Prompt)
    Prompt = "Legen Sie die Anzahl der Elemente fest:"
    ObjCount = ThisDrawing.Utility.GetInteger(Prompt)
    Center(0) = VarPoint(0)
    Center(1) = VarPoint(1)
    Center(2) = VarPoint(2)
    RetEnt = Object.ArrayPolar(ObjCount, FillAngle, Center)
nd If
End Sub

```

Nachdem alle Variablen deklariert wurden, wird mithilfe der *Utility*-Funktion *GetEntity* das Objekt abgefragt, aus dem die polare Anordnung erstellt werden soll. Anschließend werden alle Parameter für die Funktion abgefragt und in den entsprechenden Variablen gespeichert. Der Mittelpunkt wird an das 3D-Array *Center* übergeben und die Funktion *ArrayPolar* ausgeführt. Wenn die Funktion erfolgreich war, wird eine polare Anordnung erstellt und die Verweise auf die neuen Objekte finden sich in der Variablen *RetEnt* wieder. Sie können die Anzahl der Objekte, die erzeugt wurden, mithilfe der *UBound*-Anweisung ermitteln.

```
MsgBox "Anzahl der erzeugten Objekte: " & CStr(UBound(RetEnt) + 1)
```

Eine rechteckige Anordnung erstellen

Die Funktion *ArrayRectangular* hat eine ähnliche Wirkung wie die vorhergehende. Nur wird bei dieser Funktion die Anordnung nicht polar, sondern rechteckig erzeugt. Mit den Parametern *NumberOfRows* und *NumberOfColumns* legen Sie die Anzahl der Zeilen und Spalten fest. Wenn Sie eine dreidimensionale Anordnung erstellen möchten, können Sie über den Parameter *NumberOfLevels* zusätzlich die Anzahl der Z-Schichten bestimmen. Außerdem müssen noch die Parameter, die den Abstand der einzelnen Objekte untereinander festlegen, übergeben werden. Hier die einzelnen Parameter noch mal zusammengefasst.

- *NumberOfRows* – legt die Anzahl der Zeilen fest
- *NumberOfColumns* – legt die Anzahl der Spalten fest
- *NumberOfLevels* – legt die Anzahl der Ebenen fest
- *DistBetweenRows* – legt den Zeilenabstand fest
- *DistBetweenCols* – legt den Spaltenabstand fest
- *DistBetweenLevels* – legt den Abstand zwischen den Ebenen fest.

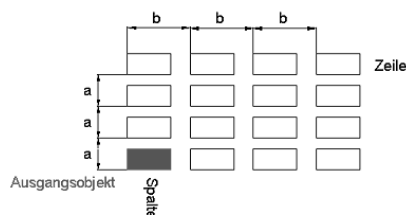


Abbildung 8.7: Rechteckige Anordnung

Tipp

Wenn Sie für die Anzahl der Reihen den Wert 1 verwenden, muss die Anzahl der Spalten größer 1 sein, damit eine Anordnung erzeugt werden kann. Genauso muss die Anzahl der Zeilen größer 1 sein, wenn die Anzahl der Spalten auf 1 gesetzt wird.

```
Public Sub CreateArrayRectangular()
Dim Object As Object
Dim PickedPoint As Variant
Dim ObjRows As Integer
Dim ObjColumns As Integer
Dim ObjHeight As Integer
Dim RowDist As Double
Dim ColumnDist As Double
Dim ZDist As Double
Dim Prompt As String
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Prompt = "Legen Sie die Anzahl der Zeilen fest:"
    ObjRows = ThisDrawing.Utility.GetInteger(Prompt)
    Prompt = "Legen Sie die Anzahl der Spalten fest:"
    ObjColumns = ThisDrawing.Utility.GetInteger(Prompt)
    Prompt = "Legen Sie die Anzahl der Schichten fest:"
    ObjHeight = ThisDrawing.Utility.GetInteger(Prompt)
    Prompt = "Geben Sie den Zeilenabstand an:"
    RowDist = ThisDrawing.Utility.GetDistance(, Prompt)
    Prompt = "Geben Sie den Abstand der Spalten an:"
    ColumnDist = ThisDrawing.Utility.GetDistance(, Prompt)
    Prompt = "Geben Sie den Höhenunterschied an:"
    ZDist = ThisDrawing.Utility.GetDistance(, Prompt)
    RetEnt = Object.ArrayRectangular ObjRows, ObjColumns, _
        ObjHeight, RowDist, ColumnDist, ZDist
End If
End Sub
```

Nachdem auch hier alle Variablen deklariert wurden, wird mithilfe der *Utility*-Funktion *GetEntity* das Objekt abgefragt, aus dem die rechteckige Anordnung erstellt werden soll. Anschließend werden alle Parameter für die Funktion abgefragt und in den entsprechenden Variablen gespeichert. Wenn die Funktion *ArrayRectangular* erfolgreich war, wird eine rechteckige Anordnung erstellt und die Verweise auf die neuen Objekte finden sich in der Variablen *RetEnt* wieder. Sie können die Anzahl der Objekte, die erzeugt wurden, mithilfe der *UBound()*-Funktion ermitteln.

```
MsgBox "Anzahl der erzeugten Objekte: " & CStr(UBound(RetEnt) + 1)
```

Objekte kopieren, verschieben und drehen

Kopieren (Copy-Funktion)

Diese Funktion dupliziert das Objekt, mit dem Sie diese Funktion ausführen und gibt als Ergebnis das neu erstellte Objekt zurück. Dabei erhält das neue Objekt die gleichen Eigenschaften (Position, Drehung, Skalierung, Layer ...) wie das Ausgangsobjekt. Da diese Funktion eine Objektvariable zurückliefert, müssen Sie eine Zuweisung mit der *Set*-Anweisung ausführen.

```

Public Sub CopyObject()
Dim Object As Object
Dim Prompt As String
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Set NewObj = Object.Copy()
End If
End Sub

```

Verschieben (Move-Funktion)

Mit dieser Funktion können Sie das Objekt, mit dem Sie diese Prozedur ausgeführt haben, von einem Ausgangspunkt auf einen Zielpunkt verschieben. Als Parameter müssen nur der Start- und der Zielpunkt jeweils als dreidimensionales Array angegeben werden.

```

Public Sub MoveObject()
Dim Object As Object
Dim Prompt As String
Dim FromPnt(2) As Double
Dim ToPnt(2) As Double
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) = "IAcadBlockReference" Then
    VarPoint = Object.InsertionPoint
    FromPnt(0) = VarPoint(0)
    FromPnt(1) = VarPoint(1)
    FromPnt(2) = VarPoint(2)
    Prompt = "Blockreferenz verschieben nach:"
    VarPoint = ThisDrawing.Utility.GetPoint(FromPnt, Prompt)
    ToPnt(0) = VarPoint(0)
    ToPnt(1) = VarPoint(1)
    ToPnt(2) = VarPoint(2)
    Object.Move FromPnt, ToPnt
End If
End Sub

```

Drehen (Rotate-Funktion)

Mit dieser Funktion können Sie das Objekt, mit dem Sie diese Prozedur ausgeführt haben, um einen Basispunkt drehen lassen. Als Parameter müssen nur der Basispunkt als dreidimensionales Array sowie der Drehwinkel, um den das Objekt gedreht werden soll, angegeben werden.

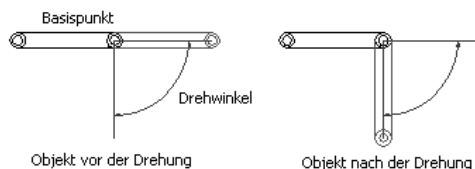


Abbildung 8.8: Objekte drehen

```

Public Sub RotateObject()
Dim Object As Object
Dim Prompt As String
Dim FromPnt(2) As Double
Dim Angle As Double
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) = "IAcadBlockReference" Then
    VarPoint = Object.InsertionPoint
    FromPnt(0) = VarPoint(0)
    FromPnt(1) = VarPoint(1)
    FromPnt(2) = VarPoint(2)
    Prompt = "Blockreferenz drehen um:"
    Angle = ThisDrawing.Utility.GetOrientation(, Prompt)
    Object.Rotate FromPnt, Angle
End If
End Sub

```

Abschließend noch eine kurze Anmerkung zur *Copy*-Funktion. Sie sollten bei der *Copy*-Funktion die Methoden *Move* und *Rotate* kombinieren, damit die duplizierten Elemente nicht übereinander liegen bleiben. Das Ganze könnte folgende Form haben:

```

Public Sub Copy_Move()
Dim OutEnt As AcadCircle
Dim Center(2) As Double
Dim Radius As Double
Dim copyEnt As AcadCircle
Dim FromPnt(2) As Double
Dim ToPnt(2) As Double
Center(0) = 0#: Center(1) = 0#: Center(2) = 0#
Radius = 1
Set OutEnt = ThisDrawing.ModelSpace.AddCircle(Center, Radius)
Set copyEnt = OutEnt.Copy()
FromPnt(0) = 0: FromPnt(1) = 0: FromPnt(2) = 0
ToPnt(0) = 2: ToPnt(1) = 0: ToPnt(2) = 0
copyEnt.Move FromPnt, ToPnt
End Sub

```

Objekte löschen

Mit der Funktion *Delete* löschen Sie das Objekt, auf das Sie diese Funktion anwenden. Wenn Sie diese Funktion ein zweites Mal mit demselben Objekt ausführen, wird dieses Objekt wiederhergestellt.

Objekt löschen:

```
Objekt.Delete
```

Objekt löschen und wiederherstellen:

```
Objekt.Delete
Objekt.Delete
```

Die Funktion GetBoundingBox

Diese Funktion ermittelt ein Rechteck zur Umgrenzung des jeweiligen Objekts. Als Ergebnis werden die linke untere sowie die rechte obere Ecke dieses Rechtecks zurückgegeben. Das Rechteck wird dabei immer in Bezug auf das WKS berechnet. Sie könnten mit dieser Funktion z.B. eine Fläche erzeugen, die das gesamte Objekt überdeckt. Die Koordinaten der Begrenzungspunkte werden als Array (X, Y, Z) zurückgegeben.

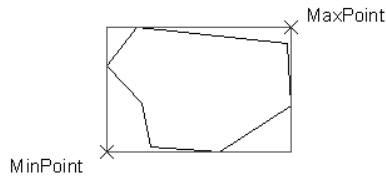


Abbildung 8.9: Objektmengengrenzung

```
Public Sub BoundingBox()
Dim Object As Object
Dim Prompt As String
Dim FromPnt(2) As Double
Dim ToPnt(2) As Double
Dim MinPoint As Variant
Dim MaxPoint As Variant
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) = "IAcadBlockReference" Then
    Object.GetBoundingBox MinPoint, MaxPoint
    FromPnt(0) = MinPoint(0)
    FromPnt(1) = MinPoint(1)
    ToPnt(0) = MaxPoint(0)
    ToPnt(1) = MaxPoint(1)
    ThisDrawing.ModelSpace.AddLine FromPnt, ToPnt
End If
End Sub
```

Objekte hervorheben (Highlight)

Mit dieser Funktion heben Sie das entsprechende Objekt von den anderen hervor, so dass der Benutzer dies optisch erkennen kann. Die Darstellung des Objekts wird nur durch eine Regenerierung oder durch die Aufhebung der Hervorhebung wieder normalisiert. Sie können dieser Prozedur folgende Parameter mitgeben:

- **TRUE** – das entsprechende Objekt wird hervorgehoben.
- **FALSE** – die Hervorhebung des Objekts wird wieder aufgehoben.

```
Public Sub HighlightObject()
Dim Object As Object
Dim Prompt As String
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
```



```

ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Object.Highlight True
End If
End Sub

```

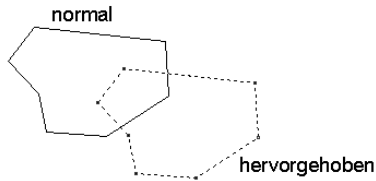


Abbildung 8.10: Objekthervorhebung

Schnittpunkte zwischen zwei Objekten ermitteln (IntersectWith)

Mit dieser Funktion ermitteln Sie den Schnittpunkt zwischen zwei Objekten. Sie können dabei festlegen, ob sich die Linien direkt überschneiden oder die beiden Objekte einen Schnittpunkt haben. Falls diese Funktion Schnittpunkte berechnet, werden diese als Array (X, Y, Z) zurückgegeben. Bei Polylinien kann diese Funktion auch mehrere Schnittpunkte berechnen. Es werden in jedem Fall alle Schnittpunkte zurückgegeben und zwar in der Reihenfolge, in der sie berechnet wurden.

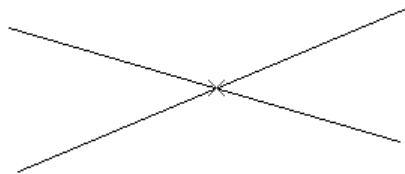
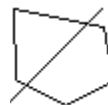


Abbildung 8.11: Schnittpunkt zwischen zwei Linien

Die Funktion kann folgende Ergebnisse zurückgeben:

VarPoint VarPoint(0) VarPoint(1) VarPoint(0)	Es wurde ein Schnittpunkt ermittelt und die X-, Y-, Z-Koordinaten wurden der Variablen <i>VarPoint</i> übergeben.
VarPoint VarPoint(0) VarPoint(1) VarPoint(2) VarPoint(3) VarPoint(4) VarPoint(5)	Es konnten mehrere Schnittpunkte ermittelt werden und die X-, Y-, Z-Koordinaten wurden der Variablen <i>VarPoint</i> übergeben.



Über die entsprechenden Parameter können Sie mit der Funktion folgende Wirkung erzielen:

- **IntersectObject** – legt das Objekt fest, das mit dem ausführenden Objekt geschnitten werden soll.
- **ExtendOption** – legt fest, wie die einzelnen Objekte beim Schnitt ausgewertet werden sollen. Sie können hierfür folgende Konstanten verwenden:
 - **acExtendNone** – beide Objekte werden nur auf ihre ursprüngliche Ausdehnung begrenzt.
 - **acExtendThisEntity** – ermittelt die Schnittpunkte so, als wenn das Basisobjekt maximal expandiert wurde.
 - **acExtendOtherEntity** – ermittelt die Schnittpunkte so, als wenn das Parameterobjekt maximal expandiert wurde.
 - **acExtendBoth** – ermittelt die Schnittpunkte so, als wenn beide Objekte maximal expandiert wurden.

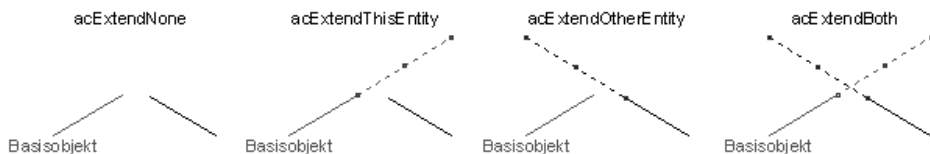


Abbildung 8.12: Schnittoptionen

```
Public Sub IntesectObject()
Dim Object1 As Object
Dim Object2 As Object
Dim Prompt As String
Dim VarPoint As Variant
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object1, PickedPoint, Prompt
If TypeName(Object1) <> "Nothing" Then
  Prompt = "Wählen Sie ein das zu schneidende Objekt:"
  ThisDrawing.Utility.GetEntity Object2, PickedPoint, Prompt
  VarPoint = Object1.IntersectWith(Object2, acExtendBoth)
End If
End Sub
```

Objekte zweidimensional spiegeln

Die Funktion *Mirror* erstellt aus dem ausführenden Objekt ein Spiegelbild und gibt dieses Objekt als Variable zurück. Dabei bleibt das ausführende Objekt erhalten. Um das Basisobjekt zu löschen, wenden Sie die Funktion *Delete* auf das ausführende Objekt an. Wenn Sie diese Funktion auf einen Text anwenden, müssen Sie auf den Wert der Systemvariablen *MIRRTXT* achten. Wenn der Wert der Systemvariablen *MIRRTXT* auf 0 gesetzt wurde, wird der Text nicht gespiegelt. Sie müssen als Parameter lediglich die Spiegelachse mitgeben, die Sie über verschiedene *Utility*-Funktionen vom Benutzer abfragen können.

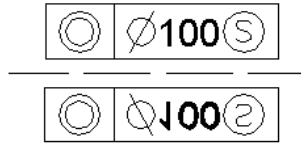


Abbildung 8.13: Gespiegeltes Toleranzobjekt

```

Public Sub IntesectObject()
Dim Object As Object
Dim Prompt As String
Dim OutObj As Object
Dim VarPoint As Variant
Dim Point1(2) As Double
Dim Point2(2) As Double
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Prompt = "Wählen Sie den Startpunkt der SpiegelAchse:"
    VarPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Point1(0)=VarPoint(0)
    Point1(1)=VarPoint(1)
    Point1(2)=VarPoint(2)
    Prompt = "Wählen Sie den Endpunkt der SpiegelAchse:"
    VarPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Point2(0)=VarPoint(0)
    Point2(1)=VarPoint(1)
    Point2(2)=VarPoint(2)
    Set OutObj = Object.Mirror(Point1, Point2)
End If
End Sub

```

Objekte räumlich spiegeln

Die Funktion *Mirror3D* erstellt, ähnlich wie die Funktion *Mirror*, aus dem ausführenden Objekt ein Spiegelbild und gibt das neue Objekt als Variable zurück. Sie verwenden bei dieser Funktion allerdings eine zweite Achse, um das Objekt dreidimensional zu spiegeln. Auch hier bleibt das ausführende Objekt erhalten. Als Parameter geben Sie lediglich die Spiegelachse 1-2 und 2-3 an, die Sie über verschiedene *Utility*-Funktionen vom Benutzer abfragen können.

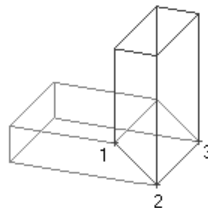


Abbildung 8.14: Räumliche Spiegelung

```

Public Sub MirrorObject3D()
Dim Object As Object
Dim Prompt As String
Dim OutObj As Object
Dim VarPoint As Variant
Dim Point1(2) As Double
Dim Point2(2) As Double
Dim Point2(2) As Double
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Prompt = "Wählen Sie den Startpunkt der Spiegelachsen:"
    VarPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Point2(0) = VarPoint(0)
    Point2(1) = VarPoint(1)
    Point2(2) = VarPoint(2)
    Prompt = "Wählen Sie den Endpunkt der ersten Spiegelachse:"
    VarPoint = ThisDrawing.Utility.GetPoint(Point2, Prompt)
    Point1(0) = VarPoint(0)
    Point1(1) = VarPoint(1)
    Point1(2) = VarPoint(2)
    Prompt = "Wählen Sie den Endpunkt der zweiten Spiegelachse:"
    VarPoint = ThisDrawing.Utility.GetPoint(Point2, Prompt)
    Point3(0) = VarPoint(0)
    Point3(1) = VarPoint(1)
    Point3(2) = VarPoint(2)
    Set OutObj = Object.Mirror3D(Point1, Point2, Point3)
End If
End Sub

```

Objekte räumlich drehen

Mithilfe der Anweisung *Rotate3D* können Sie unter Angabe eines Drehwinkels ein Objekt um eine von Ihnen festgelegte Achse rotieren lassen. Als Parameter geben Sie die Punkte der entsprechenden Achse als Array sowie den Drehwinkel als Zahl an. Nutzen Sie zur Bestimmung der einzelnen Parameter die Funktionen des *Utility*-Objekts.

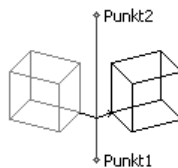


Abbildung 8.15: Räumliche Drehung

```

Public Sub RotateObject3D()
Dim Object As Object
Dim Prompt As String
Dim VarPoint As Variant
Dim Point1(2) As Double

```

```

Dim Point2(2) As Double
Dim Angle As Double
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Prompt = "Wählen Sie den Startpunkt der Drehachse:"
    VarPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Point1(0) = VarPoint(0)
    Point1(1) = VarPoint(1)
    Point1(2) = VarPoint(2)
    Prompt = "Wählen Sie den Endpunkt der Drehachse:"
    VarPoint = ThisDrawing.Utility.GetPoint(Point1, Prompt)
    Point2(0) = VarPoint(0)
    Point2(1) = VarPoint(1)
    Point2(2) = VarPoint(2)
    Prompt = "Legen Sie den Drehwinkel fest:"
    Angle = ThisDrawing.Utility.GetOrientation(, Prompt)
    Object.Rotate3D Point1, Point2, Angle
End If
End Sub

```

Objekte skalieren

Die *ScaleEntity*-Funktion entspricht dem AutoCAD-Befehl *_Scale*. Allerdings können Sie an das Objekt lediglich den Basispunkt sowie den Skalierungsfaktor übergeben. Der Skalierungsfaktor wird dabei nicht absolut, sondern relativ an das Objekt angebracht. Das bedeutet, wenn Sie einen Faktor größer als 1 verwenden, wird das Objekt um diesen Faktor vergrößert. Ist der Faktor kleiner als 1, wird um diesen Faktor verkleinert. Dabei gilt: Der Skalierfaktor muss immer größer 0 sein. Wenn Sie eine Skalierung in Bezug auf eine bestimmte Strecke erreichen wollen, können Sie hierfür die *Utility*-Funktion *GetDistance* verwenden.

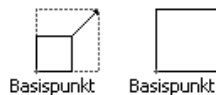


Abbildung 8.16: Skalierung

```

Public Sub ScaleObject()
Dim Object As Object
Dim Prompt As String
Dim VarPoint As Variant
Dim Point1(2) As Double
Dim ScaleF As Double
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Prompt = "Wählen Sie den Basispunkt:"
    VarPoint = ThisDrawing.Utility.GetPoint(, Prompt)

```

```

    Point1(0) = VarPoint(0)
    Point1(1) = VarPoint(1)
    Point1(2) = VarPoint(2)
    Prompt = "Geben Sie den Skalierungsfaktor ein:"
    ScaleF = ThisDrawing.Utility.GetReal(Prompt)
    Object.ScaleEntity Point1, ScaleF
End If
End Sub
Public Sub ScaleObject2()
Dim Object As Object
Dim Prompt As String
Dim VarPoint As Variant
Dim Point1(2) As Double
Dim ScaleF As Double
Dim Bezug As Double
Dim LNeu As Double
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then
    Prompt = "Wählen Sie den Basispunkt:"
    VarPoint = ThisDrawing.Utility.GetPoint(, Prompt)
    Point1(0) = VarPoint(0)
    Point1(1) = VarPoint(1)
    Point1(2) = VarPoint(2)
    Prompt = "Geben Sie die Bezugslänge ein:"
    Bezug = ThisDrawing.Utility.GetDistance(, Prompt)
    Prompt = "Geben Sie die neue Länge ein:"
    LNeu = ThisDrawing.Utility.GetDistance(, Prompt)
    ScaleF = LNeu / Bezug
    Object.ScaleEntity Point1, ScaleF
End If
End Sub

```

Die Funktion TransformBy

Diese Funktion ist eine Kombination der Funktionen *Move*, *Rotate* und *ScaleEntity*. Sie können mithilfe dieser Funktion das ausführende Objekt anhand einer Matrix (Array 4 x 4) verschieben, drehen und/oder skalieren. Die Matrix hat dabei folgendes Format:

```
Dim transMat(3, 3) As Double
```

R00	R01	R02	T0	entspricht	TrMat(0,0)	TrMat(0,1)	TrMat(0,2)	TrMat(0,3)
R10	R11	R12	T1	entspricht	TrMat(1,0)	TrMat(1,1)	TrMat(1,2)	TrMat(1,3)
R20	R21	R22	T2	entspricht	TrMat(2,0)	TrMat(2,1)	TrMat(2,2)	TrMat(2,3)
0	0	0	1	entspricht	TrMat(3,0)	TrMat(3,1)	TrMat(3,2)	TrMat(3,3)

Die Zellen R00 bis R22 enthalten die Werte für die Skalierung und Drehung. Die Zellen T0 bis T2 enthalten die Verschiebung in X, Y und Z. Sollte die Transformationsmatrix nicht korrekt sein, wird ein Fehler ausgelöst.

Im folgenden Beispiel wird das gewählte Objekt um den Faktor 10 skaliert und in X und Y um 50 verschoben.

```
Public Sub TranslateObject()
Dim Object As Object
Dim Prompt As String
Dim TransMat(3, 3) As Double
On Local Error Resume Next
Prompt = "Wählen Sie ein Objekt:"
ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) <> "Nothing" Then

    TransMat(0, 0) = 10#
    TransMat(0, 1) = 0#
    TransMat(0, 2) = 0#
    TransMat(0, 3) = 50#

    TransMat(1, 0) = 0#
    TransMat(1, 1) = 10#
    TransMat(1, 2) = 0#
    TransMat(1, 3) = 50#

    TransMat(2, 0) = 0#
    TransMat(2, 1) = 0#
    TransMat(2, 2) = 10#
    TransMat(2, 3) = 0#

    TransMat(3, 0) = 0#
    TransMat(3, 1) = 0#
    TransMat(3, 2) = 0#
    TransMat(3, 3) = 1#

    Object.TransformBy TransMat
End If
End Sub
```

Abschließend noch einige Beispiel für die Transformationsmatrix.

Drehung des Objekts 90°			
0.000000	-1.000000	0.000000	0.000000
1.000000	0.000000	0.000000	0.000000
0.000000	0.000000	1.000000	0.000000
0.000000	0.000000	0.000000	1.000000
Verschiebung um 10,10,0			
1.000000	0.000000	0.000000	10.000000
0.000000	1.000000	0.000000	10.000000
0.000000	0.000000	1.000000	0.000000
0.000000	0.000000	0.000000	1.000000

Skalierung um Faktor 10			
10.000000	0.000000	0.000000	0.000000
0.000000	10.000000	0.000000	0.000000
0.000000	0.000000	10.000000	0.000000
0.000000	0.000000	0.000000	1.000000

8.3.2 Grundlegende Objekte

Bevor wir uns den einzelnen grafischen Elementen zuwenden, gilt es noch zwei besondere Objekte zu erklären. Diese Objekte bilden die Grundlage für andere AutoCAD-Objekte, so dass diese auch als Träger der anderen Klassen dienen können. Die Ausnahme bildet der Datentyp *Object*. Bei diesen Klassen handelt es sich um *AcadObject* und *AcadEntity*. Der Unterschied zwischen beiden Klassen besteht darin, dass *AcadEntity* nur grafische Objekte abbilden kann. Um zu verstehen, wie die einzelnen Objektklassen zusammenhängen, müssen Sie sich nur eines vor Augen halten. Alle Zeichnungsobjekte enthalten die Eigenschaften und Methoden der Klasse *AcadObject*, so auch *AcadEntity*, basieren also auf dieser Klasse. *AcadEntity* hingegen umfasst zusätzliche Eigenschaften und Methoden, die nur in grafischen Objekten vorkommen. Einen Überblick über die Zusammenhänge zwischen den einzelnen Objekten finden Sie im Zeichnungsdokument „struktur.dwg“.

Die Klasse *AcadObject*

Das erste Objekt bildet die Grundlage aller AutoCAD-Objekte, die in einer Zeichnung eingefügt werden können bzw. innerhalb einer Zeichnung als Container anderer Objekte dienen. Diese Klasse kann somit auch zur Darstellung eines grafischen AutoCAD-Objekts verwendet werden.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 8.10: Eigenschaften der Klasse *AcadObject*

Eigenschaft	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

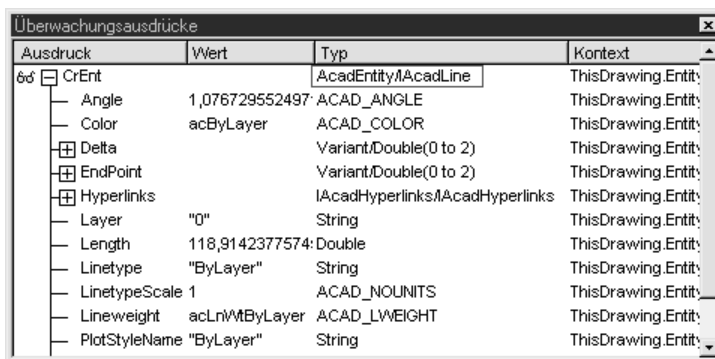
Tabelle 8.11: Methoden der Klasse *AcadObject*

Die Klasse *AcadEntity*

Diese Klasse wird häufig als Rückgabeparameter für eine Funktion verwendet. Sie stellt dabei ein allgemeines Zeichnungsobjekt in AutoCAD als ein sichtbares Zeichnungselement wie Linie, Text, Polylinie usw. dar. Das bedeutet, dass Sie an diese Objektklasse sämtliche grafischen AutoCAD-Elemente übergeben können. Da z.B. in einem Auswahlset (*SelectionSet*) Objekte verschiedenster Klasse enthalten sein können, gibt die *Item*-Anweisung eine Objektvariable vom Typ *AcadEntity* zurück. Das bedeutet aber nicht, dass Sie auf die verschiedensten Eigenschaften und Methoden eines Objekts, wie z.B. eine Polylinie, nicht mehr zugreifen können. Aber schauen wir uns das Ganze im Einzelnen an. Dazu erzeugen Sie bitte eine neue leere Zeichnung. Fügen Sie diesem Dokument einen Kreis hinzu. Anschließend wechseln Sie in die VBA-Umgebung und fügen folgenden kurzen Programmcode einem neuen VBA-Projekt hinzu.

```
Public Sub EntityDemo()  
Dim CrEnt As AcadEntity  
Set CrEnt = ModelSpace.Item(0)  
End Sub
```

Setzen Sie nun einen Haltepunkt auf die Zeile *End Sub* [F9]. Wenn Sie diese kleine Prozedur jetzt starten und das Programm in der vorgesehenen Zeile unterbricht, fügen Sie bitte die Objektvariable *CrEnt*, die ja als *AcadEntity* deklariert wurde, dem Überwachungsfenster hinzu.

Abbildung 8.17: Darstellung einer Linie mit der übergeordneten Klasse *AcadEntity*

Wie Sie im Überwachungsfenster erkennen können, werden alle Eigenschaften eines Kreises angezeigt, obwohl die Objektvariable als Klasse *AcadEntity* deklariert wurde. Außerdem wird in der Spalte *Typ* des Fensters für die Variablen *CrEnt* der deklarierte sowie der untergeord-

nete (tatsächliche) Datentyp angezeigt. Da es sich bei der Klasse *AcadEntity* um die Default-Schnittstelle für AutoCAD-Elemente handelt, enthält diese Klasse nur die Eigenschaften und Methoden, die alle anderen AutoCAD-Elemente auch besitzen. Im Folgenden erhalten Sie eine Übersicht über die Eigenschaften und Methoden.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der Elemente fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Objekts zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Objekts an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Objekts an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Objekts an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für das Objekt an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Objekt zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Objekts steuern.

Tabelle 8.12: Eigenschaften der Klasse *AcadEntity*

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.

Tabelle 8.13: Methoden der Klasse *AcadEntity*

Prozedur	Beschreibung
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Highlight</i>	Hebt das Objekt hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Objekts mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt ein Objekt als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt ein neues Objekt als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt das Objekt von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht das Objekt um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht das Objekt um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Objekt um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Objekt an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Bemaßungsobjekts im AutoCAD-Zeichnungsfenster.

Tabelle 8.13: Methoden der Klasse *AcadEntity* (Forts.)

Nachdem Sie einen Überblick über die Eigenschaften und Methoden der Klasse *AcadEntity* gewonnen haben, sollten Sie sich noch kurz die Auflistungen ansehen, die diese Klasse verwenden. Alle diese Auflistungen können demnach sämtliche AutoCAD-Elemente aufnehmen bzw. beinhalten.

Klasse / Auflistung	Methode
<i>AcadBlock</i>	<i>Item</i>
<i>AcadGroup</i>	<i>Item/AppendItems/RemoveItems</i>
<i>AcadModelSpace</i>	<i>Item</i>
<i>AcadPaperSpace</i>	<i>Item</i>
<i>AcadSelectionSet</i>	<i>Item/AddItems/RemoveItems</i>

8.3.3 3D-Flächenobjekt (*Acad3DFace*)

Die Klasse *Acad3DFace* stellt ein einzelnes 3D-Flächenobjekt dar. Sie können die Eigenschaften dieses Objekts über den VBA-Code anpassen, müssen dabei aber auf die Einstellungen der Systemvariablen *SPLFRAME* achten. Sollte diese Variable auf 1 gestellt sein, werden alle Kanten sichtbar dargestellt, unabhängig von den *Visible*-Eigenschaften der einzelnen Kanten.

Sie können einen Verweis auf eine 3D-Fläche erstellen, indem Sie die Funktion *Add3DFace* (Kapitel 6.4.2) im Modellbereich, Papierbereich oder auf einen Block anwenden oder ein vorhandenes Element über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften zur Verwaltung der Stützpunkte

Die Stützpunkte eines 3D-Flächenobjekts werden in der *Coordinates*-Eigenschaft verwaltet. Diese Eigenschaft entspricht einem Array von Koordinaten, wobei jedes Feld eine einzelne Koordinate (X, Y oder Z) enthält. Da eine 3D-Fläche immer vier Stützpunkte besitzt, ist die Obergrenze dieses Arrays fest und lässt sich ganz einfach berechnen. Bei vier vorhandenen Stützpunkten, von denen jeder drei Koordinaten aufnimmt, ergibt sich eine Array-Größe von zwölf Feldern, wobei das erste Feld mit dem Index 0 und das letzte mit dem Index 11 angesprochen wird. Wenn Sie die Koordinaten der Stützpunkte anpassen möchten, müssen Sie die Daten der *Coordinates*-Eigenschaft komplett an eine Variable übergeben.

```
PointsList = ActFace.Coordinates
```

Jetzt können Sie die entsprechenden Felder des Arrays anpassen. Wenn Sie beispielsweise die Höhe (Z) der Stützpunkte auf 0 setzen wollen, dann müssen Sie jedes dritte Feld ansprechen:

```
PointsList(2) = 0  
PointsList(5) = 0  
PointsList(8) = 0  
PointsList(11) = 0
```

und dann das Array in die Eigenschaft *Coordinates* zurückschreiben:

```
ActFace.Coordinates = PointsList
```

Mit dieser Methode können Sie die gesamte Stützpunktliste auf einmal anpassen. Dieses Objekt besitzt aber noch eine zweite Eigenschaft, mit der Sie die Koordinaten einzelner Stützpunkte abrufen können. *Coordinate* gibt den jeweiligen Stützpunkt, den Sie im Index der Eigenschaft angeben, zurück oder legt diesen fest. Sie haben nun zwei Möglichkeiten, die einzelnen Koordinatenwerte zu ändern. Die erste besteht darin, alle Koordinaten des Stützpunkts abzurufen, indem Sie die Koordinaten an eine Variable übergeben, diese ändern und dann wieder zurückschreiben. Dazu müssen Sie nur den Index des entsprechenden Stützpunkts mit der *Coordinate*-Eigenschaft abrufen.

```
PointsList = ActFace.Coordinate(0) 'ersten Stützpunkt abrufen
```

Das so entstandene Array enthält die drei Koordinaten (X, Y, Z) des jeweiligen Stützpunkts und kann nun angepasst werden.

```
PointsList(2) = 0
```

Nach der Änderung kann das Array wieder zurückgeschrieben werden.

```
ActFace.Coordinate(0) = PointsList
```

Die zweite Möglichkeit, Stützpunkte mit der *Coordinate*-Eigenschaft anzupassen, ist die direkte Eingabe der Koordinaten. Die Z-Koordinate des zweiten Stützpunkts ließe sich mit folgender Zeile abrufen:

```
Point2_Z = Object.Coordinate(1)(2)
```

Siehe auch *Kapitel 8.1.1*.

Sichtbarkeit der Kanten steuern

3D-Flächen bieten die Möglichkeit, die Sichtbarkeit der einzelnen Kanten über Eigenschaften zu steuern. Sonderbarerweise hat man für jede Kante eine eigene Sichtbarkeitseigenschaft erstellt, anstatt mit einer Eigenschaft, die mit einem Index als Parameter versehen werden kann, die Sichtbarkeit der einzelnen Kanten einzustellen. Aber sei es, wie es sei – mit den Eigenschaften *VisibleEdge1*, *VisibleEdge2*, *VisibleEdge3* und *VisibleEdge4* kann die jeweilige Kante ein- bzw. ausgeschaltet werden. Dazu müssen Sie nur den Wert der Eigenschaft anpassen. Um die Kante sichtbar zu machen, setzen Sie deren Wert auf *TRUE*.

```
ActFace.VisibleEdge1 = True
```

Der Wert *FALSE* macht die Kante hingegen unsichtbar.

```
ActFace.VisibleEdge1 = True
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der 3D-Fläche fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der 3D-Fläche zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der 3D-Fläche ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der 3D-Fläche zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der 3D-Fläche an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der 3D-Fläche an oder legt diesen fest
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der 3D-Fläche an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die 3D-Fläche an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die ObjectID der 3D-Fläche zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>Acad3DFace</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die 3D-Fläche zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der 3D-Fläche steuern.

Tabelle 8.14: Weitere Eigenschaften einer 3D-Fläche

Methoden des 3D-Flächen-Objekts

Dieses Objekt enthält eigentlich nur zwei interessante Prozeduren, außer denen, die sowieso in jedem Objekt vorhanden sind. Zusätzlich zu der Eigenschaft *VisibleEdge* können Sie die Sichtbarkeit der einzelnen Kanten mit den Anweisungen *GetInvisibleEdge* und *SetInvisibleEdge* steuern. Diese Anweisungen eignen sich oftmals besser zum Ein- oder Ausschalten der Kanten, da Sie hier den Index der entsprechenden Kante und den Modus angeben können. Mit *GetInvisibleEdge* ermitteln Sie, ob die entsprechende Kante, die Sie im Parameter *Index* angeben, sichtbar oder unsichtbar ist.

```
IsVisible = ActFace.GetInvisibleEdge(0)
```

SetInvisibleEdge hingegen legt die Sichtbarkeit der jeweiligen Kante fest.

```
ActFace.SetInvisibleEdge 0, True
```

Der erste Parameter der Prozedur bestimmt die entsprechende Kante (0–3) und der zweite die Sichtbarkeit dieser Kante. Um die Kante unsichtbar zu machen, müssen Sie hier den Wert *FALSE* angeben und *TRUE*, um die Kante wieder sichtbar zu machen.

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der 3D-Fläche ab.
<i>GetXData</i>	Ruft die erweiterten Daten der 3D-Fläche ab.
<i>Highlight</i>	Hebt die 3D-Fläche hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der 3D-Fläche mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine 3D-Fläche als Resultat einer Spiegelung mit dieser 3D-Fläche.
<i>Mirror3D</i>	Erstellt eine 3D-Fläche als Resultat einer räumlichen Spiegelung dieser 3D-Fläche.
<i>Move</i>	Verschiebt die 3D-Fläche von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die 3D-Fläche um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die 3D-Fläche um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die 3D-Fläche um einen bestimmten Faktor.

Tabelle 8.15: Weitere Prozeduren einer 3D-Fläche

Prozedur	Beschreibung
<i>SetXData</i>	Speichert die erweiterten Daten der 3D-Fläche.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die 3D-Fläche an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der 3D-Fläche im AutoCAD-Zeichnungsfenster.

Tabelle 8.15: Weitere Prozeduren einer 3D-Fläche (Forts.)

Anwendungsbeispiele

(Beispielprojekt 08\Acad3DFace.dvb)

Im Folgenden sehen Sie einige Beispiele zur Steuerung der Sichtbarkeit der einzelnen Kanten der 3D-Flächen und zur Anpassung der einzelnen Eckpunkte.

Das erste Beispiel steuert die Sichtbarkeit der Kanten über die Eigenschaft *VisibilityEdge1*:

```
Public Sub Create3DFace_V1()
Dim CrFace As Acad3DFace
Dim Pnt1(2) As Double
Dim Pnt2(2) As Double
Dim Pnt3(2) As Double
Dim Pnt4(2) As Double
On Local Error Resume Next
Pnt1(0) = 0: Pnt1(1) = 0: Pnt1(2) = 0
Pnt2(0) = 0: Pnt2(1) = 40: Pnt2(2) = 0
Pnt3(0) = 40: Pnt3(1) = 40: Pnt3(2) = 0
Pnt4(0) = 40: Pnt4(1) = 0: Pnt4(2) = 0
Set CrFace = ModelSpace.Add3DFace(Pnt1, Pnt2, Pnt3, Pnt4)
If TypeName(CrFace) <> "Nothing" Then
    CrFace.VisibilityEdge1 = True
    CrFace.VisibilityEdge2 = False
    CrFace.VisibilityEdge3 = True
    CrFace.VisibilityEdge4 = False
End If
End Sub
```

Das zweite Beispiel steuert ebenfalls die Sichtbarkeit der Kanten. Hierfür wird allerdings die Funktion *SetInvisibleEdge* eingesetzt:

```
Public Sub Create3DFace_V2()
Dim CrFace As Acad3DFace
Dim Pnt1(2) As Double
Dim Pnt2(2) As Double
Dim Pnt3(2) As Double
Dim Pnt4(2) As Double
On Local Error Resume Next
Pnt1(0) = 0: Pnt1(1) = 0: Pnt1(2) = 0
Pnt2(0) = 0: Pnt2(1) = 40: Pnt2(2) = 0
Pnt3(0) = 40: Pnt3(1) = 40: Pnt3(2) = 0
Pnt4(0) = 40: Pnt4(1) = 0: Pnt4(2) = 0
Set CrFace = ModelSpace.Add3DFace(Pnt1, Pnt2, Pnt3, Pnt4)
If TypeName(CrFace) <> "Nothing" Then
    For i = 0 To 3
```

```

        CrFace.SetInvisibleEdge i, True
    Next
End If
End Sub

```

Im dritten Beispiel werden die Koordinaten der 3D-Fläche einzeln ausgelesen und um einen bestimmten Faktor verändert, wobei die Höhe von dieser Änderung ausgeschlossen werden soll.

```

Public Sub Change3DFaceCoords()
Dim Object As Object
Dim Prompt As String
Dim VarPoint As Variant
On Local Error Resume Next
Prompt = "Wählen Sie eine 3D-Fläche:"
Utility.GetEntity Object, PickedPoint, Prompt
If TypeName(Object) = "IAcad3DFace" Then
    VarPoint = Object.Coordinates
    ReDim NewPoints(UBound(VarPoint)) As Double
    For i = 0 To UBound(VarPoint) Step 3
        NewPoints(i) = VarPoint(i) * 2
        NewPoints(i + 1) = VarPoint(i + 1) * 2
        NewPoints(i + 2) = VarPoint(i + 2)
    Next
    Object.Coordinates = NewPoints
End If
End Sub

```

8.3.4 3D-Polylinien (Acad3DPolyline)

Eine Objektvariable, die mit dieser Klasse deklariert wurde, stellt jeweils eine einzelne 3D-Polylinie dar. Sie können einen Verweis auf eine 3D-Polylinie herstellen, indem Sie die Funktion *Add3Dpoly* im Modellbereich, Papierbereich oder auf einen Block anwenden oder ein vorhandenes Element über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften zur Verwaltung der Stützpunkte

Genau wie bei 3D-Flächen können Sie auch bei 3D-Polylinien die Stützpunkte über die Eigenschaften *Coordinates* und *Coordinate* abrufen bzw. anpassen. Der Unterschied besteht darin, dass eine Polylinie eine praktisch unbegrenzte Anzahl an Stützpunkten besitzen kann. Deshalb ist es wichtig zunächst die Anzahl der Stützpunkte mithilfe der *Coordinates*-Eigenschaft zu ermitteln. Dazu müssen Sie nur den Inhalt dieser Eigenschaft an eine Variable übergeben und anschließend über die *UBound()*-Funktion die Obergrenze des Arrays ermitteln.

```

Dim PointsList
PointsList = Act3DPoly.Coordinates
CoordCount = UBound(PointsList) + 1

```

Beachten Sie dabei, dass das erste Feld des Arrays mit dem Index 0 abgerufen wird und somit die ermittelte Obergrenze des Arrays nicht der Anzahl der Felder entspricht, sondern um 1 erhöht werden muss. Aus der so ermittelten Gesamtanzahl der Koordinaten kann nun die

Anzahl der Stützpunkte ermittelt werden. Da jeder Stützpunkt drei Koordinaten enthält (X, Y, Z), müssen Sie nur die Anzahl der Koordinaten durch 3 dividieren, um zu berechnen, wie viele Stützpunkte die Polylinie besitzt.

```
Dim PointsList
PointsList = Act3DPoly.Coordinates
PtCount = (UBound(PointsList) + 1) / 3
```

Obwohl das Polylinien-Objekt eine Funktion zum Hinzufügen von Stützpunkten besitzt, eignet sich die *Coordinates*-Eigenschaft oftmals besser zur Steuerung der Stützpunktanzahl. Um beispielsweise einen weiteren Stützpunkt hinzuzufügen, müssen Sie nur die Daten der *Coordinates*-Eigenschaft an eine Variable übergeben, diese Variable neu dimensionieren

```
Dim PointsList
PointsList = Act3DPoly.Coordinates
ReDim Preserve PointsList(UBound(VarPoint) + 3) As Double
```

und nach Anpassung der Stützpunktkoordinaten das Array in die *Coordinates*-Eigenschaft zurückschreiben.

```
Act3DPoly.Coordinates = PointsList
```

Sie müssen hierbei nur darauf achten, dass Sie eine korrekte Anzahl an Feldern für das Array festlegen, also Anzahl der Stützpunkte mal drei Koordinaten minus ein Feld (das Array beginnt immer mit Index 0).

Die Verarbeitung der Stützpunkte mit der *Coordinate*-Eigenschaft funktioniert genau wie bei der 3D-Fläche. Entweder Sie arbeiten mit dem Stützpunkt-Array,

```
Dim StPoint
StPoint = Act3DPoly.Coordinate(N)
```

wobei der Parameter N die Nummer des Stützpunkts bezeichnet, oder Sie werten die einzelnen Koordinaten direkt aus, indem Sie zusätzlich zur Position des Stützpunkts die Nummer der entsprechenden Koordinate (0–2) angeben.

```
X = Act3DPoly.Coordinate(N)(0)
Y = Act3DPoly.Coordinate(N)(1)
Z = Act3DPoly.Coordinate(N)(2)
```

Siehe auch *Kapitel 8.1.1*.

Weitere Eigenschaften der 3D-Polylinie

Mit der Eigenschaft *Closed* können Sie festlegen, ob die Polylinie geschlossen dargestellt werden soll (*TRUE*) oder nicht (*FALSE*). Wenn Sie die Polylinie schließen, dann wird eine zusätzliche Verbindung vom ersten zum letzten Stützpunkt dargestellt.

```
Act3Dpoly.Closed = True
```

Mit der *Type*-Eigenschaft können Sie die Darstellungsart der Polylinie festlegen. Eine 3D-Polylinie lässt sich auf drei verschiedene Arten darstellen. Die erste ist die linienhafte Darstellung, bei der die Verbindung zwischen den einzelnen Stützpunkten durch eine direkte Verbindung angezeigt wird.

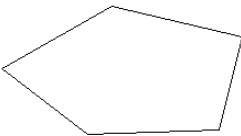


Abbildung 8.18: Linienhafte Darstellung der Polylinie

Mit den anderen beiden Darstellungen gleichen Sie die Verbindung zwischen den Stützpunkten mit einer Kurvenanpassung an, die mit einem B-Spline vergleichbar ist. AutoCAD unterscheidet dabei zwischen einer quadratischen und einer kubischen Polylinie.

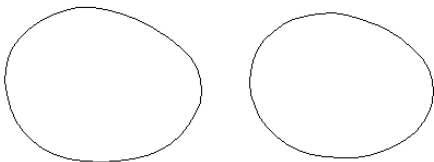


Abbildung 8.19: Quadratische und kubische Darstellung der Polylinie

Welche der drei Darstellungsarten Sie verwenden, legen Sie mit einer der Konstanten fest, die in Tabelle 8.16 aufgeführt sind. Diese Konstanten brauchen Sie nur der *Type*-Eigenschaft der Polylinie zuweisen und die Polyliniendarstellung wird durch AutoCAD angeglichen.

```
Act3DPoly.Type = acSimple3DPoly
```

Konstante	Darstellung	Wert
acSimple3DPoly	einfache Darstellung	0
acQuadSpline3DPoly	quadratische Darstellung	1
acCubicSpline3DPoly	kubische Darstellung	2

Tabelle 8.16: Type-Konstanten

Tipp Einige AutoCAD-Befehle, wie z.B. Stutzen oder Brechen, entfernen die Kurvenanpassung der Polylinie, weshalb die Verwendung eines Spline in diesem Fall besser geeignet wäre.

Eigenschaften	Beschreibung
Color	Legt die Farbe der 3D-Polylinie fest oder gibt diese zurück.
Document (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
Handle (schreibgeschützt)	Diese Eigenschaft gibt den Handle der 3D-Polylinie zurück.
HasExtensionDictionary (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der 3D-Polylinie ein Erweiterungs-wörterbuch zugeordnet wurde.

Tabelle 8.17: Weitere Eigenschaften einer 3D-Polylinie

Eigenschaften	Beschreibung
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der 3D-Polylinie zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der 3D-Polylinie an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der 3D-Polylinie an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der 3D-Polylinie an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die 3D-Polylinie an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die ObjectID der 3D-Polylinie zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>Acad3DPoly</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die 3D-Polylinie zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der 3D-Polylinie steuern.

Tabelle 8.17: Weitere Eigenschaften einer 3D-Polylinie (Forts.)

Methoden des 3D-Polylinien-Objekts

Neben der Möglichkeit die Stützpunkte über die *Coordinates*-Eigenschaft zu verwalten, besitzt die 3D-Polylinie die Prozedur *AppendVertex*. Hiermit fügen Sie der Polylinie einen weiteren Stützpunkt hinzu. Leider können Sie auch hier nicht die genaue Position des Stützpunkts angeben, so dass der neue Punkt an das Ende der Polylinie angehängt wird. Der Aufruf dieser Funktion ist recht einfach. Zunächst bestimmen Sie die Koordinaten des neuen Stützpunkts und fügen diesen dann unter Verwendung der *AppendVertex*-Anweisung an die Polylinie an.

```
Utility.GetEntity Object, PickedPoint, "3D-Polylinie wählen"
If TypeName(Object) = "IAcad3DPolyline" Then
    PointsList = Object.Coordinates
    PtCount = ((UBound(PointsList) + 1) / 3) - 1
    LastPt = Object.Coordinate(PtCount)
    NewPoint = Utility.GetPoint(LastPt, "neuen Stützpunkt angeben")
    Object.AppendVertex NewPoint
End If
```

Da auf diese Weise nur ein Stützpunkt angefügt werden kann, sollten Sie sich bei der Verarbeitung der Stützpunkte besser an die Eigenschaften *Coordinates* und *Coordinate* halten, da Sie damit auch Stützpunkte an einer beliebigen Position einfügen können.

Die nächste Prozedur, die genauer betrachtet werden soll, ist die *Explode*-Anweisung. Mit dieser Anweisung können Sie die entsprechende Polylinie in ihre einzelnen Bestandteile auflösen. Als Ergebnis erhalten Sie ein Array, das Verweise auf alle aus der Polylinie entstandenen Liniobjekte enthält. Die Polylinie selbst bleibt erhalten.

```
Dim Object As Object
Utility.GetEntity Object, PickedPoint, "3D-Polylinie wählen"
If TypeName(Object) = "IAcad3DPolyline" Then
    ObjectList = Object.Explode
    Object.Delete
End If
```

Wenn Sie eine quadratische bzw. kubische 3D-Polylinie auflösen, dann entstehen statt der Linien einzelne Kreisbögen, die dem Verlauf der Polylinie angeglichen sind.

Auf die einzelnen Objekte des Arrays können Sie wieder ganz normal zugreifen, indem Sie diese direkt ansprechen. Die Änderung der Farbe der Objekte könnte dann wie folgt aussehen.

```
For i = 0 To UBound(ObjectList)
    ObjectList(i).Color = acBlue
Next
```

Sie könnten aber auch einen Verweis auf das entsprechende Objekt an eine neue Variable übergeben. Dabei müssen Sie beachten, dass es sich bei den einzelnen Felddaten des Arrays um Objekte handelt, was das Schlüsselwort *Set* wieder ins Spiel bringt.

```
For i = 0 To UBound(ObjectList)
    Set NewObject = ObjectList(i)
    NewObject.Color = acBlue
Next
```

Prozeduren	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der 3D-Polylinie ab.
<i>GetXData</i>	Ruft die erweiterten Daten der 3D-Polylinie ab.
<i>Highlight</i>	Hebt die 3D-Polylinie hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der 3D-Polylinie mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine 3D-Polylinie als Resultat einer Spiegelung mit dieser 3D-Polylinie.
<i>Mirror3D</i>	Erstellt eine 3D-Polylinie als Resultat einer räumlichen Spiegelung dieser 3D-Polylinie.
<i>Move</i>	Verschiebt die 3D-Polylinie von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die 3D-Polylinie um einen bestimmten Winkel.

Tabelle 8.18: Weitere Prozeduren einer 3D-Fläche

Prozeduren	Beschreibung
<i>Rotate3D</i>	Dreht die 3D-Polylinie um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die 3D-Polylinie um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der 3D-Polylinie.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die 3D-Polylinie an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der 3D-Polylinie im AutoCAD-Zeichnungsfenster.

Tabelle 8.18: Weitere Prozeduren einer 3D-Fläche (Forts.)

Anwendungsbeispiele

(Beispielprojekt 08\Acad3DPoly.dvb)

Es folgen wieder einige Beispiele zur Steuerung der Eigenschaften der 3D-Polylinie, insbesondere die Anpassung der Koordinaten der Polylinie.

Im ersten Beispiel wird eine 3D-Polylinie mit drei Stützpunkten erzeugt. Jeder Stützpunkt enthält jeweils einen X-, einen Y- und einen Z-Wert. Nachdem die Polylinie erstellt wurde, wird sie geschlossen, ihre Farbe auf Blau festgelegt und abschließend hervorgehoben.

```
Public Sub Create3DPoly()
Dim Cr3DPoly As Acad3DPolyline
Dim VertList(8) As Double
On Local Error Resume Next
VertList(0) = 0: VertList(1) = 0: VertList(2) = 0
VertList(3) = 0: VertList(4) = 40: VertList(5) = 5
VertList(6) = 40: VertList(7) = 40: VertList(8) = 2
Set Cr3DPoly = ModelSpace.Add3DPoly(VertList)
If TypeName(Cr3DPoly) <> "Nothing" Then
    Cr3DPoly.Closed = True
    Cr3DPoly.Color = acBlue
    Cr3DPoly.Highlight True
End If
End Sub
```

Im zweiten Beispiel fügen wir der Polylinie einen Stützpunkt zwischen Punkt 1 und 2 ein. Dies erreichen wir, indem die Stützpunkte einzeln durchlaufen werden. Wird der zweite Stützpunkt erreicht, werden schnell die Koordinaten des neuen Stützpunkts eingefügt und anschließend wird das Array mit den restlichen Stützpunkten aufgefüllt.

```
Public Sub Append3DPolyVertex()
Dim Cr3DPoly As Acad3DPolyline
Dim VertList(8) As Double
Dim VarPoint As Variant
On Local Error Resume Next
VertList(0) = 0: VertList(1) = 0: VertList(2) = 0
VertList(3) = 0: VertList(4) = 40: VertList(5) = 5
VertList(6) = 40: VertList(7) = 40: VertList(8) = 2
Set Cr3DPoly = ThisDrawing.ModelSpace.Add3DPoly(VertList)
If TypeName(Cr3DPoly) <> "Nothing" Then
    VarPoint = Cr3DPoly.Coordinates
```

```

ReDim NewPoints(UBound(VarPoint) + 3) As Double
x = 0
For i = 0 To UBound(VarPoint) Step 3
    If i = 3 Then
        NewPoints(x) = 10
        x = x + 1
        NewPoints(x) = 20
        x = x + 1
        NewPoints(x) = 3
        x = x + 1
    End If
    NewPoints(x) = VarPoint(i)
    x = x + 1
    NewPoints(x) = VarPoint(i + 1)
    x = x + 1
    NewPoints(x) = VarPoint(i + 2)
    x = x + 1
Next
Cr3DPoly.Coordinates = NewPoints
End If
End Sub

```

8.3.5 Kreisbögen (AcadArc)

Kreisbögen werden in AutoCAD durch die Klasse *AcadArc* definiert. Sie können einen Verweis auf einen Kreisbogen erstellen, indem Sie die Funktion *AddArc* im Modellbereich, im Papierbereich oder auf einen Block anwenden. Sie können auch einen vorhandenen Kreisbogen über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln. Nachdem Sie die Objektvariable initialisiert haben, lassen sich folgende Eigenschaften und Methoden auf den Kreisbogen anwenden.

Kreisbogeneigenschaften verarbeiten

Der Kreisbogen enthält verschiedene Eigenschaften, mit denen Sie seine Position ermitteln bzw. anpassen. Die wichtigsten sind hierbei die Eigenschaften *Center* mit den Koordinaten des Zentrums sowie *StartAngle* und *EndAngle* zur Festlegung der Position des Start- und Endpunkts des Kreisbogens. Der Zentrums punkt wird als 3D-Array, das die X-, Y- und Z-Koordinate enthält, ausgegeben bzw. festgelegt.

```

Dim Center(2) As Double
Center(0) = 100
Center(1) = 120
Center(2) = 10
CrArc.Center = Center

```

Wenn Sie diese Koordinaten ändern, dann entspricht das einer Verschiebung des Objekts. Die Konfiguration an sich bleibt erhalten. Um den Start- und Endpunkt des Bogens anzupassen, sind einige Details zu beachten. Das Objekt enthält zwar die Eigenschaften *StartPoint* und *EndPoint*, die die Koordinaten dieser Punkte enthalten. Diese sind aber schreibgeschützt, so dass eine direkte Angabe neuer Koordinaten nicht möglich ist. Sie müssen vielmehr den indirekten Weg über die Eigenschaften *StartAngle*, *EndAngle* und *Radius* gehen. Diese drei Eigen-

schaften zusammen bestimmen die Position der beiden Bogenendpunkte. *StartAngle* bestimmt dabei die Eingangsrichtung des Bogenanfangs und *EndAngle* die Ausgangsrichtung des Bogenendes (siehe Abbildung 8.20). Schon die Änderung einer dieser Eigenschaften hat Auswirkungen auf die Länge und den Flächeninhalt des Bogens.

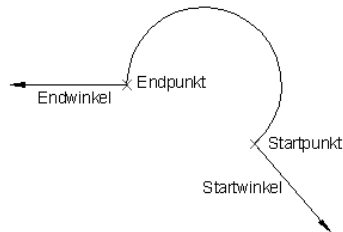


Abbildung 8.20: Bogendefinition

Die Werte der Winkel werden in Radiant angegeben. Wenn Sie Werte an die Eigenschaften übergeben, müssen Sie also darauf achten, dass Sie die richtige Einheit für die Winkel verwenden.

Siehe auch *Kapitel 8.1.3*.

Eigenschaften des Kreisbogens

Zusätzlich zu den Eigenschaften, die Position und Größe des Bogens festlegen, besitzt dieses Objekt noch rein informative Eigenschaften. Diese sind alle schreibgeschützt und können somit nicht direkt angepasst werden. *ArcLength* gibt die aktuelle Bogenlänge, *Area* den Flächeninhalt und *TotalAngle* den Innenwinkel des Bogens zurück. Bei der Fläche des Bogens müssen Sie darauf achten, dass diese nicht den Inhalt des Vollkreises angibt, sondern nur bis zu den Bogenendpunkten berechnet wird.

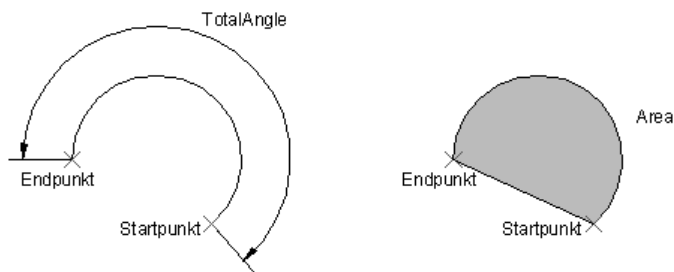


Abbildung 8.21: Definition der Eigenschaften *TotalAngle* und *Area*

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Kreisbogens fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Kreisbogens zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Kreisbogen ein Erweiterungs- wörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Kreisbogens zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Kreisbogens an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Kreisbogens an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Kreisbogens an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für den Kreisbogen an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Kreisbogens zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadArc</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identi- fizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für den Kreisbogen zurück oder legt diesen fest.
<i>Thickness</i>	Diese Eigenschaft speichert die Objekthöhe des Kreisbogens. Sie kön- nen den aktuellen Wert auslesen oder zurückschreiben.
<i>Visible</i>	Mit dieser Eigenschaft steuern Sie die Sichtbarkeit des Kreisbogens.

Tabelle 8.19: Weitere Eigenschaften eines Kreisbogens

Methoden des Bogen-Objekts

Neben den Prozeduren, die jedes grafische Objekt in AutoCAD besitzt, enthält der Bogen die *Offset*-Anweisung. Mit dieser Anweisung können Sie eine Kopie des Bogens erzeugen, die um einen angegebenen Betrag versetzt wird, ähnlich wie dies auch der AutoCAD-Befehl *_Offset* bewerkstelligt. Als Ergebnis dieser Anweisung können Sie einen Verweis auf den neuen Bogen in eine Objektvariable übergeben lassen, um den neuen Bogen weiterzuverarbeiten. Als Parameter dieser Anweisung müssen Sie lediglich den Abstand festlegen, um den das Ursprungsobjekt versetzt werden soll. Ein Betrag > 0 versetzt den Bogen nach außen. Ist der angegebene Wert < 0 wird der neue Bogen nach innen versetzt.

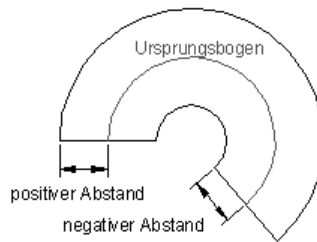


Abbildung 8.22: Kreisbogen versetzen

Die Ausführung dieser Anweisung ist wieder recht einfach. Zunächst müssen Sie den Versatz ermitteln, was beispielsweise durch eine Benutzerabfrage mit der *GetDistance*-Funktion des *Utility*-Objekts erfolgen könnte. Dann müssen Sie festlegen, in welche Richtung der Bogen versetzt werden soll. Dies könnte entweder der Benutzer tun, indem er gleich einen positiven oder negativen Wert festlegt oder indem Sie (wie beim *_Offset*-Befehl) die Seite durch die Auswahl eines Punkts bestimmen lassen.

```
Dim Object As Object, PickPoint, Center
On Error Resume Next
Offset = Utility.GetDistance("Abstand eingeben:")
If Offset <= 0 Then Exit Sub
Utility.GetEntity Object, pickedpoint, "Wählen Sie einen Bogen:"
Prompt = "Punkt auf Seite angeben, auf die versetzt werden soll:"
If TypeName(Object) = "IAcadArc" Then
    PickPoint = Utility.GetPoint(, Prompt)
    Center = Object.Center
    If GetDistance(Center, PickPoint) < Object.Radius Then
        Offset = -Offset
    End If
    Object.Offset Offset
End If
```

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechteckige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Kreisbogens ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Kreisbogens ab.
<i>Highlight</i>	Hebt den Kreisbogen hervor.

Tabelle 8.20: Weitere Prozeduren eines Kreisbogens

Prozedur	Beschreibung
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Kreisbogens mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt einen Kreisbogens als Resultat einer Spiegelung mit diesem Kreisbogen.
<i>Mirror3D</i>	Erstellt einen Kreisbogens als Resultat einer räumlichen Spiegelung dieses Kreisbogens.
<i>Move</i>	Verschiebt den Kreisbogen von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht den Kreisbogen um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Kreisbogen um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert den Kreisbogen um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Kreisbogens.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf den Kreisbogen an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Kreisbogens im AutoCAD-Zeichnungsfenster.

Tabelle 8.20: Weitere Prozeduren eines Kreisbogens (Forts.)

8.3.6 Kreise (AcadCircle)

Kreise werden durch die Klasse *AcadCircle* dargestellt, die eine Reihe von speziellen Eigenschaften aufweist, deren Änderung Auswirkung auf die Darstellung des Kreises hat. Sie können einen Verweis auf einen Kreis erstellen, indem Sie die Funktion *AddCircle* im Modell- bzw. Papierbereich oder auf einen Block anwenden. Sie können auch einen vorhandenen Kreisbogen über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln. Nachdem Sie die Objektvariable initialisiert haben, können Sie folgende Eigenschaften und Methoden auf den Kreisbogen anwenden.

Kreiseigenschaften verarbeiten

Der Mittelpunkt des Kreises ist in der Eigenschaft *Center* gespeichert und als 3D-Array definiert. Genau wie bei einem Bogen beeinflusst die Änderung der Werte dieser Eigenschaft die Position des Kreises.

```
Dim Center(2) As Double
Center(0) = 100
Center(1) = 120
Center(2) = 10
CrCircle.Center = Center
```

Das Kreis-Objekt besitzt außerdem die Eigenschaften *Area*, *Circumference*, *Diameter* und *Radius*. Im Gegensatz zum Bogen ist die Eigenschaft *Area*, die den Flächeninhalt des Kreises bestimmt, nicht schreibgeschützt, so dass Sie den Flächeninhalt direkt anpassen können.

```
CrCircle.Area = 200
```

Circumference bestimmt den Umfang des Kreises, *Diameter* den Durchmesser und *Radius* den Radius des Kreises. Alle diese Eigenschaften sind eng miteinander verknüpft. Die Änderung einer dieser Eigenschaften hat unmittelbare Auswirkungen auf den Wert der anderen Eigenschaften. Diese Eigenschaften sind *Radius*, *Area*, *Diameter* und *Circumference*. Gültige Werte für diese Eigenschaften sind alle reellen Zahlen, die größer 0 sind.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Kreises fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Kreises zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Kreis ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Kreises zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Kreises an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Kreises an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Kreises an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für den Kreis an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die ObjectID des Kreises zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadCircle</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für den Kreis zurück oder legt diesen fest.
<i>Thickness</i>	Diese Eigenschaft speichert die Objekthöhe des Kreises.
<i>Visible</i>	Mit dieser Eigenschaft steuern Sie die Sichtbarkeit des Kreises.

Tabelle 8.21: Weitere Eigenschaften eines Kreises

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechteckige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Kreises ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Kreises ab.
<i>Highlight</i>	Hebt den Kreis hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Kreises mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt einen Kreis als Resultat einer Spiegelung mit diesem Kreis.
<i>Mirror3D</i>	Erstellt einen neuen Kreis als Resultat einer räumlichen Spiegelung dieses Kreises.
<i>Move</i>	Verschiebt den Kreis von Ursprungs- auf Zielkoordinaten.
<i>Offset</i>	Versetzt den Kreis um einen bestimmten Betrag. Nach Ausführung der Funktion wird ein neuer Kreis erstellt und an eine Objektvariable zurückgegeben.
<i>Rotate</i>	Dreht den Kreis um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Kreis um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert den Kreis um einen bestimmten Faktor. Achtung: Die Änderung dieser Eigenschaft hat Auswirkungen auf die Eigenschaften <i>Area</i> , <i>Diameter</i> , <i>Circumference</i> und <i>Radius</i> .
<i>SetXData</i>	Speichert die erweiterten Daten des Kreises.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf den Kreis an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Kreises im AutoCAD-Zeichnungsfenster.

Tabelle 8.22: Methoden eines Kreises

Anwendungsbeispiel

(Beispielprojekt 08\AcadCircle.dvb)

Im folgenden Beispielprojekt werden wir den Benutzer einen Kreis wählen und die Eigenschaften *Radius*, *Umfang*, *Fläche* und *Durchmesser* in einem Formular anzeigen lassen. Der Benutzer soll dann die Möglichkeit erhalten, diese Werte in dem Formular zu ändern. Dazu wird zunächst ein Formular benötigt, auf dem Sie vier Textfelder und zwei Schaltflächen positionieren (siehe Abbildung 8.23).

Abbildung 8.23: Beispielformular

Modulcode

Zum Starten des Projekts ist eine Prozedur im Klassenmodul *ThisDrawing* erforderlich. In dieser Prozedur wird zunächst der Benutzer aufgefordert, einen Kreis in der Zeichnung auszuwählen. Hat er dies getan, wird das Kreisobjekt an die Objektvariable *CrCircle* übergeben, die im Formular deklariert wurde. Danach wird die Formularprozedur *initValues* ausgeführt und abschließend das Formular gestartet.

```
Public Sub EditCircle()
Dim Object As Object
On Error Resume Next
Utility.GetEntity Object, PickedPoint, "Kreis wählen"
If TypeName(Object) = "IAcadCircle" Then
    Set UserForm1.CrCircle = Object
    UserForm1.initValues
    UserForm1.Show
End If
End Sub
```

Die folgende Prozedur, die ja bereits in der Startprozedur aufgerufen wurde, übergibt die Werte der einzelnen Eigenschaften an die entsprechenden Textfelder. Da es sich bei den Werten um Zahlen handelt, werden diese mit der *RealToString*-Funktion in Zeichenfolgen umgewandelt.

```
Public Sub initValues()
TextBox1 = ThisDrawing.Utility.RealToString(CrCircle.radius, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
TextBox2 = ThisDrawing.Utility.RealToString(CrCircle.Diameter, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
TextBox3 = ThisDrawing.Utility.RealToString(CrCircle. _
    Circumference, acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))>
TextBox4 = ThisDrawing.Utility.RealToString(CrCircle.Area, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
End Sub
```

Die folgenden vier Ereignisprozeduren reagieren jeweils auf die Änderung eines Werts durch den Benutzer. Das Prinzip ist aber immer das gleiche. Zuerst wird der eingegebene Wert in eine reelle Zahl mithilfe der *DistanceToReal*-Funktion umgewandelt und an die entsprechende Eigenschaft übergeben. Hat der Benutzer einen ungültigen Wert festgelegt, egal für welche Eigenschaft, dann löst AutoCAD einen Fehler aus. Diesen Umstand können wir nutzen und geben in diesem Fall die Fehlerbeschreibung (*Description*-Eigenschaft des *Err*-Objekts) in einem Meldungsfeld aus. Außerdem wird die *Cancel*-Eigenschaft auf *TRUE* gesetzt, um den Änderungsvorgang abzubrechen.

Radius wurde geändert:

```
Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _  
MSForms.ReturnBoolean)  
On Error Resume Next  
CrCircle.radius = ThisDrawing.Utility.DistanceToReal(TextBox1, _  
acDefaultUnits)  
If Err <> 0 Then  
    MsgBox Err.Description  
    Cancel = True  
End If  
initValues  
End Sub
```

Durchmesser wurde geändert:

```
Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _  
MSForms.ReturnBoolean)  
On Error Resume Next  
CrCircle.Diameter = ThisDrawing.Utility.DistanceToReal(TextBox2, _  
acDefaultUnits)  
If Err <> 0 Then  
    MsgBox Err.Description  
    Cancel = True  
End If  
initValues  
End Sub
```

Umfang wurde geändert:

```
Private Sub TextBox3_BeforeUpdate(ByVal Cancel As _  
MSForms.ReturnBoolean)  
On Error Resume Next  
CrCircle.Circumference = ThisDrawing.Utility.DistanceToReal(_  
    TextBox3, acDefaultUnits)  
If Err <> 0 Then  
    MsgBox Err.Description  
    Cancel = True  
End If  
initValues  
End Sub
```

Flächeninhalt wurde geändert:

```
Private Sub TextBox4_BeforeUpdate(ByVal Cancel As _  
MSForms.ReturnBoolean)  
On Error Resume Next  
CrCircle.Area = ThisDrawing.Utility.DistanceToReal(TextBox4, _  
acDefaultUnits)  
If Err <> 0 Then  
    MsgBox Err.Description  
    Cancel = True  
End If  
initValues  
End Sub
```

Schaltfläche „Wählen“

Der Code dieser Schaltfläche wurde bereits in der Startprozedur dieses Projekts verwendet und dient lediglich der Auswahl eines neuen Kreisobjekts. Wichtig hierbei ist nur, dass das Formular zunächst ausgeblendet, dann die Auswahlfunktion ausgeführt und abschließend das Formular wieder eingeblendet wird.

```
Private Sub CommandButton2_Click()
Dim Object As Object
On Error Resume Next
Me.Hide
ThisDrawing.Utility.GetEntity Object, PickedPoint, "Kreis wählen"
If TypeName(Object) = "IAcadCircle" Then
    Set CrCircle = Object
    initValues
End If
Me.Show
End Sub
```

8.3.7 Ellipse (AcadEllipse)

Sie erstellen einen Verweis auf eine Ellipse, indem Sie mit der Funktion *AddEllipse* eine neue Ellipse dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene Ellipse über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln. Nachdem Sie die Objektvariable initialisiert haben, können Sie folgende Eigenschaften und Methoden auf die Ellipse anwenden.

Ellipsen verfügen über einige zusätzliche Eigenschaften, mit denen Sie ihre Position und Darstellung anpassen können. Die Position der Ellipse ergibt sich aus der Eigenschaft *Center*, die die Koordinaten des Zentrums enthält. Wenn Sie den Wert dieser Eigenschaft ändern, dann ist das so, als ob die Ellipse verschoben wird. Die Ellipsenform bleibt erhalten. Die Konfiguration der Ellipse, also die eiförmige Darstellung, erhält die Ellipse erst durch Angabe der Werte für die Haupt- und Nebenachse. Für beide Achsen müssen Sie jeweils zwei Eigenschaften festlegen: zum einen den Radius und zum anderen die Achsrichtung, ausgehend vom Zentrumspunkt (Abbildung 8.24 und Abbildung 8.25). Die Eigenschaften, die diese Daten enthalten, sind *MajorRadius* und *MajorAxis* für die Hauptachse und *MinorRadius* und *MinorAxis* für die Nebenachse. Der Radius ist klar, etwas schwieriger erscheint die Definition der Richtungswerte (*MajorAxis* und *MinorAxis*). Diese Eigenschaften enthalten keine reinen Koordinaten, sondern die Richtungsvektoren ausgehend vom Zentrumspunkt der Ellipse. Die jeweilige Entfernung wird ja bekanntlich durch die Radien der Achsen bestimmt.

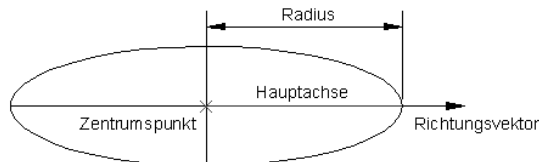


Abbildung 8.24: Definition der Hauptachse

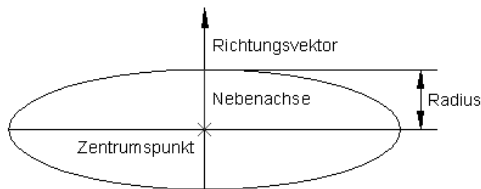


Abbildung 8.25: Definition der Nebenachse

Die Eigenschaft *RadiusRatio* speichert das Verhältnis der beiden Radien zueinander. Somit könnten Sie den Nebenradius auch durch Änderung dieser Eigenschaft bestimmen. Da der Nebenradius kleiner oder gleich dem Hauptradius sein muss, darf das Radienverhältnis nicht größer als 1.0 sein, wobei dieser Wert einen Kreis definieren würde. Vorrang vor dem Seitenverhältnis hat bei der Änderung einer der Radien allerdings immer der andere Radius.

Ein Beispiel: Angenommen, Sie legen ein Seitenverhältnis von 0.5 fest, bei einem Hauptradius von 10.0. Dann berechnet AutoCAD den Nebenradius neu, was 5.0 ergeben würde. Wenn Sie nun den Wert einer der Radien neu vergeben, dann wird das Seitenverhältnis nach den Werten der beiden Radien neu bestimmt und nicht der Radius nach dem Seitenverhältnis.

Offene Ellipse definieren (elliptischer Bogen)

Um aus der Darstellung einer Vollellipse einen elliptischen Bogen zu erzeugen, haben Sie zwei Möglichkeiten. Zum einen können Sie mit den Eigenschaften *StartAngle* und *EndAngle* den Öffnungswinkel der Ellipse bestimmen. Diese Winkelangaben beziehen sich auf die Hauptachse, womit klar ist, dass zunächst die Richtung dieser Achse bestimmt werden muss, um anschließend den Öffnungswinkel zu berechnen.

```
Basis = Object.Center
MajorAxis = Object.MajorAxis
MajorPoint(0) = basis(0) + MajorAxis(0)
MajorPoint(1) = basis(1) + MajorAxis(1)
MajorPoint(2) = basis(2) + MajorAxis(2)
StartAngle = Utility.GetOrientation(basis)
EndAngle = Utility.GetOrientation(basis)
Object.StartAngle = StartAngle - Utility.AngleFromXAxis(Basis, _
    MajorPoint)
Object.EndAngle = EndAngle - Utility.AngleFromXAxis(basis, _
    MajorPoint)
```

Als Alternative zur Bestimmung der beiden Winkel besitzt das Ellipsenobjekt die Eigenschaften *EndParameter* und *StartParameter*. Der Parameter liegt dabei im Bereich $0 - 2 * \pi$. Die Berechnung der Parameters basiert auf folgender parametrischen Vektorformel:

$$\text{Parameter} = \text{AchseH} * \cos(\text{Winkel}) + \text{AchseN} * \sin(\text{Winkel})$$

Daraus ergibt sich, dass beide Eigenschaftstypen eng miteinander verbunden sind. Ändert sich der Wert eines Winkels, dann wird auch der dazugehörige Parameter neu berechnet. Der Vorteil bei der Verwendung der Parameter ist, dass Sie keine Rücksicht auf die derzeitigen Winklereinstellungen des Systems nehmen müssen. Der angegebene Parameterwert legt immer direkt den Öffnungswinkel ausgehend von der Hauptachse fest, was die Bestimmung der Winkel wesentlich vereinfacht.

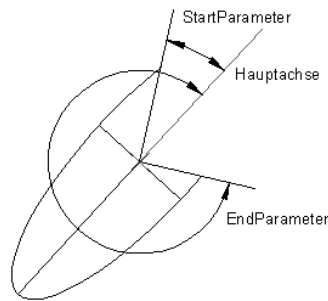


Abbildung 8.26: Definition der Ellipsen-Parameter

```
StartParameter = Utility.GetReal("Startwinkel eingeben:");
```

```
EndParameter = Utility.GetReal("Endwinkel eingeben:");
```

```
Object.StartParameter = StartParameter * PI / 200
```

```
Object.EndParameter = EndParameter * PI / 200
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Area</i> (schreibgeschützt)	Diese Eigenschaft enthält den aktuellen Flächeninhalt der Ellipse.
<i>Color</i>	Legt die Farbe der Ellipse fest oder gibt diese zurück.
<i>EndPoint</i> (schreibgeschützt)	Gibt die Koordinaten des Endpunkts des elliptischen Bogens zurück.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Ellipse zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Ellipse ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Ellipse zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der Ellipse an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Ellipse an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Ellipse an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Ellipse an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die ObjectID der Ellipse zurück.

Tabelle 8.23: Standardeigenschaften der Ellipse

Eigenschaft	Beschreibung
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadEllipse</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Ellipse zurück oder legt diesen fest.
<i>StartPoint</i> (schreibgeschützt)	Gibt die Koordinaten des Startpunkts des elliptischen Bogens zurück.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der Ellipse steuern.

Tabelle 8.23: Standardeigenschaften der Ellipse (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens der Ellipse.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Ellipse ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Ellipse ab.
<i>Highlight</i>	Hebt die Ellipse hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Ellipse mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Ellipse als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine Ellipse als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt die Ellipse von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die Ellipse um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Ellipse um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Ellipse um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Ellipse.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Ellipse an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Ellipse im AutoCAD-Zeichnungsfenster.

Tabelle 8.24: Standardmethoden der Ellipse

Anwendungsbeispiel

(Beispielprojekt: 08\AcadEllipse.dvb)

Mit den folgenden Beispielprogrammen kann der Benutzer jeweils eine gewählte Ellipse öffnen. Der Unterschied zwischen beiden ist die Art und Weise, wie die Öffnung der Ellipse definiert wird. Der Start der Prozeduren ist jeweils gleich. Zunächst wird der Benutzer mittels der *GetEntity*-Funktion der Klasse *MyUtility* (siehe *Kapitel A.3*) aufgefordert, eine Ellipse in der Zeichnung zu wählen. Wurde dies erfolgreich ausgeführt, was wir mit der *TypeName()*-Funktion überprüfen, wird der Code innerhalb des *If...End If*-Blocks fortgesetzt.

```
Public Sub Ellipse_Sample1()
Dim Object As Object
Dim MajorPoint(2) As Double
On Error Resume Next
Set Util = New MyUtility
Util.GetEntity Object, PickedPoint
If TypeName(Object) = "IAcadEllipse" Then
```

Von hieran unterscheiden sich dann die beiden Programme. In der ersten Prozedur werden nun die Koordinaten des Zentrums der gewählten Ellipse an die Variable *Basis* übergeben. Zusätzlich wird noch der Vektor der Hauptachse benötigt, welcher in der Variablen *MajorAxis* gespeichert wird.

```
Basis = Object.Center
MajorAxis = Object.MajorAxis
```

Um nun die Koordinate des Endpunkts der Hauptachse zu ermitteln, müssen wir nur den Koordinaten des Basispunkts die Vektoren der Hauptachse hinzuaddieren.

```
MajorPoint(0) = Basis(0) + MajorAxis(0)
MajorPoint(1) = Basis(1) + MajorAxis(1)
MajorPoint(2) = Basis(2) + MajorAxis(2)
```

Jetzt brauchen wir nur noch die Richtungswinkel für den Anfang und das Ende der Öffnung mithilfe der *GetOrientation*-Funktion durch den Benutzer angeben zu lassen. Dabei erhält dieser eine optische Hilfe durch die Anzeige eines Gummibands, ausgehend vom Zentrums der Ellipse.

```
StartAngle = Util.GetOrientation(Basis, "Startwinkel:")
EndAngle = Util.GetOrientation(Basis, "Endwinkel:")
```

Nachdem der Benutzer diesen Richtungswinkel bestimmt hat, kann der Öffnungswinkel der Ellipse angepasst werden. Da die Hauptachse nicht unbedingt in der Nullrichtung der Zeichnung verlaufen muss, wird vom gewählten Start- und Endwinkel jeweils der Richtungswinkel der Hauptachse abgezogen.

```
Object.StartAngle = StartAngle - Util.AngleFromXAxis(Basis, _
MajorPoint)
Object.EndAngle = EndAngle - Util.AngleFromXAxis(Basis, _
MajorPoint)
End If
End Sub
```

Das zweite Beispielprojekt ist nicht allzu umfangreich. Hier wird der Benutzer nur aufgefordert, einen Start- und Endwinkel einzugeben.

```
Public Sub Ellipse_Sample2()
Dim Object As Object
Dim MajorPoint(2) As Double
Dim StartParameter As Double, EndParameter As Double
On Error Resume Next
Set Util = New MyUtility
Utility.GetEntity Object, PickedPoint
If TypeName(Object) = "IAcadEllipse" Then
```

Die eingegebenen Winkel werden dann als Parameter der Ellipse zugewiesen.

```
StartParameter = Util.GetOrientation(, "Startwinkel eingeben:")
EndParameter = Util.GetOrientation(, "Endwinkel eingeben:")
Object.StartParameter = StartParameter
Object.EndParameter = EndParameter
End If
End Sub
```

8.3.8 Bemaßungsobjekte

Bemaßungsobjekte sind komplexe Elemente, die sich aus den verschiedensten Objekten zusammensetzen. So enthalten alle einen Text und Führungslinien, womit sie auch über gewisse Gemeinsamkeiten, Eigenschaften, die jede Bemaßung besitzt, verfügen. Aus diesem Grund wurde in die Objektbibliothek von AutoCAD die Ausgangsklasse *AcadDimension* implementiert, die diese gemeinsamen Eigenschaften bereits enthält. Auf dieser Klasse basieren alle anderen Bemaßungsobjekte. Bevor wir uns den einzelnen Bemaßungen zuwenden, werde ich näher auf diese Eigenschaften eingehen.

Eigenschaften der Bemaßungsobjekte

Die Eigenschaft *DecimalSeparator* legt das Dezimaltrennzeichen metrischer Bemaßungen fest. Der Ausgangswert dieser Eigenschaft ist zwar ein Punkt ("."), Sie können allerdings auch ein anderes beliebiges Zeichen festlegen. Verwendet wird der Wert dieser Eigenschaft, wenn *UnitsFormats* auf *acDimDecimal* festgelegt wurde.

Mit den Eigenschaften *SuppressLeadingZeros* und *SuppressTrailingZeros* können Sie die Anzeige der Vorkomma- bzw. Nachkommanulln steuern. Wenn Sie *SuppressLeadingZeros* auf *TRUE* setzen, werden die führenden Nullen vor dem Komma unterdrückt. *FALSE* macht dies wieder rückgängig.

- *TRUE* – Nullen unterdrücken (der Wert *0.0123* würde als *.0123* dargestellt)
- *FALSE* – Nullen nicht unterdrücken (der Wert *0.0123* würde als *0.0123* dargestellt)

SuppressTrailingZeros bewirkt einen ähnlichen Effekt, nur das hier die nachfolgenden Nullen aktiviert bzw. deaktiviert werden.

- *TRUE* – Nullen unterdrücken (der Wert *0.1200* würde als *0.12* dargestellt)
- *FALSE* – Nullen nicht unterdrücken (der Wert *0.1200* würde als *0.1200* dargestellt)

Wie auch immer der Bemaßungstext formatiert wird, den tatsächlich gemessenen Wert können Sie immer über die Eigenschaft *Measurement* abrufen. Bei linearen Bemaßungsobjekten enthält diese Eigenschaft einen metrischen Wert, bei Winkelbemaßungen einen Wert in Radiant.

Eigenschaften des Bemaßungstextes

Neben der *Color*-Eigenschaft, die die Farbe des Bemaßungsobjekts festlegt, können Sie zusätzlich die Farbe des Bemaßungstextes explizit mit *TextColor* festlegen. Die Farbwerte liegen zwischen 0 und 256. Mit der Eigenschaft *TextGap* bestimmen Sie die Größe der Lücken, die links und rechts des Maßtextes in die Maßlinie eingefügt werden sollen. Der Bemaßungstext wird dann automatisch außerhalb der Bemaßungslinien positioniert, wenn der Abstandswert, den Sie in dieser Eigenschaft festgelegt haben, zwischen Text und Linie unterschritten wird.

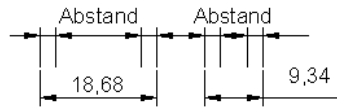


Abbildung 8.27: Eigenschaft *TextGap*

TextHeight bestimmt die Höhe des Textes, *TextRotation* den Drehwinkel und *TextStyle* den Textstil des Bemaßungsobjekts. Wenn Sie den Wert der Eigenschaft *TextRotation* anpassen, müssen Sie darauf achten, dass dessen Wert in Radiant angegeben wird und Sie gegebenenfalls eine Umrechnung vornehmen müssen. Wenn Sie einen anderen Text als Messergebnis in dem Bemaßungsobjekt darstellen wollen, können Sie diese Zeichenfolge in der Eigenschaft *TextOverride* festlegen. Der Wert dieser Eigenschaft ersetzt dann den ursprünglichen Bemaßungstext. Sie können allerdings den gemessenen Wert einschließlich der Toleranzen in dieser Eigenschaft mit einbeziehen, indem Sie die Zeichenfolge „<>“ an der Stelle eingeben, an der der Bemaßungstext dargestellt werden soll, z. B.:

```
DimObject.TextOverride = "L=<>m"
```

Um den ursprünglichen Text wiederherzustellen, müssen Sie lediglich die festgelegte Zeichenfolge wieder entfernen:

```
DimObject.TextOverride = ""
```

Die Eigenschaften *TextPrefix* und *TextSuffix* legen die Zeichenfolgen fest, die dem Bemaßungstext vorangestellt (*TextPrefix*) bzw. nachfolgen (*TextSuffix*) sollen. Die Werte werden nur berücksichtigt, wenn der Bemaßungstext nicht durch die *TextOverride*-Eigenschaft bestimmt wird. Die gebräuchlichsten Präfixe bzw. Suffixe können Sie der Tabelle 8.25 entnehmen.

Name	Symbol	Eingabe in AutoCAD	Unicode
Grad	°	%%d	\U+0080
Durchmesser	Ø	%%c	\U+2205
Toleranz	±	%%p	\U+0081
Prozent	%	%%%	

Tabelle 8.25: Gebräuchliche Präfixe und Suffixe

Die Position (3D-Array mit den Koordinaten des Einfügepunkts) des Bemaßungstextes können Sie auch über die Eigenschaft *TextPosition* abrufen oder mit dieser Eigenschaft manuell festlegen. Die *TextMovement*-Eigenschaft bestimmt zusätzlich, wie die Bemaßungslinie auf das Verschieben des Textes reagieren soll. Je nachdem, welchen Wert (Tabelle 8.26) Sie für diese Eigenschaft verwenden, wird die Darstellung der Bemaßungslinie angepasst.

Konstante	Beschreibung	Wert
<i>AcDimLineWithText</i>	Die Bemaßungslinie folgt dem verschobenen Bemaßungstext.	0
<i>AcMoveTextAddLeader</i>	Eine Bezugslinie wird von der Bemaßungslinie zum verschobenen Text gezogen.	1
<i>AcMoveTextNoLeader</i>	Der Text wird ohne Bezugslinie verschoben.	2

Tabelle 8.26: Konstanten der TextMovement-Eigenschaft

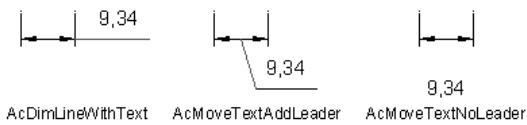


Abbildung 8.28: Der Wert der TextMovement-Eigenschaft bewirkt eine unterschiedliche Liniendarstellung

Mit den folgenden Eigenschaften legen Sie fest, wo der Bemaßungstext im Verhältnis zur Maßlinie positioniert wird. *VerticalTextPosition* bestimmt dabei die vertikale und *HorizontalTextPosition* die horizontale Ausrichtung des Bemaßungstextes, wobei Letztere nicht in jedem Bemaßungsobjekt verfügbar ist. Für beide Eigenschaften stehen gewisse Konstantenaufstellungen zur Verfügung, die die entsprechende Textposition festlegen. Welche Wirkung welche Konstante auf den Bemaßungstext hat, können Sie der Tabelle 8.27 für die horizontale und der Tabelle 8.28 für die vertikale Ausrichtung entnehmen.

Konstante	Beschreibung	Wert
<i>acVertCentered</i>	Ausrichtung zentriert zwischen den Verbindungslinien.	0
<i>acAbove</i>	Positioniert den Text oberhalb der Bemaßungslinie.	1
<i>acOutside</i>	Positioniert den Text an der Linie, die am weitesten von der Textposition des Maßtextes entfernt ist.	2
<i>acJIS</i>	Ausrichtung nach dem Japanischen Industriestandard.	3

Tabelle 8.27: Konstanten der Eigenschaft HorizontalTextPosition

Konstante	Beschreibung	Wert
<i>acHorzCentered</i>	Zentriert	0
<i>acFirstExtensionLine</i>	Erste Hilfslinie	1
<i>acSecondExtensionLine</i>	Zweite Hilfslinie	2
<i>acOverFirstExtension</i>	Über der ersten Hilfslinie	3
<i>acOverSecondExtension</i>	Über der zweiten Hilfslinie	4

Tabelle 8.28: Konstanten der Eigenschaft VerticalTextPosition

In der Eigenschaft *TextPrecision* wird die Anzahl der Dezimalstellen des Bemaßungstextes angegeben. Es gibt zwar eine Reihe von Konstanten (Tabelle 8.29), die Sie für diese Eigenschaft verwenden können, es ist allerdings bequemer, den Wert der jeweiligen Konstante zu verwenden, da diese der Anzahl der Nachkommastellen entspricht. Gültige Werte liegen im Bereich von 0 bis 8.

Konstante	Beschreibung	Wert
<i>acDimPrecisionZero</i>	0.	0 Stellen
<i>acDimPrecisionOne</i>	0.0	1 Stelle
<i>acDimPrecisionTwo</i>	0.00	2 Stellen
<i>acDimPrecisionThree</i>	0.000	3 Stellen
<i>acDimPrecisionFour</i>	0.0000	4 Stellen
<i>acDimPrecisionFive</i>	0.00000	5 Stellen
<i>acDimPrecisionSix</i>	0.000000	6 Stellen
<i>acDimPrecisionSeven</i>	0.0000000	7 Stellen
<i>acDimPrecisionEight</i>	0.00000000	8 Stellen

Tabelle 8.29: Konstanten der Precision-Eigenschaften

RoundDistance bestimmt den Rundungswert bei linearen Bemaßungseinheiten. Wenn Sie hier beispielsweise den Wert 0.25 festlegen, werden die Einheiten des Bemaßungsobjekts auf die nächste Viertelstufe gerundet. Die hat allerdings keinen Einfluss auf den tatsächlich gemessenen Wert, der in der Eigenschaft *Measurement* gespeichert ist.

UnitsFormat bestimmt die zu verwendende Einheit der linearen Bemaßung, *AngleFormat* die der Winkelbemaßung. Sie könnten so rein theoretisch für jedes Bemaßungsobjekt eine eigene Einheit festlegen. Die Konstanten, die Sie für die jeweilige Eigenschaft verwenden können, entnehmen Sie den folgenden Tabellen.

Konstante	Beschreibung	Wert
<i>acDimLScientific</i>	Wissenschaftlich	1
<i>acDimLDecimal</i>	Dezimal	2
<i>acDimLEngineering</i>	Engineering	3
<i>acDimLArchitectural</i>	Architectural	4
<i>acDimLFractional</i>	Bruch	5
<i>acDimLWindowsDesktop</i>	Windows-Desktop	6

Tabelle 8.30: Konstanten für lineare Einheiten

Konstante	Beschreibung	Wert
<i>acDegrees</i>	Dezimalgrad	0
<i>acDegreeMinuteSeconds</i>	Grad Minuten Sekunden	1
<i>AcGrads</i>	Neugrad	2
<i>AcRadians</i>	Radian	3

Tabelle 8.31: Konstanten für Winkeleinheiten

TextInside legt fest, ob die Bemaßungstexte erzwungenermaßen innerhalb des Bemaßungsobjekts angezeigt werden sollen, egal, ob die Werte der *TextGap*-Eigenschaft unterschritten wurden oder nicht. Legen Sie für *TextInside* den Wert *TRUE* fest, werden die Texte immer innerhalb der Hilfslinien, wie in Abbildung 8.29 zu sehen, positioniert.

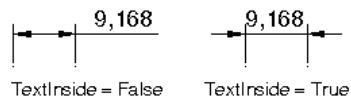


Abbildung 8.29: Auswirkung der Eigenschaft *TextInside*

Wenn Sie die Eigenschaft *TextInside* auf *TRUE* gesetzt haben, steht Ihnen mit *TextInsideAlign* eine weitere Eigenschaft zur Verfügung. Mit dieser bestimmen Sie die Ausrichtung des Maßtextes innerhalb der Hilfslinien. Folgende Einstellungen sind für diese Eigenschaft möglich:

- *TRUE* – Text wird horizontal ausgerichtet
- *FALSE* – Text wird an der Maßlinie ausgerichtet

TextOutsideAlign hingegen bestimmt die Ausrichtung des Bemaßungstextes, wenn dieser außerhalb der Hilfslinien positioniert wurde. Auch hier gelten die folgenden Einstellungsmöglichkeiten:

- *TRUE* – Text wird horizontal ausgerichtet
- *FALSE* – Text wird an der Maßlinie ausgerichtet

Eigenschaften zur Steuerung der Toleranzwerte

Zusätzlich zu den gemessenen Werten kann das Bemaßungsobjekt auch Toleranzbereiche anzeigen, die individuell festgelegt werden. Um beispielsweise in einer Streckenmessung darzustellen, dass diese mit einer Genauigkeit von $\pm 5\text{cm}$ bestimmt wurde, können Sie die Toleranzeigenschaften des jeweiligen Bemaßungsobjekts anpassen.

Wenn Sie in der *TextOverride*-Eigenschaft eine alternative Zeichenfolge für das Bemaßungsobjekt festlegen, dann werden die Toleranzeigenschaften ignoriert und somit auch keine Toleranzwerte angezeigt.

Mit den Eigenschaften *ToleranceLowerLimit* und *ToleranceUpperLimit* können Sie den Wert der unteren und oberen Toleranz festlegen bzw. abrufen. Mit diesen Werten wird dann die Toleranz im Bemaßungstext dargestellt, wobei diese Darstellung mit der Eigenschaft *ToleranceDisplay* bestimmt wird. Diese legt fest, ob überhaupt eine Toleranz angezeigt werden soll und wenn ja, wie diese Toleranzwerte angezeigt werden sollen. Folgende Konstanten steuern diese Eigenschaft.

- *acTolNone* Die Toleranzanzeige ist abgeschaltet. Das Bemaßungsobjekt zeigt nur den direkt gemessenen Wert an.
- *acTolSymmetrical* (symmetrische Toleranzanzeige): Bei dieser Darstellungsart wird nur der obere Toleranzwert berücksichtigt. AutoCAD nimmt an, dass sowohl die obere als auch die untere Toleranz den gleichen absoluten Wert besitzen und die Toleranz wird zu einem Wert zusammengefasst (Abbildung 8.30).

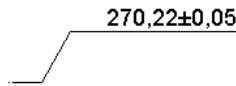


Abbildung 8.30: Symmetrische Darstellung

- *acTolDeviation* (Abweichung): Beide Toleranzwerte werden übereinander angezeigt, auch wenn beide denselben absoluten Wert besitzen sollten (Abbildung 8.31).

Wenn der untere Toleranzwert (*ToleranceLowerLimit*) sich im absoluten Wert vom oberen Toleranzwert (*ToleranceUpperLimit*) unterscheidet, stellt AutoCAD automatisch den Wert *acTolDeviation* für diese Eigenschaft ein, wenn zuvor eine symmetrische Anzeige festgelegt war.

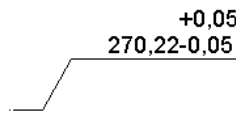


Abbildung 8.31: Abweichende Darstellung

- *acTolLimits* (Toleranzgrenzen): Wenn Sie diese Konstante der *ToleranceDisplay*-Eigenschaft zuweisen, dann werden die Toleranzgrenzen direkt an den gemessenen Wert angebracht und ein Wertebereich als Bemaßungstext angezeigt. Der gemessene Wert ist aber immer noch über die Eigenschaft *Measurement* abrufbar.

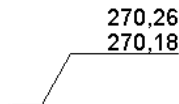


Abbildung 8.32: Toleranzgrenzen

- *acTolBasic* (Grundtoleranz): Mit dieser Konstante legen Sie fest, dass der gemessene Wert als theoretisch genaues Maß angezeigt werden soll. Der Maßtext wird daraufhin umrahmt dargestellt, was darauf hinweist, dass dieser gemessene Wert vom tatsächlichen Wert abweicht oder abweichen kann, der gemessene Wert aber als richtig angenommen wird.

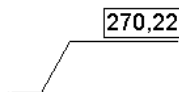


Abbildung 8.33: Grundtoleranz

Nachdem festgelegt wurde, ob und wie die Toleranzwerte anzuzeigen sind, können Sie mit den folgenden Eigenschaften die Darstellung noch ein wenig präzisieren. Mit den Eigenschaften *ToleranceSuppressLeadingZeros* und *ToleranceSuppressTrailingZeros* kann festgelegt werden, ob die führenden und/oder nachfolgenden Nullen angezeigt werden sollen. Die Werte dieser Eigenschaften haben die gleiche Wirkung auf die Darstellung der Toleranzwerte wie die Eigenschaften *SuppressLeadingZeros* und *SuppressTrailingZeros* auf den Bemaßungstext. *Toleran-*

cePrecision legt die Anzahl der angezeigten Nachkommastellen fest. Gültige Werte hierfür liegen im Bereich von 0 bis 8. Mit *ToleranceHeightScale* bestimmen Sie die Höhe des Toleranztextes. Dieser Wert legt allerdings nicht direkt die Höhe fest, sondern bestimmt lediglich die Skalierung des Textes, die sich auf die Höhe des Bemaßungstextes bezieht. Wurde beispielsweise für *ToleranceHeightScale* ein Wert von 0.5 festgelegt, dann werden die Toleranzen nur mit der halben Größe des Bemaßungstextes (*TextHeight*) dargestellt.

Die Eigenschaft *ToleranceJustification* bestimmt die vertikale Ausrichtung für Toleranzwerte im Verhältnis zur nominalen Maßtexthöhe. Sie können für diese Eigenschaft auf folgende Konstanten zurückgreifen:

Konstante	Beschreibung	Wert
<i>AcTolBottom</i>	Ausrichtung unterhalb	0
<i>AcTolMiddle</i>	Ausrichtung zentriert	1
<i>AcTolTop</i>	Ausrichtung oberhalb	2

Tabelle 8.32: Konstanten der *ToleranceJustification*-Eigenschaft

Darstellung der Bemaßungspfeile steuern

Die folgenden Eigenschaften steuern die Darstellung der Bemaßungslinien und der Pfeile. Die meisten der Bemaßungsobjekte besitzen entweder alle oder zumindest einige dieser Eigenschaften, je nachdem, welche der Zusatzdarstellungen das Objekt benutzen kann.

Arrowhead1Type und *Arrowhead2Type* bestimmen den Typ der ersten bzw. zweiten Pfeilspitze. Sie können für diese Eigenschaft eine der Konstanten der Tabelle 8.33 verwenden, um die Darstellung der jeweiligen Pfeilspitzen anzupassen.

	Konstante	Beschreibung	Wert	Blockname
	<i>acArrowDefault</i>	Vorgabe	0	
	<i>acArrowClosedBlank</i>	Geschlossen leer	1	_CLOSEDBLANK
	<i>acArrowClosed</i>	Geschlossen	2	_CLOSED
	<i>acArrowDot</i>	Punkt	3	_DOT
	<i>acArrowArchTick</i>	Architektonisch	4	_ARCHTICK
	<i>acArrowOblique</i>	Schräg	5	_OBLIQUE
	<i>acArrowOpen</i>	Offen	6	_OPEN
	<i>acArrowOrigin</i>	Ausgangspunkt	7	_ORIGIN

Tabelle 8.33: Konstanten der *ArrowheadType*-Eigenschaft

	Konstante	Beschreibung	Wert	Blockname
	<i>acArrowOrigin2</i>	Ausgangspunkt 2	8	_ORIGIN2
	<i>acArrowOpen90</i>	Rechter Winkel	9	_OPEN90
	<i>acArrowOpen30</i>	Offen 30	10	_OPEN30
	<i>acArrowDotSmall</i>	Punkt klein	11	_DOTSMALL
	<i>acArrowDotBlank</i>	Punkt leer	12	_DOTBLANK
	<i>acArrowSmall</i>	Punkt klein leer	13	_SMALL
	<i>acArrowBoxBlank</i>	Quadrat	14	_BOXBLANK
	<i>acArrowBoxFilled</i>	Quadrat gefüllt	15	_BOXFILLED
	<i>acArrowDatumBlank</i>	Bezugsdreieck	16	_DATUMBLANK
	<i>acArrowDatumFilled</i>	Bezugsdreieck gefüllt	17	_DATUMFILLED
	<i>acArrowIntegral</i>	Integral	18	_INTEGRAL
	<i>acArrowNone</i>	Keine	19	_NONE
	<i>acArrowUserDefined</i>	Benutzerdefiniert	20	

Tabelle 8.33: Konstanten der ArrowheadType-Eigenschaft (Forts.)

Neben den Standardpfeilspitzen haben Sie auch die Möglichkeit, eine benutzerdefinierte Spitze festzulegen. Dazu müssen Sie zunächst den Block bestimmen, der als Pfeilspitze dienen soll. Je nachdem, welchen Pfeil Sie bestücken wollen, müssen Sie den entsprechenden Blocknamen an die Eigenschaften *Arrowhead1Block* und *Arrowhead2Block* übergeben. Wenn Sie für diese Eigenschaft einen gültigen Blocknamen vergeben, wird die Eigenschaft *Arrowhead1Type* respektive *Arrowhead2Type* auf den Wert *acArrowUserDefined* gesetzt. Nachdem Sie die Pfeilspitzen bestimmt haben, können Sie mit *ArrowheadSize* noch deren Größe festlegen. Gültige Werte für diese Eigenschaft sind alle positiven reellen Zahlen. Je größer dieser Wert ist, umso größer wird die Pfeilspitze dargestellt. Sie müssen also darauf achten, dass die Blöcke für benutzerdefinierte Pfeile ungefähr in der Größe der Standardspitzen definiert wurden, damit Sie eine korrekte Darstellung erhalten, wenn Sie in einem Bemaßungsobjekt einen benutzerdefinierten und einen Standardpfeil verwenden. Sobald Sie einen der Standardpfeile verwenden, wird durch AutoCAD die entsprechende Blockdefinition der Blockauflistung hinzugefügt.

Darstellung der Bemaßungslinien steuern

Bemaßungslinien unterteilen sich in zwei Kategorien, Maßlinien und Hilfslinien. Als Maßlinien werden die Linien bezeichnet, die den gemessenen Wert grafisch untermauern und an deren Ende jeweils ein Bemaßungspfeil dargestellt werden kann. Je nachdem, welches Bemaßungsobjekt Sie verwenden, besitzt das Objekt entweder zwei (z.B. Winkel oder lineare Bemaßungen), eine (z.B. Radius- oder Durchmesserbemaßung) oder gar keine Maßlinien (Koordinatenbemaßung).

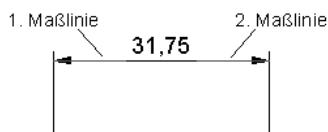


Abbildung 8.34: Maßlinien einer ausgerichteten Bemaßung

Entsprechend der Anzahl der vorhandenen Maßlinien, verfügt das Objekt auch über eine gewisse Anzahl an Eigenschaften, die die Darstellung dieser Linien beeinflussen. Die Farbe und die Linienstärken der Maßlinien werden mit den Eigenschaften *DimensionLineColor* und *DimensionLineWeight* gesteuert. Dabei ist es egal, ob eine oder zwei Maßlinien vorliegen. *DimLineInside* bestimmt die Anzeige der Maßlinien, wenn der Bemaßungstext innerhalb der Maßlinien positioniert wird (*TextInside* = *TRUE*). Wird der Wert dieser Eigenschaft auf *TRUE* gesetzt, werden die Maßlinien ebenfalls angezeigt.

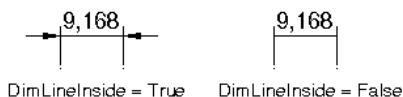


Abbildung 8.35: Auswirkung der Eigenschaft *DimLineInside*

Um dennoch eine Maßlinie unterhalb der Bemaßungstexte anzuzeigen (Abbildung 8.35), müssen Sie die Eigenschaft *ForceLineInside* auf *TRUE* setzen. Der jeweilige Wert dieser Eigenschaften gilt generell für alle Maßlinien des Objekts.

DimLine1Supress und *DimLine2Supress* steuern die Sichtbarkeit der Maßlinien. Wenn Sie eine der Eigenschaften auf den Wert *TRUE* setzen, wird die Anzeige der entsprechenden Maßlinie unterdrückt.

Darstellung der Hilfslinien steuern

Hilfslinien stellen eine optische Verbindung zwischen bemessenem Objekt und Bemaßungstext her. Meist werden sie lotrecht zur Maßlinie gezeichnet, sie können aber auch wahlweise schräg dargestellt werden. Zur Anpassung dieser Linien stellen die Bemaßungsobjekte die verschiedensten Eigenschaften zur Verfügung.

ExtensionLineColor und *ExtensionLineWeight* legen wieder die Farbe und die Linienstärke der Linien fest. Mit *ExtLine1Supress* und *ExtLine2Supress* kann die Anzeige der Hilfslinien unterdrückt werden, indem Sie diesen Eigenschaften den Wert *TRUE* zuweisen.

Der Wert für die Verlängerung der Hilfslinien, wie in Abbildung 8.36 zu sehen, wird mit der Eigenschaft *ExtensionLineExtend* festgelegt. *ExtensionLineOffset* bestimmt hingegen die Lücke zwischen den Hilfslinien und den Ursprungspunkten (Abbildung 8.36). Verfügt ein Bemaßungsobjekt über mehr als eine Hilfslinie, gelten die Einstellungen der Eigenschaften für alle Hilfslinien.

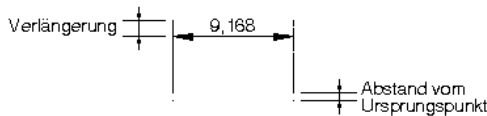


Abbildung 8.36: Definition der Verlängerung der Hilfslinien und des Abstands vom Ursprungspunkt

Mit der *Fit*-Eigenschaft können Sie bestimmen, welche Elemente verschoben werden müssen, damit Text und Pfeile zwischen die Hilfslinien passen. Folgende Konstanten können dabei verwendet werden:

- *acTextAndArrows* – diese Option platziert Text und Maßpfeil innerhalb des Bemaßungsobjekts.
- *acArrowsOnly* – diese Option platziert nur den Maßpfeil innerhalb des Bemaßungsobjekts.
- *acTextOnly* – diese Option platziert nur den Text innerhalb des Bemaßungsobjekts.
- *acBestFit* – diese Option platziert Text und/oder Maßpfeil innerhalb des Bemaßungsobjekts, wenn genügend Platz für das entsprechende Element vorhanden ist.

Alternative Einheiten bestimmen

Mit alternativen Bemaßungseinheiten können den gemessenen Wert eines Bemaßungsobjekts im Format eines weiteren Meßsystems anzeigen lassen. In der Regel wird dies das britische System (Zoll) bei metrischen Bemaßungstexten und umgekehrt sein. Der alternative Maßtext wird dabei zusätzlich zum normalen Maßtext in eckigen Klammer angezeigt. Da sowohl die primären als auch die alternativen Einheiten zusammen im Bemaßungstext angezeigt werden, überschreibt die Eigenschaft *TextOverride* auch die alternative Einheit, wenn Sie dieser eine Zeichenfolge zuweisen. Verwenden Sie wieder die Zeichen „<>“, um den Bemaßungstext in diese Zeichenfolge einzubinden.

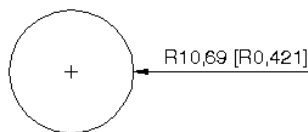


Abbildung 8.37: Anzeige der alternative Einheiten

Mit dem Wert, den Sie der Eigenschaft *AltUnits* zuweisen, können Sie die Anzeige dieser Einheiten aktivieren (*TRUE*) bzw. deaktivieren (*FALSE*) oder kontrollieren, ob diese bereits aktiviert bzw. deaktiviert wurden. Alle anderen Eigenschaften sind so oder so ähnlich bereits von den primären Einheiten her bekannt.

AltSuppressLeadingZeros und *AltSuppressTrailingZeros* bestimmen beispielsweise, ob die führenden bzw. nachfolgenden Nullen angezeigt werden sollen. Die Eigenschaften *AltTextPrefix* und *AltTextSuffix* legen die Zeichenfolgen fest, die dem alternativen Bemaßungstext vorangestellt werden (*AltTextPrefix*) bzw. nachfolgen (*AltTextSuffix*) sollen. Die gängigsten Werte finden Sie in Tabelle 8.25. Mit *AltUnitsPrecision* steuern Sie die Anzahl der Nachkommastellen der alternativen Einheit. Wie bei *TextPrecision* können Sie entweder eine Zahl zwischen 0 und 8 vergeben oder die Konstanten der Tabelle 8.29 verwenden. *AltUnitsFormat* legt das Format der alternativen Einheiten fest. Sie können bei dieser Eigenschaft auf die Konstanten von *UnitsFormat* zurückgreifen, die in Tabelle 8.30 aufgelistet sind.

AltRoundDistance bestimmt den Rundungswert der Alternativeinheiten. Wenn Sie hier beispielsweise den Wert 0.25 festlegen, werden die Alternativeinheiten des Bemaßungsobjekts auf die nächste Viertelstufe gerundet.

Den eigentlichen Wert, der in der Zeichenfolge der alternativen Einheit angezeigt werden soll, bestimmt – neben dem gemessenen Wert – die Eigenschaft *AltUnitsScale*. Mit dem Wert, der hier festgelegt wurde, wird der tatsächlich gemessene Wert skaliert, woraus sich schließlich das Ergebnis für die alternative Einheit ergibt. Angenommen, Sie haben eine metrische Strecke bemaßt und wollen diese dann alternativ in Zoll darstellen.

Die Eigenschaften *AltSuppressZeroFeet* und *AltSuppressZeroInches* schalten die Unterdrückung von 0 Fuß (*AltSuppressZeroFeet*) bzw. 0 Zoll (*AltSuppressZeroInches*) für die Alternativeinheit ein (*TRUE*) oder aus (*FALSE*).

Eigenschaften zur Steuerung der Toleranzwerte von Alternativeinheiten

Wenn Sie die Anzeige von Toleranzen für das Bemaßungsobjekt aktiviert haben (Eigenschaft *ToleranceDisplay*), werden auch für die alternativen Einheiten Toleranzwerte angezeigt. Zusätzlich zu den Toleranzeigenschaften des Bemaßungsobjekts, die dann ebenfalls für die Toleranzen der Alternativeinheiten verwendet werden (z.B. Ober- und Untergrenze der Toleranz oder die vertikale Textposition), verfügt das Bemaßungsobjekt über einige Eigenschaften, die speziell die Darstellungen der alternativen Toleranzen steuern.

AltTolerancePrecision legt die Anzahl der Dezimalstellen für die Toleranzwerte der Alternativeinheiten fest. Genau wie bei den *Precision*-Eigenschaften zuvor, können Sie entweder eine Ganzzahl zwischen 0 und 8 vergeben oder die Konstanten der Tabelle 8.29 verwenden.

AltToleranceSuppressLeadingZeros und *AltToleranceSuppressTrailingZeros* bestimmen, ob die führenden bzw. nachfolgenden Nullen der Toleranz unterdrückt werden sollen (*TRUE*) oder nicht (*FALSE*).

Mit den Eigenschaften *AltToleranceSuppressZeroFeet* und *AltToleranceSuppressZeroInches* bestimmen Sie wiederum, ob die Anzeige von 0 Fuß bzw. 0 Zoll unterdrückt werden soll.

Winkelbemaßung (AcadDim3PointAngular)

Diese Klasse *AcadDim3PointAngular* stellt ein Winkelbemaßungsobjekt in AutoCAD dar. Dieser Bemaßungstyp entspricht der Winkelbemaßung über einen Scheitelpunkt (AutoCAD-Befehl: *bemwinkel*).

Sie können einen Verweis auf solch ein Bemaßungsobjekt erstellen, indem Sie mit der Funktion *AddDim3PointAngular* ein neues Bemaßungsobjekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes 3-Punkte-Bemaßungsobjekt über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

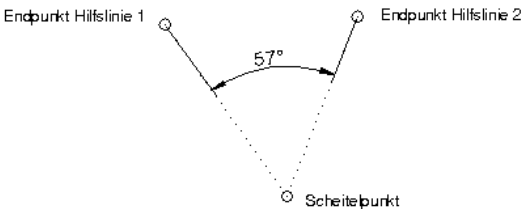


Abbildung 8.38: Winkelbemaßung über drei Punkte

Diese Objekt enthält einige spezielle Eigenschaften, die direkt auf die Winkelbemaßung zugeschnitten sind. *AngleFormat* bestimmt beispielsweise die Winkleinheit für das Bemaßungsobjekt. In der Einheit, die Sie in dieser Eigenschaft festlegen, wird der Messergebnistext im Bemaßungsobjekt angezeigt. Die jeweiligen Winkleinheiten können Sie der Tabelle 8.31 entnehmen.

Der Scheitelpunkt der Winkelbemaßung ist in der Eigenschaft *AngleVertex* gespeichert, die Endpunkte der Hilfslinien in den Eigenschaften *ExtLine1EndPoint* und *ExtLine2EndPoint*. Alle drei Punkte werden als 3D-Array mit WKS-Koordinaten zurückgegeben oder festgelegt.

Hilfslinieneigenschaften	
<i>ExtensionLineColor</i>	Bestimmt die Farbe der Hilfslinien.
<i>ExtensionLineExtend</i>	Bestimmt den aktuellen Wert zur Verlängerung der Hilfslinie über die Maßlinie hinaus.
<i>ExtensionLineOffset</i>	Bestimmt die Lücke zwischen Hilfslinien und Ursprungspunkten.
<i>ExtensionLineWeight</i>	Diese Eigenschaft legt die Linienstärke für die Hilfslinien des Bemaßungsobjekts fest oder ruft sie ab.
<i>ExtLine1Point</i>	Ursprungspunkt der ersten Hilfslinie.
<i>ExtLine1Suppress</i>	Steuert die Unterdrückung der ersten Hilfslinie.
<i>ExtLine2Point</i>	Ursprungspunkt der zweiten Hilfslinie.
<i>ExtLine2Suppress</i>	Steuert die Unterdrückung der zweiten Hilfslinie.
<i>Fit</i>	Bestimmt, welche Elemente verschoben werden müssen, damit Text und Pfeile zwischen die Hilfslinien passen.

Pfeilspitzeneigenschaften	
<i>Arrowhead1Block</i>	Blockname für den ersten benutzerdefinierten Pfeil.
<i>Arrowhead1Type</i>	Bestimmt den Typ der ersten Pfeilspitze.
<i>Arrowhead2Block</i>	Blockname für den zweiten benutzerdefinierten Pfeil.
<i>Arrowhead2Type</i>	Bestimmt den Typ der zweiten Pfeilspitze.
<i>ArrowheadSize</i>	Bestimmt die Größe der Pfeilspitzen.

Tabelle 8.34: Eigenschaften der 3-Punkt-Winkelbemaßung

Maßlinieneigenschaften	
<i>DimensionLineColor</i>	Bestimmt die Farbe der Maßlinien.
<i>DimensionLineWeight</i>	Bestimmt die Linienstärke der Maßlinien.
<i>DimLine1Suppress</i>	Steuert die Unterdrückung der ersten Maßlinie.
<i>DimLine2Suppress</i>	Steuert die Unterdrückung der zweiten Maßlinie.
<i>DimLineInside</i>	Bestimmt die Darstellung der Maßlinien außerhalb der Hilfslinien.
<i>ForceLineInside</i>	Mit dieser Eigenschaft können Sie die Darstellung einer Maßlinie zwischen Hilfslinien ein- (<i>TRUE</i>) bzw. ausschalten (<i>FALSE</i>), selbst wenn der Text außerhalb platziert wird.
Texteigenschaften	
<i>DecimalSeparator</i>	Dezimaltrennzeichen für metrische Bemaßungen.
<i>SuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen.
<i>SuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen.
<i>TextColor</i>	Farbe der Maßtextes.
<i>TextGap</i>	Größe der Lücke in der Maßlinie links und rechts vom Maßtext.
<i>TextHeight</i>	Höhe des Bemaßungstextes.
<i>TextInside</i>	Erzwingt die Positionierung der Maßtexte innerhalb der Hilfslinien.
<i>TextInsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes innerhalb der Hilfslinien.
<i>TextMovement</i>	Textposition beim manuellen oder automatischen Verschieben
<i>TextOutsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes außerhalb der Hilfslinien.
<i>TextOverride</i>	Ersetzt den gemessenen Text des Bemaßungsobjekts durch einen frei definierbaren Text.
<i>TextPosition</i>	Textpositionspunkt.
<i>TextPrecision</i>	Bestimmt die Dezimalstellengenauigkeit für den Bemaßungstext.
<i>TextPrefix</i>	Textpräfix für das Bemaßungsobjekt.
<i>TextRotation</i>	Aktueller Drehwinkel des Bemaßungstextes.
<i>TextStyle</i>	Textstil des Bemaßungstextes.
<i>TextSuffix</i>	Textsuffix für das Bemaßungsobjekt.
<i>VerticalTextPosition</i>	Vertikale Position des Bemaßungstextes.
<i>HorizontalTextPosition</i>	Horizontale Position des Bemaßungstextes.

Tabelle 8.34: Eigenschaften der 3-Punkt-Winkelbemaßung (Forts.)

Toleranzeigenschaften	
<i>ToleranceDisplay</i>	Steuert den Anzeigemodus von Toleranzwerten.
<i>ToleranceHeightScale</i>	Skalierfaktor für die Texthöhe von Toleranzen relativ zur Höhe des Bemaßungstextes.
<i>ToleranceJustification</i>	Vertikale Ausrichtung für Toleranzwerte.
<i>ToleranceLowerLimit</i>	Unterer Toleranzwert.
<i>TolerancePrecision</i>	Anzahl der Dezimalstellen für die Toleranzwerte.
<i>ToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen der Toleranzen.
<i>ToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen der Toleranzen.
<i>ToleranceUpperLimit</i>	Oberer Toleranzwert.

Allgemeine Eigenschaften	
Alle Eigenschaften der Klasse <i>AcadEntity</i>	
<i>AngleFormat</i>	Bestimmt die Winkleinheit für das Bemaßungsobjekt.
<i>AngleVertex</i>	Scheitelpunkt der 3-Punkt-Winkelbemaßung.
<i>Measurement</i> (schreibgeschützt)	Diese Eigenschaft enthält den gemessenen Wert des Bemaßungsobjekts.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>Rotation</i>	Aktueller Drehwinkel des Bemaßungsobjekts.
<i>ScaleFactor</i>	Übergeordneter Skalierfaktor für Größen- oder Abstandseigenschaften.
<i>StyleName</i>	Name des zugeordneten Bemaßungsstils.

Tabelle 8.34: Eigenschaften der 3-Punkt-Winkelbemaßung (Forts.)

Prozeduren
Alle Prozeduren der Klasse <i>AcadEntity</i>

Winkelbemaßung (AcadDimAngular)

Die zweite Form der Winkelbemaßung wird durch die Klasse *AcadDimAngular* dargestellt. Dieser Bemaßungstyp entspricht der Winkelbemaßung, die über zwei Linien oder einen Kreisbogen bestimmt wurde (AutoCAD-Befehl: *bemwinkel*).

Sie können einen Verweis auf solch ein Bemaßungsobjekt erstellen, indem Sie mit der Funktion *AddDimAngular* ein neues Bemaßungsobjekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Winkelbemaßungsobjekt über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

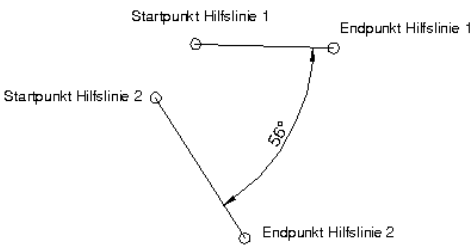


Abbildung 8.39: Winkelbemaßung über Objekte ohne gemeinsamen Scheitelpunkt

Im Gegensatz zum Winkelbemaßungsobjekt, das über drei Punkte bestimmt wurde, verwaltet dieses Objekt nicht den Scheitelpunkt des eingeschlossenen Winkels und verfügt somit auch nicht über die Eigenschaft *AngleVertex*. Da der Winkel über zwei AutoCAD-Objekte ermittelt wird, die keinen gemeinsamen Scheitelpunkt besitzen, werden vielmehr die Endpunkte der ersten und zweiten Hilfslinie verwaltet. Start- und Endpunkt der ersten Hilfslinie werden in den Eigenschaften *ExtLine1StartPoint* und *ExtLine1EndPoint*, die der zweiten in den Eigenschaften *ExtLine2StartPoint* und *ExtLine2EndPoint* gespeichert. Alle vier Punkte werden über ein 3D-Array zurückgegeben bzw. festgelegt, das die X-, Y- und Z-Koordinaten im WKS enthalten. Wenn Sie versuchen, die Anfangs- und Endpunkte der Winkelbemaßung anzupassen, sollten Sie darauf achten, dass die Linien nicht parallel verlaufen.

Hilfslinieneigenschaften	
<i>ExtensionLineColor</i>	Bestimmt die Farbe der Hilfslinien.
<i>ExtensionLineExtend</i>	Bestimmt den aktuellen Wert zur Verlängerung der Hilfslinie über die Maßlinie hinaus.
<i>ExtensionLineOffset</i>	Bestimmt die Lücke zwischen Hilfslinien und Ursprungspunkten.
<i>ExtensionLineWeight</i>	Diese Eigenschaft legt die Linienstärke für die Hilfslinien des Bemaßungsobjekts fest oder ruft sie ab.
<i>ExtLine1Point</i>	Ursprungspunkt der ersten Hilfslinie.
<i>ExtLine1Suppress</i>	Steuert die Unterdrückung der ersten Hilfslinie.
<i>ExtLine2Point</i>	Ursprungspunkt der zweiten Hilfslinie.
<i>ExtLine2Suppress</i>	Steuert die Unterdrückung der zweiten Hilfslinie.
<i>Fit</i>	Bestimmt, welche Elemente verschoben werden müssen, damit Text und Pfeile zwischen die Hilfslinien passen.

Pfeilspitzeneigenschaften	
<i>Arrowhead1Block</i>	Blockname für den ersten benutzerdefinierten Pfeil.
<i>Arrowhead1Type</i>	Bestimmt den Typ der ersten Pfeilspitze.
<i>Arrowhead2Block</i>	Blockname für den zweiten benutzerdefinierten Pfeil.
<i>Arrowhead2Type</i>	Bestimmt den Typ der zweiten Pfeilspitze.
<i>ArrowheadSize</i>	Bestimmt die Größe der Pfeilspitzen.

Tabelle 8.35: Eigenschaften der Winkelbemaßung

Maßlinieneigenschaften	
<i>DimensionLineColor</i>	Bestimmt die Farbe der Maßlinie.
<i>DimensionLineWeight</i>	Bestimmt die Linienstärke der Maßlinien.
<i>DimLine1Suppress</i>	Steuert die Unterdrückung der ersten Maßlinie.
<i>DimLine2Suppress</i>	Steuert die Unterdrückung der zweiten Maßlinie.
<i>DimLineInside</i>	Bestimmt die Darstellung der Maßlinien außerhalb der Hilfslinien.
<i>ForceLineInside</i>	Mit dieser Eigenschaft können Sie die Darstellung einer Maßlinie zwischen Hilfslinien ein- (<i>TRUE</i>) bzw. ausschalten (<i>FALSE</i>), selbst wenn der Text außerhalb platziert wird.
Texteigenschaften	
<i>DecimalSeparator</i>	Dezimaltrennzeichen für metrische Bemaßungen.
<i>SuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen.
<i>SuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen.
<i>TextColor</i>	Farbe der Maßtextes.
<i>TextGap</i>	Größe der Lücke in der Maßlinie links und rechts vom Maßtext.
<i>TextHeight</i>	Höhe des Bemaßungstextes.
<i>TextInside</i>	Erzwingt die Positionierung der Maßtexte innerhalb der Hilfslinien.
<i>TextInsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes innerhalb der Hilfslinien.
<i>TextMovement</i>	Textposition beim manuellen oder automatischen Verschieben.
<i>TextOutsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes außerhalb der Hilfslinien.
<i>TextOverride</i>	Ersetzt den gemessenen Text des Bemaßungsobjekts durch einen frei definierbaren Text.
<i>TextPosition</i>	Textpositionspunkt.
<i>TextPrecision</i>	Bestimmt die Dezimalstellengenauigkeit für den Bemaßungstext.
<i>TextPrefix</i>	Textpräfix für das Bemaßungsobjekt.
<i>TextRotation</i>	Aktueller Drehwinkel des Bemaßungstextes.
<i>TextStyle</i>	Textstil des Bemaßungstextes.
<i>TextSuffix</i>	Textsuffix für das Bemaßungsobjekt.
<i>VerticalTextPosition</i>	Vertikale Position des Bemaßungstextes.
<i>HorizontalTextPosition</i>	Horizontale Position des Bemaßungstextes.

Tabelle 8.35: Eigenschaften der Winkelbemaßung (Forts.)

Toleranzeigenschaften	
<i>ToleranceDisplay</i>	Steuert den Anzeigemodus von Toleranzwerten.
<i>ToleranceHeightScale</i>	Skalierfaktor für die Texthöhe von Toleranzen relativ zur Höhe des Bemaßungstextes.
<i>ToleranceJustification</i>	Vertikale Ausrichtung für Toleranzwerte.
<i>ToleranceLowerLimit</i>	Unterer Toleranzwert.
<i>TolerancePrecision</i>	Anzahl der Dezimalstellen für die Toleranzwerte.
<i>ToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen der Toleranzen.
<i>ToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen der Toleranzen.
<i>ToleranceUpperLimit</i>	Oberer Toleranzwert.

Allgemeine Eigenschaften	
Alle Eigenschaften der Klasse <i>AcadEntity</i>	
<i>AngleFormat</i>	Bestimmt die Winkleinheit für das Bemaßungsobjekt.
<i>Measurement</i> (schreibgeschützt)	Diese Eigenschaft enthält den gemessenen Wert des Bemaßungsobjekts.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>Rotation</i>	Aktueller Drehwinkel des Bemaßungsobjekts.
<i>ScaleFactor</i>	Übergeordneter Skalierfaktor für Größen- oder Abstandseigenschaften.
<i>StyleName</i>	Name des zugeordneten Bemaßungsstils.

Tabelle 8.35: Eigenschaften der Winkelbemaßung (Forts.)

Prozeduren
Alle Prozeduren der Klasse <i>AcadEntity</i>

Lineare Bemaßung (*AcadDimAligned* / *AcadDimRotated*)

Bei linearen Bemaßungen wird in AutoCAD zwischen zwei verschiedenen Klassen unterschieden. Die erste Klasse (*AcadDimAligned*) wird verwendet, wenn es sich um eine ausgerichtete lineare Bemaßung handelt (AutoCAD-Befehl: *bemausg*). Mit der zweiten (*AcadDimRotated*) wird hingegen eine gedrehte lineare Bemaßung abgebildet (AutoCAD-Befehl: *bemausg*).

Sie können einen Verweis auf solch ein Bemaßungsobjekt erstellen, indem Sie mittels der Funktion *AddDimAligned* bzw. *AddDimRotated* ein neues Bemaßungsobjekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Bemaßungsobjekt über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

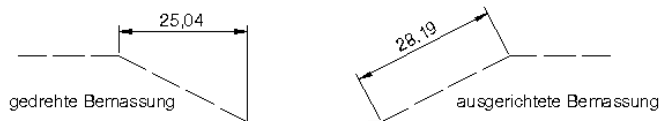


Abbildung 8.40: Beispiel linearer Bemaßungen

Beide Bemaßungsobjekte enthalten so ziemlich alle Eigenschaften, die eine Bemaßung überhaupt anbieten kann. Dadurch wird die Steuerung dieser Objekte recht komplex. Der Unterschied zwischen beiden Objekten besteht zunächst einmal in der Verwaltung.

Hilfslinieneigenschaften	
<i>ExtensionLineColor</i>	Bestimmt die Farbe der Hilfslinien.
<i>ExtensionLineExtend</i>	Bestimmt den aktuellen Wert zur Verlängerung der Hilfslinie über die Maßlinie hinaus.
<i>ExtensionLineOffset</i>	Bestimmt die Lücke zwischen Hilfslinien und Ursprungspunkten.
<i>ExtensionLineWeight</i>	Diese Eigenschaft legt die Linienstärke für die Hilfslinien des Bemaßungsobjekts fest oder ruft sie ab.
<i>ExtLine1Point</i>	Ursprungspunkt der ersten Hilfslinie.
<i>ExtLine1Suppress</i>	Steuert die Unterdrückung der ersten Hilfslinie.
<i>ExtLine2Point</i>	Ursprungspunkt der zweiten Hilfslinie.
<i>ExtLine2Suppress</i>	Steuert die Unterdrückung der zweiten Hilfslinie.
<i>Fit</i>	Bestimmt, welche Elemente verschoben werden müssen, damit Text und Pfeile zwischen die Hilfslinien passen.
Pfeilspitzeneigenschaften	
<i>Arrowhead1Block</i>	Blockname für den ersten benutzerdefinierten Pfeil.
<i>Arrowhead1Type</i>	Bestimmt den Typ der ersten Pfeilspitze.
<i>Arrowhead2Block</i>	Blockname für den zweiten benutzerdefinierten Pfeil.
<i>Arrowhead2Type</i>	Bestimmt den Typ der zweiten Pfeilspitze.
<i>ArrowheadSize</i>	Bestimmt die Größe der Pfeilspitzen.

Tabelle 8.36: Eigenschaften der linearen Bemaßung

Maßlinieneigenschaften	
<i>DimensionLineColor</i>	Bestimmt die Farbe der Maßlinie.
<i>DimensionLineExtend</i>	Legt den Wert zur Verlängerung der Maßlinie über die Hilfslinie hinaus fest.
<i>DimensionLineWeight</i>	Bestimmt die Linienstärke der Maßlinien.
<i>DimLine1Suppress</i>	Steuert die Unterdrückung der ersten Maßlinie.
<i>DimLine2Suppress</i>	Steuert die Unterdrückung der zweiten Maßlinie.
<i>DimLineInside</i>	Bestimmt die Darstellung der Maßlinien außerhalb der Hilfslinien.
<i>ForceLineInside</i>	Mit dieser Eigenschaft können Sie die Darstellung einer Maßlinie zwischen Hilfslinien ein- (<i>TRUE</i>) bzw. ausschalten (<i>FALSE</i>), selbst wenn der Text außerhalb platziert wird.
Texteigenschaften	
<i>DecimalSeparator</i>	Dezimaltrennzeichen für metrische Bemaßungen.
<i>SuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanulln.
<i>SuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanulln.
<i>SuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Bemaßungseinheiten.
<i>SuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Bemaßungseinheiten.
<i>TextColor</i>	Farbe der Maßtextes.
<i>TextGap</i>	Größe der Lücke in der Maßlinie links und rechts vom Maßtext.
<i>TextHeight</i>	Höhe des Bemaßungstextes.
<i>TextInside</i>	Erzwingt die Positionierung der Maßtexte innerhalb der Hilfslinien.
<i>TextInsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes innerhalb der Hilfslinien.
<i>TextMovement</i>	Textposition beim manuellen oder automatischen Verschieben.
<i>TextOutsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes außerhalb der Hilfslinien.
<i>TextOverride</i>	Ersetzt den gemessenen Text des Bemaßungsobjekts durch einen frei definierbaren Text.
<i>TextPosition</i>	Textpositionspunkt.
<i>TextPrefix</i>	Textpräfix für das Bemaßungsobjekt.
<i>TextRotation</i>	Aktueller Drehwinkel des Bemaßungstextes.
<i>TextStyle</i>	Textstil des Bemaßungstextes.

Tabelle 8.36: Eigenschaften der linearen Bemaßung (Forts.)

Texteigenschaften	
<i>TextSuffix</i>	Textsuffix für das Bemaßungsobjekt.
<i>VerticalTextPosition</i>	Vertikale Position des Bemaßungstextes.
<i>HorizontalTextPosition</i>	Horizontale Position des Bemaßungstextes.
Toleranzeigenschaften	
<i>ToleranceDisplay</i>	Steuert den Anzeigemodus von Toleranzwerten.
<i>ToleranceHeightScale</i>	Skalierfaktor für die Texthöhe von Toleranzen relativ zur Höhe des Bemaßungstextes.
<i>ToleranceJustification</i>	Vertikale Ausrichtung für Toleranzwerte.
<i>ToleranceLowerLimit</i>	Unterer Toleranzwert.
<i>TolerancePrecision</i>	Anzahl der Dezimalstellen für die Toleranzwerte.
<i>ToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen der Toleranzen.
<i>ToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen der Toleranzen.
<i>ToleranceSuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Toleranzeinheiten.
<i>ToleranceSuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Toleranzeinheiten.
<i>ToleranceUpperLimit</i>	Oberer Toleranzwert.
Eigenschaften für alternative Einheiten	
<i>AltRoundDistance</i>	Bestimmt den Rundungswert für Alternativeinheiten.
<i>AltSuppressLeadingZeros</i>	Bestimmt die Unterdrückung von Vorkommanullen für Alternativeinheiten.
<i>AltSuppressTrailingZeros</i>	Bestimmt die Unterdrückung von Nachkommanullen für Alternativeinheiten.
<i>AltSuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Alternativeinheiten.
<i>AltSuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Alternativeinheiten.
<i>AltTextPrefix</i>	Textpräfix der Alternativeinheiten.
<i>AltTextSuffix</i>	Textsuffix der Alternativeinheiten.
<i>AltTolerancePrecision</i>	Bestimmt die Anzahl der Dezimalstellen für die Toleranzwerte der Alternativeinheiten.
<i>AltToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen der alternativen Toleranzwerte.

Tabelle 8.36: Eigenschaften der linearen Bemaßung (Forts.)

Eigenschaften für alternative Einheiten	
<i>AltToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommastellen der alternativen Toleranzwerte.
<i>AltToleranceSuppressZeroFeet</i>	Bestimmt die Unterdrückung von 0 Fuß für Toleranzwerte der Alternativeinheiten.
<i>AltToleranceSuppressZeroInches</i>	Bestimmt die Unterdrückung von 0 Zoll für Toleranzwerte der Alternativeinheiten.
<i>AltUnits</i>	Bestimmt, ob die Bemaßung mit Alternativeinheiten ein- bzw. ausgeschaltet wird.
<i>AltUnitsFormat</i>	Bestimmt das Einheitenformat der Alternativeinheiten.
<i>AltUnitsPrecision</i>	Bestimmt die Anzahl der Dezimalstellen für die Alternativeinheiten.
<i>AltUnitsScale</i>	Bestimmt den Skalierfaktor für Alternativeinheiten.

Allgemeine Eigenschaften	
Alle Eigenschaften der Klasse <i>AcadEntity</i>	
<i>FractionFormat</i>	Bestimmt den Bruchtyp.
<i>Measurement</i> (schreibgeschützt)	Diese Eigenschaft enthält den gemessenen Wert des Bemaßungsobjekts.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>PrimaryUnitsPrecision</i>	Bestimmt die Genauigkeit der Primäreinheiten.
<i>Rotation</i>	Aktueller Drehwinkel des Bemaßungsobjekts.
<i>RoundDistance</i>	Bestimmt den Rundungswert.
<i>ScaleFactor</i>	Übergeordneter Skalierfaktor für Größen- oder Abstandseigenschaften.
<i>StyleName</i>	Name des zugeordneten Bemaßungsstils.
<i>UnitsFormat</i>	Bestimmt die Einheit für das Bemaßungsobjekt.

Tabelle 8.36: Eigenschaften der linearen Bemaßung (Forts.)

Prozeduren
Alle Prozeduren der Klasse <i>AcadEntity</i>

Durchmesserbemaßung (*AcadDimDiametric*)

Die Bemaßung eines Durchmessers wird in AutoCAD durch die Klasse *AcadDimDiametric* dargestellt. Dieser Bemaßungstyp entspricht der Bemaßung, die der Benutzer über den AutoCAD-Befehl `bemdurchm` erstellt hat.

Sie können einen Verweis auf solch ein Bemaßungsobjekt erstellen, indem Sie mit der Funktion *AddDimDiametric* ein neues Bemaßungsobjekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Durchmesser-Bemaßungsobjekt

über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) abrufen. Nachdem Sie die Objektvariable initialisiert haben, stehen Ihnen wieder eine Reihe von Eigenschaften und Methoden zur Verarbeitung des Bemaßungsobjekts zur Verfügung.

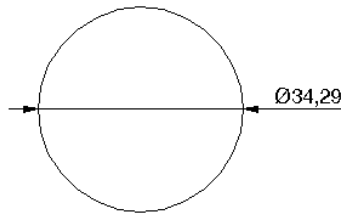


Abbildung 8.41: Durchmesserbemaßung

Zentrumsmarken festlegen

Normalerweise wird die Durchmesserbemaßung innerhalb des bemaßten Kreises bzw. Bogens durch eine Bemaßungslinie angezeigt. Wenn Sie allerdings den Bemaßungstext außerhalb des Kreises positionieren und die Bemaßungslinie deaktivieren, werden die Einstellungen zusätzlicher Eigenschaften berücksichtigt, mit der die Darstellung einer Zentrumsmarke gesteuert werden können.

CenterType bestimmt den Zentrumsmarkentyp dieses Bemaßungsobjekts. Zu unterscheiden sind drei Einstellungsmöglichkeiten, die durch folgende Konstanten vorgenommen werden können:

Konstante	Beschreibung	Wert
<i>acCenterMark</i>	Zentrumsmarke	 0
<i>AcCenterLine</i>	Zentrumslinien	 1
<i>AcCenterNone</i>	Keine Zentrumsmarke	2

Tabelle 8.37: Konstanten der *CenterType*-Eigenschaft

Mit *CenterMarkSize* bestimmen Sie die Größe dieser Zentrumsmarke. Diese Eigenschaft wird allerdings nicht berücksichtigt, wenn die Eigenschaft *CenterType* auf den Wert *acCenterNone* gesetzt wurde.

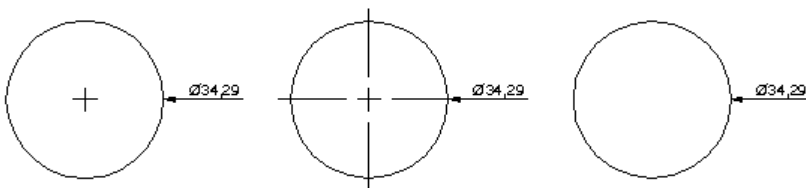


Abbildung 8.42: Darstellung der unterschiedlichen Zentrumsmarken

Tipp

Obwohl die Klasse *AcadDimDiametric* die Eigenschaft *LeaderLength* besitzt, können Sie den Wert dieser Eigenschaft nur bestimmen, wenn Sie die Durchmesserbemaßung mit der Anweisung *AddDimDiametric* der Zeichnung hinzufügen. Wenn das Objekt erst einmal erstellt wurde, kann dieser Wert weder abgerufen noch geändert werden.

Hilfslinieneigenschaften	
<i>Fit</i>	Bestimmt, welche Elemente verschoben werden müssen, damit Text und Pfeile zwischen die Hilfslinien passen.
Pfeilspitzeneigenschaften	
<i>Arrowhead1Block</i>	Blockname für den ersten benutzerdefinierten Pfeil.
<i>Arrowhead1Type</i>	Bestimmt den Typ der ersten Pfeilspitze.
<i>Arrowhead2Block</i>	Blockname für den zweiten benutzerdefinierten Pfeil.
<i>Arrowhead2Type</i>	Bestimmt den Typ der zweiten Pfeilspitze.
<i>ArrowheadSize</i>	Bestimmt die Größe der Pfeilspitzen.
Maßlinieneigenschaften	
<i>DimensionLineColor</i>	Bestimmt die Farbe der Maßlinie.
<i>DimensionLineWeight</i>	Bestimmt die Linienstärke der Maßlinien.
<i>DimLine1Suppress</i>	Steuert die Unterdrückung der ersten Maßlinie.
<i>DimLine2Suppress</i>	Steuert die Unterdrückung der zweiten Maßlinie.
<i>ForceLineInside</i>	Mit dieser Eigenschaft können Sie die Darstellung einer Maßlinie zwischen Hilfslinien ein- (<i>TRUE</i>) bzw. ausschalten (<i>FALSE</i>), selbst wenn der Text außerhalb platziert wird.
Texteigenschaften	
<i>DecimalSeparator</i>	Dezimaltrennzeichen für metrische Bemaßungen.
<i>SuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen.
<i>SuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen.
<i>SuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Bemaßungseinheiten.
<i>SuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Bemaßungseinheiten.
<i>TextColor</i>	Farbe der Maßtextes.
<i>TextGap</i>	Größe der Lücke in der Maßlinie links und rechts vom Maßtext.
<i>TextHeight</i>	Höhe des Bemaßungstextes.
<i>TextInside</i>	Erzwingt die Positionierung der Maßtexte innerhalb der Hilfslinien.

Tabelle 8.38: Eigenschaften der Durchmesserbemaßung

Texteigenschaften	
<i>TextInsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes innerhalb der Hilfslinien.
<i>TextMovement</i>	Textposition beim manuellen oder automatischen Verschieben.
<i>TextOutsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes außerhalb der Hilfslinien.
<i>TextOverride</i>	Ersetzt den gemessenen den Text des Bemaßungsobjekts durch einen frei definierbaren Text.
<i>TextPosition</i>	Textpositionspunkt.
<i>TextPrefix</i>	Textpräfix für das Bemaßungsobjekt.
<i>TextRotation</i>	Aktueller Drehwinkel des Bemaßungstextes.
<i>TextStyle</i>	Textstil des Bemaßungstextes.
<i>TextSuffix</i>	Textsuffix für das Bemaßungsobjekt.
<i>VerticalTextPosition</i>	Vertikale Position des Bemaßungstextes.
Toleranzeigenschaften	
<i>ToleranceDisplay</i>	Steuert den Anzeigemodus von Toleranzwerten.
<i>ToleranceHeightScale</i>	Skalierfaktor für die Texthöhe von Toleranzen relativ zur Höhe des Bemaßungstextes.
<i>ToleranceJustification</i>	Vertikale Ausrichtung für Toleranzwerte.
<i>ToleranceLowerLimit</i>	Unterer Toleranzwert.
<i>TolerancePrecision</i>	Anzahl der Dezimalstellen für die Toleranzwerte.
<i>ToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanulln der Toleranzen.
<i>ToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanulln der Toleranzen.
<i>ToleranceSuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Toleranzeinheiten.
<i>ToleranceSuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Toleranzeinheiten.
<i>ToleranceUpperLimit</i>	Oberer Toleranzwert.
Eigenschaften für Alternative Einheiten	
<i>AltRoundDistance</i>	Bestimmt den Rundungswert für Alternativeinheiten.
<i>AltSuppressLeadingZeros</i>	Bestimmt die Unterdrückung von Vorkommanulln für Alternativeinheiten.
<i>AltSuppressTrailingZeros</i>	Bestimmt die Unterdrückung von Nachkommanulln für Alternativeinheiten.
<i>AltSuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Alternativeinheiten.

Tabelle 8.38: Eigenschaften der Durchmesserbemaßung (Forts.)

Eigenschaften für Alternative Einheiten	
<i>AltSuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Alternativeinheiten.
<i>AltTextPrefix</i>	Textpräfix der Alternativeinheiten.
<i>AltTextSuffix</i>	Textsuffix der Alternativeinheiten.
<i>AltTolerancePrecision</i>	Bestimmt die Anzahl der Dezimalstellen für die Toleranzwerte der Alternativeinheiten.
<i>AltToleranceSuppressLeading-Zeros</i>	Steuert die Unterdrückung der Vorkommanulln der alternativen Toleranzwerte.
<i>AltToleranceSuppressTrailing-Zeros</i>	Steuert die Unterdrückung der Nachkommanulln der alternativen Toleranzwerte.
<i>AltToleranceSuppressZeroFeet</i>	Bestimmt die Unterdrückung von 0 Fuß für Toleranzwerte der Alternativeinheiten.
<i>AltToleranceSuppressZeroInches</i>	Bestimmt die Unterdrückung von 0 Zoll für Toleranzwerte der Alternativeinheiten.
<i>AltUnits</i>	Bestimmt, ob die Bemaßung mit Alternativeinheiten ein- bzw. ausgeschaltet wird.
<i>AltUnitsFormat</i>	Bestimmt das Einheitenformat der Alternativeinheiten.
<i>AltUnitsPrecision</i>	Bestimmt die Anzahl der Dezimalstellen für die Alternativeinheiten.
<i>AltUnitsScale</i>	Bestimmt den Skalierfaktor für Alternativeinheiten.
Allgemeine Eigenschaften	
Alle Eigenschaften der Klasse <i>AcadEntity</i>	
<i>CenterMarkSize</i>	Bestimmt die Größe der Zentrumsmarke.
<i>CenterType</i>	Legt den Zentrumsmarkentyp fest.
<i>FractionFormat</i>	Bestimmt den Bruchtyp.
<i>LeaderLength</i>	Bestimmt die Länge der Führungslinie.
<i>LinearScaleFactor</i>	Bestimmt einen globalen Skalierfaktor für die Bemaßung.
<i>Measurement</i> (schreibgeschützt)	Diese Eigenschaft enthält den gemessenen Wert des Bemaßungsobjekts.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>PrimaryUnitsPrecision</i>	Bestimmt die Genauigkeit der Primäreinheiten.
<i>Rotation</i>	Aktueller Drehwinkel des Bemaßungsobjekts.
<i>RoundDistance</i>	Bestimmt den Rundungswert.
<i>ScaleFactor</i>	Übergeordneter Skalierfaktor für Größen- oder Abstandseigenschaften.
<i>StyleName</i>	Name des zugeordneten Bemaßungsstils.
<i>UnitsFormat</i>	Bestimmt die Einheit für das Bemaßungsobjekt.

Tabelle 8.38: Eigenschaften der Durchmesserbemaßung (Forts.)

Prozeduren

Alle Prozeduren der Klasse *AcadEntity*

Radiusbemaßung (*AcadDimRadial*)

Ein Radiusbemaßungsobjekt wird durch die Klasse *AcadDimRadial* in AutoCAD dargestellt. Dieser Bemaßungstyp entspricht der Bemaßung, die der Benutzer über den AutoCAD-Befehl *bemradius* erstellt hat.

Sie können einen Verweis auf solch ein Bemaßungsobjekt erstellen, indem Sie mit der Funktion *AddDimRadial* ein neues Bemaßungsobjekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Radial-Bemaßungsobjekt über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

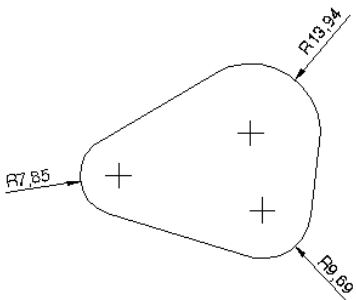


Abbildung 8.43: Beispiel einer Radiusbemaßung

Dieses Objekt besitzt ungefähr dieselben Eigenschaften wie die Durchmesserbemaßung. Außer dass mit dem einen der Durchmesser und mit dem anderen der Radius eines Kreises bzw. Bogens gemessen und dargestellt wird, besteht der Unterschied zwischen beiden Objekten darin, dass die Radiusbemaßung nur eine Pfeilspitze anzeigen kann und damit auch nur über die Eigenschaften *Arrowhead1Block* und *Arrowhead1Type* zur Definition dieser Pfeilspitze verfügt. Außerdem ist auch bei diesem Objekt die Eigenschaft *LeaderLength* zwar in der Klasse selbst definiert, kann aber nur bei der Erzeugung des Objekts mit der *AddDimRadial* vergeben werden.

Hilfslinieneigenschaften

<i>Fit</i>	Bestimmt, welche Elemente verschoben werden müssen, damit Text und Pfeile zwischen die Hilfslinien passen.

Pfeilspitzeneigenschaften

<i>ArrowheadBlock</i>	Blockname für den ersten benutzerdefinierten Pfeil.
<i>ArrowheadSize</i>	Bestimmt die Größe der Pfeilspitzen.
<i>ArrowheadType</i>	Bestimmt den Typ der ersten Pfeilspitze.

Tabelle 8.39: Eigenschaften der Durchmesserbemaßung

Maßlinieneigenschaften	
<i>DimensionLineColor</i>	Bestimmt die Farbe der Maßlinie.
<i>DimensionLineWeight</i>	Bestimmt die Linienstärke der Maßlinien.
<i>DimLineSuppress</i>	Steuert die Unterdrückung der ersten Maßlinie.
<i>ForceLineInside</i>	Mit dieser Eigenschaft können Sie die Darstellung einer Maßlinie zwischen Hilfslinien ein- (<i>TRUE</i>) bzw. ausschalten (<i>FALSE</i>), selbst wenn der Text außerhalb platziert wird.
Texteigenschaften	
<i>DecimalSeparator</i>	Dezimaltrennzeichen für metrische Bemaßungen.
<i>SuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen.
<i>SuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen.
<i>SuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Bemaßungseinheiten.
<i>SuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Bemaßungseinheiten.
<i>TextColor</i>	Farbe der Maßtextes.
<i>TextGap</i>	Größe der Lücke in der Maßlinie links und rechts vom Maßtext.
<i>TextHeight</i>	Höhe des Bemaßungstextes.
<i>TextInside</i>	Erzwingt die Positionierung der Maßtexte innerhalb der Hilfslinien.
<i>TextInsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes innerhalb der Hilfslinien.
<i>TextMovement</i>	Textposition beim manuellen oder automatischen Verschieben.
<i>TextOutsideAlign</i>	Bestimmt die Ausrichtung des Maßtextes außerhalb der Hilfslinien.
<i>TextOverride</i>	Ersetzt den gemessenen Text des Bemaßungsobjekts durch einen frei definierbaren Text.
<i>TextPosition</i>	Textpositionspunkt.
<i>TextPrefix</i>	Textpräfix für das Bemaßungsobjekt.
<i>TextRotation</i>	Aktueller Drehwinkel des Bemaßungstextes.
<i>TextStyle</i>	Textstil des Bemaßungstextes.
<i>TextSuffix</i>	Textsuffix für das Bemaßungsobjekt.
<i>VerticalTextPosition</i>	Vertikale Position des Bemaßungstextes.

Tabelle 8.39: Eigenschaften der Durchmesserbemaßung (Forts.)

Toleranzeigenschaften	
<i>ToleranceDisplay</i>	Steuert den Anzeigemodus von Toleranzwerten.
<i>ToleranceHeightScale</i>	Skalierfaktor für die Texthöhe von Toleranzen relativ zur Höhe des Bemaßungstextes.
<i>ToleranceJustification</i>	Vertikale Ausrichtung für Toleranzwerte.
<i>ToleranceLowerLimit</i>	Unterer Toleranzwert.
<i>TolerancePrecision</i>	Anzahl der Dezimalstellen für die Toleranzwerte.
<i>ToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanulln der Toleranzen.
<i>ToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanulln der Toleranzen.
<i>ToleranceSuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Toleranzeinheiten.
<i>ToleranceSuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Toleranzeinheiten.
<i>ToleranceUpperLimit</i>	Oberer Toleranzwert.
Eigenschaften für Alternative Einheiten	
<i>AltRoundDistance</i>	Bestimmt den Rundungswert für Alternativeinheiten.
<i>AltSuppressLeadingZeros</i>	Bestimmt die Unterdrückung von Vorkommanulln für Alternativeinheiten.
<i>AltSuppressTrailingZeros</i>	Bestimmt die Unterdrückung von Nachkommanulln für Alternativeinheiten.
<i>AltSuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Alternativeinheiten.
<i>AltSuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Alternativeinheiten.
<i>AltTextPrefix</i>	Textpräfix der Alternativeinheiten.
<i>AltTextSuffix</i>	Textsuffix der Alternativeinheiten.
<i>AltTolerancePrecision</i>	Bestimmt die Anzahl der Dezimalstellen für die Toleranzwerte der Alternativeinheiten.
<i>AltToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanulln der alternativen Toleranzwerte.
<i>AltToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanulln der alternativen Toleranzwerte.
<i>AltToleranceSuppressZeroFeet</i>	Bestimmt die Unterdrückung von 0 Fuß für Toleranzwerte der Alternativeinheiten.
<i>AltToleranceSuppressZeroInches</i>	Bestimmt die Unterdrückung von 0 Zoll für Toleranzwerte der Alternativeinheiten.
<i>AltUnits</i>	Bestimmt, ob die Bemaßung mit Alternativeinheiten ein- bzw. ausgeschaltet wird.

Tabelle 8.39: Eigenschaften der Durchmesserbemaßung (Forts.)

Eigenschaften für Alternative Einheiten	
<i>AltUnitsFormat</i>	Bestimmt das Einheitenformat der Alternativeinheiten.
<i>AltUnitsPrecision</i>	Bestimmt die Anzahl der Dezimalstellen für die Alternativ-einheiten.
<i>AltUnitsScale</i>	Bestimmt den Skalierfaktor für Alternativeinheiten.
Allgemeine Eigenschaften	
Alle Eigenschaften der Klasse <i>AcadEntity</i>	
<i>CenterMarkSize</i>	Bestimmt die Größe der Zentrumsmarke.
<i>CenterType</i>	Legt den Zentrumsmarkentyp fest.
<i>FractionFormat</i>	Bestimmt den Bruchtyp.
<i>LeaderLength</i>	Bestimmt die Länge der Führungslinie.
<i>LinearScaleFactor</i>	Legt den globalen Skalierfaktor für die Bemaßung fest.
<i>Measurement</i> (schreibgeschützt)	Diese Eigenschaft enthält den gemessenen Wert des Bema-ßungsobjekts.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordi-naten des normalen Richtungsvektors abrufen oder festle-gen.
<i>Rotation</i>	Aktueller Drehwinkel des Bemaßungsobjekts.
<i>ScaleFactor</i>	Übergeordneter Skalierfaktor für Größen- oder Abstandsei-genschaften.
<i>StyleName</i>	Name des zugeordneten Bemaßungsstils.
<i>UnitsFormat</i>	Bestimmt die Einheit für das Bemaßungsobjekt.

Tabelle 8.39: Eigenschaften der Durchmesserbemaßung (Forts.)

Prozeduren
Alle Prozeduren der Klasse <i>AcadEntity</i>

Koordinatenbemaßung (AcadDimOrdinate)

Die Klasse *AcadDimOrdinate* stellt ein Koordinatenbemaßungsobjekt in AutoCAD dar. Dieser Bemaßungstyp entspricht der Bemaßung, die der Benutzer über den AutoCAD-Befehl *bemord* erstellt hat.

Sie können einen Verweis auf solch ein Bemaßungsobjekt erstellen, indem Sie mit der Funk-tion *AddDimOrdinate* ein neues Bemaßungsobjekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Koordinatenbemaßungsobjekt über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

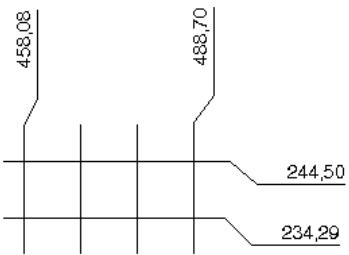


Abbildung 8.44: Koordinatenbemaßung

Hilfslinieneigenschaften	
<i>ExtensionLineColor</i>	Bestimmt die Farbe der Hilfslinien.
<i>ExtensionLineOffset</i>	Bestimmt die Lücke zwischen Hilfslinien und Ursprungspunkten.
<i>ExtensionLineWeight</i>	Diese Eigenschaft legt die Linienstärke für die Hilfslinien des Bemaßungsobjekts fest oder ruft sie ab.
Pfeilspitzeigenschaften	
<i>ArrowheadSize</i>	Bestimmt die Größe der Pfeilspitzen.
Texteigenschaften	
<i>DecimalSeparator</i>	Dezimaltrennzeichen für metrische Bemaßungen.
<i>SuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen.
<i>SuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen.
<i>SuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Bemaßungseinheiten.
<i>SuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Bemaßungseinheiten.
<i>TextColor</i>	Farbe der Maßtextes.
<i>TextGap</i>	Größe der Lücke in der Maßlinie links und rechts vom Maßtext.
<i>TextHeight</i>	Höhe des Bemaßungstextes.
<i>TextMovement</i>	Textposition beim manuellen oder automatischen Verschieben.
<i>TextOverride</i>	Ersetzt den gemessenen Text des Bemaßungsobjekts durch einen frei definierbaren Text.
<i>TextPosition</i>	Textpositionspunkt.
<i>TextPrefix</i>	Textpräfix für das Bemaßungsobjekt.
<i>TextRotation</i>	Aktueller Drehwinkel des Bemaßungstextes.

Tabelle 8.40: Eigenschaften der Radiusbemaßung

Texteigenschaften	
<i>TextStyle</i>	Textstil des Bemaßungstextes.
<i>TextSuffix</i>	Textsuffix für das Bemaßungsobjekt.
<i>VerticalTextPosition</i>	Vertikale Position des Bemaßungstextes.
Toleranzeigenschaften	
<i>ToleranceDisplay</i>	Steuert den Anzeigemodus von Toleranzwerten.
<i>ToleranceHeightScale</i>	Skalierfaktor für die Texthöhe von Toleranzen relativ zur Höhe des Bemaßungstextes.
<i>ToleranceJustification</i>	Vertikale Ausrichtung für Toleranzwerte.
<i>ToleranceLowerLimit</i>	Unterer Toleranzwert.
<i>TolerancePrecision</i>	Anzahl der Dezimalstellen für die Toleranzwerte.
<i>ToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen der Toleranzen.
<i>ToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen der Toleranzen.
<i>ToleranceSuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Toleranzeinheiten.
<i>ToleranceSuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Toleranzeinheiten.
<i>ToleranceUpperLimit</i>	Oberer Toleranzwert.
Eigenschaften für Alternative Einheiten	
<i>AltRoundDistance</i>	Bestimmt den Rundungswert für Alternativeinheiten.
<i>AltSuppressLeadingZeros</i>	Bestimmt die Unterdrückung von Vorkommanullen für Alternativeinheiten.
<i>AltSuppressTrailingZeros</i>	Bestimmt die Unterdrückung von Nachkommanullen für Alternativeinheiten.
<i>AltSuppressZeroFeet</i>	Steuert die Unterdrückung von 0 Fuß der Alternativeinheiten.
<i>AltSuppressZeroInches</i>	Steuert die Unterdrückung von 0 Zoll der Alternativeinheiten.
<i>AltTextPrefix</i>	Textpräfix der Alternativeinheiten.
<i>AltTextSuffix</i>	Textsuffix der Alternativeinheiten.
<i>AltTolerancePrecision</i>	Bestimmt die Anzahl der Dezimalstellen für die Toleranzwerte der Alternativeinheiten.
<i>AltToleranceSuppressLeadingZeros</i>	Steuert die Unterdrückung der Vorkommanullen der alternativen Toleranzwerte.
<i>AltToleranceSuppressTrailingZeros</i>	Steuert die Unterdrückung der Nachkommanullen der alternativen Toleranzwerte.

Tabelle 8.40: Eigenschaften der Radiusbemaßung (Forts.)

Eigenschaften für Alternative Einheiten	
<i>AltToleranceSuppressZeroFeet</i>	Bestimmt die Unterdrückung von 0 Fuß für Toleranzwerte der Alternativeinheiten.
<i>AltToleranceSuppressZeroInches</i>	Bestimmt die Unterdrückung von 0 Zoll für Toleranzwerte der Alternativeinheiten.
<i>AltUnits</i>	Bestimmt, ob die Bemaßung mit Alternativeinheiten ein- bzw. ausgeschaltet wird.
<i>AltUnitsFormat</i>	Bestimmt das Einheitenformat der Alternativeinheiten.
<i>AltUnitsPrecision</i>	Bestimmt die Anzahl der Dezimalstellen für die Alternativeinheiten.
<i>AltUnitsScale</i>	Bestimmt den Skalierfaktor für Alternativeinheiten.

Allgemeine Eigenschaften	
Alle Eigenschaften der Klasse <i>AcadEntity</i>	
<i>FractionFormat</i>	Bestimmt den Bruchtyp.
<i>Measurement</i> (schreibgeschützt)	Diese Eigenschaft enthält den gemessenen Wert des Bemaßungsobjekts.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>Rotation</i>	Aktueller Drehwinkel des Bemaßungsobjekts.
<i>ScaleFactor</i>	Übergeordneter Skalierfaktor für Größen- oder Abstandseigenschaften.
<i>StyleName</i>	Name des zugeordneten Bemaßungsstils.
<i>UnitsFormat</i>	Bestimmt die Einheit für das Bemaßungsobjekt.

Tabelle 8.40: Eigenschaften der Radiusbemaßung (Forts.)

Prozeduren
Alle Prozeduren der Klasse <i>AcadEntity</i>

Anwendungsbeispiele zu Bemaßungsobjekten

(Beispielprojekt 08\AcadDimension.dvb)

Im Beispielprojekt zu diesem Thema werden wir die einzelnen Segmente einer Polylinie mit linearen Bemaßungsobjekten beschriften. Zunächst muss der Benutzer eine Polylinie in der Zeichnung wählen. Dabei ist es egal, ob es sich um eine 3D-, eine 2D- oder optimierte Polylinie handelt.

```
Public Function DimPolylineLinear()  
On Error Resume Next  
Set Util = New MyUtility  
Do  
    Set Object = Nothing  
    Util.GetEntity Object, PickedPoint, "Polylinie wählen:"
```

Hat er dies erfolgreich ausgeführt, prüfen wir die Anzahl der Stützpunkte des gewählten Objekts mit der *GetVertexCount*-Funktion der Klasse *MyUtility* (siehe Kapitel A.3). Handelt es sich um eine Polylinie und verfügt diese über mindestens zwei Stützpunkte, kann mit der Erzeugung der Bemaßungsobjekte fortgefahren werden.

```
If Util.GetVertexCount(Object) > 2 Then
```

Jetzt werden alle Stützpunkte nacheinander in einer *For...Next*-Schleife abgerufen. Da wir für eine lineare Bemaßung immer einen Anfangs- und einen Endpunkt benötigen, wird als erster Punkt immer der Stützpunkt verwendet, der dem Wert des Zählers der Schleife entspricht, und als zweiter Punkt der jeweils folgende ($i + 1$). Aus diesem Grund läuft die Schleife auch nicht bis zum letzten, sondern nur bis zum vorletzten Stützpunkt der Polylinien.

```
For i = 0 To Util.GetVertexCount(Object) - 2
    FirstPoint = Object.Coordinate(i)
    SecondPoint = Object.Coordinate(i + 1)
```

Bevor nun das Bemaßungsobjekt endgültig erzeugt wird, berechnen wir noch die Position des Bemaßungstextes, da die Funktion *AddDimAligned* diese als weiteren Parameter erwartet.

```
    TextPosition = Util.PolarPoint(FirstPoint, _
        Util.AngleFromXAxis(FirstPoint, SecondPoint), _
        Util.CalcDistance(FirstPoint, SecondPoint) / 2)
```

Nachdem nun alle Parameter bestimmt wurden, kann das Bemaßungsobjekt erzeugt werden. Dies geschieht entweder im Modell- oder im Papierbereich, je nachdem, in welchem sich die Zeichnung gerade befindet.

```
    If ActiveSpace = acModelSpace Then
        Set NewDim = ModelSpace.AddDimAligned(FirstPoint, _
            SecondPoint, TextPosition)
    Else
        Set NewDim = PaperSpace.AddDimAligned(FirstPoint, _
            SecondPoint, TextPosition)
    End If
```

Jetzt müssen nur noch die Eigenschaften des neuen Bemaßungsobjekts entsprechend der Vorgaben des Benutzers, die er im Formular vorgenommen hat, angepasst werden. Dieser Vorgang wird in der Funktion *SetAlignedProperties* ausgeführt, weshalb an dieser Stelle nur der Objektverweis (*NewDim*) und die Koordinaten des Start- und Endpunkts an diese Funktion übergeben werden müssen.

```
    SetAlignedProperties NewDim, FirstPoint, SecondPoint
Next
```

Bleibt zum Schluss nur noch die Beschriftung des letzten Teilstücks (falls vorhanden). Sollte die Polylinie nämlich geschlossen sein, muss auch der Abschnitt vom letzten zum ersten Stützpunkt bemaßt werden.

```
If Object.Closed = True Then
    FirstPoint = Object.Coordinate(Util.GetVertexCount( _
        Object) - 1)
    SecondPoint = Object.Coordinate(0)
    TextPosition = Util.PolarPoint(FirstPoint, _
        Util.AngleFromXAxis(FirstPoint, SecondPoint), _
        Util.CalcDistance(FirstPoint, SecondPoint) / 2)
```

```

If ActiveSpace = acModelSpace Then
    Set NewDim = ModelSpace.AddDimAligned(FirstPoint, _
        SecondPoint, TextPosition)
Else
    Set NewDim = PaperSpace.AddDimAligned(FirstPoint, _
        SecondPoint, TextPosition)
End If
SetAlignedProperties NewDim, FirstPoint, SecondPoint
End If
ElseIf TypeName(Object) = "Nothing" Then
    Exit Do
End If
Loop
End Function

```

In der folgenden Prozedur werden die Eigenschaften des Bemaßungsobjekts mit den Einstellungen, die der Benutzer im Formular vornehmen kann, angepasst. Im Normalfall kann der Inhalt der Steuerelemente direkt an die entsprechende Eigenschaft weitergegeben werden, wie beispielsweise beim Bemaßungspräfix bzw. -suffix oder der horizontalen und vertikalen Textposition.

Abbildung 8.45: Einstellungsformular des Beispielprojekts

Eine Ausnahme bilden hier die Kontrollkästchen im Rahmen OPTIONEN. Mit dem ersten Kontrollkästchen (ANZEIGE DER BEMASSUNGSLINIEN UNTERDRÜCKEN) werden beide Bemaßungslinien angesprochen. Genauso verhält es sich mit dem zweiten Kontrollkästchen (ANZEIGE DER HILFSLINIEN UNTERDRÜCKEN). Damit wird die Anzeige beider Hilfslinien gesteuert.

```

Public Sub SetAlignedProperties(DimAligned As AcadDimAligned, _
    Optional Point1, Optional Point2)
    On Error Resume Next
    With AlignedPropFrm

```

Durch Aktivierung der Option RÄUMLICHE ENTFERNUNG BESCHRIFTEN soll das Bemaßungsobjekt den räumlichen Abstand der einzelnen Polylinienabschnitte darstellen (macht natürlich nur bei 3D-Polylinien Sinn). Da dies auf direktem Wege nicht möglich ist, müssen wir den Umweg über den linearen Bemaßungsfaktor einschlagen. Da der direkt gemessene Wert multipliziert mit diesem Faktor als Bemaßungstext angezeigt wird, brauchen wir nur das Verhältnis von räumlicher Strecke zur Lagentfernung zu bestimmen und müssen dann den ermittelten Faktor in die Eigenschaft *LinearScaleFactor* eintragen.

```
If .CheckBox3.Value = True Then
    DimAligned.LinearScaleFactor = Util.CalcDistance(Point1, _
        Point2, True) / Util.CalcDistance(Point1, Point2)
End If
```

Den restlichen Eigenschaften können wir direkt die Einstellungen der Steuerelemente aus dem Formular zuweisen.

```
DimAligned.DimLine1Suppress = .CheckBox1
DimAligned.DimLine2Suppress = .CheckBox1
DimAligned.ExtLine1Suppress = .CheckBox2
DimAligned.ExtLine2Suppress = .CheckBox2
DimAligned.HorizontalTextPosition = .ComboBox1.ListIndex
DimAligned.VerticalTextPosition = .ComboBox2.ListIndex
DimAligned.TextPrefix = .TextBox1
DimAligned.TextSuffix = .TextBox2
DimAligned.UnitsFormat = .ComboBox3.ListIndex + 1
DimAligned.PrimaryUnitsPrecision = CLng(.TextBox3)
End With
End Sub
```

Systemvariablen für Bemaßungsobjekte

Für fast jede Eigenschaft eines Bemaßungsobjekts existiert auch eine Systemvariable, mit der Sie gegebenenfalls einen Vorgabewert in der AutoCAD-Befehlszeile anzeigen oder in einem Formular einem Steuerelement zuordnen können. In der folgenden Tabelle sind diese Variablen den Eigenschaften gegenübergestellt, für die sie den Ausgangswert bei der Erzeugung des Bemaßungsobjekts bestimmen.

Eigenschaft	Variable	Eigenschaft	Variable
Hilfslinieneigenschaften			
<i>ExtensionLineColor</i>	DIMCLRE	<i>ExtLine1Suppress</i>	DIMSE1
<i>ExtensionLineExtend</i>	DIMEXE	<i>ExtLine2Suppress</i>	DIMSE2
<i>ExtensionLineOffset</i>	DIMEXO	<i>Fit</i>	DIMATFIT
<i>ExtensionLineWeight</i>	DIMLWE		
Pfeilspitzeneigenschaften			
<i>Arrowhead1Block</i>	DIMBLK1	<i>Arrowhead2Type</i>	DIMBLK2
<i>Arrowhead1Type</i>	DIMBLK1	<i>ArrowheadSize</i>	DIMASZ
<i>Arrowhead2Block</i>	DIMBLK2		
Maßlinieneigenschaften			
<i>DimensionLineColor</i>	DIMCLRD	<i>DimLine2Suppress</i>	DIMSD2
<i>DimensionLineExtend</i>	DIMDLE	<i>DimLineInside</i>	DIMSOXD
<i>DimensionLineWeight</i>	DIMLWD	<i>ForceLineInside</i>	DIMTOFL
<i>DimLine1Suppress</i>	DIMSD1		

Tabelle 8.41: Systemvariablen

Eigenschaft	Variable	Eigenschaft	Variable
Texteigenschaften			
<i>DecimalSeparator</i>	DIMDSEP	<i>TextInsideAlign</i>	DIMTIH
<i>SuppressLeadingZeros</i>	DIMZIN	<i>TextMovement</i>	DIMT-MOVE
<i>SuppressTrailingZeros</i>	DIMZIN	<i>TextOutsideAlign</i>	DIMTOH
<i>SuppressZeroFeet</i>	DIMZIN	<i>TextPrefix</i>	DIMPOST
<i>SuppressZeroInches</i>	DIMZIN	<i>TextStyle</i>	DIMTXSTY
<i>TextColor</i>	DIMCLRT	<i>TextSuffix</i>	DIMPOST
<i>TextGap</i>	DIMGAP	<i>VerticalTextPosition</i>	DIMTAD
<i>TextHeight</i>	DIMTXT	<i>HorizontalTextPosition</i>	DIMJUST
<i>TextInside</i>	DIMTIX		
Toleranzeigenschaften			
<i>ToleranceDisplay</i>	DIMTOL	<i>ToleranceSuppressLeadingZeros</i>	DIMTZIN
<i>ToleranceHeightScale</i>	DIMTFAC	<i>ToleranceSuppressTrailingZeros</i>	DIMTZIN
<i>ToleranceJustification</i>	DIMTOLJ	<i>ToleranceSuppressZeroFeet</i>	DIMTZIN
<i>ToleranceLowerLimit</i>	DIMTM	<i>ToleranceSuppressZeroInches</i>	DIMTZIN
<i>TolerancePrecision</i>	DIMTDEC	<i>ToleranceUpperLimit</i>	DIMTP
Eigenschaften für Alternative Einheiten			
<i>AltRoundDistance</i>	DIMALTRND	<i>AltToleranceSuppressLeadingZeros</i>	DIMALTTZ
<i>AltSuppressLeadingZeros</i>	DIMALTZ	<i>AltToleranceSuppressTrailingZeros</i>	DIMALTTZ
<i>AltSuppressTrailingZeros</i>	DIMALTZ	<i>AltToleranceSuppressZeroFeet</i>	DIMALTTZ
<i>AltSuppressZeroFeet</i>	DIMALTZ	<i>AltToleranceSuppressZeroInches</i>	DIMALTTZ
<i>AltSuppressZeroInches</i>	DIMALTZ	<i>AltUnits</i>	DIMALT
<i>AltTextPrefix</i>	DIMAPOST	<i>AltUnitsFormat</i>	DIMALTU
<i>AltTextSuffix</i>	DIMAPOST	<i>AltUnitsPrecision</i>	DIMALTD
<i>AltTolerancePrecision</i>	DIMTDEC	<i>AltUnitsScale</i>	DIMALTF
Allgemeine Eigenschaften			
<i>FractionFormat</i>	DIMFRAC	<i>ScaleFactor</i>	DIMSCALE
<i>PrimaryUnitsPrecision</i>	DIMDEC	<i>StyleName</i>	DIMSTYLE
<i>RoundDistance</i>	DIMRND	<i>UnitsFormat</i>	DIMLUNIT

Tabelle 8.41: Systemvariablen (Forts.)

8.3.9 Führungslinienobjekt (AcadLeader)

Die Klasse *AcadLeader* stellt in AutoCAD ein Führungslinienobjekt dar und enthält somit alle Eigenschaften und Methoden, die Sie auf ein Führungslinienobjekt anwenden können. Da eine Führungslinie einem Bemaßungsobjekt gleicht, können verschiedene Bemaßungsvariablen Auswirkungen auf die Darstellung des Leader-Objekts haben.

Sie können einen Verweis auf eine Führung erstellen, indem Sie mit der Funktion *AddLeader* ein neues Führungsobjekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene Führung über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Stützpunkte einer Führungslinie verwalten

Da ein Führungsobjekt mehrere Stützpunkte enthalten kann, werden alle Koordinaten in der Eigenschaft *Coordinates* verwaltet. Mit der *Coordinate*-Eigenschaft können Sie einen bestimmten Stützpunkt ansprechen. Der entsprechende Punkt wird dabei als 3D-Array ausgegeben. Um die Koordinaten des ersten Stützpunkts zu erhalten, müssen Sie wie folgt vorgehen:

```
Dim VarPoint As Variant
VarPoint = ActLeader.Coordinate(0)
```

Anmerkungsbjekte zuordnen

Die *Annotation*-Eigenschaft bestimmt das Anmerkungsobjekt für eine Führung. Solche Objekte können Klassen vom Typ *AcadTolerance*, *AcadMLine* oder *AcadBlockReferences* sein. Wenn die Führung kein Anmerkungsobjekt enthält, wird in diesem Fall *Nothing* zurückgegeben. Sie sollten also vor Verwendung eines Anmerkungsobjekts mit der *TypeName*-Anweisung prüfen, ob die Führung tatsächlich ein Anmerkungsobjekt enthält.

Führungsliniendarstellung anpassen

Mit dieser Eigenschaft bestimmen Sie die Art der Darstellung des Führungsobjekts. Es kann entweder als Linie mit oder ohne Pfeilspitze bzw. als Spline mit oder ohne Pfeil angezeigt werden. Die einzelnen Typen der Führung werden durch folgende Konstanten festgelegt:

Darstellung	Konstante	Beschreibung
	<i>AcLineNoArrow</i>	Führungsobjekt als Linienzug und ohne Pfeil
	<i>AcLineWithArrow</i>	Führungsobjekt als Linienzug und mit Pfeil
	<i>AcSplineNoArrow</i>	Führungsobjekt als Spline und ohne Pfeil
	<i>AcSplineWithArrow</i>	Führungsobjekt als Spline und mit Pfeil

Tabelle 8.42: Weitere Eigenschaften eines Führungslinienobjekts

Maßlinieneigenschaften	
<i>DimensionLineColor</i>	Bestimmt die Farbe der Maßlinie.
<i>DimensionLineWeight</i>	Bestimmt die Linienstärke der Maßlinien
Pfeilspitzeneigenschaften	
<i>ArrowheadBlock</i>	Blockname für den benutzerdefinierten Pfeil.
<i>ArrowheadType</i>	Bestimmt den Typ der Pfeilspitze.
<i>ArrowheadSize</i>	Bestimmt die Größe der Pfeilspitzen.
Texteigenschaften	
<i>TextGap</i>	Größe der Lücke in der Maßlinie links und rechts vom Maßtext.
<i>VerticalTextPosition</i>	Vertikale Position des Bemaßungstextes.
Eigenschaften	
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Führungsobjekts fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Evaluate</i>	Diese Anweisung berechnet die Führung nach den vorgegebenen Eigenschaften.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Führungsobjekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Führungsobjekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Führungsobjekts zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Führungsobjekts an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Führungsobjekts an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Führungsobjekts an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für das Führungsobjekt an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Führungsobjekts zurück.

Tabelle 8.42: Weitere Eigenschaften eines Führungslinienobjekts (Forts.)

Eigenschaften	
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadLeader</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Führungsobjekt zurück oder legt diesen fest.
<i>ScaleFactor</i>	Übergeordneter Skalierfaktor für Größen- oder Abstandseigenschaften.
<i>StyleName</i>	Name des zugeordneten Bemaßungsstils.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Führungsobjekts steuern.

Tabelle 8.42: Weitere Eigenschaften eines Führungslinienobjekts (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Führungsobjekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Führungsobjekts ab.
<i>Highlight</i>	Hebt das Führungsobjekt hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Führungsobjekts mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt ein Führungsobjekt als Resultat einer Spiegelung mit diesem Führungsobjekt.
<i>Mirror3D</i>	Erstellt ein neues Führungsobjekt als Resultat einer räumlichen Spiegelung mit diesem Führungsobjekt.
<i>Move</i>	Verschiebt das Führungsobjekt von den Ursprungs- zu den Zielkoordinaten.
<i>Rotate</i>	Dreht das Führungsobjekt um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht das Führungsobjekt um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Führungsobjekt um einen bestimmten Faktor.

Tabelle 8.43: Prozeduren eines Führungslinienobjekts

Prozedur	Beschreibung
<i>SetXData</i>	Speichert die erweiterten Daten des Führungsobjekts.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Führungsobjekt an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Führungsobjekts im AutoCAD-Zeichnungsfenster.

Tabelle 8.43: Prozeduren eines Führungslinienobjekts (Forts.)

Anwendungsbeispiel

(Beispielprojekt 08\AcadLeader.dvb!ThisDrawing.CreateLeader)

Ziel dieses Projekts ist es, dem Benutzer die Auswahl beliebiger Koordinaten im WKS zu ermöglichen, und auf der Basis dieser Stützpunkte ein Führungslinienobjekt zu erzeugen. Zusätzlich kann der Benutzer in einem Formular auswählen, welches Objekt an die Führung gebunden werden soll. Je nachdem, welches Objekt er gewählt hat, wird entweder ein MText, eine Blockreferenz oder eine Toleranz erzeugt.

Startprozedur

Die Startprozedur stellt den Hauptbestandteil des Projekts dar. Hier werden die einzelnen Stützpunkte des Führungslinienobjekts ausgewählt und der Aufruf der Prozedur zum Erzeugen des Objekts gesteuert.

```
Public Sub CreateLeader()
Dim StartPoint, EndPoint, PointsArray
On Error Resume Next
LeaderFrm.Show 0
```

In der äußeren Schleife wird zunächst der Startpunkt des Objekts bestimmt. Damit die Schleife nicht ungewollt vorzeitig verlassen wird (z.B. durch den Aufruf eines transparenten Befehls), benutzen wir in diesem Fall einmal Schlüsselwörter zur Steuerung der Schleifen. Der Benutzer kann nun das Programm nur beenden, wenn er „X“ oder „exit“ in der Befehlszeile eingibt.

```
Do
Utility.InitializeUserInput 128, "eXit"
Prompt = "Startpunkt der Führung oder eXit:"
StartPoint = Empty
Err.Clear
StartPoint = Utility.GetPoint(, Prompt)
If Err.Description = "Benutzereingabe ist ein Schlüsselwort" _
Then
Exit Do
Else
If TypeName(StartPoint) = "Double()" Then
ReDim PointsArray(2) As Double
PointsArray = StartPoint
```

Die innere Schleife wird gestartet, nachdem der Benutzer einen gültigen Stützpunkt gewählt hat.

Als Erstes wird hier die Zahl der Schlüsselwörter erhöht. Zusätzlich zum Schlüsselwort „Exit“, mit dem der Benutzer diese Schleife verlassen und zur äußeren Schleife zurückkehren kann, bekommt er auch die Möglichkeit, durch Eingabe von „Z“ bzw. „erzeugen“ das Führungslinienobjekt zu erzeugen. Hat er hingegen einen Punkt gewählt, wird die Variable *PointsArray* um drei Felder erhöht und die gewählten Koordinaten werden in die neuen Felder eingetragen.

```

Do
    Utility.InitializeUserInput 128, "Exit erzeugen"
    Prompt = "nächsten Punkt der Führung <Exit, " & _
        "erzeugen>:"
    EndPoint = Empty
    Err.Clear
    EndPoint = Utility.GetPoint(StartPoint, Prompt)
    If Err.Description = "Benutzereingabe ist ein " & _
        "Schlüsselwort" Then
        keyword = Utility.GetInput
        Select Case UCase(keyword)
            Case "EXIT"
                Exit Do
            Case "ERZEUGEN"
                If UBound(PointsArray) < 5 Then
                    MsgBox "", vbCritical
                Else
                    CreateLeaderObject PointsArray, StartPoint
                    Exit Do
                End If
            End Select
        End Select
    Else
        If TypeName(EndPoint) = "Double()" Then
            StartPoint = EndPoint
            ReDim Preserve PointsArray(UBound(PointsArray) _
                + 3) As Double
            PointsArray(UBound(PointsArray) - 2) = _
                EndPoint(0)
            PointsArray(UBound(PointsArray) - 1) = _
                EndPoint(1)
            PointsArray(UBound(PointsArray)) = EndPoint(2)
        End If
    End If
Loop
End If
End If
Loop
SaveSetting "BspLeader", "Position", "Left", LeaderFrm.Left
SaveSetting "BspLeader", "Position", "Top", LeaderFrm.top
Unload LeaderFrm
End Sub

```

Prozedur zur Erzeugung der Führungslinie

Die folgende Prozedur wird zur Erzeugung der Führungslinie verwendet. Als Parameter werden das Array, das die Stützpunkte der zu erzeugenden Führungslinie enthält, und der letzte gewählte Punkt übergeben. Alle anderen Parameter werden aus dem *LeaderFrm*-Formular übernommen.

```
Public Sub CreateLeaderObject(PointsArray, InsPoint)
Dim NewLeader As AcadLeader
Dim Annotation As AcadEntity
Dim Direction(2) As Double
On Error Resume Next
```

Zunächst wird überprüft, welche Art von Führungsobjekt der Benutzer gewählt hat. Dementsprechend wird das jeweilige Formular aufgerufen, in dem der Benutzer die zusätzlichen Angaben zu diesem Objekt festlegen kann. Diese Objekte werden dann mit den zusätzlichen Angaben an der zuletzt ausgewählten Position eingefügt und der zurückgegebene Verweis wird in der Objektvariablen *Annotation* gespeichert. Mehr über die verwendeten Objekte erfahren Sie im Verlauf dieses Kapitels.

```
Select Case LeaderFrm.ComboBox2.ListIndex
Case 0
    ToleranceFrm.Show
    If ToleranceFrm.OK = False Then Exit Sub
    sValue = ToleranceFrm.TextBox1.Text
    Direction(0) = 1
    Direction(1) = 0
    Direction(2) = 0
    Set Annotation = ModelSpace.AddTolerance(sValue, InsPoint, _
        Direction)
Case 1
    BlockFrm.Show
    If BlockFrm.OK = False Then Exit Sub
    Set Annotation = ModelSpace.InsertBlock(InsPoint, _
        BlockFrm.ComboBox1.Text, 1, 1, 1, 0)
    Unload BlockFrm
Case 2
    MTextFrm.Show
    If MTextFrm.OK = False Then Exit Sub
    sValue = MTextFrm.RichTextBox1.Text
    Set Annotation = ModelSpace.AddMText(InsPoint, Width, _
        sValue)
End Select
```

Abschließend wird nur noch das Führungslinienobjekt mit den entsprechenden Parametern erzeugt und der Typ und die Pfeilspitzengröße dem erzeugten Führungsobjekt zugeordnet.

```
Set NewLeader = ModelSpace.AddLeader(PointsArray, Annotation, _
    LeaderFrm.ComboBox1.ListIndex)
NewLeader.ArrowheadType = LeaderFrm.ImageCombo1.SelectedItem. _
    Index - 1
NewLeader.ArrowheadSize = Utility.DistanceToReal( _
    LeaderFrm.TextBox1, acDefaultUnits)
End Sub
```

Blockauswahlformular

Diese Formular wird immer dann eingeblendet, wenn der Benutzer einen Blocknamen auswählen soll. Es enthält zu diesem Zweck eine ComboBox, die alle in der aktuellen Zeichnung vorhandenen Blocknamen aufnimmt, mit Ausnahme anonymer Blöcke und externer Referenzen. Der Formularcode ist recht einfach gehalten. Zunächst wird die globale Variable *OK*

deklariert. Diese Variable wird von der Prozedur kontrolliert, die dieses Formular gestartet hat, und soll diese über die Auswahl des Benutzers informieren. Die beiden Schaltflächenprozeduren setzen den jeweiligen Wert dieser Variablen und blenden dann das Formular aus.

```
Public OK As Boolean
```

Schaltfläche OK:

```
Private Sub CommandButton1_Click()  
OK = True  
Me.Hide  
End Sub
```

Schaltfläche ABBRECHEN:

```
Private Sub CommandButton2_Click()  
OK = False  
Me.Hide  
End Sub
```

In der Initialisierungsprozedur des Formulars wird lediglich die ComboBox mit den Blocknamen der aktuellen Zeichnung gefüllt und anschließend der erste Block der Liste als Auswahlname gesetzt. Dabei werden alle anonymen Blöcke (der Name des Blocks beginnt mit einem Stern) und alle externen Referenzen (enthalten das Zeichen „|“) ausgeschlossen.

```
Private Sub UserForm_Initialize()  
On Error Resume Next  
For Each Blk In ThisDrawing.Blocks  
    If Left(Blk.Name, 1) <> "*" And InStr(1, Blk.Name, "|") = 0 _  
        Then  
        ComboBox1.AddItem Blk.Name  
    End If  
Next  
ComboBox1.ListIndex = 0  
End Sub
```

Texteingabeformular

Dieses Formular wird sowohl zur Eingabe der Zeichenfolge eines MTextes als auch für die Zeichenfolge einer Toleranz verwendet. Das Besondere an diesem Formular ist nicht der Quellcode, der recht einfach gestaltet ist, sondern das Steuerelement *RichTextbox*. Dieses Steuerelement funktioniert in Bezug auf die Eingabe durch den Benutzer genau wie eine normale Textbox. Der Unterschied liegt wieder einmal im Detail. Diese Form der Textbox speichert die Eingabedaten im Richttext-Format, was bedeutet, dass Sie umfassende Formatierungen an dem Text vornehmen können – Formatierungen, wie sie das MText-Objekt in seiner *Text*-Eigenschaft verwendet.

```
Public OK As Boolean
```

Schaltfläche OK:

```
Private Sub CommandButton1_Click()  
OK = True  
Me.Hide  
End Sub
```

Schaltfläche ABBRECHEN:

```
Private Sub CommandButton2_Click()
    OK = False
    Me.Hide
End Sub
```

Führungslinienformular

In diesem Formular kann der Benutzer die verschiedenen Einstellungen für das neue Führungslinienobjekt vornehmen. Diese Vorgaben werden in der Initialisierungsprozedur des Formulars zusammengestellt und stehen dem Benutzer dann zur Verfügung. In der zweiten Prozedur reagieren wir auf die Auswahl einer Pfeilspitze durch den Benutzer. Hat dieser den benutzerdefinierten Pfeil gewählt, wird das Blockauswahlformular gestartet und der Benutzer kann hier den Block für den benutzerdefinierten Pfeil auswählen.

```
Dim LastIDX As Integer
```

Auswahl einer Pfeilspitze

```
Private Sub ImageCombo1_Click()
    On Error Resume Next
    If ImageCombo1.SelectedItem.Index = 21 Then
        BlockFrm.ComboBox1.Text = ThisDrawing.GetVariable("DIMLDRBLK")
        BlockFrm.Show
        If BlockFrm.OK = False Then
            ImageCombo1.SelectedItem = ImageCombo1.ComboItems(LastIDX)
        Else
            ThisDrawing.GetVariable("DIMLDRBLK") = _
                BlockFrm.ComboBox1.Text
        End If
    End If
    LastIDX = ImageCombo1.SelectedItem.Index
End Sub
```

Initialisierungsprozedur

```
Private Sub UserForm_Initialize()
    On Error Resume Next
    Me.Left = GetSetting("BspLeader", "Position", "Left", Me.Left)
    Me.top = GetSetting("BspLeader", "Position", "Top", Me.top)
    ComboBox1.AddItem "Linien ohne Pfeil"
    ComboBox1.AddItem "Spline ohne Pfeil"
    ComboBox1.AddItem "Linien mit Pfeil"
    ComboBox1.AddItem "Spline mit Pfeil"
    ComboBox1.ListIndex = 2
    ComboBox2.AddItem "Toleranz"
    ComboBox2.AddItem "Blockreferenz"
    ComboBox2.AddItem "MText"
    ComboBox2.ListIndex = 0
    Set ImageCombo1.ImageList = ImageList1
    ImageCombo1.ComboItems.Add , , "Geschlossen gefüllt", 1
    ImageCombo1.ComboItems.Add , , "Geschlossen leer", 2
    ImageCombo1.ComboItems.Add , , "Geschlossen", 3
```

```

ImageComb1.ComboItems.Add , , "Punkt", 4
ImageComb1.ComboItems.Add , , "Architektonisch", 5
ImageComb1.ComboItems.Add , , "Schräg", 6
ImageComb1.ComboItems.Add , , "Offen", 7
ImageComb1.ComboItems.Add , , "Ausgangspunkt", 8
ImageComb1.ComboItems.Add , , "Ausgangspunkt2", 9
ImageComb1.ComboItems.Add , , "Rechter Winkel", 10
ImageComb1.ComboItems.Add , , "Offen 30", 11
ImageComb1.ComboItems.Add , , "Punkt klein", 12
ImageComb1.ComboItems.Add , , "Punkt leer", 13
ImageComb1.ComboItems.Add , , "Punkt klein leer", 14
ImageComb1.ComboItems.Add , , "Quadrat", 15
ImageComb1.ComboItems.Add , , "Quadrat gefüllt", 16
ImageComb1.ComboItems.Add , , "Bezugsdreieck", 17
ImageComb1.ComboItems.Add , , "Bezugsdreieck gefüllt", 18
ImageComb1.ComboItems.Add , , "Integral", 19
ImageComb1.ComboItems.Add , , "keine", 20
ImageComb1.ComboItems.Add , , "Benutzerdefiniert", 21
ImageComb1.SelectedItem = ImageComb1.ComboItems(1)
LeaderFrm.TextBox1 = Utility.RealToString(GetVariable("DIMASZ"), _
    acDefaultUnits, GetVariable("LUPREC"))
End Sub

```

8.3.10 Schraffuren (AcadHatch)

Bei dieser Klasse handelt es sich um ein Schraffurobjekt. Eine Schraffur in AutoCAD zu erstellen, erfordert eine etwas umfangreichere Vorgehensweise. Zuerst müssen Sie die Schraffur mit der Funktion *AddHatch* dem Modellbereich, Papierbereich oder einem Block hinzufügen. Nachdem das Objekt erstellt wurde, müssen Sie die Funktion *AppendOuterLoop* mit dem neuen Schraffurobjekt anwenden, um der Schraffur eine Umgrenzung (rote Umgrenzung im Bild) zu geben. Sie können auch eine vorhandene Schraffur über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Tipp

Nicht jede Schraffur in AutoCAD ist auch ein Hatch-Objekt. Vielmehr kann es sich bei einigen Schraffuren auch um Blockreferenzen handeln. Diese Schraffuren werden dann auch als solche verwaltet. Das bedeutet, dass Sie die Eigenschaften und Methoden des Hatch-Objekts in diesem Fall nicht verwenden können. Solche Objekten sind daran zu erkennen, dass der Name des Blocks generell mit einem „*U“ beginnt.

Schraffureigenschaften

Die Eigenschaft *AssociativeHatch* zeigt an, ob es sich bei der Schraffur um eine Assoziativschraffur handelt oder nicht. Sie können die Assoziativität einer Schraffur entfernen, indem Sie diese Eigenschaft auf *FALSE* setzen. Um die Assoziativität einer Schraffur wiederherzustellen, müssen Sie die ursprüngliche Schraffur löschen und eine neue erstellen.

Mit *HatchStyle* lässt sich der Stil der Schraffur festlegen oder abrufen. Sie können dabei auf die Konstanten der Tabelle 8.46 zurückgreifen.

Schraffurdarstellung

Die *PatternType*-Eigenschaft gibt den Mustertyp des Schraffurobjekts zurück. Der Mustertyp bestimmt, aus welcher Mustervorlage das Schraffurmuster entnommen bzw. ob das Schraffurmuster aus den aktuellen Einstellungen generiert werden soll. Da Sie den Wert dieser Eigenschaft nicht direkt über die Eigenschaft anpassen können, müssen Sie den Typ einer Schraffur bei deren Erzeugung (*AddHatch*) oder mit der Anweisung *SetPattern* festlegen. Folgende Konstanten kann diese Eigenschaft zurückgeben:

Konstante	Beschreibung	Wert
<i>acHatchPatternTypeUserDefined</i>	Definiert ein Schraffurmuster unter Verwendung der aktuellen Linientypen.	0
<i>acHatchPatternTypePreDefined</i>	Verwendet die Schraffurmuster der Datei <i>acad.pat</i> .	1
<i>acHatchPatternTypeCustom-Defined</i>	Verwendet die Schraffurmuster einer benutzerdefinierten PAT-Datei.	2

Tabelle 8.44: Konstanten der *PatternType*-Eigenschaft

Mit der *PatternName*-Eigenschaft können Sie den aktuell verwendeten Musternamen der Schraffur abrufen. *PatternName* ist außerdem von der Einstellung der Eigenschaft *PatternType* abhängig (Tabelle 8.44). Auch diese Eigenschaft ist schreibgeschützt und kann nur vergeben werden, wenn Sie oder der Benutzer die Schraffur erstellen oder in Zusammenhang mit der Eigenschaft *PatternType* über die *SetPattern*-Anweisung.

Die Eigenschaft *PatternAngle* legt den Winkel des Schraffurmusters fest. *PatternScale* bestimmt die Skalierung des Schraffurmusters. Ist die Eigenschaft *PatternType* auf den Wert *AcHatchPatternTypeUserDefined* gesetzt, wird *PatternScale* auf die Schraffur nicht angewendet, da der Abstand der Musterlinien in der Musterdatei festgelegt wurde. Stattdessen müssten Sie mit der Eigenschaft *PatternSpace* den Abstand zwischen den einzelnen Linien vorgeben. Weiterhin können Sie in diesem Fall in der Eigenschaft *PatternDouble* festlegen, ob eine Doppelschraffur angezeigt werden soll (*TRUE*) oder nur eine einfache Schraffur (*FALSE*).

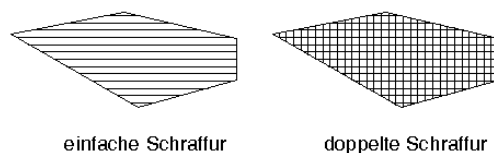


Abbildung 8.46: Darstellung einer einfachen und doppelten Schraffur

Wenn Sie für die Schraffur ein ISO-Schraffurmuster verwenden, können Sie mit der Eigenschaft *ISOPenWidth* die ISO-Stiftbreite bestimmen. Für diese Eigenschaft gelten folgende Konstanten:

Konstante	Beschreibung	Wert
<i>acPenWidthUnk</i>	Stiftbreite unbekannt	-1
<i>acPenWidth013</i>	Stiftbreite 0.13 mm	13
<i>acPenWidth018</i>	Stiftbreite 0.18 mm	18

Tabelle 8.45: Konstanten der Eigenschaft *ISOPenWidth*

Konstante	Beschreibung	Wert
<i>acPenWidth025</i>	Stiftbreite 0.25 mm	25
<i>acPenWidth035</i>	Stiftbreite 0.35 mm	35
<i>acPenWidth050</i>	Stiftbreite 0.50 mm	50
<i>acPenWidth070</i>	Stiftbreite 0.70 mm	70
<i>acPenWidth100</i>	Stiftbreite 1.00 mm	100
<i>acPenWidth140</i>	Stiftbreite 1.40 mm	140
<i>acPenWidth200</i>	Stiftbreite 2.00 mm	200

Tabelle 8.45: Konstanten der Eigenschaft *ISOPenWidth* (Forts.)

Die Konstante *AcPenWidthUnk* stellt eine unbekannte Stiftbreite dar. Wenn Sie diese Konstante bei der Abfrage der Stiftbreite erhalten, müssen Sie die Eigenschaft *PatternScale* zur Bestimmung der Stiftbreite verwenden. Das Setzen der Eigenschaft mit dieser Konstante hat dagegen keine Wirkung.

Die letzte Eigenschaft, mit der Sie die Darstellung der Schraffur beeinflussen können, ist *HatchStyle*. Mit dieser Eigenschaft können Sie den Inselerkennungsstil festlegen. Die Inselerkennung bewirkt dabei Folgendes:

Konstante	Beschreibung	Wert
<i>acHatchStyleNormal</i>	Normal	0
<i>acHatchStyleOuter</i>	Äußere	1
<i>acHatchStyleIgnore</i>	Ignorieren	2

Tabelle 8.46: Konstanten der Eigenschaft *HatchStyle*

Es ist empfehlenswert, den Stil *acHatchStyleNormal* zu verwenden, weil Sie damit Umgrenzungslinien genau definieren können.



Abbildung 8.47: Auswirkung der Eigenschaft *HatchStyle* auf das Schraffurobjekt

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der Schraffur fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.

Tabelle 8.47: Weitere Eigenschaften des Schraffurobjekts

Eigenschaft	Beschreibung
<i>Elevation</i>	Diese Eigenschaft bestimmt die Objekthöhe der Schraffur, relativ zur Z-Achse des Koordinatensystems der Objekte.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Schraffur zurück. Der Handle ist der eindeutige Identifier eines AutoCAD-Elements.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Schraffur ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Schraffur zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der Schraffur an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Schraffur an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Schraffur an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Schraffur an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>NumberOfLoops</i> (schreibgeschützt)	Diese Eigenschaft enthält die Anzahl der Schleifen der Schraffurumgrenzung.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Schraffur zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadHatch</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Schraffur zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der Schraffur steuern.

Tabelle 8.47: Weitere Eigenschaften des Schraffurobjekts (Forts.)

Methoden des Schraffurobjekts

Das Schraffurobjekt stellt neben den Standardprozeduren noch einige zusätzliche Methoden zur Verfügung, wobei der größte Teil dieser Methoden der Verwaltung der Schraffurumgrenzungen dienen. Diese Umgrenzungen werden immer durch andere AutoCAD-Objekte definiert, die zu einem Array zusammengefasst als Schleife hinzugefügt werden können. Diese Schleifen werden in dem Schraffurobjekt gesammelt, womit sich folgende Struktur in der Schraffur ergibt.

Jede Schleife muss dabei eine in sich geschlossene Umgrenzung bilden. Alle Schleifen zusammen ergeben dann die entsprechende Schraffurdarstellung, wobei der Wert der Eigenschaft *HatchStyle* wesentlichen Einfluss nimmt.

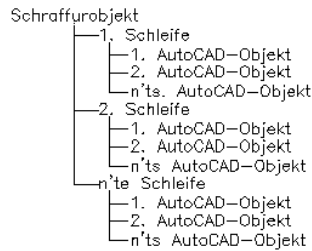


Abbildung 8.48: Verwaltung der einzelnen Schraffurschleifen

Tipp

Wichtig bei der Arbeit mit Schraffuren ist die Anweisung *Evaluate*. Mit dieser Anweisung wird die Schraffur neu berechnet. Nachdem Sie eine Änderung an der Umgrenzung der Schraffur vorgenommen haben, sollten Sie immer diese Anweisung ausführen, damit die Änderungen in der Zeichnung sichtbar werden.

Sie können folgende AutoCAD-Elemente für die Erstellung der ersten Schraffurumgrenzung verwenden: Bögen, Linien, Polylinien, Kreise, Ellipsen, Splines und Regionen. Nachdem die erste Umgrenzung erstellt wurde und die Schraffur damit sichtbar ist, können Sie auch Textobjekte (*AcadText*, *AcadMText*) als Schraffurschleifenobjekte definieren. Versuchen Sie bereits bei der Definition der ersten Schleife Texte mit einzubinden, löst AutoCAD einen Fehler aus.

Schraffurschleifen hinzufügen

Das Hinzufügen einer Schleife kann über drei verschiedene Funktionen erfolgen. Die ersten beiden Prozeduren fügen dem Schraffurobjekt eine weitere Schleife am Ende der Schleifen hinzu.

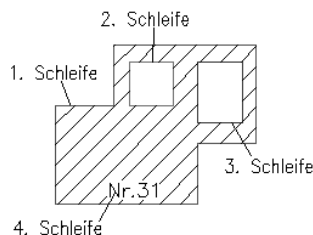


Abbildung 8.49: Schleifendefinitionen einer Schraffur

Sowohl *AppendOuterLoop* als auch *AppendInnerLoop* erwarten ein Array von einem oder mehreren Objekten des Typs *AcadEntity*, die alle zusammen eine geschlossene Umgrenzung ergeben müssen.

Beispiel für *AppenOuterLoop*:

```

Public Sub Sample_AppendOuterLoop()
Dim Object As Object, HatchSSet As AcadSelectionSet
Dim ActHatch As AcadHatch, ObjArray
On Error Resume Next
  
```

```

Utility.GetEntity Object, PickedPoint
If TypeName(Object) = "IAcadHatch" Then
    Set ActHatch = Object
    SelectionSets.Add "Hatch"
    Set HatchSSet = SelectionSets("Hatch")
    HatchSSet.Clear
    HatchSSet.SelectOnScreen
    ReDim ObjArray(HatchSSet.Count - 1) As AcadEntity
    For i = 0 To HatchSSet.Count - 1
        Set ObjArray(i) = HatchSSet.Item(i)
    Next
    ActHatch.AppendOuterLoop ObjArray
    ActHatch.Evaluate
    HatchSSet.Delete
End If
End Sub

```

Beispiel für *AppenInnerLoop*:

```

Public Sub Sample_AppendInnerLoop()
Dim Object As Object, HatchSSet As AcadSelectionSet
Dim ActHatch As AcadHatch, ObjArray
On Error Resume Next
Utility.GetEntity Object, PickedPoint
If TypeName(Object) = "IAcadHatch" Then
    Set ActHatch = Object
    SelectionSets.Add "Hatch"
    Set HatchSSet = SelectionSets("Hatch")
    HatchSSet.Clear
    HatchSSet.SelectOnScreen
    ReDim ObjArray(HatchSSet.Count - 1) As AcadEntity
    For i = 0 To HatchSSet.Count - 1
        Set ObjArray(i) = HatchSSet.Item(i)
    Next
    ActHatch.AppendInnerLoop ObjArray
    ActHatch.Evaluate
    HatchSSet.Delete
End If
End Sub

```

Eine weitere Möglichkeit, eine Schraffurumgrenzung hinzuzufügen, besteht in der Verwendung der Anweisung *InsertLoopAt*. Diese fügt an einem bestimmten Index, der im ersten Parameter der Anweisung festgelegt wird, eine Schleife in der Schraffur ein. Im zweiten Parameter müssen Sie den Typ der hinzuzufügenden Schleife festlegen. Egal, welchen Typ Sie hier wählen, das Ergebnis ist immer dasselbe. Aus diesem Grund reicht generell die Angabe der Konstanten *AcHatchLoopTypeDefault*. Als dritten Parameter übergeben Sie schließlich ein Array der Objekte, die die einzufügende Schleife definieren, genau wie bei den anderen beiden Prozeduren.

```

Public Sub Sample_InsertLoop()
Dim Object As Object, HatchSSet As AcadSelectionSet
Dim ActHatch As AcadHatch, ObjArray
On Error Resume Next
Utility.GetEntity Object, PickedPoint

```

```

If TypeName(Object) = "IAcadHatch" Then
    Set ActHatch = Object
    SelectionSets.Add "Hatch"
    Set HatchSSet = SelectionSets("Hatch")
    HatchSSet.Clear
    HatchSSet.SelectOnScreen
    ReDim ObjArray(HatchSSet.Count - 1) As AcadEntity
    For i = 0 To HatchSSet.Count - 1
        Set ObjArray(i) = HatchSSet.Item(i)
    Next
    ActHatch.InsertLoopAt ActHatch.NumberOfLoops, _
        acHatchLoopTypeDefault, ObjArray
    ActHatch.Evaluate
    HatchSSet.Delete
End If
End Sub

```

Schraffurschleifen abrufen

Die Anweisung *GetLoopAt* ruft eine Schleife an einem bestimmten Index ab. Mit dem ersten Parameter geben Sie die entsprechende Nummer der gesuchten Schleife an. Im zweiten Parameter wird eine Variable festgelegt, die das Array mit den Verweisen auf die Objekte aufnimmt, die die Schleife definieren.

```

ActHatch.GetLoopAt N, ObjList
For I = 0 To UBound(ObjList)
    ObjList(i).Delete
Next
ActHatch.Evaluate

```

Auf diese Weise können Sie ermitteln, welche Objekte die Schraffur erzeugen, und gegebenenfalls einige Schleifen entfernen, indem Sie die Objekte löschen.

Schraffurmuster anpassen

Da sich mit den Eigenschaften *PatternName* und *PatternType* das Muster und der Typ einer Schraffur nur abrufen, aber nicht verändern lassen, stellt das Schraffurobjekt die Anweisung *SetPattern* zur Verfügung. Mit dieser Anweisung können Sie einen Mustertyp (*PatternType*) sowie einen Musternamen (*PatternName*) für die Schraffur in einem Gang festlegen. Diese Vorgehensweise hat einen ganz einfachen Grund. Da AutoCAD das Muster immer nach dem mit *PatternType* festgelegten Typ bestimmt (siehe Tabelle 8.44), sind diese Eigenschaften eng miteinander verknüpft.

Im zweiten Parameter übergeben Sie schließlich den Namen des Musters, das verwendet werden soll. Nachdem Sie diese Funktion ausgeführt haben, sollten Sie auch hier die Anweisung *Evaluate* auf die Schraffur ausführen, um diese neu zu berechnen.

```

ActHatch.SetPattern acHatchPatternTypePreDefined, "Ansi31"
ActHatch.Evaluate

```

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechteckige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>Evaluate</i>	Diese Anweisung berechnet die Schraffur nach den vorgegebenen Eigenschaften.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Schraffur ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Schraffur ab.
<i>Highlight</i>	Hebt die Schraffur hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Schraffur mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Schraffur als Resultat einer Spiegelung mit dieser Schraffur.
<i>Mirror3D</i>	Erstellt eine neue Schraffur als Resultat einer räumlichen Spiegelung dieser Schraffur.
<i>Move</i>	Verschiebt die Schraffur von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die Schraffur um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Schraffur um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Schraffur um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Schraffur.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Schraffur an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Schraffur im AutoCAD-Zeichnungsfenster.

Tabelle 8.48: Weitere Methoden des Schraffurobjekts

Anwendungsbeispiele

(Beispielprojekt: 08\AcadHatch.dvb!ThisDrawing.createHatch)

Betrachten wir uns doch einmal anhand eines Beispiels, wie eine Schraffur mithilfe von VBA in AutoCAD entsteht und was Sie danach alles damit anstellen können. Zu diesem Zweck erstellen wir ein Programm, das eine Gebäudeschraffur zeichnet. Ein wichtiger Punkt bei der Erstellung einer solchen Schraffur ist die automatische Bestimmung der Neigung der Schraffurlinien. Das Programm soll dem Benutzer die Möglichkeit bieten, die Schraffurlinien senkrecht zur längsten Linie oder im Winkel von 45° zur längsten Linie des zu schraffierenden Polygons zu zeichnen.

Als Erstes erstellen Sie ein neues Projekt und fügen ein Formular ein, in dem wir später die Einstellungen zur Erzeugung der Schraffur vornehmen werden. Benutzer der AutoCAD-Version ab 2000i können diesmal das Formular ungebunden anzeigen lassen. Diejenigen, die noch AutoCAD 2000 benutzen, müssen natürlich das Formular wie immer einblenden. Der Vorteil des nicht modalen Anzeigens des Formulars ist, dass Sie die Einstellungen interaktiv verändern können. Im nachfolgenden Beispiel werde ich die 2000i-Version verwenden. Auf der Buch-CD finden Sie natürlich beide Versionen, also die modale und die nicht modale. Das Grundprinzip der beiden Versionen ist das gleiche, bis auf einige Kleinigkeiten, die zum Anzeigen des Formulars verwendet werden.

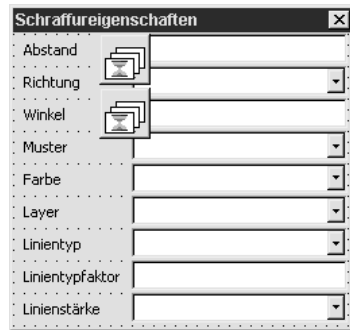


Abbildung 8.50: Formular des Beispielprojekts

Dieses Formular enthält wieder die *ImageList*-Steuerelemente, die die Grundfarben sowie die Linienstärken als kleine Bildchen verwalten.

Wieder wird die Ereignisprozedur *UserForm_Initialize* genutzt, um die Steuerelemente zu füllen, deren Inhalt sich zur Laufzeit nicht mehr ändert. In den Textboxen 1 & 2 werden einfach die entsprechenden Vorgabewerte festgelegt. Sie können für den Schraffurwinkel (*TextBox2*) auch den Inhalt der Systemvariable *HPANG* verwenden, die ja den aktuellen Winkel für Schraffuren enthält. Dazu müssen Sie nur die Zahl 0 durch die Anweisung *ThisDrawing.GetVariable("HPANG")* ersetzen. Für den Linientypfaktor in *TextBox3* bedienen wir uns des aktuellen Linientypfaktors des Systems (Variable *LTSCALE*). Die Prozeduren zum Füllen der *ImageCombos* kennen Sie ja bereits aus *Kapitel 3.3.6*, so dass ich darauf nicht weiter eingehen muss. Bleibt zum Schluss noch die Prozedur *setHatchPatternList*. Mit dieser Prozedur werden alle Schraffurmuster aus der Datei *Acad.pat* der *ComboBox2* hinzugefügt.

```
Private Sub UserForm_Initialize()
On Local Error Resume Next
Text1.Text = ThisDrawing.Utility.RealToString(1, acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
ComboBox1.AddItem "Automatisch 45°"
ComboBox1.AddItem "Automatisch 90°"
ComboBox1.AddItem "freier Winkel"
ComboBox1.ListIndex = 0
Text2.Text = ThisDrawing.Utility.AngleToString(0, _
    ThisDrawing.GetVariable("AUNITS"), _
    ThisDrawing.GetVariable("AUPREC"))
setHatchPatternList
ComboBox2.ListIndex = 0
```



```

Text3.Text = ThisDrawing.Utility.RealToString( _
ThisDrawing.GetVariable("LTSCALE"), _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
nitColorCombo ImageCombo1
InitLWCombo ImageCombo2
ImageCombo1.ComboItems(1).Selected = True
ImageCombo2.ComboItems(1).Selected = True
End Sub

```

Schraffurmuster ermitteln

In dieser Prozedur werden alle Supportverzeichnisse von AutoCAD, die ja unter dem *Preferences*-Objekt zu finden sind (*Kapitel 5.3.3*), einzeln nach der Datei *acad.pat* durchsucht. Wenn die Funktion *Dir()* die Datei gefunden hat, wird diese geöffnet und Zeile für Zeile nach den Musternamen durchsucht. Wenn Sie sich diese ASCII-Datei schon einmal angeschaut haben, werden Sie wissen, dass die Zeilen der Datei, die mit einem Stern (*) beginnen (wird mit der *Left*-Anweisung geprüft), den Musternamen enthalten. Wird also ein Musternamen gefunden, schneiden wir das erste Zeichen ab und übergeben den Rest (den Musternamen selbst) an die *ComboBox2*, die ja eine Liste der Musternamen enthalten soll. Als Letztes benötigen wir im Formularcode nur noch die Formatierungen der Zahlen, falls der Benutzer diese geändert haben sollte.

```

Public Sub setMusterHatch()
Dim SupportPathes As String
Dim SupportPath As String
On Local Error Resume Next
ComboBox2.Clear
SupportPathes = hisDrawing.Application.Preferences.Files. _
    SupportPath
For I = 1 To Len(SupportPathes)
    If Mid$(SupportPathes, I, 1) = ";" Then
        If Dir$(SupportPath & "\acad.pat") <> "" Then
            Open SupportPath & "\acad.pat" For Input As #1
            Do While Not EOF(1)
                Line Input #1, PatName
                If Left$(PatName, 1) = "*" Then
                    ComboBox2.AddItem Mid$(PatName, 2, _
                        InStr(1, PatName, ",") - 2)
                End If
            Loop
            Close
            Exit For
        End If
    Else
        SupportPath = SupportPath & Mid$(SupportPathes, I, 1)
    End If
End For
Next
End Sub

```

Abstand wurde geändert

```

Private Sub Text1_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Local Error Resume Next

```

```

If ThisDrawing.Utility.DistanceToReal(Text1.Text, _
    acDefaultUnits) <= 0 Then
    MsgBox "Der Abstand muß größer 0 sein!", vbCritical
    Cancel = True
Else
    Text1.Text = ThisDrawing.Utility.RealToString( _
        ThisDrawing.Utility.DistanceToReal(Text1.Text, _
            acDefaultUnits), acDefaultUnits, _
            ThisDrawing.GetVariable("LUPREC"))
End If
End Sub

```

Winkel wurde geändert

```

Private Sub Text2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Local Error Resume Next
Text2.Text = ThisDrawing.Utility.AngleToString(_
    ThisDrawing.Utility.AngleToReal(Text2.Text, _
        ThisDrawing.GetVariable("AUNITS")), _
        ThisDrawing.GetVariable("AUNITS"), _
        ThisDrawing.GetVariable("AUPREC"))
End Sub

```

Linientypfaktor wurde geändert

```

Private Sub Text3_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Local Error Resume Next
If ThisDrawing.Utility.DistanceToReal(Text3.Text, _
    acDefaultUnits) = 0 Then
    MsgBox "Linientypfaktor muß größer 0 sein!", vbCritical
    Cancel = True
Else
    Text3.Text = ThisDrawing.Utility.RealToString( _
        ThisDrawing.Utility.DistanceToReal(Text3.Text, _
            acDefaultUnits), acDefaultUnits, _
            ThisDrawing.GetVariable("LUPREC"))
End If
End Sub

```

Das wäre also der gesamte Formularcode. Die einzelnen Prozeduren dürften mittlerweile kein Problem mehr darstellen, so dass wir unverzüglich mit dem Modulcode fortfahren können. Den gesamten Code fügen Sie bitte wieder in das Modul *ThisDrawing* ein. Die erste Prozedur des Modulcodes, mit der wir uns beschäftigen wollen, ist die Startprozedur.

Startprozedur

Im ersten Teil der Prozedur werden die Layer und Linientypen des aktuellen Dokuments in die entsprechenden Comboboxen des Formulars *HatchSetupFrm* übertragen. Als ausgewähltes Element der jeweiligen Combobox werden der aktuelle Layer (*ActiveLayer.Name*) sowie der aktuelle Linientyp (*ActiveLinetype.Name*) festgelegt. Mit der Anweisung *HatchSetupFrm.Show 0* wird das Formular schließlich ungebunden (nicht modal) eingeblendet. Aus diesem Grund wird die Ausführung des Programmcodes fortgesetzt, obwohl das Formular

angezeigt wird. Nach dem Einblenden des Formulars wird ein Auswahl Satz initialisiert. In der *Do...Loop*-Schleife wird schließlich die Auswahl durchgeführt (*AcSSet.SelectOnScreen*). Hat der Benutzer kein Objekt gewählt, der Auswahl Satz enthält also keine Elemente, wird die Schleife verlassen, das Formular ausgeblendet und das Programm beendet. Andernfalls wird der Auswahl Satz der Objekte an folgende Prozedur weitergegeben.

```
Public Sub createHatch()
Dim AcSSet As AcadSelectionSet
On Local Error Resume Next

Set AllLayers = Layers
HatchSetupFrm.ComboBox3.Clear
For Each l In AllLayers
    If InStr(1, l.Name, "|") = 0 Then
        HatchSetupFrm.ComboBox3.AddItem l.Name
    End If
Next
HatchSetupFrm.ComboBox3.Text = ActiveLayer.Name

Set AllLStyles = Linetypes
HatchSetupFrm.ComboBox4.Clear
For Each l In AllLStyles
    If InStr(1, l.Name, "|") = 0 Then
        HatchSetupFrm.ComboBox4.AddItem l.Name
    End If
Next
HatchSetupFrm.ComboBox4.Text = ActiveLinetype.Name
HatchSetupFrm.Show 0

'Auswahl Satz initialisieren
If TypeName(SelectionSets("Auswahl")) = "Nothing" Then
    SelectionSets.Add "Auswahl"
End If
Set AcSSet = SelectionSets("Auswahl")
'Auswahl Satz initialisiert

Do
    Err.Clear
    AcSSet.Clear
    AcSSet.SelectOnScreen
    If AcSSet.Count > 0 Then
        CreateHatchPattern AcSSet
    Else
        Exit Do
    End If
Loop
Unload HatchSetupFrm
End Sub
```

Prozedur zur Erzeugung der Schraffur

Da der Auswahl Satz an diese Prozedur als Parameter übergeben wurde, können die ausgewählten Objekte weiter genutzt werden. Als Erstes wird der Auswahl Satz an die Prozedur *getPatternAngle* übergeben, um den Drehwinkel für das Schraffurmuster zu ermitteln. Mit dieser Prozedur werden wir uns weiter unten befassen. Gehen Sie einfach davon aus, dass diese Funktion den entsprechenden Winkel bestimmt und ihn dann an die Variable *HatchAngle* weiterreicht. Als Nächstes wird das Array *pSelSet* deklariert. Da dieses Array alle

Objekte aufnehmen soll, die der Auswahl Satz enthält, muss das Array dieselbe Größe annehmen wie der Auswahl Satz. Da die Anzahl der Elemente des Auswahl Satzes in der Eigenschaft *Count* gespeichert ist und das erste Element des Arrays, wie Sie ja bereits wissen, mit dem Index 0 beginnt, muss das Array von 0 bis *Count - 1* deklariert werden. Nachdem das Array deklariert wurde, werden alle Objekte mittels *For...Next*-Schleife vom Auswahl Satz zum Array übertragen. Da es sich bei den Elementen des Auswahl Satzes um Objekte handelt, ist die Anweisung *Set* (Erstellung eines Objektverweises) zwingend notwendig.

Über die Anweisung *AddHatch*, die in diesem Fall auf den Modellbereich angewendet wird (Sie können die Schraffur auch in einem Block oder im Papierbereich erstellen), wird zunächst die Schraffur erstellt. Da als dritter Parameter *TRUE* festgelegt wurde, wird eine Assoziativität für diese Schraffur erzeugt. Mit der folgenden Anweisung *AppendOuterLoop*, die auf die neue Schraffur angewendet wird, werden der Schraffur schließlich die AutoCAD-Objekte übergeben, die als äußere Umgrenzung ausgewählt wurden. Die restlichen Zeilen dieser Prozedur übergeben dann nur noch die Eigenschaften, die der Benutzer im Formular festgelegt hat, an die neue Schraffur. Um die Schraffur nun endgültig vor dem Auge des Benutzers erscheinen zu lassen, wird die letzte Anweisung (*NewHatch.Evaluate*) auf die Schraffur angewendet. Wenn Sie dies nicht tun, muss der Benutzer die Zeichnung erst regenerieren, bevor die Schraffur sichtbar wird.

```
Public Sub CreateHatchPattern(AcSSet As AcadSelectionSet)
    Dim NewHatch As AcadHatch
    Dim HatchAngle As Double
    On Local Error Resume Next
    HatchAngle = getPatternAngle(AcSSet)
    ReDim pSelSet(AcSSet.Count - 1) As AcadEntity
    For i = 0 To AcSSet.Count - 1
        Set pSelSet(i) = AcSSet.Item(i)
    Next
    Set NewHatch = ModelSpace.AddHatch(1, _
        HatchSetupFrm.ComboBox2.Text, True)
    NewHatch.AppendOuterLoop pSelSet
    NewHatch.PatternSpace = Utility.DistanceToReal( _
        HatchSetupFrm.Text1.Text, acDefaultUnits) * 1
    NewHatch.PatternAngle = HatchAngle
    NewHatch.Color = HatchSetupFrm.GetColorIndex( _
        HatchSetupFrm.ImageCombo1.SelectedItem.Index)
    NewHatch.Layer = HatchSetupFrm.ComboBox3.Text
    NewHatch.Linetype = HatchSetupFrm.ComboBox4.Text
    NewHatch.LinetypeScale = Utility.DistanceToReal( _
        HatchSetupFrm.Text3.Text, acDefaultUnits)
    NewHatch.Lineweight = HatchSetupFrm.GetLWIndex( _
        HatchSetupFrm.ImageCombo2.SelectedItem.Index)
    NewHatch.Evaluate
End Sub
```

Prozedur zur Ermittlung des Schraffurwinkels

Da mit dieser Prozedur die längste Seite der Schraffurumgrenzung ermittelt werden soll, werden alle Objekte des Auswahl Satzes in einer *For Each...Next*-Schleife durchlaufen. Jedes Objekt wird nach Übergabe an die Objektvariable *Obj* auf seine Länge überprüft. Da eine Auswertung aller in AutoCAD vorhandenen Objekttypen den Rahmen dieses Beispiels sprengen würde, beschränken wir uns auf Linien und Polylinien. Diese Objekttypen werden mit der

TypeName-Funktion herausgefiltert. Bei Linien die Länge zu ermitteln, ist relativ einfach. Dazu nehmen wir den Start- und den Endpunkt der jeweiligen Linie und berechnen damit die Entfernung zwischen den Punkten. (Sie können auch die Linieneigenschaft *Length* zur Ermittlung der Linienlänge verwenden, müssen dann aber darauf achten, dass es sich dabei um die räumliche Entfernung zwischen den beiden Punkten handelt). Ist die berechnete Länge größer als die zuvor ermittelte, stellt diese Linie die aktuell längste Seite dar. Also wird der Richtungswinkel zwischen Start- und Endpunkt mithilfe der Funktion *AngleFromXAxis* (Kapitel 6.3) berechnet und der Funktion als vorläufiger Rückgabewert zugewiesen. Die aktuelle Länge wird an die Variable *LastDist* zwischengespeichert, um im weiteren Verlauf prüfen zu können, ob Objekte ausgewählt wurden, die noch größere Längen aufweisen. Das gleiche Prinzip wird auch auf die Polylinien angewendet, nur dass dort alle Stützpunkte durchlaufen werden.

Nachdem die längste Seite ermittelt wurde, wird zum Schluss noch der endgültige Winkel festgelegt. Dazu prüfen wir mit der *Select Case*-Anweisung, welche Auswahl der Benutzer in der *ComboBox1* des Formulars vorgenommen hat. Sie erinnern sich: „Automatisch 45°“, „Automatisch 90°“ oder „freier Winkel“. Im ersten Fall wird zur ermittelten Richtung der längsten Seite (wurde ja der Funktion bereits übergeben) ein Winkel von 45° addiert, im zweiten 90° und im dritten wird der Winkel, den der Benutzer in der *Textbox2* eingetragen hat, verwendet.

```
Public Function getPatternAngle(AcSSet As AcadSelectionSet) As _
    Double
    Const PI = 3.14159265358979
    Dim FirstPt(2) As Double
    Dim SecondPt(2) As Double
    On Local Error Resume Next
    LastDist = 0
    For Each Obj In AcSSet
        Select Case TypeName(Obj)
            Case "IAcadLine"
                varPoint = Obj.StartPoint
                FirstPt(0) = varPoint(0)
                FirstPt(1) = varPoint(1)
                varPoint = Obj.EndPoint
                SecondPt(0) = varPoint(0)
                SecondPt(1) = varPoint(1)
                If LastDist < GetDistance(FirstPt, SecondPt) Then
                    getPatternAngle = Utility.AngleFromXAxis(FirstPt, _
                        SecondPt)
                    LastDist = GetDistance(FirstPt, SecondPt)
                End If
            Case "IAcadPolyline", "IAcadLWPolyline", "IAcad3DPolyline"
                varPoint = Obj.Coordinate(0)
                FirstPt(0) = varPoint(0)
                FirstPt(1) = varPoint(1)
                PointsList = Obj.Coordinates
                X = UBound(Obj.Coordinate(0)) + 1
                For i = X To UBound(PointsList) Step X
                    SecondPt(0) = PointsList(i)
                    SecondPt(1) = PointsList(i + 1)
                    If LastDist < GetDistance(FirstPt, SecondPt) Then
                        getPatternAngle = Utility.AngleFromXAxis(FirstPt, _
                            SecondPt)
                        LastDist = GetDistance(FirstPt, SecondPt)
                    End If
                Next i
            End Select
        Next Obj
    End For
```

```

        End If
        FirstPt(0) = SecondPt(0)
        FirstPt(1) = SecondPt(1)
    Next
    If Obj.Closed = True Then
        varPoint = Obj.Coordinate(0)
        SecondPt(0) = varPoint(0)
        SecondPt(1) = varPoint(1)
        If LastDist < GetDistance(FirstPt, SecondPt) Then
            getPatternAngle = Utility.AngleFromXAxis(FirstPt, _
                SecondPt)
            LastDist = GetDistance(FirstPt, SecondPt)
        End If
    End If
End Select
Next
Select Case HatchSetupFrm.ComboBox1.ListIndex
    Case 0
        getPatternAngle = getPatternAngle + PI / 4 'automatisch 45°
    Case 1
        getPatternAngle = getPatternAngle + PI / 2 'automatisch 90°
    Case 2
        getPatternAngle = Utility.AngleToReal(HatchSetupFrm.Text2. _
            Text, GetVariable("AUNITS")) 'benutzerdefiniert
End Select
End Function

```

Tipp

Beachten Sie bei der Festlegung von Winkeln, dass immer mit Radiant-Werten gerechnet wird. $45^\circ = \text{PI} / 4$, $90^\circ = \text{PI} / 2$, $180^\circ = \text{PI}$, $360^\circ = 2 * \text{PI}$ usw.

In diesem Beispiel wird der Winkel generell addiert. Sie müssten also darauf achten, welchen Wert die Systemvariable *ANGDIR* besitzt, da im obigen Beispiel davon ausgegangen wird, dass das System im Urzeigersinn arbeitet. Sie könnten also diese Systemvariable prüfen und müssten dann, je nach Rückgabe (0 – gegen den Uhrzeigersinn, 1 – im Uhrzeigersinn), den Winkel subtrahieren oder addieren.

8.3.11 Linie (AcadLine)

Diese Klasse enthält alle Eigenschaften und Methoden, die Sie auf eine einfache Linie anwenden können.

Sie können einen Verweis auf eine Linie erstellen, indem Sie mit der Funktion *AddLine* eine neue Linie dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene Linie über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Die wichtigsten Eigenschaften dieses Objekts sind *StartPoint* und *EndPoint*, die die Position der Linie in der Zeichnung festlegen. In beiden sind die jeweiligen Koordinaten als 3D-Array gespeichert.

Zu den weiteren Eigenschaften des Objekts gehört *Angle*, die den aktuellen Richtungswinkel der Linie enthält. Dieser Winkel wird vom Startpunkt zum Endpunkt bestimmt und in Radiant zurückgegeben. Um den Winkel in eine andere Maßeinheit umzurechnen, benutzen Sie das

Utility-Objekt. *Length* gibt die aktuelle räumliche Länge der Linie zurück, gemessen vom Start- zum Endpunkt, und die Eigenschaft *Delta* enthält die Koordinatenunterschiede zwischen Start- und Endpunkt.

```
Dim DelatVec As Variant
DeltaVec = ActLine.Delta
DeltaX = DeltaVec(0)
DeltaY = DeltaVec(1)
DeltaZ = DeltaVec(2)
```

Alle diese Eigenschaften haben rein informativen Charakter und können nicht direkt geändert werden, sondern passen sich an, wenn Sie die Position der Linie verändern.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der Linie fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Linie zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Linie ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Linie zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der Linie an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Linie an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Linie an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Linie an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Linie zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadLine</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Linie zurück oder legt diesen fest.
<i>Thickness</i>	Diese Eigenschaft speichert die Objekthöhe der Linie. Sie können den aktuellen Wert auslesen oder zurückschreiben.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der Linie steuern.

Tabelle 8.49: Weitere Eigenschaften eines Linienobjekts

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Linie ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Linie ab.
<i>Highlight</i>	Hebt die Linie hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Linie mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Linie als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine neue Linie als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt die Linie von Ursprungs- auf Zielkoordinaten.
<i>Offset</i>	Diese Anweisung versetzt die Linie um einen bestimmten Betrag. Nach Ausführung der Funktion wird eine neue Linie erstellt und als Objektvariable zurückgegeben.
<i>Rotate</i>	Dreht die Linie um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Linie um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Linie um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Linie.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Linie an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Linie im AutoCAD-Zeichnungsfenster.

Tabelle 8.50: Prozeduren des Linienobjekts

Anwendungsbeispiel

(Beispielprojekt: 08\AcadLine.dvb\ThisDrawing.CreateLines)

Sehen wir uns zu dem Linienobjekt ein kleines Beispiel an. Wir werden ein Programm erstellen, das Linienzüge im Dokument erstellt. Diesen Linien werden Eigenschaften wie Layer und Linientyp über ein Dialogfenster zur Laufzeit zugeordnet. Damit der Benutzer diese Eigenschaften interaktiv ändern kann, benötigen wir wieder ein Formular, das ungebunden angezeigt wird (ab AutoCAD 2000i). Die AutoCAD2000-User finden das 2000-kompatible Programm ebenfalls auf der Buch-CD.

Wenn Sie sich den Formularcode ansehen, werden Sie erkennen, dass die einzelnen Prozeduren ähnlich wie im vorherigen Beispiel sind. Fangen wir wieder mit der Initialisierungsprozedur für das Formular an:

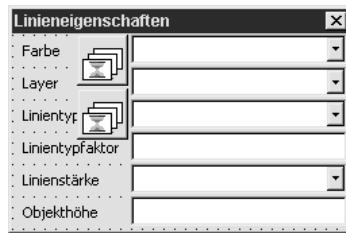


Abbildung 8.51: Formular des Beispielprojekts

```

Private Sub UserForm_Initialize()
On Local Error Resume Next
Me.Left = GetSetting("BspLine", "Position", "Left", Me.Left)
Me.top = GetSetting("BspLine", "Position", "Top", Me.top)
Text1.Text = ThisDrawing.Utility.RealToString(1, acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
Text2.Text = ThisDrawing.Utility.RealToString(0, acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
InitColorCombo ImageCombo1
InitLWCombo ImageCombo2
ImageCombo1.ComboItems(1).Selected = True
ImageCombo2.ComboItems(1).Selected = True
End Sub

```

Sie enthält in diesem Beispiel deutlich weniger Codezeilen, da wir diesmal ja auch weniger Steuerelemente verwenden. Die Prozeduren zum Füllen der *ImageCombos* ist dieselbe wie im Beispiel zum *Kapitel 3.3.6*. Die Auflistung und Erklärung der einzelnen Prozeduren sparen wir uns an dieser Stelle. Vielmehr soll der Modulcode, wie immer im Modul *ThisDrawing* abgelegt, einer näheren Betrachtung unterzogen werden.

```

Public Sub CreateLines()
Dim VarPoint As Variant
Dim StartPt(2) As Double
Dim EndPt(2) As Double
Dim NewLine As AcadLine
On Local Error Resume Next
LineSetupFrm.Show 0
Set AllLayers = Layers
LineSetupFrm.ComboBox1.Clear
For Each l In AllLayers
    If InStr(1, l.Name, "|") = 0 Then
        LineSetupFrm.ComboBox1.AddItem l.Name
    End If
Next
LineSetupFrm.ComboBox1.Text = ActiveLayer.Name
Set AllStyles = Linetypes
LineSetupFrm.ComboBox2.Clear
For Each l In AllStyles
    LineSetupFrm.ComboBox2.AddItem l.Name
Next
LineSetupFrm.ComboBox2.Text = ActiveLinetype.Name
Do

```

```

Prompt = "Wählen Sie den Startpunkt des Linienzuges:"
VarPoint = Empty
VarPoint = Utility.GetPoint(, vbCrLf & Prompt)
If TypeName(VarPoint) = "Double()" Then
    StartPt(0) = VarPoint(0)
    StartPt(1) = VarPoint(1)
    StartPt(2) = VarPoint(2)
Do
    Prompt = "Wählen Sie den nächsten Punkt des Linienzuges:"
    VarPoint = Empty
    VarPoint = Utility.GetPoint(StartPt, vbCrLf & Prompt)
    If TypeName(VarPoint) = "Double()" Then
        EndPt(0) = VarPoint(0)
        EndPt(1) = VarPoint(1)
        EndPt(2) = VarPoint(2)
        Set NewLine = ModelSpace.AddLine(StartPt, EndPt)
        NewLine.Color = LineSetupFrm.GetColorIndex( _
            LineSetupFrm.ImageCombo1.SelectedItem.Index)
        NewLine.Layer = LineSetupFrm.ComboBox1.Text
        NewLine.Linetype = LineSetupFrm.ComboBox2.Text
        NewLine.LinetypeScale = Utility.DistanceToReal( _
            LineSetupFrm.Text1.Text, acDefaultUnits)
        NewLine.Lineweight = LineSetupFrm.GetLWIndex( _
            LineSetupFrm.ImageCombo2.SelectedItem.Index)
        NewLine.Thickness = Utility.DistanceToReal( _
            LineSetupFrm.Text2.Text, acDefaultUnits)
        StartPt(0) = EndPt(0)
        StartPt(1) = EndPt(1)
        StartPt(2) = EndPt(2)
    Else
        Exit Do
    End If
Loop
Else
    Exit Do
End If
Loop
SaveSetting "BspLine", "Position", "Left", LineSetupFrm.Left
SaveSetting "BspLine", "Position", "Top", LineSetupFrm.top
Unload LineSetupFrm
End Sub

```

Nachdem die Variablen deklariert wurden, unter anderem *NewLine* (als Klasse *AcadLine*) für die jeweils neu erzeugte Linie, und die Comboboxen initialisiert wurden, genau wie in den vorhergehenden Beispielen, beginnt die Abfrage der Linienendpunkte. Da wir ja einen Linienzug erzeugen wollen, müssen zwei *Do...Loop*-Schleifen ineinander verschachtelt werden. Die erste Schleife wird so lange durchlaufen, bis kein Startpunkt für einen Linienzug zurückgegeben wird. Wie Sie ja bereits wissen, können wir mithilfe der Funktion *GetPoint* des *Utility*-Objekts einen Punkt in der Zeichnung abfragen. Wir nutzen also diese Funktion für den Startpunkt:

```

Prompt = "Wählen Sie den Startpunkt des Linienzugs:"
VarPoint = Empty
VarPoint = Utility.GetPoint(, vbCrLf & Prompt)

```

Bevor die Funktion *GetPoint* ausgeführt wird, muss die Variable *VarPoint*, die ja den gewählten Punkt enthalten soll, geleert werden. Das hat einen ganz einfachen Grund. Beim nächsten Durchlauf durch die Schleife enthält die Variable ja immer noch ein Array von Koordinaten aus der vorhergehenden Abfrage. Wurde die Funktion *GetPoint* nicht erfolgreich ausgeführt, z.B. Abbruch durch den Benutzer, wird die Variable *VarPoint* unverändert weitergereicht. Wenn diese dann noch Werte aus der vorhergehenden Abfrage enthält, kommen wir nicht mehr aus der Schleife, was auch die nächste Codezeile erklärt.

```
If TypeName(VarPoint) = "Double()" Then
```

Wurde die Funktion *GetPoint* erfolgreich ausgeführt, übergibt diese ein Array mit Zahlen an die Variable *VarPoint*. Die Anweisung *TypeName* gibt in diesem Fall die Zeichenfolge *Double()* als Ergebnis ihrer Prüfung zurück und das Programm kann fortgesetzt werden. Andernfalls wird die Schleife, und damit das Programm, mit der Anweisung *Exit Do* beendet.

In der inneren Schleife werden nun die weiteren Punkte des Linienzugs abgefragt. Bevor allerdings die Schleife beginnt, müssen wir die Koordinaten des zuvor gewählten Punkts an das Array *StartPt* übergeben, um den entsprechenden Punkt weiter nutzen zu können. Dieser Startpunkt wird als Erstes dazu genutzt, der Funktion *GetPoint* einen Ausgangspunkt (von diesem Punkt wird ein Gummiband gezogen, siehe *Kapitel 6.3*) mitzugeben.

```
StartPt(0) = VarPoint(0)
StartPt(1) = VarPoint(1)
StartPt(2) = VarPoint(2)
Do
    Prompt = "Wählen Sie den nächsten Punkt des Linienzuges:"
    VarPoint = Empty
    VarPoint = Utility.GetPoint(StartPt, vbCrLf & Prompt)
    If TypeName(VarPoint) = "Double()" Then
```

Nach der erneuten Prüfung, wie für den Startpunkt, wird die Linie in der Zeichnung erzeugt und der Objektvariablen *NewLine* übergeben. Dieser Objektvariablen – und somit auch der neuen Linie – werden anschließend noch die Eigenschaften zugewiesen, die der Benutzer im Formular festgelegt hat.

```
Set NewLine = ModelSpace.AddLine(StartPt, EndPt)
NewLine.Color = LineSetupFrm.GetColorIndex( _
    LineSetupFrm.ImageCombo1.SelectedItem.Index)
NewLine.Layer = LineSetupFrm.ComboBox1.Text
NewLine.Linetype = LineSetupFrm.ComboBox2.Text
NewLine.LinetypeScale = Utility.DistanceToReal( _
    LineSetupFrm.Text1.Text, acDefaultUnits)
NewLine.Lineweight = LineSetupFrm.GetLWIndex( _
    LineSetupFrm.ImageCombo2.SelectedItem.Index)
NewLine.Thickness = Utility.DistanceToReal( _
    LineSetupFrm.Text2.Text, acDefaultUnits)
```

Jetzt wird die Schleife damit abgeschlossen, dass der eben gewählte Endpunkt zum Startpunkt der neuen Linie wird.

```
StartPt(0) = EndPt(0)
StartPt(1) = EndPt(1)
StartPt(2) = EndPt(2)
```

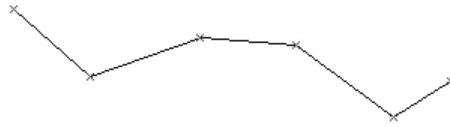


Abbildung 8.52: Linienzug über mit *GetPoint* gewählte Punkte

Diese Schleife wird so lange ausgeführt, bis der Benutzer keinen Endpunkt mehr auswählt. Ist dies der Fall, setzt das Programm die Ausführung mit der Auswahl eines neuen Startpunkts in der äußeren Schleife fort.

So nun noch ein Punkt zum Abschluss dieses Beispiels. Bevor das Formular geschlossen wird, speichern wir diesmal die Position des Formulars in der Registrierung.

```
SaveSetting "BspLine", "Position", "Left", LineSetupFrm.Left
SaveSetting "BspLine", "Position", "Top", LineSetupFrm.top
Unload LineSetupFrm
```

8.3.12 2D-Polylinien

Optimierte Polylinie (*AcadLWPolyline*)

Die Lightweight-Polylinie ist ein besonderes Polylinie-Objekt in AutoCAD. Denn im Gegensatz zur „normalen“ und zur 3D-Polylinie speichert diese nur zwei Koordinaten (X, Y) pro Stützpunkt. Aus diesem Grund wird die LWPolylinie auch als optimierte Polylinie bezeichnet. Die Klasse *AcadLWPolyline* enthält alle Eigenschaften und Methoden, die Sie auf diese Art der Polylinie anwenden können.

Sie erstellen einen Verweis auf eine LWPolylinie, indem Sie mit der Funktion *AddLightWeightPolyline* eine neue LWPolylinie dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene LWPolylinie über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Auch bei dieser Art Polylinie werden die Stützpunkte, über die der Linienzug verläuft, in den Eigenschaften *Coordinate* (einzelner Stützpunkt) und *Coordinates* (alle Stützpunkte) verwaltet. Der Unterschied zu den anderen Objekten besteht darin, dass die einzelnen Stützpunkte keine Z-Koordinaten enthalten. Diese werden in der Eigenschaft *Elevation* gespeichert, die somit die Gesamthöhe der Polylinie wiedergibt. Aus der Trennung der Koordinaten ergibt sich auch eine neue Variante der Bestimmung der Gesamtzahl an Stützpunkten über die *Coordinates*-Eigenschaft.

```
Dim PointsList As Variant
PointsList = ActLWPoly.Coordinates
StpCount = (UBound(PointsList)+ 1) / 2
```

Siehe auch *Kapitel 8.1.1*.

Normale Polylinie (*AcadPolyline*)

Im Gegensatz zur LWPolylinie, kann diese 2D-Polylinie pro Stützpunkt drei Koordinaten aufnehmen. Allerdings wird die Z-Koordinate in allen Stützpunkten ignoriert. Diese Art der Polylinie ist ein Überbleibsel älterer AutoCAD-Versionen und wird aus Kompatibilitätsgründen

weiterhin verwendet. Welche Art der Polylinie in einer AutoCAD-Zeichnung verwendet wird, hängt von der Einstellung der Systemvariablen *PLINETYPE* ab.

Sie können einen Verweis auf solch eine Polylinie erstellen, indem Sie mit der Funktion *AddPolyline* eine neue 2D-Polylinie dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene 2D-Polylinie über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften der Polylinie

Mit der Eigenschaft *ConstantWidth* können Sie eine konstante Breite für die Polylinie festlegen oder die aktuelle Breite abrufen. Da Sie hiermit nur die Breite für die gesamte Polylinie verarbeiten können, müssen Sie, wenn Sie einzelne Segmente ansprechen möchten, die Funktionen *SetWidth* bzw. *GetWidth* verwenden.

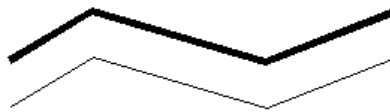


Abbildung 8.53: Polylinien mit unterschiedlicher globaler Breite

Als letzte interessante Eigenschaft stellt das Polylinienobjekt *LinetypeGeneration* zur Verfügung. Mit dieser Eigenschaft können Sie die Linientypgenerierung für die Polylinie aktivieren (*TRUE*) oder deaktivieren (*FALSE*). Ist *LinetypeGeneration* deaktiviert, wird mit dem Muster der Polylinie an jedem Stützpunkt neu begonnen. Eine Aktivierung dieser Eigenschaft bewirkt, dass das Linienmuster über den Stützpunkt hinaus fortgesetzt wird.



Abbildung 8.54: Polylinien mit und ohne Generierung

Genau wie die 3D-Polylinie können Sie auch bei 2D-Polylinien unterschiedliche Darstellungen der Linienverbindungen mit der *Type*-Eigenschaft festlegen. Diese Eigenschaft ist bei optimierten Polylinienobjekten nicht verfügbar.

Konstante	Beschreibung	Wert
<i>acSimplePoly</i>	Keine Glättung	0
<i>acFitCurvePoly</i>	Kurve	1
<i>acQuadSplinePoly</i>	Quadratisch	2
<i>acCubicSplinePoly</i>	Kubisch	3

Tabelle 8.51: Konstanten der *Type*-Eigenschaft

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Area</i> (schreibgeschützt)	Mit dieser Eigenschaft ermitteln Sie den aktuellen Flächeninhalt der optimierten Polylinie.
<i>Closed</i>	Mit dieser Eigenschaft können Sie feststellen, ob die Polylinie geschlossen ist oder es festlegen.
<i>Color</i>	Legt die Farbe der Polylinie fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Polylinie zurück. Der Handle ist der eindeutige Identifier eines AutoCAD-Elements.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Polylinie ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Polylinie zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der Polylinie an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Polylinie an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Polylinie an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Polylinie an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Polylinie zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>LWPolyline</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück. Die OwnerID kann die ID eines Blocks, eines Xref, des Modell- oder Papierbereichs sein, je nachdem, wo dieses Objekt erzeugt wurde.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Polylinie zurück oder legt diesen fest.
<i>Thickness</i>	Diese Eigenschaft speichert die Objekthöhe der Polylinie.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der Polylinie steuern.

Tabelle 8.52: Weitere Polylinieneigenschaften

Methoden der Polylinie

Das Polylinienobjekt verfügt neben den Standardmethoden über eine Reihe von zusätzlichen Prozeduren, mit denen die verschiedensten Anpassungen an dem Objekt vorgenommen werden können.

Stützpunkte hinzufügen

Mit der *AddVertex*-Anweisung lässt sich der Polylinie ein weiterer Stützpunkt hinzufügen. Der Unterschied zur *AppendVertex*-Anweisung besteht darin, dass Sie die genaue Position des Stützpunkts innerhalb der Stützpunktauflistung der Polylinie bestimmen können. Diese Position wird im ersten Parameter (Index) festgelegt. Im zweiten Parameter müssen Sie dann wieder die Koordinaten des hinzuzufügenden Stützpunkts übergeben. Beachten Sie hierbei, dass Sie nur 2D-Arrays verwenden dürfen, die die X- und Y-Koordinate enthalten. Zur Festlegung der Z-Koordinate müssen Sie die Eigenschaft *Elevation* verwenden.

```
Dim NewVertex(1) As Double  
NewVertex(0) = 100.234  
NewVertex(1) = 130.423  
ActLwPoly.AddVertex 2, NewVertex
```

Segmentbreite anpassen

Neben der globalen Breite, die für die gesamte Polylinie gilt, können Sie auch die Breiten einzelner Polyliniensegmente abrufen bzw. anpassen. Dazu stellt das Objekt die Methoden *GetWidth* und *SetWidth* zur Verfügung. *GetWidth* gibt die aktuelle Anfangs- und Endbreite des im Parameter *Index* angegebenen Segments zurück. *SetWidth* setzt hingegen die Breite des Segments, das auch hier im Parameter *Index* angegeben wurde.



Abbildung 8.55: Polylinie mit unterschiedlichen Segmentbreiten

Der Aufruf der Prozeduren ist in etwa identisch. Im ersten Parameter wird der Index des Segments festgelegt, der zweite und dritte Parameter legen entweder Start- und Endbreite zurück (*GetWidth*) oder legen diese fest (*SetWidth*).

Kreisbögen einfügen

Die Definition von Kreisbögen wird über den Wert der Ausbuchtung des jeweiligen Segments gesteuert. Diese Ausbuchtung ergibt sich aus dem Verhältnis von Ausbuchtungshöhe zur halben Stützpunktdistanz, was Abbildung 8.56 verdeutlicht.

In welche Richtung das Segment ausgebuchtet werden soll, bestimmt das Vorzeichen des jeweiligen Werts in Abhängigkeit von der Richtung, in der die Polylinien gezogen wurden. Wenn der Bogen im Uhrzeigersinn gezeichnet werden soll, muss der Wert positiv sein, gegen den Uhrzeigersinn hingegen negativ. Eine Ausbuchtung von 0 stellt eine Gerade, eine Ausbuchtung von 1 einen Halbkreis dar.

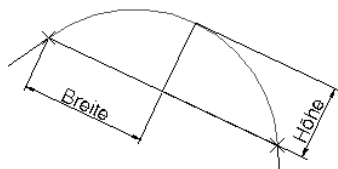


Abbildung 8.56: Ausbuchtungsdefinition

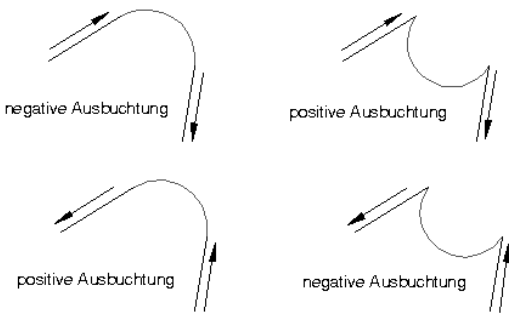


Abbildung 8.57: Ausbuchtungen einer Polylinie

Mit der Eigenschaft *GetBulge* können Sie die Ausbuchtung eines bestimmten Segments der Polylinie ermitteln.

```
Dim Bulge As Double
Bulge = ActLWPoly.GetBulge(1)
```

Mit der Anweisung *SetBulge* können Sie die Ausbuchtung eines bestimmten Segments der Polylinie bestimmen.

```
ActLWPoly.SetBulge 1, 0.5
```

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>Explode</i>	Diese Anweisung löst die Polylinie auf und gibt die daraus entstandenen Linien und Kreisbögen als Array zurück. Dim PLEnt As Variant PLEnt = ActLWPoly.Explode For i = 0 To UBound(PLEnt) Next

Tabelle 8.53: Weitere Prozeduren einer optimierten Polylinie

Prozedur	Beschreibung
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Polylinie ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Polylinie ab.
<i>Highlight</i>	Hebt die Polylinie hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Polylinie mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Polylinie als Resultat einer Spiegelung mit dieser Polylinie.
<i>Mirror3D</i>	Erstellt eine neue Polylinie als Resultat einer räumlichen Spiegelung dieser Polylinie.
<i>Move</i>	Verschiebt die Polylinie von Ursprungs- auf Zielkoordinaten.
<i>Offset</i>	Versetzt die Polylinie um einen bestimmten Betrag. Nach Ausführung der Funktion wird eine neue Polylinie erstellt und kann an eine Objektvariable zurückgegeben werden.
<i>Rotate</i>	Dreht die Polylinie um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Polylinie um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Polylinie um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Polylinie.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Polylinie an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Polylinie im AutoCAD-Zeichnungsfenster.

Tabelle 8.53: Weitere Prozeduren einer optimierten Polylinie (Forts.)

Anwendungsbeispiele

Optimierte Polylinie

(Beispielprojekt: 08\AcadPolyline.dvb!Konvert3DPolyToLWPoly)

Im nachfolgenden Beispiel werden 3D-Polylinien in optimierte Polylinien umgewandelt, um den Unterschied in der Verwaltung der einzelnen Stützpunkte dieser beiden Polylinientypen zu verdeutlichen. Für dieses relative einfache Programm benötigen wir lediglich ein neues Projekt. Der Code des Programms wird wie gehabt im Modul *ThisDrawing* abgelegt.

```
Public Sub Konvert3DPolyToLWPoly()
Dim VarPoints As Variant
Dim Object As Object
Dim Prompt As String
Dim mHeight As Double
Dim LWPoly As AcadLWPolyline
Dim sValue As VbMsgBoxResult
On Local Error Resume Next
sValue = MsgBox("Konvertierte 3D-Polylinien löschen?", _
vbYesNoCancel, "Konvert")
If sValue = vbCancel Then Exit Sub
```

```

Do
  Prompt = "Wählen Sie eine 3D-Polylinie:"
  ThisDrawing.Utility.GetEntity Object, PickedPoint, Prompt
  If TypeName(Object) = "IAcad3DPolyline" Then
    VarPoints = Object.Coordinates
    ReDim VertList(((UBound(VarPoints) + 1) / 3) * 2 - 1) As _
      Double
    x = 0
    mHeight = 0
    For i = 0 To UBound(VarPoints) Step 3
      VertList(x) = VarPoints(i)
      x = x + 1
      VertList(x) = VarPoints(i + 1)
      x = x + 1
      mHeight = mHeight + VarPoints(i + 2)
    Next
    Set LWPoly = ModelSpace.AddLightWeightPolyline(VertList)
    LWPoly.Elevation = mHeight / ((UBound(VarPoints) + 1) / 3)
    LWPoly.Closed = Object.Closed
    LWPoly.Color = Object.Color
    LWPoly.Layer = Object.Layer
    LWPoly.Linetype = Object.Linetype
    LWPoly.LinetypeScale = Object.LinetypeScale
    LWPoly.Lineweight = Object.Lineweight
    LWPoly.PlotStyleName = Object.PlotStyleName
    If sValue = vbYes Then Object.Delete
  Else
    Exit Do
  End If
Loop
End Sub

```

Nach der Deklaration der einzelnen Variablen beginnen wir mit der Abfrage, ob die gewählten 3D-Polylinien nach der Konvertierung gelöscht werden sollen. Da der Funktion *MsgBox* im zweiten Parameter mitgeteilt wird, dass die Schaltflächen JA, NEIN und ABBRECHEN zur Auswahl eingeblendet werden sollen, kommen als Rückgabe die Werte *vbYes* (Ja), *vbNo* (Nein) oder *vbCancel* (Abbrechen) in Frage. Speichern wir zunächst die Rückgabe in der Variablen *sValue* (deklariert als *VbMsgBoxResult*). Als Nächstes wird geprüft, ob der Benutzer die Schaltfläche ABBRECHEN (*sValue = vbCancel*) gewählt hat. In diesem Fall kann das Programm beendet werden (Anweisung *Exit Sub*). Ist dies nicht der Fall, wird eine *Do...Loop*-Schleife begonnen, die beendet wird, sobald der Benutzer keine 3D-Polylinien mehr ausgewählt hat. Im nächsten Abschnitt wird, wie immer, ein Objekt durch den Benutzer gewählt und geprüft, ob es sich dabei um eine 3D-Polylinie handelt. Ist dies der Fall, beginnt die Umwandlung in eine optimierte Polylinie. Es ist natürlich keine Umwandlung im herkömmlichen Sinne, vielmehr wird eine neue Polylinie über die Stützpunkte der 3D-Polylinie erzeugt. Also werden erst einmal die Koordinaten aller Stützpunkte der gewählten Polylinie an die Variable *VarPoints* übergeben. Das so entstandene Array wird für die folgenden Anweisungen genutzt.

```

ReDim VertList(((UBound(VarPoints) + 1) / 3) * 2 - 1) As Double
x = 0
mHeight = 0
For i = 0 To UBound(VarPoints) Step 3
  VertList(x) = VarPoints(i)

```

```

x = x + 1
VertList(x) = VarPoints(i + 1)
x = x + 1
mHeight = mHeight + VarPoints(i + 2)
Next

```

Mit *ReDim* deklarieren wir ein neues Array, das nur noch die X- und Y-Koordinaten, also wesentlich weniger Felder besitzen wird. Die Anzahl der Felder wird mit folgender Formel bestimmt:

```
Anzahl = ((n + 1) / 3) * 2 - 1
```

Aus dieser Tatsache ergibt sich auch, dass für das Hochzählen der Felder des Arrays der neuen Polylinie ein eigener Zähler (X) verwendet werden muss. Beim Durchlaufen der einzelnen Koordinaten werden die X- und Y-Koordinaten der 3D-Polylinie an die Felder der Polylinie übergeben. Die Höhen der 3D-Polylinie werden in einer eigenen Variable (*mHeight*) zu einem Wert aufaddiert. Nachdem das Array initialisiert wurde, kann die neue Polylinie erstellt werden.

```

Set LWPoly = ModelSpace.AddLightWeightPolyline(VertList)
LWPoly.Elevation = mHeight / ((UBound(VarPoints) + 1) / 3)

```

Im nächsten Schritt werden alle Höhen der 3D-Polylinien, die ja alle der Variablen *mHeight* hinzugefügt wurden, gemittelt und der Eigenschaft *Elevation*, die die Höhe der LWPolylinie darstellt, übergeben. Nach der Übergaben einiger Eigenschaften der 3D-Polylinie an die neue LWPolylinie (soll ja den gleichen Linientyp haben und auf dem gleichen Layer liegen) wird nochmals der Inhalt der Variablen *sValue* (enthält den Rückgabewert der Benutzerabfrage *MsgBox*) geprüft. Enthält diese Variable den Wert *vbYes*, kann die 3D-Polylinie gelöscht werden.

Normale Polylinie

Da diese Art der Polylinie, bis auf die Eigenschaft *Type* und die Anzahl der Koordinaten in den Stützpunkten, genau der optimierten Polylinie entspricht, werde ich als Beispiel ein Konvertierungsprogramm beschreiben. In diesem Programm werden LWPolylinien in normale Polylinien oder normale Polylinien in LWPolylinien umgewandelt. Der Programmcode des Beispiels besteht lediglich aus einer Prozedur, die Sie dem Modul *ThisDrawing* hinzufügen müssen.

```

Public Sub KonvertPolylines()
Dim VarPoints As Variant
Dim Object As Object
Dim Prompt As String
Dim LWPoly As AcadLWPolyline
Dim Poly As AcadPolyline
Dim OutStr As String
Dim ModeStr As String
Dim AcSSet As AcadSelectionSet
Dim KMode As Integer
On Local Error Resume Next
If TypeName(SelectionSets("Auswahl")) = "Nothing" Then
    SelectionSets.Add "Auswahl"
End If
Set AcSSet = SelectionSets("Auswahl")
Do
    Abfrage:
    Prompt = vbCrLf & "Typ der zu konvertierenden Polylinie" & _

```

```

    " eingeben" & vbCrLf & "[eXit Polylinie Lwpolylinien]")
Utility.InitializeUserInput 128, & _
    "eXit Polylinie Lwpolylinie"
OutStr = Utility.GetKeyword(Prompt)
Select Case UCase(OutStr)
    Case "LWPOLYLINE"
        Utility.InitializeUserInput 128, & _
            "KEine KURve Quadratisch KUBisch"
        Prompt = vbCrLf & "Vorwahl der Polylinienglättung " & _
            & festlegen" & vbCrLf _
            & "[KEine KURve Quadratisch KUBisch]"
        ModeStr = Utility.GetKeyword(Prompt)
        Select Case UCase(ModeStr)
            Case "KEINE"
                KMode = 0
            Case "KURVE"
                KMode = 1
            Case "QUADRATISCH"
                KMode = 2
            Case "KUBISCH"
                KMode = 3
        End Select
    Case "POLYLINE"
    Case "EXIT"
        Exit Do
    Case Else
        GoTo Abfrage
End Select
AcSSet.Clear
AcSSet.SelectOnScreen
If AcSSet.Count > 0 Then
    For x = 0 To AcSSet.Count - 1
        Set Object = AcSSet.Item(x)
        If UCase(OutStr) = "POLYLINE" Then
            If TypeName(Object) = "IAcadPolyline" Then
                VarPoints = Object.Coordinates
                ReDim VertList(((UBound(VarPoints) + 1) / 3) * _
                    2 - 1) As Double
                x = 0
                For i = 0 To UBound(VarPoints) Step 3
                    VertList(x) = VarPoints(i)
                    x = x + 1
                    VertList(x) = VarPoints(i + 1)
                    x = x + 1
                Next
                Set OwnerObj = ObjectIdToObject(Object.OwnerID)
                Set LWPoly = OwnerObj.AddLightWeightPolyline( _
                    VertList)
                LWPoly.Elevation = Object.Elevation
                LWPoly.Closed = Object.Closed
                LWPoly.Color = Object.Color
                LWPoly.Layer = Object.Layer
                LWPoly.Linetype = Object.Linetype
            End If
        End If
    Next
End If

```

```

        LWPoly.LinetypeScale = Object.LinetypeScale
        LWPoly.Lineweight = Object.Lineweight
        LWPoly.PlotStyleName = Object.PlotStyleName
        Object.Delete
    End If
Else
    If TypeName(Object) = "IAcadLWPolyline" Then
        VarPoints = Object.Coordinates
        ReDim VertList(((UBound(VarPoints) + 1) / 2) * _
            3 - 1) As Double
        x = 0
        For i = 0 To UBound(VarPoints) Step 2
            VertList(x) = VarPoints(i)
            x = x + 1
            VertList(x) = VarPoints(i + 1)
            x = x + 1
            VertList(x) = 0
            x = x + 1
        Next
        Set OwnerObj = Object.IdToObject(Object.OwnerID)
        Set Poly = OwnerObj.AddPolyline(VertList)
        Poly.Elevation = Object.Elevation
        Poly.Closed = Object.Closed
        Poly.Color = Object.Color
        Poly.Layer = Object.Layer
        Poly.Linetype = Object.Linetype
        Poly.LinetypeScale = Object.LinetypeScale
        Poly.Lineweight = Object.Lineweight
        Poly.PlotStyleName = Object.PlotStyleName
        Poly.Type = KMode
        Object.Delete
    End If
End If
Next
End If
Loop
End Sub

```

Nachdem die verschiedensten Variablen deklariert wurden, unter anderem eine Objektvariable für eine Polylinie, eine für eine LWPolylinie und eine für einen Auswahlsatz, beginnt das Programm mit der Erstellung eines Verweises auf einen Auswahlsatz mit dem Namen „Auswahl“. Zuerst wird geprüft, ob ein Auswahlsatz mit diesem Namen bereits existiert. Ist dies nicht der Fall, wird ein Auswahlsatz mit diesem Namen der Auflistung von Auswahlsätzen hinzugefügt. Nachdem ein Verweis auf diesen Auswahlsatz in der Objektvariable *AcSSet* gespeichert wurde, beginnt eine *Do...Loop*-Schleife. Diese Schleife wird so lange durchlaufen, bis der Benutzer in der AutoCAD-Befehlszeile die Option EXIT wählt. Also muss als nächster Schritt eine Eingabe entwickelt werden, bei der der Benutzer mittels Eingabe verschiedene Optionen auswählen kann.

```

Prompt = vbCrLf & "Typ der zu konvertierenden Polylinie " & _
    "eingeben" & vbCrLf & "[Exit Polylinie Lwpolylinien]"
Utility.InitializeUserInput 128, "Exit Polylinie Lwpolylinie"
OutStr = Utility.GetKeyword(Prompt)

```

Zunächst legen wir einen Text für die Eingabeaufforderung fest, wobei die späteren Optionen separat angezeigt werden sollen.

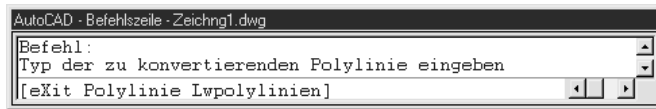


Abbildung 8.58: Eingabeaufforderung

Anschließend legen wir die Optionen mithilfe der Anweisung *InitializeUserInput* des *Utility*-Objekts (*Kapitel 6.3*) fest. So kann dann der Benutzer über die Funktion *GetKeyword* eine entsprechende Eingabe tätigen. Welche Option gewählt wurde, können wir anschließend der Variablen *OutStr* entnehmen. Prüfen wir also die Eingabe mithilfe der *Select...Case*-Anweisung. Hat der Benutzer die Option *LWPOLYLINE* gewählt, so möchte er optimierte Polylinien in normale Polylinien umwandeln. Da die normale Polylinie die zusätzliche Eigenschaft *Type* enthält, kann dem Benutzer eine weitere Auswahl zur Verfügung gestellt werden, in der der Polylinientyp (keine, Kurve, quadratisch oder kubisch) festgelegt werden kann.

```
Utility.InitializeUserInput 128, "KEine KURve Quadratisch KUBisch"
Prompt = vbCrLf & "Vorwahl der Polylinienglättung festlegen" & _
        vbCrLf & "[KEine KURve Quadratisch KUBisch]"
ModeStr = Utility.GetKeyword(Prompt)
```

Bei dieser Auswahl lässt sich an dieser Stelle gleich ein weiteres Merkmal der Benutzerabfrage festhalten. Wie Sie sehen, beginnen drei der Optionen mit dem gleichen Anfangsbuchstaben und zwei weitere haben sogar zwei gleiche Anfangsbuchstaben. Aus diesem Grund müssen Sie alle Buchstaben, die der Benutzer für die Auswahl der Option eingeben muss, als Großbuchstaben festlegen. AutoCAD kann so auf die Eingabe des Benutzers reagieren und im Falle einer unkonkreten Eingabe eine Aufforderung zur genaueren Angabe der Option ausgeben:

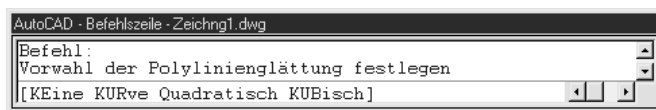


Abbildung 8.59: Eingabeaufforderung

Nachdem der Benutzer nun die entsprechende Auswahl getroffen hat, müssen wir dieser Auswahl den entsprechenden Polylinientyp zuordnen. Dies geschieht wiederum mit der *Select Case*-Anweisung:

```
Select Case UCase(ModeStr)
    Case "KEINE"
        KMode = 0
    Case "KURVE"
        KMode = 1
    Case "QUADRATISCH"
        KMode = 2
    Case "KUBISCH"
        KMode = 3
End Select
```

Bei der Zuordnung des jeweiligen Typs können Sie entweder die Werte 0–3 verwenden oder die entsprechenden Konstanten, wie sie in der Tabelle der Eigenschaften der Polylinie beschrieben sind.

```
Select Case UCase(ModeStr)
  Case "KEINE"
    KMode = acSimplePoly
  Case "KURVE"
    KMode = acFitCurvePoly
  Case "QUADRATISCH"
    KMode = acQuadSplinePoly
  Case "KUBISCH"
    KMode = acCubicSplinePoly
End Select
```

Sie können außerdem die Variable *KMode* statt als Datentyp *Integer* auch als Konstantenauf-listung *AcPolylineType* deklarieren. Das hätte dann den Vorteil, dass alle Konstanten der Auf-listung als Vorgaben angezeigt werden.

Wenden wir uns nun wieder den anderen Abschnitten der ersten *Select...Case*-Anweisung zu. Im zweiten Teil wird zwar in den Vergleich „Polylinie“ verzweigt, dieser Abschnitt wird aber nur verwendet, damit diese Option nicht über den *Case Else*-Abschnitt abgehandelt wird. Im dritten Anweisungsteil (*Case „EXIT“*) wird die Schleife einfach verlassen. Der vierte und letzte Abschnitt (*Case Else*) soll eine ungültige Eingabe durch den Benutzer verhindern und in solch einem Fall an der Sprungmarke *Abfrage* das Programm fortsetzen.

Nachdem eine Auswahl von AutoCAD-Objekten dem Auswahlsatz *AcSSet* übergeben wurden, beginnen die Umwandlungsroutinen.

Routine 1: normale Polylinien in LWPolylinien

```
VarPoints = Object.Coordinates
ReDim VertList(((UBound(VarPoints) + 1) / 3) * 2 - 1) As Double
x = 0
For i = 0 To UBound(VarPoints) Step 3
  VertList(x) = VarPoints(i)
  x = x + 1
  VertList(x) = VarPoints(i + 1)
  x = x + 1
Next
Set OwnerObj = ObjectIdToObject(Object.OwnerID)
Set LWPoly = OwnerObj.AddLightWeightPolyline(VertList)
LWPoly.Elevation = Object.Elevation
LWPoly.Closed = Object.Closed
LWPoly.Color = Object.Color
LWPoly.Layer = Object.Layer
LWPoly.Linetype = Object.Linetype
LWPoly.LinetypeScale = Object.LinetypeScale
LWPoly.Lineweight = Object.Lineweight
LWPoly.PlotStyleName = Object.PlotStyleName
Object.Delete
```

Zunächst werden die Koordinaten der normalen Polylinie einem Array übergeben und anschließend wird mithilfe dieses Arrays ein neues Array, das für die LWPolylinie geeignet ist, deklariert. Nachdem über die *If...End If*-Schleife die Koordinaten dem neuen Array übergeben wurden, wird eine LWPolylinie erzeugt. Dabei legen wir die Polylinie im gleichen Bereich (Modell-, Papierbereich oder Block) ab, in dem die ursprüngliche Polylinie gelegen hat.

```
Set OwnerObj = ObjectIdToObject(Object.OwnerID)
Set LWPoly = OwnerObj.AddLightWeightPolyline(VertList)
```

Dazu ermitteln wir zuerst die ObjectID des Eigentümerobjekts der Polylinie und erstellen mithilfe der Anweisung *ObjectIdToObject* einen Verweis (Variable *OwnerObj*) auf dieses Objekt. Diesem Eigentümerobjekt wird nun die neue LWPolylinie hinzugefügt. Zum Schluss werden noch die Eigenschaften der normalen Polylinie auf die LWPolylinie übertragen und dann die alte Polylinie gelöscht.

Routine 2: LWPolylinien in normale Polylinien

```
If TypeName(Object) = "IAcadLWPolyline" Then
  VarPoints = Object.Coordinates
  ReDim VertList(((UBound(VarPoints) + 1) / 2) * 3 - 1) As Double
  x = 0
  For i = 0 To UBound(VarPoints) Step 2
    VertList(x) = VarPoints(i)
    x = x + 1
    VertList(x) = VarPoints(i + 1)
    x = x + 1
    VertList(x) = 0
    x = x + 1
  Next
  Set OwnerObj = ObjectIdToObject(Object.OwnerID)
  Set Poly = OwnerObj.AddPolyline(VertList)
  Poly.Elevation = Object.Elevation
  Poly.Closed = Object.Closed
  Poly.Color = Object.Color
  Poly.Layer = Object.Layer
  Poly.Linetype = Object.Linetype
  Poly.LinetypeScale = Object.LinetypeScale
  Poly.LineWeight = Object.LineWeight
  Poly.PlotStyleName = Object.PlotStyleName
  Poly.Type = KMode
  Object.Delete
End If
```

Die zweite Routine funktioniert auf dieselbe Art wie die erste, mit dem Unterschied, dass hier die *Type*-Eigenschaft belegt wird und dass die Stützpunkte mit jeweils drei Koordinaten belegt werden.

8.3.13 Multilinie (AcadMLine)

Multilinen werden in VBA durch die Klasse *AcadMLine* dargestellt. Linien dieser Klasse können in AutoCAD mit dem Befehl `_mline` eingefügt werden.

Sie können einen Verweis auf eine Multilinie erstellen, indem Sie mit der Funktion *AddMLine* eine neue Multilinie dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene Multilinie über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Verwaltung der Stützpunkte

Da die Multilinie auch über mehrere Stützpunkte verfügen kann, ist es logisch, dass die einzelnen Punkte in der Eigenschaft *Coordinates* verwaltet werden. Leider enthält dieses Objekt keine *Coordinate*-Eigenschaft zum Abrufen bzw. Setzen einzelner Stützpunkte (warum auch immer), so dass die komplette Verwaltung auf *Coordinates* zurückfällt.

Multiliniestil

Die *StyleName*-Eigenschaft gibt den verwendeten Namen des Multiliniestils für diese Multilinie zurück. Da diese Eigenschaft schreibgeschützt ist, sollten Sie den aktuell eingestellten Multiliniestil überprüfen, bevor eine Multilinie erzeugt wird (*MLSTIL*).

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der Multilinie fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Multilinie zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Multilinie ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Multilinie zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der Multilinie an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Multilinie an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Multilinie an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuellen Linienstärke für die Multilinie an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Multilinie zurück.

Tabelle 8.54: Weitere Eigenschaften der Multilinienobjekte

Eigenschaft	Beschreibung
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadMLine</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerId</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Multilinie zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der Multilinie steuern.

Tabelle 8.54: Weitere Eigenschaften der Multilinienobjekte (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Multilinie ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Multilinie ab.
<i>Highlight</i>	Hebt die Multilinie hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Multilinie mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Multilinie als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine neue Multilinie als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt die Multilinie von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die Multilinie um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Multilinie um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Multilinie um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Multilinie.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Multilinie an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Multilinie im AutoCAD-Zeichnungsfenster.

Tabelle 8.55: Methoden der Multilinienobjekte

Anwendungsbeispiel

(Beispielprojekt: 08\AcadMLine.dvb!\ThisDrawing.CreateMLines)

Da dieses Objekt nur sehr wenig Eigenschaften und Methoden besitzt, bleiben nur wenig Möglichkeiten, ein komplexes Beispiel zu diesem Thema zu entwickeln. Erstellen wir doch einfach ein Programm, das Multilinienzüge im Dokument erstellt. Diesen Multilinen werden Eigenschaften wie Layer und Linientyp über ein Dialogfenster zur Laufzeit zugeordnet. Damit der Benutzer diese Eigenschaften interaktiv ändern kann, benutzen wir wieder ein ungebundenes Formular. Erstellen Sie ein Projekt und fügen Sie diesem ein Formular hinzu. Das Formular sollte wie folgt aufgebaut sein:

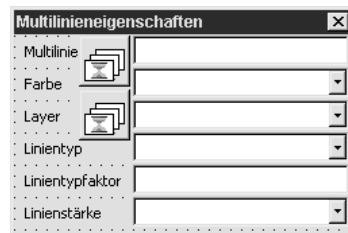


Abbildung 8.60: Beispielformular

Das Formular gleicht im Aussehen doch fast dem Formular des Projekts aus *Kapitel 8.3.11*, es sind also nicht viel Worte über den Aufbau des Formularcodes zu verlieren. Schauen Sie sich nur kurz die Initialisierungsroutine des Formulars an:

```
Private Sub UserForm_Initialize()
On Local Error Resume Next
Me.Left = GetSetting("BspMLine", "Position", "Left", Me.Left)
Me.Top = GetSetting("BspMLine", "Position", "Top", Me.Top)
Text1.Text = "STANDARD"
Text2.Text = ThisDrawing.Utility.RealToString(1, acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
InitColorCombo ImageCombo1
InitLWCombo ImageCombo2
ImageCombo1.ComboItems(1).Selected = True
ImageCombo2.ComboItems(1).Selected = True
End Sub
```

In dieser Routine werden, wie immer, die Comboboxen, Textfelder und ImageCombos eingerichtet. Außerdem werden auch hier die Registrierungsfunktionen zum Auslesen der gespeicherten Positionen eingesetzt, da es sich um ein ungebundenes Formular handelt. Kommen wir nun umgehend zum Modulcode, der wieder dem Klassenmodul *ThisDrawing* hinzugefügt wurde.

```
Public Sub CreateMLines()
Dim VarPoint As Variant
Dim VarPoints As Variant
Dim StartPt(2) As Double
Dim VertList(5) As Double
Dim NewMLine As AcadMLine
On Local Error Resume Next
```

```

MLineSetupFrm.Show 0
Set AllLayers = Layers
MLineSetupFrm.ComboBox1.Clear
For Each l In AllLayers
    If InStr(1, l.Name, "|") = 0 Then
        MLineSetupFrm.ComboBox1.AddItem l.Name
    End If
Next
MLineSetupFrm.ComboBox1.Text = ActiveLayer.Name
Set AllLStyles = Linetypes
MLineSetupFrm.ComboBox2.Clear
For Each l In AllLStyles
    MLineSetupFrm.ComboBox2.AddItem l.Name
Next
MLineSetupFrm.ComboBox2.Text = ActiveLinetype.Name
Do
    Prompt = "Wählen Sie den Startpunkt des Multilinienzuges:"
    VarPoint = Empty
    VarPoint = Utility.GetPoint(, vbCrLf & Prompt)
    If TypeName(VarPoint) = "Double()" Then
        StartPt(0) = VarPoint(0)
        StartPt(1) = VarPoint(1)
        StartPt(2) = VarPoint(2)
    Do
        Prompt = "Wählen Sie den nächsten Punkt des „ & _
        Multilinienzuges:"
        VarPoint = Empty
        VarPoint = Utility.GetPoint(StartPt, vbCrLf & Prompt)
        If TypeName(VarPoint) = "Double()" Then
            If TypeName(NewMLine) = "Nothing" Then
                VertList(0) = StartPt(0)
                VertList(1) = StartPt(1)
                VertList(2) = StartPt(2)
                VertList(3) = VarPoint(0)
                VertList(4) = VarPoint(1)
                VertList(5) = VarPoint(2)
                Set NewMLine = ModelSpace.AddMLine(VertList)
            Else
                VarPoints = NewMLine.Coordinates
                ReDim NewPoints(UBound(VarPoints) + 3) As Double
                For i = 0 To UBound(VarPoints)
                    NewPoints(i) = VarPoints(i)
                Next
                NewPoints(UBound(VarPoints) + 1) = VarPoint(0)
                NewPoints(UBound(VarPoints) + 2) = VarPoint(1)
                NewPoints(UBound(VarPoints) + 3) = VarPoint(2)
                NewMLine.Coordinates = NewPoints
                NewMLine.Update
            End If
            NewMLine.Color = MLineSetupFrm.GetColorIndex( _
            MLineSetupFrm.ImageComb1.SelectedItem.Index)
            NewMLine.Layer = MLineSetupFrm.ComboBox1.Text
            NewMLine.Linetype = MLineSetupFrm.ComboBox2.Text
        End If
    End Do

```

```

        NewMLine.LinetypeScale = Utility.DistanceToReal( _
        MLineSetupFrm.Text2.Text, acDefaultUnits)
        NewMLine.Lineweight = MLineSetupFrm.GetLWIndex( _
        MLineSetupFrm.ImageCombo2.SelectedItem.Index)
        StartPt(0) = VarPoint(0)
        StartPt(1) = VarPoint(1)
        StartPt(2) = VarPoint(2)
    Else
        Set AcadMLine = Nothing
        Exit Do
    End If
Loop
Else
    Exit Do
End If
Loop
SaveSetting "BspMLine", "Position", "Left", MLineSetupFrm.Left
SaveSetting "BspMLine", "Position", "Top", MLineSetupFrm.Top
Unload MLineSetupFrm
End Sub

```

Wie Sie bereits bemerkt haben dürften, erinnert dieses Programm doch stark an das Beispiel zur Erstellung von Linienzügen. Das zeigt uns, dass sich mit einigen wenigen Handgriffen ein bereits erstelltes Programm auf ein neues Objekt (z.B. Multilinie) umschreiben lässt. Aus diesem Grund folgen nur einige Erläuterungen zu den Änderungen, die am Linienprogramm vorgenommen werden mussten, damit wir dieses Ergebnis erhalten. Zum einen sind da natürlich die Änderungen am Formular, aber das ist ja das geringste Problem. Der weitaus wichtigere Teil der Programmanpassung liegt im Inneren der Startprozedur:

```

If TypeName(NewMLine) = "Nothing" Then
    VertList(0) = StartPt(0)
    VertList(1) = StartPt(1)
    VertList(2) = StartPt(2)
    VertList(3) = VarPoint(0)
    VertList(4) = VarPoint(1)
    VertList(5) = VarPoint(2)
    Set NewMLine = ModelSpace.AddMLine(VertList)
Else
    VarPoints = NewMLine.Coordinates
    ReDim NewPoints(UBound(VarPoints) + 3) As Double
    For i = 0 To UBound(VarPoints)
        NewPoints(i) = VarPoints(i)
    Next
    NewPoints(UBound(VarPoints) + 1) = VarPoint(0)
    NewPoints(UBound(VarPoints) + 2) = VarPoint(1)
    NewPoints(UBound(VarPoints) + 3) = VarPoint(2)
    NewMLine.Coordinates = NewPoints
    NewMLine.Update
End If

```

Da Multilinen, anders als Linien, mehrere Stützpunkte haben können, die alle in der Objekteigenschaft *Coordinates* gespeichert werden, müssen wir uns diese Eigenschaft zu Nutze machen. Sie werden sich natürlich fragen, warum sammeln wir nicht einfach alle Koordinaten und erstellen zum Schluss mit dieser Koordinatenliste die Multilinie, dann brauchen wir die Eigenschaft ja nicht? Nun der Benutzer soll natürlich sofort ein Ergebnis auf der Zeichnungs-

fläche erhalten. Aus diesem Grund wird eine Multiline erstellt, sobald der Benutzer die ersten beiden Punkte bestimmt hat. Alle weiteren Punkte werden nach und nach der neuen Multiline hinzugefügt. So wird die Multiline immer bis zum letzten gewählten Punkt gezogen. Aber im Einzelnen:

```
If TypeName(NewMLine) = "Nothing" Then
    VertList(0) = StartPt(0)
    VertList(1) = StartPt(1)
    VertList(2) = StartPt(2)
    VertList(3) = VarPoint(0)
    VertList(4) = VarPoint(1)
    VertList(5) = VarPoint(2)
    Set NewMLine = ModelSpace.AddMLine(VertList)
Else
```

Im ersten Teil der *If*-Anweisung wird geprüft, ob bereits eine neue Multiline erzeugt wurde. Hat der Benutzer die ersten beiden Punkte gewählt, ist dies nicht der Fall, da ja noch kein Verweis auf eine Multiline der Objektvariablen *NewMLine* übergeben wurde (*NewMLine* = *Nothing*). Also werden alle Koordinaten in einem Array (*VertList*) gesammelt und mithilfe der Anweisung *AddMLine*, die in diesem Beispiel auf den Modellbereich angewendet wird, eine neue Multiline erzeugt. Als Ergebnis wird der Objektvariablen ein Verweis auf diese Multiline zugeordnet (*TypeName(NewMLine)* = „IAcadMLine“). Diese Multiline wird jetzt für die nächsten Punkte verwendet. Jedes Mal, wenn der Benutzer einen Punkt gewählt hat, werden folgende Anweisungen ausgeführt:

```
VarPoints = NewMLine.Coordinates
ReDim NewPoints(UBound(VarPoints) + 3) As Double
For i = 0 To UBound(VarPoints)
    NewPoints(i) = VarPoints(i)
Next
NewPoints(UBound(VarPoints) + 1) = VarPoint(0)
NewPoints(UBound(VarPoints) + 2) = VarPoint(1)
NewPoints(UBound(VarPoints) + 3) = VarPoint(2)
NewMLine.Coordinates = NewPoints
```

Zuerst werden die bereits vorhandenen Koordinaten der Multiline ermittelt (Variable *VarPoints*). Da wir ja jetzt mehr Koordinaten haben, als die Multiline derzeit aufnehmen kann, muss ein neues Array deklariert werden, das die vorhandenen Koordinaten und zusätzlich die drei neuen (X, Y, Z) aufnehmen kann. Diesem Array übergeben wir einfach die vorhandenen Koordinaten der Multiline und hängen dann den gewählten Punkt hinten dran. Dieses Array kann nun an die Eigenschaft *Coordinates* weitergereicht werden, womit die Multiline einen weiteren Stützpunkt erhält. Wenn der Benutzer diese Multiline abschließt, indem die Schleife verlassen wird, und bevor mit einer neuen Multiline begonnen wird, muss der Verweis auf die Multiline in der Objektvariablen entfernt werden.

```
Set AcadMLine = Nothing
```

Dadurch ist gewährleistet, dass beim nächsten Durchlauf eine neue Multiline erzeugt wird, statt mit dem Anhängen an die vorherige fortzusetzen.

8.3.14 Punktojekt (AcadPoint)

Die Klasse *AcadPoint* stellt einen einfachen Punkt in AutoCAD dar. Aus diesem Grund enthält diese Klasse auch nur wenige Eigenschaften und Methoden.

Sie können einen Verweis auf einen Punkt erstellen, indem Sie mit der Funktion *AddPoint* einen neuen Punkt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch einen vorhandenen Punkt über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Einfügekpunkt des Punktojekts bestimmen

Der Einfügekpunkt dieses Objekts wird mit der Eigenschaft *Coordinates* verwaltet. Warum in das Objekt diese Eigenschaft und nicht *InsertionPoint* integriert wurde, das ist schwer nachzuvollziehen, aber sei es, wie es sei. Im Gegensatz zur *Coordinates*-Eigenschaft anderer Objekte können Sie für einen Punkt auch nur eine 3D-Koordinate festlegen. Das Array hat dabei einen Bereich von 0 bis 2 (X, Y, Z).

```
Dim InsPoint(2) As Double
InsPoint(0) = 100.137
InsPoint(1) = 125.125
InsPoint(2) = 20.114
ActPoint.Coordinates = VarPoint
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Punkts fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Punkts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Punkt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Punkts zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Punkts an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Punkts an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Punkts an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke des Punkts an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Punkts zurück.

Tabelle 8.56: Eigenschaften eines Punktojekts

Eigenschaft	Beschreibung
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadPoint</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für den Punkt zurück oder legt diesen fest.
<i>Thickness</i>	Legt die Objekthöhe des Punkts fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Punkts steuern.

Tabelle 8.56: Eigenschaften eines Punktobjekts (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Punkts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Punkts ab.
<i>Highlight</i>	Hebt den Punkt hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Punkts mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt einen Punkt als Resultat einer Spiegelung mit diesem Punkt.
<i>Mirror3D</i>	Erstellt einen neuen Punkt als Resultat einer räumlichen Spiegelung dieses Punkts.
<i>Move</i>	Verschiebt den Punkt von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht den Punkt um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Punkt um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert den Punkt um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Punkts.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf den Punkt an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Punkts im AutoCAD-Zeichnungsfenster.

Tabelle 8.57: Methoden eines Punktobjekts

8.3.15 Symbolobjekt (AcadShape)

Die Klasse *AcadShape* stellt Ihnen die Eigenschaften und Methoden eines *Shape*-Objekts, auch Symbol genannt, zur Verfügung. Im Folgenden wird deswegen auch nur noch von Symbolen die Rede sein. Symbole sind Objekte, die aus Linien, Bögen und Kreisen bestehen, die in einer *shx*-Datei definiert wurden.

Sie können einen Verweis auf ein solches Symbol erstellen, indem Sie mit der Funktion *AddShape* eine begrenzte Fläche dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene Region über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften des Symbolobjekts

Der Einfügekpunkt dieses Objekts wird in der Eigenschaft *InsertionPoint* gespeichert. Die Koordinaten dieses Punkts werden als 3D-Array zurückgegeben bzw. müssen als 3D-Array an die Eigenschaft übergeben werden, wenn Sie den Einfügekpunkt anpassen möchten.

```
Dim InsPoint(2) As Double
varPt = Utility.GetPoint(, "Einfügekpunkt:")
If TypeName(varPt) = "Double()" Then
    InsPoint(0) = varPt(0)
    InsPoint(1) = varPt(1)
    InsPoint(2) = varPt(2)
    ActShape.InsertionPoint = InsPoint
End If
```

Mithilfe der Eigenschaft *Height* können Sie die Größe des Symbols bestimmen. Die Größe eines Symbols entspricht in etwa den Eigenschaften *XScaleFactor*, *YScaleFactor* einer Blockreferenz. Da ein Symbol keine gesonderten Faktoren in X und Y aufnehmen kann, wird die Gesamtskalierung in der Eigenschaft *Height* verwaltet. *ScaleFactor* bestimmt hingegen den Breitenfaktor des Symbols oder gibt den aktuellen Wert zurück. Im Gegensatz zur Eigenschaft *Height* wird hiermit nur ein Breitenfaktor bestimmt, der eine ähnliche Wirkung hat wie bei Textobjekten.

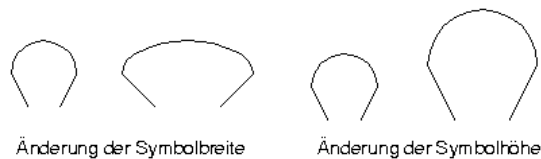


Abbildung 8.61: Änderungen an den Eigenschaften *ScaleFactor* und *Height*

Rotation legt den aktuellen Drehwinkel des Symbols fest oder gibt diesen zurück. Der Wert des Drehwinkels hängt dabei von der Systemvariablen *ANGBASE* ab. Der Neigungswinkel des Symbols kann mit der Eigenschaft *ObliqueAngle* festgelegt werden. Beide Winkel, sowohl *Rotation* als auch *ObliqueAngle*, werden in Radiant angegeben. Zur Umwandlung in ein anderes Format können Sie die entsprechenden Funktionen der Klasse *Utility* verwenden (Kapitel 6.3).

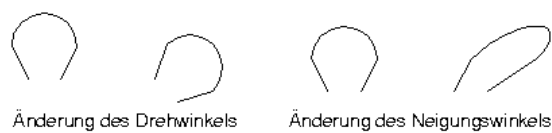


Abbildung 8.62: Änderungen der Winkeleigenschaften

Die wohl wichtigste Eigenschaft des Symbolobjekts ist die *Name*-Eigenschaft. Diese bestimmt, wie das Symbol an sich aussehen soll. Der Name, den Sie hier festlegen, entspricht dem Namen eines Symbols aus einer *shx*-Datei. Bevor Sie den Namen ändern, sollten Sie sicher sein, dass das Symbol geladen wurde. Gegebenenfalls können Sie die *LoadShapeFile*-Anweisung (Kapitel 6.2.3) des *Dokument*-Objekts verwenden, um dies nachzuholen.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Symbols fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Symbols zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Symbol ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Symbols zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Symbols an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Symbols an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Symbols an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für das Symbol an oder legt diese fest.
<i>Normal</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Symbols zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadShape</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 8.58: Weitere Eigenschaften des Symbolobjekts

Eigenschaft	Beschreibung
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Symbol zurück oder legt diesen fest.
<i>Thickness</i>	Legen Sie mithilfe dieser Eigenschaft die Objekthöhe des Symbols fest oder rufen Sie den aktuellen Wert ab.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Symbols steuern.

Tabelle 8.58: Weitere Eigenschaften des Symbolobjekts (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Symbols ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Symbols ab.
<i>Highlight</i>	Hebt das Symbol hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Symbols mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Kopie des Symbols als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine Kopie des Symbols als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt das Symbol von den Ursprungs- auf die Zielkoordinaten.
<i>Rotate</i>	Dreht das Symbol um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht das Symbol um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Symbol um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Symbols.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Symbol an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Symbols im AutoCAD-Zeichnungsfenster.

Tabelle 8.59: Eigenschaften des Symbolobjekts

8.3.16 Textobjekte

In AutoCAD wird zwischen zwei verschiedenen Textobjekten unterschieden. Zum einen gibt es den einzeiligen Text, mit dem eine einzeilige Zeichenfolge dargestellt werden kann. Das zweite Textobjekt ist der Absatztext, mit dem formatierte Absätze in der Zeichnung dargestellt werden.

Einzeiliger Text (AcadText)

Diese Klasse stellt Ihnen die Eigenschaften und Methoden eines einzeiligen Textes zur Verfügung.

Sie können einen Verweis auf einen Text erstellen, indem Sie mit der Funktion *AddText* unter Angabe der Zeichenfolge, des Einfügepunkts und der Texthöhe einen Text dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch einen vorhandenen Text über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Einfügepunkt bestimmen

Das Textobjekt bietet zwei unterschiedliche Einfügepunkte, die in den Eigenschaften *InsertionPoint* und *TextAlignmentPoint* verwaltet werden. Welcher Einfügepunkt zur Positionierung des Textes verwendet wird, hängt von der Einstellung der *Alignment*-Eigenschaft ab. In jedem Fall geben die Eigenschaften ein 3D-Array mit den entsprechenden Koordinaten zurück bzw. erwarten ein solches zur Festlegung eines neuen Einfügepunkts.

Textausrichtung festlegen

Einzeilige Text können horizontal ausgerichtet werden, wobei die aktuelle Einstellung in der Eigenschaft *Alignment* gespeichert wird. Je nachdem, welchen Wert Sie für diese Eigenschaft setzen, werden die Eigenschaften *InsertionPoint* und *TextAlignmentPoint* verwendet.

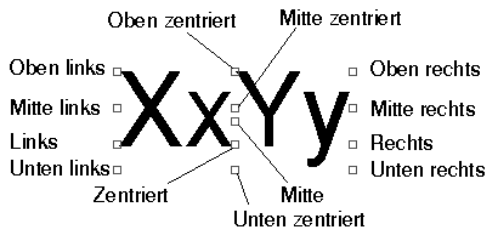


Abbildung 8.63: Textausrichtungen

Für die Eigenschaft *Alignment* stehen Ihnen eine Reihe von Konstanten zur Verfügung, wobei die Konstanten *acAlignmentAligned* und *acAlignmentFit* besondere Ausrichtungen einstellen. In beiden Fällen wird der Text zwischen die Punkte, die in *InsertionPoint* und *TextAlignmentPoint* gespeichert sind, eingepasst. Der Unterschied ist, dass bei der Ausrichtung mit *acAlignmentFit* die ursprüngliche Texthöhe erhalten bleibt, im anderen Fall die Texthöhe automatisch angepasst wird. Welche Konstante Sie für welche Ausrichtung verwenden können, entnehmen Sie der Tabelle 8.60.

Konstante	Beschreibung	Wert
<i>acAlignmentLeft</i>	Links	0
<i>acAlignmentCenter</i>	Zentriert	1
<i>acAlignmentRight</i>	Rechts	2
<i>acAlignmentAligned</i>	Ausgerichtet	3
<i>acAlignmentMiddle</i>	Mitte	4
<i>acAlignmentFit</i>	Eingepasst	5
<i>acAlignmentTopLeft</i>	Oben Links	6
<i>acAlignmentTopCenter</i>	Oben Zentriert	7
<i>acAlignmentTopRight</i>	Oben Rechts	8
<i>acAlignmentMiddleLeft</i>	Mitte Links	9
<i>acAlignmentMiddleCenter</i>	Mitte Zentriert	10
<i>acAlignmentMiddleRight</i>	Mitte Rechts	11
<i>acAlignmentBottomLeft</i>	Unten Links	12
<i>acAlignmentBottomCenter</i>	Unten Zentriert	13
<i>acAlignmentBottomRight</i>	Unten Rechts	14

Tabelle 8.60: Konstanten der Alignment-Eigenschaft

Weitere Texteigenschaften

Die Höhe des Textes wird mit der *Height*-Eigenschaft festgelegt, wenn *Alignment* nicht auf den Wert *acAlignmentAligned* eingestellt wurde. Sie müssen für diese Eigenschaft positive reelle Zahlen verwenden, wenn Sie die Höhe des Textes ändern. *Rotation* legt den Drehwinkel und *ObliqueAngle* den Neigungswinkel des Textes fest. Die *ScaleFactor*-Eigenschaft enthält den Breitenfaktor.

Die *Backward*-Eigenschaft gibt zurück, ob der Text rückwärts oder vorwärts geschrieben wird, bzw. legt dies fest. Mit *UpsideDown* können Sie festlegen bzw. kontrollieren, ob der Text auf dem Kopf (*TRUE*) oder normal (*FALSE*) dargestellt wird.

Die Zeichenfolge, die mit einem Textobjekt angezeigt werden soll, wird in der Eigenschaft *TextString* verwaltet, wobei deren Länge 256 Zeichen nicht überschreiten darf. Der Stil des Textes kann mit *StyleName* angepasst werden. Dabei wird wie bei der *Linetype*- oder *Layer*-Eigenschaft nur der Name des Textstils zugewiesen und nicht ein Verweis auf das Textstil-Objekt, das in der *TextStyles*-Auflistung vorhanden ist. Sie sollten also gegebenenfalls vorher prüfen, ob der angegebene Textstilname wirklich in der Zeichnung vorhanden ist. Folgende Codezeilen lösen diese Aufgabe:

```
Dim ActStil As AcadTextStyle
Set ActStil = TextStyles(StyleName)
If TypeName(ActStil) = "Nothing" Then
    MsgBox "Textstil konnte nicht gefunden werden", vbCritical
Else
    ActText.StyleName = Name
End If
```

Der Zugriff auf einzelne Textstile über die *TextStyles*-Auflistung wird in *Kapitel 7.7* genauer beschrieben.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Textes fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Textes zurück. Der Handle ist der eindeutige Identifier eines AutoCAD-Objekts.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Text ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Textes zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Textes an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Textes an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Textes an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für den Text an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Textes zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadText</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für den Text zurück oder legt diesen fest.
<i>Thickness</i>	Diese Eigenschaft speichert die Objekthöhe des Textes. Sie können den aktuellen Wert auslesen oder zurückschreiben.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Textes steuern.

Tabelle 8.61: Weitere Eigenschaften eines einzeiligen Textes

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.

Tabelle 8.62: Methoden eines einzeiligen Textes

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Textes ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Textes ab.
<i>Highlight</i>	Hebt den Text in der AutoCAD-Zeichnung hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Textes mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt einen Text als Resultat einer Spiegelung mit diesem Text.
<i>Mirror3D</i>	Erstellt einen neuen Text als Resultat einer räumlichen Spiegelung dieses Textes.
<i>Move</i>	Verschiebt den Text von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht den Text um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Text um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert den Text um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Textes.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf den Text an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Textes im AutoCAD-Zeichnungsfenster.

Tabelle 8.62: Methoden eines einzeiligen Textes (Forts.)

Anwendungsbeispiel

(Beispielprojekt: 08\AcadText.dvb!ThisDrawing.CreateText)

Im folgenden Beispielprojekt erhält der Benutzer die Möglichkeit, einzeilige Texte in der aktuellen Zeichnung zu erstellen. Dafür kann er in einem Formular parallel zur Erstellung bereits einige Einstellungen vornehmen, darunter auch die Auswahl eines Modus zur Drehwinkelauswahl.

The image shows a small dialog box titled "neuer Text" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the dialog, there are six labeled input fields arranged vertically: "Textstil:" (a dropdown menu), "Texthöhe:" (a text input field), "Drehwinkel:" (a dropdown menu), "Ausrichtung:" (a dropdown menu), "Textbreite:" (a text input field), and "Neigung:" (a text input field). The dialog has a dotted border and a close button (X) in the top right corner.

Abbildung 8.64: Formular des Beispielprojekts

Initialisierungsprozedur des Formulars

In dieser Prozedur, die ja bekanntlich beim Laden des Formulars gestartet wird, werden die Inhalte der einzelnen Steuerelemente eingerichtet. Zu beachten gilt hier insbesondere, dass bei der Zusammenstellung der Textstilnamen aus der *TextStyles*-Auflistung die Textstile ausgeschlossen werden, die aus externen Referenzen stammen.

```

Private Sub UserForm_Initialize()
On Error Resume Next
Me.Left = GetSetting("BspText", "Position", "Left", Me.Left)
Me.top = GetSetting("BspText", "Position", "Top", Me.top)
For Each TStil In ThisDrawing.TextStyles
    If Instr(1, TStil.Name, "|") = 0 Then
        ComboBox1.AddItem TStil.Name
    End If
Next
ComboBox1.Text = ThisDrawing.ActiveTextStyle.Name
ComboBox2.AddItem "in der Zeichnung bestimmen"
ComboBox2.AddItem "nach Zeichnungsrand ausrichten"
ComboBox2.ListIndex = 0
ComboBox3.AddItem "Links"
ComboBox3.AddItem "Zentriert"
ComboBox3.AddItem "Rechts"
ComboBox3.AddItem "Ausgerichtet"
ComboBox3.AddItem "Mitte"
ComboBox3.AddItem "Einpassen"
ComboBox3.AddItem "Oben links"
ComboBox3.AddItem "Oben zentriert"
ComboBox3.AddItem "Oben rechts"
ComboBox3.AddItem "Mitte links"
ComboBox3.AddItem "Mitte zentriert"
ComboBox3.AddItem "Mitte rechts"
ComboBox3.AddItem "Unten links"
ComboBox3.AddItem "Unten zentriert"
ComboBox3.AddItem "Unten rechts"
ComboBox3.ListIndex = 0
TextBox1.Text = ThisDrawing.Utility.RealToString( _
    ThisDrawing.GetVariable("TEXTSIZE"), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
TextBox2.Text = ThisDrawing.Utility.RealToString(1, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
TextBox3.Text = ThisDrawing.Utility.AngleToString(0, _
    ThisDrawing.GetVariable("AUNITS"), _
    ThisDrawing.GetVariable("AUPREC"))
End Sub

```

Startprozedur des Projekts

Dieses kleine Programm enthält nur eine Prozedur, die gleichzeitig die Startprozedur ist. Die Funktionsweise ist dabei recht einfach. Zunächst wird das Formular gestartet, in dem der Benutzer die Einstellungen für den neuen Text vornehmen kann. Anschließend wird eine *Do..Loop*-Schleife geöffnet, die so lange durchlaufen wird, bis der Benutzer die Anweisung „X“ bzw. „exit“ in der Eingabeaufforderung eingibt.

```

Public Sub CreateText()
Dim InsPoint As Variant
Dim NewText As AcadText
Dim Prompt As String
Const PI = 3.14159265358979
On Error Resume Next
TextFrm.Show 0

```



```

Do
    InsPoint = Empty
    Utility.InitializeUserInput 128, "eXit"
    Err.Clear
    Prompt = "Texteinfügemarke wählen oder eXit:"
    InsPoint = Utility.GetPoint(, Prompt)
    If TypeName(InsPoint) = "Double()" Then

```

Hat der Benutzer hingegen einen Einfügemarke für den Text in der Zeichnung gewählt, wird mit der Texterstellung fortgefahren. Die interessanteste Aufgabe fällt hierbei der Bestimmung des Drehwinkels für den Text zu. Hat der Benutzer die Auswahl IN DER ZEICHNUNG BESTIMMEN getroffen, wird der Benutzer aufgefordert, einen Winkel in der Zeichnung auszuwählen. Im anderen Fall bedienen wir uns des Inhalts der Systemvariablen *VIEWTWIST*, die den aktuellen Drehwinkel der Zeichnung beinhaltet. Um den korrekten Winkel für den Text zu erhalten, muss nun von einem Vollkreis ($2 * \text{PI}$) dieser Wert abgezogen werden.

```

    If TextFrm.ComboBox2.ListIndex = 0 Then
        Prompt = "Winkel wählen:"
        Angle = Utility.GetOrientation(InsPoint, Prompt)
    Else
        Angle = 2 * PI - GetVariable("VIEWTWIST")
    End If

```

Die Zeichenfolge des neuen Textes fragen wir einfach mit einer Eingabebox ab. Hat der Benutzer keine leere Zeichenfolge ("") festgelegt, kann mit der Erzeugung des Textes fortgefahren werden.

```

    Prompt = "Geben Sie die Zeichenfolge für den neuen Text ein"
    sValue = InputBox(Prompt, "Neuer Text")
    If sValue <> "" Then

```

Die restlichen Einstellungen werden dem Formular entnommen und dem neuen Textobjekt zugewiesen.

```

        tHeight = Utility.DistanceToReal(TextFrm.TextBox1.Text, _
            acDefaultUnits)
        Set NewText = ModelSpace.AddText(sValue, InsPoint, _
            tHeight)
        NewText.Rotation = Angle
        NewText.ObliqueAngle = Utility.AngleToReal( _
            TextFrm.TextBox3.Text, GetVariable("AUNITS"))
        NewText.StyleName = TextFrm.ComboBox1.Text
        NewText.Alignment = TextFrm.ComboBox3.ListIndex
    End If
Else
    If Err.Description = "Benutzereingabe " & _
        "ist ein Schlüsselwort" Then
        Exit Do
    End If
End If

Loop
SaveSetting "BspText", "Position", "Left", TextFrm.Left
SaveSetting "BspText", "Position", "Top", TextFrm.top
Unload TextFrm
End Sub

```

Absatztext (AcadMText)

Ein mehrzeiliger Text, so genannter Absatztext, wird in AutoCAD durch die Klasse *AcadMText* dargestellt. Die Klasse *AcadMText* enthält alle Eigenschaften und Methoden, die Sie auf einen Absatztext anwenden können.

Sie erstellen einen Verweis auf einen Absatztext, indem Sie mit der Funktion *AddMText* ein neues MText-Objekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch einen vorhandenen Absatztext über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Absatztexte unterscheiden sich von einzeiligen Texten nicht nur in der Darstellung, sondern auch in den Eigenschaften. Da dieses Objekt den Text anders verwaltet als der einzeilige Text, besitzt es auch ganz spezielle Eigenschaften, mit denen das Objekt in seiner Darstellung angepasst werden kann.

MText-Eigenschaften

Die *AttachmentPoint*-Eigenschaft bestimmt neben der Position des Einfügepunkts am Textobjekt auch die Ausrichtung des Textes innerhalb des Absatztextes. Sie können für diese Eigenschaft die verschiedensten Konstanten verwenden, die folgende Wirkung auf die Darstellung des Textes haben.

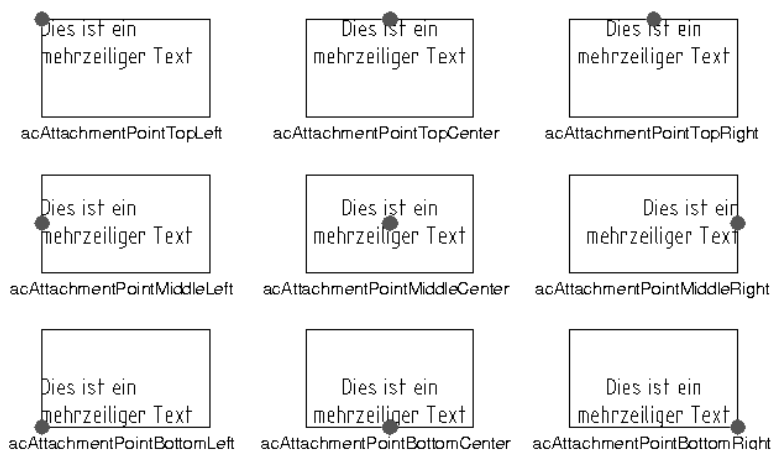


Abbildung 8.65: Konstanten und deren Wirkung auf die AttachmentPoint-Eigenschaft

Die *InsertionPoint*-Eigenschaft speichert den Einfügepunkt des Textes als 3D-Array. Wie bereits erwähnt, hat die Einstellung von *AttachmentPoint* Auswirkungen auf dessen Positionsdefinition innerhalb des MText-Objekts. Ändern Sie diese Eigenschaft, dann ändern sich ebenfalls die Koordinaten des Einfügepunkts und somit auch die Eigenschaft *InsertionPoint*.

Mit der *LineSpacingFactor*-Eigenschaft bestimmen Sie den Abstand zwischen den einzelnen Zeilen des Absatztextes. Sie können den Zeilenabstand als Vielfaches des Einzelzeilenabstands oder als absoluten Wert angeben. Der angezeigte Zeilenabstand ist abhängig vom aktuellen Wert der Eigenschaft *LineSpacingStyle*. Diese legt den Typ für den Zeilenabstand fest. Sie können folgende Konstanten für diese Eigenschaft verwenden.

Konstante	Beschreibung	Wert
<i>AcLineSpacingStyleAtLeast</i>	Zeilenabstand wird automatisch angepasst, wenn bestimmte Zeichen zu groß für den Zeilenversatz sind.	1
<i>acLineSpacingStyleExactly</i>	Legt einen einheitlichen Zeilenabstand für das MText-Objekt fest.	2

Tabelle 8.63: Konstanten der *LineSpacingStyle*-Eigenschaft

Width legt die Breite des Absatztextes fest oder gibt die aktuelle Breite zurück. *Height* bestimmt die Höhe des Textes, nicht die des MText-Objekts. Die Höhe wird dabei als eine Art Skalierfaktor für die Breite und Höhe des MText-Objekts verwendet.

DrawingDirection bestimmt die Schreibrichtung innerhalb des Absatztextes. Je nachdem, welche Einstellung Sie für diese Eigenschaft vornehmen, entspricht die Einstellung der Eigenschaft *Width* der Breite (*acLeftToRight*) oder der Höhe (*acBottomToTop*) des Textobjekts. Es gelten folgende Einstellungskonstanten:

Konstante	Beschreibung	Wert
<i>acLeftToRight</i>	Der Text wird von links nach rechts geschrieben.	1
<i>acRightToLeft</i>	Wurde reserviert, wird aber noch nicht verwendet.	2
<i>acTopToBottom</i>	Der Text wird von oben nach unten geschrieben.	3
<i>acBottomToTop</i>	Wurde reserviert, wird aber noch nicht verwendet.	4
<i>acByStyle</i>	Wurde reserviert, wird aber noch nicht verwendet.	5

Tabelle 8.64: Konstanten der *DrawingDirection*-Eigenschaft

Rotation, *StyleName* und *TextString* haben dieselbe Funktion wie beim einzeligen Text. *Rotation* legt den Drehwinkel des Objekts fest, *StyleName* bestimmt den aktuellen Textstil und *TextString* enthält die Zeichenfolge des MTextes, wobei diese Eigenschaft viel komplexere Zeichenfolgen aufnehmen kann. Da der Text auch sämtliche Steuerzeichen enthält, sollten Sie vorsichtig mit der *TextString*-Eigenschaft umgehen und zur Bearbeitung ein Editor-Steurelement verwenden, das diese Steuerzeichen verarbeiten kann.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Absatztextes fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Absatztextes zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Absatztext ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Absatztextes zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).

Tabelle 8.65: Weitere Eigenschaften des Absatztextes

Eigenschaft	Beschreibung
<i>Layer</i>	Zeigt den Layer des Absatztextes an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Absatztextes an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Absatztextes an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für den Absatztext an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Absatztextes zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadMText</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für den Absatztext zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Absatztextes steuern.

Tabelle 8.65: Weitere Eigenschaften des Absatztextes (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Absatztextes ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Absatztextes ab.
<i>Highlight</i>	Hebt den Absatztext hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Absatztextes mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt einen Absatztext als Resultat einer Spiegelung mit diesem Absatztext.
<i>Mirror3D</i>	Erstellt einen neuen Absatztext als Resultat einer räumlichen Spiegelung dieses Absatztextes.

Tabelle 8.66: Methoden des Absatztextes

Prozedur	Beschreibung
<i>Move</i>	Verschiebt den Absatztext von den Ursprungs- auf die Zielkoordinaten.
<i>Rotate</i>	Dreht den Absatztext um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Absatztext um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert den Absatztext um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Absatztextes.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf den Absatztext an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Absatztextes im AutoCAD-Zeichnungsfenster.

Tabelle 8.66: Methoden des Absatztextes (Forts.)

Anwendungsbeispiel

(Beispielprojekt: 08\AcadText.dvb!ThisDrawing.CreateMText)

Das Beispielprojekt zum Thema Absatztext ist ähnlich aufgebaut wie das zum einzeiligen Text. Auch hier kann der Benutzer über ein Formular die Eigenschaften des zu erzeugenden Textes festlegen.

Initialisierungsprozedur des Formulars

In der Initialisierungsprozedur werden wieder einige Einstellungen am Formular vorgenommen, wenn dieses geladen wird. Als Erstes wird das Steuerelement *ComboBox1* mit den Textstilen der Zeichnung gefüllt und der aktuelle Textstil der Zeichnung (*ActiveTextStyle.Name*) als ausgewählter Stil festgelegt.

```
Private Sub UserForm_Initialize()
On Error Resume Next
For Each TStil In ThisDrawing.TextStyles
    If InStr(1, TStil.Name, "|") = 0 Then
        ComboBox1.AddItem TStil.Name
    End If
Next
ComboBox1.Text = ThisDrawing.ActiveTextStyle.Name
```

Als Nächstes werden die Comboboxen für die Auswahl des Modus zur Ermittlung des Drehwinkels, der Ausrichtung und des Zeilenabstandsstils des neuen MTextes mit den entsprechenden Auswahlmöglichkeiten gefüllt.

```
ComboBox2.AddItem "in der Zeichnung bestimmen"
ComboBox2.AddItem "nach Zeichnungsrand ausrichten"
ComboBox2.ListIndex = 0
ComboBox3.AddItem "Oben links"
ComboBox3.AddItem "Oben zentriert"
ComboBox3.AddItem "Oben rechts"
ComboBox3.AddItem "Mitte links"
ComboBox3.AddItem "Mitte zentriert"
ComboBox3.AddItem "Mitte rechts"
ComboBox3.AddItem "Unten links"
ComboBox3.AddItem "Unten zentriert"
```

```

ComboBox3.AddItem "Unten rechts"
ComboBox3.ListIndex = 0
ComboBox4.AddItem "Mindestens"
ComboBox4.AddItem "Genau"
ComboBox4.ListIndex = 0

```

Zum Schluss werden noch die Vorgabewerte für Texthöhe, Textbreite und Zeilenabstandsfaktor an die entsprechende Textbox übergeben.

```

TextBox1.Text = Util.RealToString(ThisDrawing.GetVariable( _
    "TEXTSIZE"))
TextBox2.Text = Util.RealToString(1)
TextBox3.Text = Util.RealToString(1)
End Sub

```

Startprozedur

Die Startprozedur ist auch gleichzeitig die Hauptprozedur dieses kleinen Programms, in der nicht nur der MText erzeugt, sondern auch dessen Eigenschaften mit den Einstellungen des Formulars angepasst werden. Der Code läuft dabei wieder in einer *Do Loop*-Schleife ab, die erst beendet wird, wenn der Benutzer keinen Punkt mehr in der Zeichnung gewählt hat. Der erste Teil des Programmes ist dabei fast mit dem des Beispielprojektes für einzeliligen Text identisch.

```

Public Sub CreateMText()
Dim InsPoint As Variant
Dim NewMText As AcadMText
Dim ActStil As AcadTextStyle
Const PI = 3.14159265358979
On Error Resume Next
Set Util = MyUtil
MTextFrm.Show 0
MTextFrm.Left = GetSetting("BspMText", "Position", "Left", _
    MTextFrm.Left)
MTextFrm.top = GetSetting("BspMText", "Position", "Top", _
    MTextFrm.top)
Do
    Err.Clear
    InsPoint = Empty
    Prompt = "Texteinfügapunkt wählen oder eXit:"
    InsPoint = Util.GetPoint(, Prompt, "eXit")
    If TypeName(InsPoint) = "Double()" Then
        If TextFrm.ComboBox2.ListIndex = 0 Then
            Prompt = "Winkel wählen:"
            Angle = Util.GetOrientation(InsPoint, Prompt)
        Else
            Angle = 2 * PI - GetVariable("VIEWTWIST")
        End If
        tWidth = Util.DistanceToReal(MTextFrm.TextBox2.Text)
        MTextEditFrm.Show
        If MTextEditFrm.OK = True Then

```

Der einzige Unterschied zum einzeliligen Text besteht in der Verarbeitung der Eigenschaften für das MText-Objekt.

```

With NewMText
  If ActiveSpace = acModelSpace Then
    Set NewMText = ModelSpace.AddMText(InsPoint, _
      tWidth, MTextEditFrm.GetMText)
  Else
    Set NewMText = PaperSpace.AddMText(InsPoint, _
      tWidth, MTextEditFrm.GetMText)
  End If
  .Rotation = Angle
  .StyleName = MTextFrm.ComboBox1.Text
  .AttachmentPoint = MTextFrm.ComboBox3.ListIndex
  .LineSpacingFactor = Util.DistanceToReal( _
    .TextBox3.Text)
  .LineSpacingStyle = MTextFrm.ComboBox4.ListIndex
End With
End If
ElseIf UCase(InsPoint) = "EXIT"
  Exit Do
End If
Loop
SaveSetting "BspMText", "Position", "Left", MTextFrm.Left
SaveSetting "BspMText", "Position", "Top", MTextFrm.top
Unload MTextFrm
End Sub

```

8.3.17 Strahl (AcadRay)

Die Klasse *AcadRay* stellt Ihnen die Eigenschaften und Methoden von einseitig begrenzten Linien, den so genannten Strahlen, zur Verfügung. Diese Objekte dienen im Allgemeinen als Konstruktionslinien, haben einen festen Startpunkt und erstrecken sich in einer Richtung bis ins Unendliche. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen `strahl` oder `_ray` erzeugen.

Sie erstellen einen Verweis auf einen solchen Strahl, indem Sie mit der Funktion *AddRay* einen Strahl dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch einen vorhandenen Strahl über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften eines Strahlobjekts

Der Startpunkt des Strahls wird in der Eigenschaft *BasePoint* verwaltet. Wenn Sie die Koordinaten dieser Eigenschaft anpassen, dann bleibt die aktuelle Richtung des Strahls erhalten, der Strahl wird sozusagen auf die neue Position verschoben. Die Eigenschaft *SecondPoint* speichert den aktuellen Richtungspunkt des Strahls. Der Richtungspunkt besitzt allerdings keinen festen Wert, sondern ändert sich immer, wenn der Benutzer den Zeichnungsausschnitt vergrößert oder verkleinert. Dieser Punkt speichert wie gesagt nur die Koordinaten, durch die der Strahl verläuft.

Mit der *DirectionVector*-Eigenschaft können Sie den Richtungsvektor des Strahls bestimmen oder abrufen. Der Richtungsvektor zeigt dabei die Ausrichtung des Strahls unabhängig von den Einheiteneinstellungen des Systems an. Die jeweiligen Vektoren liegen in einem Wertebereich von -1 bis 1 (siehe Kapitel 8.1.2).

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Strahls fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Strahls zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Strahl ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Strahls zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Strahls an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Strahls an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Strahls an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für den Strahl an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die ObjectID des Strahls zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadRay</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für den Strahl zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Strahls steuern.

Tabelle 8.67: Weitere Eigenschaften eines Strahls

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Strahls ab.

Tabelle 8.68: Methoden eines Strahls

Prozedur	Beschreibung
<i>GetXData</i>	Ruft die erweiterten Daten des Strahls ab.
<i>Highlight</i>	Hebt den Strahl hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Strahls mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Kopie des Strahls als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine Kopie des Strahls als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt den Strahl von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht den Strahl um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Strahl um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert den Strahl um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Strahls.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf den Strahl an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Strahls im AutoCAD-Zeichnungsfenster.

Tabelle 8.68: Methoden eines Strahls (Forts.)

8.3.18 Konstruktionslinien (AcadXLine)

Die Klasse *AcadXLine* stellt Ihnen die Eigenschaften und Methoden von Konstruktionslinien zur Verfügung. Diese Objekte haben einen festen Startpunkt und erstrecken sich in beide Richtungen bis ins Unendliche. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen *klinie* oder *_xline* erzeugen.

Sie erstellen einen Verweis auf eine Konstruktionslinie, indem Sie mit der Funktion *AddXLine* unter Angabe des Start- und Endpunkts als 3D-Array eine Konstruktionslinie dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene Konstruktionslinie über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften der Konstruktionslinie

Dieses Objekt hat dieselben Eigenschaften zur Definition des Einfügepunkts und der Richtung, in die die Linie gezogen werden soll, wie der Strahl. Der einzige Unterschied zu den Strahlen besteht lediglich in der Anzeige einer zweiten unendlichen Seite.

Der Startpunkt der Linie wird in der Eigenschaft *BasePoint* verwaltet. Wenn Sie die Koordinaten dieser Eigenschaft anpassen, dann bleibt die aktuelle Richtung der Konstruktionslinie erhalten, sie wird sozusagen auf die neue Position verschoben. Die Eigenschaft *SecondPoint* speichert den aktuellen Richtungspunkt der Konstruktionslinie. Mit der *DirectionVector*-Eigenschaft können Sie den Richtungsvektor der Konstruktionslinie bestimmen oder abrufen. Der Richtungsvektor zeigt dabei die Ausrichtung der Konstruktionslinie unabhängig von den Einheiteneinstellungen des Systems an. Die jeweiligen Vektoren liegen in einem Wertebereich von -1 bis 1 (siehe Kapitel 8.1.2).

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der Konstruktionslinie fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Konstruktionslinie zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Konstruktionslinie ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Konstruktionslinie zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der Konstruktionslinie an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Konstruktionslinie an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Konstruktionslinie an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Konstruktionslinie an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Konstruktionslinie zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadXLine</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Konstruktionslinie zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der Konstruktionslinie steuern.

Tabelle 8.69: Weitere Eigenschaften der Konstruktionslinie

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechteckige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.

Tabelle 8.70: Methoden der Konstruktionslinie

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Konstruktionslinie ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Konstruktionslinie ab.
<i>Highlight</i>	Hebt die Konstruktionslinie in der AutoCAD-Zeichnung hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Konstruktionslinie mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Konstruktionslinie als Resultat einer Spiegelung mit dieser Konstruktionslinie.
<i>Mirror3D</i>	Erstellt eine neue Konstruktionslinie als Resultat einer räumlichen Spiegelung dieser Konstruktionslinie.
<i>Move</i>	Verschiebt die Konstruktionslinie von Ursprungs- auf Zielkoordinaten.
<i>Offset</i>	Diese Anweisung versetzt die Konstruktionslinie um einen bestimmten Betrag.
<i>Rotate</i>	Dreht die Konstruktionslinie um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Konstruktionslinie um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Konstruktionslinie um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Konstruktionslinie.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Konstruktionslinie an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Konstruktionslinie im AutoCAD-Zeichnungsfenster.

Tabelle 8.70: Methoden der Konstruktionslinie (Forts.)

8.3.19 Band (AcadTrace)

Bandobjekte sind vierseitig begrenzte Flächen, ähnlich wie 2D-Flächen, wobei diese gefüllt oder nur deren Umriss dargestellt werden kann, je nach Einstellung der Systemvariablen *FILLMODE*. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen *band* oder *_trace* erzeugen.

Sie erstellen einen Verweis auf ein Band, indem Sie mit der Funktion *AddTrace* unter Angabe der vier Eckpunkte als 3D-Array ein Band dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Band über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Die vier Stützpunkte des Bandobjekts können Sie mit den Eigenschaften *Coordinates* und *Coordinate* verwalten. *Coordinates* gibt wie immer die gesamte Liste der Stützpunkte als 3D-Array an, wobei die Höhe (Z-Koordinate) in allen Stützpunkten gleich ist. Aus der Tatsache, dass dieses Objekt immer vier Stützpunkte besitzt, ergibt sich auch eine eindeutige Obergrenze für das Array der *Coordinates*-Eigenschaft (4 Stützpunkte * 3 Koordinaten pro Stützpunkt – 1). Besser eignen würde sich in diesem Fall aber die Verwendung der *Coordinate*-Eigenschaft, da die Anzahl der Stützpunkte ja bekannt ist.

```
Stp1 = ActTrace.Coordinate(0)
Stp2 = ActTrace.Coordinate(1)
Stp3 = ActTrace.Coordinate(2)
Stp4 = ActTrace.Coordinate(3)
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Bands fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Bands zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Band ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Bands zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Bands an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Bands an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Bands an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für das Band an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Bands zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadTrace</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Band zurück oder legt diesen fest.
<i>Thickness</i>	Diese Eigenschaft speichert die Objekthöhe des Bands. Sie können den aktuellen Wert auslesen oder zurückschreiben.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Bands steuern.

Tabelle 8.71: Weitere Eigenschaften des Bands

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechteckige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Bands ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Bands ab.
<i>Highlight</i>	Hebt das Band in der AutoCAD-Zeichnung hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Bands mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt ein Band als Resultat einer Spiegelung mit diesem Band.
<i>Mirror3D</i>	Erstellt ein neues Band als Resultat einer räumlichen Spiegelung dieses Bands.
<i>Move</i>	Verschiebt das Band von den Ursprungs- auf die Zielkoordinaten.
<i>Rotate</i>	Dreht das Band um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht das Band um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Band um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Bands.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Band an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Bands im AutoCAD-Zeichnungsfenster.

Tabelle 8.72: Methoden des Bands

8.3.20 2D-Fläche (AcadSolid)

Bei 2D-Flächen handelt es sich ebenfalls um Polygone, die in Abhängigkeit von der Einstellung der Systemvariablen *FILLMODE* gefüllt dargestellt werden. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen *solid* oder *_solid* erzeugen.

Sie können einen Verweis auf solch ein Objekt erstellen, indem Sie mit der Funktion *AddSolid* unter Angabe der vier Eckpunkte eine 2D-Fläche dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene 2D-Fläche über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Genau wie ein Bandobjekt, verwaltet auch dieses Objekt seine vier Stützpunkte mit den Eigenschaften *Coordinates* und *Coordinate*. Der Unterschied zu einem Band besteht in der Darstellungsreihenfolge dieser Stützpunkte in der Zeichnung. Während bei einem Band ein durchgehender Umring definiert wird, ergeben sie bei einer 2D-Fläche keinen direkten Umring, wie in Abbildung 8.66 zu sehen.



Abbildung 8.66: Stützpunktdarstellung der 2D-Fläche

Aus diesem Grund müssen Sie genau darauf achten, welchen Stützpunkt Sie derzeit bearbeiten.

```
Stp1 = ActTrace.Coordinate(0) '1. Stützpunkt
Stp2 = ActTrace.Coordinate(1) '2. Stützpunkt
Stp3 = ActTrace.Coordinate(2) '3. Stützpunkt
Stp4 = ActTrace.Coordinate(3) '4. Stützpunkt
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Solid fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Solid zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Solid ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Solid zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Solid an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Solid an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Solid an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die 2D-Fläche an oder legt diese fest.
<i>Normal</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der 2D-Fläche zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadSolid</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Solid zurück oder legt diesen fest.

Tabelle 8.73: Weitere Eigenschaften der 2D-Fläche

Eigenschaft	Beschreibung
<i>Thickness</i>	Legen Sie mithilfe dieser Eigenschaft die Objekthöhe des Solid fest oder rufen Sie den aktuellen Wert ab.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der 2D-Fläche steuern.

Tabelle 8.73: Weitere Eigenschaften der 2D-Fläche (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechteckige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der 2D-Fläche ab.
<i>GetXData</i>	Ruft die erweiterten Daten der 2D-Fläche ab.
<i>Highlight</i>	Hebt die 2D-Fläche in der Zeichnung hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der 2D-Fläche mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Kopie des Rasterbilds als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine Kopie der 2D-Fläche als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt die 2D-Fläche von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die 2D-Fläche um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die 2D-Fläche um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die 2D-Fläche um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der 2D-Fläche.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die 2D-Fläche an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der 2D-Fläche im AutoCAD-Zeichnungsfenster.

Tabelle 8.74: Methoden der 2D-Fläche

8.3.21 Vielflächennetz (AcadPolyfaceMesh)

Mit der Klasse *AcadPolyfaceMesh* können Sie die Eigenschaften eines Vielflächennetzes bearbeiten und mit den verschiedenen Anweisungen, die diese Klasse enthält, die Darstellung eines Vielflächennetzes beeinflussen. In AutoCAD kann der Benutzer dieses Objekt mit dem Befehl `pnetz erzeugen`.

Sie können einen Verweis auf ein solches Netz erstellen, indem Sie mit der Funktion *AddPolyfaceMesh* (siehe Kapitel 6.4.2) ein neues Vielflächennetz dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Vielflächennetz über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Die Koordinaten dieses Objekts werden wie gehabt mit den Eigenschaften *Coordinates* bzw. *Coordinate* verwaltet, wobei Sie schnell den Überblick darüber verlieren können, welche Stützpunkte zu welcher Fläche gehören, was die Abbildung 8.67 verdeutlicht. Aus diesem Grund werden Objekte dieser Klasse meist direkt durch Applikationen erzeugt, die sie dann auch weiterverarbeiten.

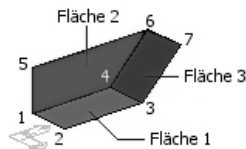


Abbildung 8.67: Beispiel eines Vielflächennetzes

Zusätzlich zu den Stützpunkteigenschaften besitzt diese Klasse zwei weitere Eigenschaften. *NumberOfFaces* enthält die Anzahl der gebildeten Flächen und *NumberOfVertices* die Anzahl der Kontrollpunkte (obwohl Sie diese auch über die *Coordinates*-Eigenschaft mithilfe der *UBound()*-Funktion ermitteln könnten).

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Netzes fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Netzes zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Netz ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Netzes zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Netzes an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Netzes an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Netzes an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke des Netzes an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Vielflächennetzes zurück.

Tabelle 8.75: Weitere Eigenschaften eines Vielflächennetzes

Eigenschaft	Beschreibung
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadPolyFaceMesh</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Netz zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Netzes steuern.

Tabelle 8.75: Weitere Eigenschaften eines Vielflächennetzes (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Netzes ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Netzes ab.
<i>Highlight</i>	Hebt das Netz hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Netzes mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt ein Vielflächennetz als Resultat einer Spiegelung mit diesem Netz.
<i>Mirror3D</i>	Erstellt ein neues Vielflächennetz als Resultat einer räumlichen Spiegelung dieses Netzes.
<i>Move</i>	Verschiebt das Netz von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht das Netz um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht das Netz um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Netz um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Netzes.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Vielflächennetz an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Netzes im AutoCAD-Zeichnungsfenster.

Tabelle 8.76: Methoden eines Vielflächennetzes

8.3.22 Polygonnetz (AcadPolygonMesh)

Mit dieser Klasse können Sie die Eigenschaften eines Polygonnetzes bearbeiten und mit den verschiedenen Anweisungen, die diese Klasse enthält, die Darstellung des Polygonnetzes beeinflussen. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen `3dnetz` oder `_3dmesh` erzeugen.

Sie können einen Verweis auf ein solches Netz erstellen, indem Sie mit der Funktion `Add3DMesh` ein neues Polygonnetz dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Polygonnetz über die `Item`-Funktion der entsprechenden Auflistung (`ModelSpace`, `PaperSpace`, `Block`, `SelectionSet`, `Group`) ermitteln.

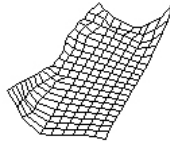


Abbildung 8.68: Polygonnetz

Verwaltung der Stützpunkte

Da ein Polygonnetz aus einer ganzen Reihe von Stützpunkten zusammengestellt wird, die wieder über die Eigenschaften `Coordinates` und `Coordinate` abgerufen oder angepasst werden können, enthält dieses Objekt noch zusätzliche Eigenschaften, die die Verarbeitung der einzelnen Stützpunkte vereinfachen. Dazu zählen `MVertexCount` und `NVertexCount`, die die Anzahl der Reihen und Spalten des Netzes angeben. Diese Eigenschaften können Sie an einem bereits erstellten Netz nicht mehr direkt beeinflussen, sondern Sie müssen sie bei der Erstellung des Netzes festlegen. Für die Verwaltung der Stützpunkte eignen sie sich aber hervorragend, da sich hiermit die Position einzelner Stützpunkte ermitteln lässt. Nehmen wir beispielsweise die äußeren Eckpunkte des Netzes. Ohne die Angabe der Anzahl der Stützpunkte in M- und N-Richtung wird es schwierig, diese zu ermitteln (mal ausgenommen des ersten und des letzten). So müssen Sie nur folgende Codezeilen benutzen, um die Stützpunkte der äußeren Ecken auszulesen oder anzupassen.

Stützpunkte auslesen:

```
Stp1 = Act3DMesh.Coordinate(0)
Stp2 = Act3DMesh.Coordinate(NVertexCount - 1)
Stp3 = Act3DMesh.Coordinate(NVertexCount * MVertexCount - _
    NVertexCount)
Stp4 = NVertexCount * MVertexCount - 1
```

Stützpunkte anpassen:

```
With Act3DMesh
    .Coordinate(0) = Stp1
    .Coordinate(.NVertexCount - 1) = Stp2
    .Coordinate(.NVertexCount * .MVertexCount - _
        .NVertexCount) = Stp3
    .Coordinate(NVertexCount * MVertexCount - 1) = Stp4
End With
```

Ähnlich funktioniert auch die Ermittlung der Stützpunkte einzelner Felder. Nehmen wir an Sie wollen die vier Eckpunkte des 14. Felds eines 3D-Netzes abrufen.

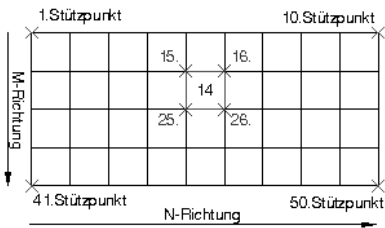


Abbildung 8.69: Stützpunktdefinitionen des 3D-Netzes

Zunächst müssen Sie die „Spalte“ ermitteln, in der sich das gesuchte Feld befindet. Dazu benötigen Sie die Anzahl der Stützpunkte in N-Richtung und natürlich die Nummer des gesuchten Felds. Mit diesen Werten berechnet sich die Spalte, in der sich das Feld befindet, nach folgender Formel:

Spalte = Fix((FeldNr - 1) / (Act3DMesh.NVertexCount - 1))

Die ersten beiden Stützpunktpositionen sind nun relativ einfach zu ermitteln:

StpNr1 = FeldNr + Spalte - 1
StpNr2 = FeldNr + Spalte

Die anderen Stützpunktpositionen lassen sich wie folgt errechnen:

StpNr3 = FeldNr + Act3DMesh.NVertexCount + Spalte - 1
StpNr4 = FeldNr + Act3DMesh.NVertexCount + Spalte

Mit diesen Positionswerten können Sie nun die entsprechenden Stützpunktkoordinaten auslesen.

```
StpPt1 = Act3DMesh.Coordinate(StpNr1)
StpPt2 = Act3DMesh.Coordinate(StpNr2)
StpPt3 = Act3DMesh.Coordinate(StpNr3)
StpPt4 = Act3DMesh.Coordinate(StpNr4)
```

Im Beispielprojekt zu diesem Objekt werden Ihnen weitere Möglichkeiten zur Bestimmung ganz spezieller Stützpunkte dargelegt.

Weitere Eigenschaften eines 3D-Netzes

Mit der *Type*-Eigenschaft können Sie die Art der Angleichung/Glättung des Polygonnetzes bestimmen, ähnlich wie bei Polylinien. Sie können für diese Eigenschaft folgende Konstanten verwenden.

Konstante	Beschreibung	Wert
acSimpleMesh	Keine Glättung	0
acQuadSurfaceMesh	Quadratisch	5
acCubicSurfaceMesh	Kubisch	6
acBezierSurfaceMesh	Bézier	8

Tabelle 8.77: Konstanten der Type-Eigenschaften

Mit den Eigenschaften *MClose* und *NClose* können Sie bestimmen, ob das Netz in M- bzw. N-Richtung geöffnet (*FALSE*) oder geschlossen (*TRUE*) ist. Ist das Netz in M- oder N-Richtung geschlossen, werden die jeweils außen liegenden Stützpunkte miteinander verbunden.

Die Eigenschaften *MDensity* und *NDensity* bestimmen die Dichte des Polygonnetzes in M- und N-Richtung. Sie können für diese Eigenschaften Werte zwischen 2 und 256 verwenden.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Netzes fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Netzes zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Netz ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Netzes zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Netzes an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Netzes an oder legt diesen fest..
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Netzes an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke des Netzes an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Vielflächennetzes zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadPolyFaceMesh</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Netz zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Netzes steuern.

Tabelle 8.78: Weitere Eigenschaften des Polygonnetzes

Methoden des Polygonnetzes

Neben den Standardmethoden besitzt diese Objekt noch die Anweisungen *AppendVertex* und *Explode*. Die *Explode*-Anweisung löst das Netz in einzelne 3D-Flächen auf und gibt ein Array mit Verweisen auf die entstandenen 3D-Flächen zurück. Das Polygonnetz selbst bleibt allerdings erhalten, so dass Sie es mit der *Delete*-Anweisung löschen müssen, um es aus der Zeichnung zu entfernen.

```
ObjList = Act3DMesh.Explode
```

Mit der *AppendVertex*-Anweisung können Sie dem Netz eine weitere N-Reihe hinzufügen. Dadurch erhöht sich natürlich die Anzahl der Spalten in der Eigenschaft *MVertexCount*. Die Anweisung erfordert dafür die Angabe einer Liste von Stützpunktkoordinaten, die dem Netz hinzugefügt werden sollen. Da immer eine komplette Spalte hinzugefügt wird, erfordert dies die Deklaration eines Arrays, das genauso viele Stützpunkte mit jeweils drei Koordinaten beinhaltet, wie das Netz N-Reihen (Eigenschaft *NVertexCount*) besitzt. Daraus ergibt sich folgende Deklaration:

```
ReDim Vertex(Act3DMesh.NVertexCount * 3 - 1) As Double
```

Danach müssen Sie dieses Array nur noch mit den entsprechenden Koordinaten füllen und dann als Parameter der *AppendVertex*-Anweisung verwenden.

```
Act3DMesh.AppendVertex Vertex
```

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Netzes ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Netzes ab.
<i>Highlight</i>	Hebt das Netz hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Netzes mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt ein Vielflächennetz als Resultat einer Spiegelung mit diesem Netz.
<i>Mirror3D</i>	Erstellt ein neues Vielflächennetz als Resultat einer räumlichen Spiegelung
<i>Move</i>	Verschiebt das Netz von den Ursprungs- auf die Zielkoordinaten.
<i>Rotate</i>	Dreht das Netz um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht das Netz um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Netz um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Netzes.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Vielflächennetz an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Netzes im AutoCAD-Zeichnungsfenster.

Tabelle 8.79: Weitere Methoden des Polygonnetzes

Anwendungsbeispiel

(Beispielprojekt: 08\Acad3DMesh.dvb)

Im Folgenden werden wir einige zuvor besprochene Eigenschaften und Methoden anhand eines praktischen Anwendungsbeispiels betrachten. In dieses Projekt wurde die Klasse *MyUtility* implementiert, um ungewollte Beendigungen, beispielsweise durch ZOOM-Befehle, zu unterbinden (siehe *Kapitel A.3*). Außerdem wurde ein Formular integriert, das die verschiedensten Koordinaten des gewählten Netzes anzeigen und es dem Benutzer über einige Schaltflächen ermöglichen soll, dieses Netz zu manipulieren.

Startprozedur

In der Startprozedur wird der Benutzer zunächst aufgefordert, ein 3D-Netz auszuwählen, das dann im Formular weiterverarbeitet werden soll.

```
Public Sub GetOuterPoints()
Dim Object As Object
Dim Act3DMesh As AcadPolygonMesh
On Error Resume Next
Set Util = New MyUtility
Util.GetEntity Object, PickedPoint, "3D-Netz wählen:"
OrgMode = GetVariable("PDMODE")
SetVariable "PDMODE", 3
If TypeName(Object) = "IAcadPolygonMesh" Then
```

Im zweiten Teil der Prozedur werden nur noch die Einstellungen des Formulars angepasst, die direkt durch das ausgewählte Netz beeinflusst werden. Dazu zählen die Objektvariable, die den Verweis auf das 3D-Netz aufnehmen soll, das *ScrollBar*-Steuerelement, mit dem der Benutzer sich durch die einzelnen Felder des Netzes bewegen kann, und der Aufruf der Anweisung *ReadCoordinates*, die die Umringstützpunkte des Netzes in die Liste des Formulars eintragen wird. Abschließend wird das Formular eingeblendet.

```
With MeshFrm
Set .Act3DMesh = Object
.ScrollBar1.Min = 1
.ScrollBar1.Max = (.Act3DMesh.NVertexCount - 1) * _
    (.Act3DMesh.MVertexCount - 1)
.ScrollBar1.SmallChange = 1
.ScrollBar1.LargeChange = .Act3DMesh.NVertexCount - 1
.ReadCoordinates
.Show
```

Da dies gebunden (modal) geschieht, werden alle nachfolgenden Anweisungen erst nach Beendigung des Programms ausgeführt.

```
.SelMesh.Delete
.SelPoint.Delete
End With
End If
SetVariable "PDMODE", OrgMode
Unload MeshFrm
End Sub
```

Umringpunkte des Netzes ermitteln

Diese Prozedur ermittelt alle Stützpunkte des Umrings und trägt die Koordinaten in das *List-View*-Steuerelement ein (erste Spalte → X, zweite Spalte → Y, dritte Spalte → Z). Die Ermittlung dieser Stützpunkte unterteilt sich in vier Abschnitte.

Jeder Abschnitt wird in einer *For...Next*-Schleife abgearbeitet und beginnt beim Startpunkt des Netzes.

```
Public Sub ReadCoordinates()
Dim LV As MSComctlLib.ListItem
Dim StpPt
On Error Resume Next
ListView1.ListItems.Clear
With Util
```

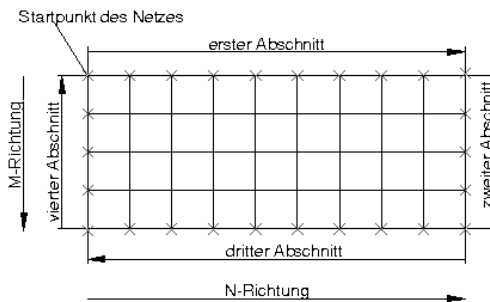


Abbildung 8.70: Umringpunkte bestimmen

Erster Abschnitt:

```
For i = 0 To Act3DMesh.NVertexCount - 1
    StpPt = Act3DMesh.Coordinate(i)
    Set LV = ListView1.ListItems.Add(, , .RealToString _
        (StpPt(0)))
    LV.ListSubItems.Add , , .RealToString(StpPt(1))
    LV.ListSubItems.Add , , .RealToString(StpPt(2))
Next
```

Zweiter Abschnitt:

```
For i = (Act3DMesh.NVertexCount * 2) - 1 To _
    Act3DMesh.MVertexCount * Act3DMesh.NVertexCount - _
    Act3DMesh.NVertexCount Step Act3DMesh.NVertexCount
    StpPt = Act3DMesh.Coordinate(i)
    Set LV = ListView1.ListItems.Add(, , .RealToString _
        (StpPt(0)))
    LV.ListSubItems.Add , , .RealToString(StpPt(1))
    LV.ListSubItems.Add , , .RealToString(StpPt(2))
Next
```

Dritter Abschnitt:

```
For i = (Act3DMesh.NVertexCount * Act3DMesh.MVertexCount) - 1 _
    To (Act3DMesh.NVertexCount * Act3DMesh.MVertexCount - _
    Act3DMesh.NVertexCount) + 1 Step -1
```

```

StpPt = Act3DMesh.Coordinate(i)
Set LV = ListView1.ListItems.Add(, , .RealToString _
    (StpPt(0)))
LV.ListSubItems.Add , , .RealToString(StpPt(1))
LV.ListSubItems.Add , , .RealToString(StpPt(2))
Next

```

Vierter Abschnitt:

```

For i = Act3DMesh.MVertexCount * Act3DMesh.NVertexCount - _
    Act3DMesh.NVertexCount To Act3DMesh.NVertexCount Step - _
    Act3DMesh.NVertexCount
    StpPt = Act3DMesh.Coordinate(i)
    Set LV = ListView1.ListItems.Add(, , .RealToString _
        (StpPt(0)))
    LV.ListSubItems.Add , , .RealToString(StpPt(1))
    LV.ListSubItems.Add , , .RealToString(StpPt(2))
Next
End With
ListView1.SelectedItem = ListView1.ListItems(0)
ListView1_ItemClick ListView1.SelectedItem
End Sub

```

Auf Feldauswahl reagieren

Diese Prozedur wird immer dann ausgeführt, wenn der Benutzer ein Feld des Netzes selektiert. Dies geschieht entweder direkt durch die Veränderung des Schiebers bzw. Eingabe einer Feldnummer oder indirekt durch Auswahl eines neuen Netzes.

In dieser Prozedur werden sowohl die Textboxen (zur Aufnahme der Koordinaten des gewählten Felds) initialisiert als auch das Objekt, das das gewählte Feld (Bandobjekt) anzeigt, entweder erzeugt (wenn dies noch nicht geschehen ist) oder neu positioniert.

```

Public Sub ShowFieldAndPoint()
Dim OwnerObj As AcadObject, Spalte As Long
Dim PointsArray(11) As Double
On Error Resume Next

```

Zunächst müssen wir die gewählte Feldposition ermitteln, um mit dieser Information die Stützpunkte dieses Felds aus dem 3D-Netz auszulesen

```

Spalte = Fix((ScrollBar1 - 1) / (Act3DMesh.NVertexCount - 1))
Stp1 = Act3DMesh.Coordinate(ScrollBar1 - 1 + Spalte)
Stp2 = Act3DMesh.Coordinate(ScrollBar1 + Spalte)
Stp3 = Act3DMesh.Coordinate(ScrollBar1 + Act3DMesh.NVertexCount _
    - 1 + Spalte)
Stp4 = Act3DMesh.Coordinate(ScrollBar1 + Act3DMesh.NVertexCount _
    + Spalte)

```

und dann das Array, das dem Bandobjekt (zeigt die Feldposition optischen) zugewiesen wird, mit diesen Koordinaten zu füllen.

```

PointsArray(0) = Stp1(0)
PointsArray(1) = Stp1(1)
PointsArray(2) = Stp1(2)
PointsArray(3) = Stp2(0)

```



```

PointsArray(4) = Stp2(1)
PointsArray(5) = Stp2(2)
PointsArray(6) = Stp3(0)
PointsArray(7) = Stp3(1)
PointsArray(8) = Stp3(2)
PointsArray(9) = Stp4(0)
PointsArray(10) = Stp4(1)
PointsArray(11) = Stp4(2)

```

Sollte dieses Objekt noch nicht existieren (wird mit *TypeName()* geprüft), wird es mit der *AddTrace*-Anweisung zu dem Bereich hinzugefügt, in dem sich auch das Netz befindet. Dazu nutzen wir die Eigenschaft *OwnerID*, die ja die *ObjectID* des Eigentümerobjekts zurückgibt. Sollte das Objekt bereits existieren, dann enthält die Variable *SelMesh* einen Verweis auf das Band und es muss nur noch die *Coordinates*-Eigenschaft angepasst werden. Um das Band in der Zeichnung optisch etwas hervorzuheben, wird es rot dargestellt. Sie könnten aber auch eine andere Farbe verwenden oder die *Highlight*-Anweisung mit dem Parameter *TRUE* aufrufen.

```

If TypeName(SelMesh) = "Nothing" Then
    Set OwnerObj = ThisDrawing.ObjectIdToObject(Act3DMesh.OwnerID)
    Set SelMesh = OwnerObj.AddTrace(PointsArray)
    SelMesh.Layer = Act3DMesh.Layer
    SelMesh.Color = acRed
Else
    SelMesh.Coordinates = PointsArray
End If
SelMesh.Update

```

Zum Schluss werden noch die Stützpunktkoordinaten an die entsprechenden Textfelder übergeben. Dazu nutzen wir wieder die Funktionen der Klasse *MyUtility*.

```

With Util
    TextBox2 = .RealToString(Stp1(0))
    TextBox3 = .RealToString(Stp1(1))
    TextBox4 = .RealToString(Stp1(2))
    TextBox5 = .RealToString(Stp2(0))
    TextBox6 = .RealToString(Stp2(1))
    TextBox7 = .RealToString(Stp2(2))
    TextBox8 = .RealToString(Stp3(0))
    TextBox9 = .RealToString(Stp3(1))
    TextBox10 = .RealToString(Stp3(2))
    TextBox11 = .RealToString(Stp4(0))
    TextBox12 = .RealToString(Stp4(1))
    TextBox13 = .RealToString(Stp4(2))
End With
End Sub

```

Auswahl eines Umringpunkts

Wenn der Benutzer einen Eintrag des *ListView*-Steuerelements wählt, dann wird die Ereignisprozedur *ItemClick* ausgelöst. Diesen Umstand können wir nutzen, um den gewählten Punkt in der Zeichnung optisch anzeigen zu lassen.

```

Private Sub ListView1_ItemClick(ByVal Item As _
MSComctlLib.ListItem)
Dim StpPt(2) As Double
On Error Resume Next

```

Zunächst stellen wir ein 3D-Array mit den Koordinaten des gewählten Stützpunkts zusammen. Dieses Array wird dann an ein Punktobjekt übergeben, das den Stützpunkt optisch anzeigen soll.

```
With ThisDrawing.Utility
    StpPt(0) = .DistanceToReal(ListView1.SelectedItem.Text, _
        acDefaultUnits)
    StpPt(1) = .DistanceToReal(ListView1.SelectedItem. _
        ListSubItems(1), acDefaultUnits)
    StpPt(2) = .DistanceToReal(ListView1.SelectedItem. _
        ListSubItems(2), acDefaultUnits)
End With
```

Sollte dieses Punktobjekt noch nicht erzeugt worden sein (direkt nach dem Programmstart), wird wieder, wie beim Band, die *OwnerID* dazu benutzt, einen Verweis auf den Container des Netzes in der Objektvariablen *OwnerObj* zu erstellen und diesem Eigentümerobjekt den Punkt hinzuzufügen. Andernfalls braucht nur noch der Stützpunkt des Punkts angepasst zu werden.

```
If TypeName(SelPoint) = "Nothing" Then
    Set OwnerObj = ThisDrawing.ObjectIdToObject(Act3DMesh.OwnerID)
    Set SelPoint = OwnerObj.AddPoint(StpPt)
    SelPoint.Layer = Act3DMesh.Layer
    SelPoint.Color = acRed
Else
    SelPoint.Coordinates = StpPt
End If
SelPoint.Update
End Sub
```

Feldauswahl durch den Benutzer

Der Benutzer erhält zwei Möglichkeiten, ein Feld des Netzes zu wählen. Als Erstes könnte er den Schieberegler der Bildlaufleiste auf die Feldposition bewegen, die im nebenstehenden Textfeld angezeigt wird.

```
Private Sub ScrollBar1_Change()
On Error Resume Next
TextBox1.Text = ScrollBar1
ShowFieldAndPoint
End Sub
```

Andererseits könnte er auch die Feldnummer direkt im Textfeld eingeben. In beiden Fällen wird die Anweisung *ShowFieldAndPoint* aufgerufen, um die optische Anzeige des ausgewählten Felds zu aktualisieren. Im Falle der Eingabe in das Textfeld geschieht dies indirekt, da durch die Übergabe des Wertes an das *ScrollBar*-Steuerelement, dessen *Change*-Ereignis ausgelöst wird.

```
Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
ScrollBar1 = TextBox1
TextBox1 = ScrollBar1
End Sub
```

Änderung der Stützpunktkoordinaten

Um die Koordinaten einzelner Stützpunkte des ausgewählten Felds anzupassen, muss der Benutzer nur die entsprechenden Werte in die Textfelder des Formulars eingeben. In der daraufhin ausgelösten Ereignisprozedur wird entsprechend auf die Änderung reagiert.

Y-Koordinate des ersten Stützpunkts wurde geändert

Die Textfelder, die die Y-Koordinaten der einzelnen Stützpunkte enthalten, übernehmen den Hauptteil der Arbeit. Dort werden die Koordinaten zu einem 3D-Array zusammengefasst und an den entsprechenden Stützpunkt übergeben. Zusätzlich wird auch die Position des Bands, das ja zur optischen Anzeige des Felds dienen soll, aktualisiert.

```
Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim Point(2) As Double
On Error Resume Next
Point(0) = Util.DistanceToReal(TextBox2)
Point(1) = Util.DistanceToReal(TextBox3)
Point(2) = Util.DistanceToReal(TextBox4)
Act3DMesh.Coordinate(ScrollBar1 - 1 + Spalte) = Point
Act3DMesh.Update
ScrollBar1_Change
ReadCoordinates
End Sub
```

X-Koordinate des ersten Stützpunkts wurde geändert

```
Private Sub TextBox3_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
TextBox2_BeforeUpdate Cancel
End Sub
```

Z-Koordinate des ersten Stützpunkts wurde geändert

```
Private Sub TextBox4_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
TextBox2_BeforeUpdate Cancel
End Sub
```

Die anderen Stützpunktänderungen funktionieren auf dieselbe Art und Weise, nur dass dort jeweils ein anderer Netzstützpunkt angesprochen wird.

Schaltfläche „Wählen“

Über diese Schaltfläche kann der Benutzer ein anderes Netz in der Zeichnung auswählen. Der größte Teil dieser Prozedur stammt aus der Startprozedur des Programms. Zusätzlich muss natürlich das Formular ausgeblendet werden, um dem Benutzer die Auswahl in der Zeichnung zu ermöglichen.

```
Private Sub CommandButton1_Click()
Dim Object As Object
On Error Resume Next
Me.hide
Util.GetEntity Object, PickedPoint, "3D-Netz wählen:"
```

```

If TypeName(Object) = "IAcadPolygonMesh" Then
    Set Act3DMesh = Object
    ScrollBar1.Min = 1
    ScrollBar1.Max = (Act3DMesh.NVertexCount - 1) * _
        (Act3DMesh.MVertexCount - 1)
    ScrollBar1.SmallChange = 1
    ScrollBar1.LargeChange = Act3DMesh.NVertexCount - 1
    ScrollBar1.Value = 1
    ScrollBar1_Change
    ReadCoordinates
End If
Me.Show
End Sub

```

Schaltfläche „Polylinie erzeugen“

Über diese Schaltfläche kann der Benutzer aus den Umringkoordinaten des Netzes eine 3D-Polylinie erzeugen lassen. Die Erstellung ist dabei recht einfach. Da die Koordinaten bereits im *Listview*-Steuerelement erfasst wurden, braucht nur noch ein Array mit den einzelnen Koordinaten dieser Liste zusammengestellt zu werden. Dieses Array wird dann als Parameter der *Add3DPoly*-Anweisung verwendet, womit die Polylinie bereits erstellt wäre. Danach muss sie nur noch geschlossen werden, damit eine Verbindung zwischen Anfangs- und Endpunkt hergestellt wird.

```

Private Sub CommandButton3_Click()
    Dim PointsList, New3DPoly As Acad3DPolyline
    On Error Resume Next
    ReDim PointsList((ListView1.ListItems.Count * 3) - 1) As Double
    x = 0
    For i = 1 To ListView1.ListItems.Count
        PointsList(x) = Util.DistanceToReal(ListView1.ListItems(i))
        x = x + 1
        PointsList(x) = Util.DistanceToReal(ListView1.ListItems(i). _
            ListSubItems(1))
        x = x + 1
        PointsList(x) = Util.DistanceToReal(ListView1.ListItems(i). _
            ListSubItems(2))
        x = x + 1
    Next
    Set OwnerObj = ThisDrawing.ObjectIdToObject(Act3DMesh.OwnerID)
    Set New3DPoly = OwnerObj.Add3DPoly(PointsList)
    New3DPoly.Closed = True
End Sub

```

Schaltfläche „Auflösen“

Die Auflösung des 3D-Netzes in einzelne 3D-Flächen ist eine einfache Angelegenheit. Nach einer Sicherheitsabfrage rufen wir nur die *Explode*-Anweisung des Netzobjekts auf und übergeben eine Liste der entstandenen Einzelflächen an die Variable *ObjList*. Sie könnten nun die einzelnen Flächen weiterverarbeiten, indem Sie die Verweise der einzelnen Felder des Arrays *ObjList* abrufen.

```

Set Act3DFace = ObjList(n)

```

Damit das Netz aus der Zeichnung entfernt wird (*Explode* erstellt nur die neuen Elemente), müssen Sie noch die *Delete*-Anweisung ausführen. Da das Netz nun verschwunden ist, kann das Formular ausgeblendet und damit das Programm beendet werden.

```
Private Sub CommandButton4_Click()
If MsgBox("Soll das 3D-Polygonnetz aufgelöst werden?", vbYesNo, _
    "Auflösen") = vbYes Then
    ObjList = Act3DMesh.Explode
    Act3DMesh.Delete
    Me.hide
End If
End Sub
```

Schaltfläche „Neue Punkte“

Über diese Schaltfläche kann der Benutzer dem gewählten Netz eine weitere Reihe von Stützpunkten hinzufügen. Nachdem das Formular ausgeblendet wurde (ist notwendig, da die Auswahl der Koordinaten in der Zeichnung erfolgt), wird zunächst ein Array deklariert, das die Koordinaten der neuen Stützpunkte aufnehmen soll. Dabei ist es wichtig, dass dieses Array dieselbe Anzahl an Stützpunkten erhält, wie für eine komplette neue Reihe erforderlich sind.

```
Private Sub CommandButton5_Click()
Dim Vertex
On Error Resume Next
Me.hide
ReDim Vertex(Act3DMesh.NVertexCount * 3 - 1) As Double
x = 0
```

In der folgenden Schleife wird zunächst der dem neuen Punkt gegenüberliegende Punkt ausgelesen und an die Variable *LastPoint* weitergegeben.

```
For i = 0 To Act3DMesh.NVertexCount - 1
    LastPoint = Act3DMesh.Coordinate((Act3DMesh.MVertexCount * _
        Act3DMesh.NVertexCount) - Act3DMesh.NVertexCount + i)
```

Dann wird der Benutzer aufgefordert, die Position des jeweiligen neuen Punkts auszuwählen. Dabei kommt wieder die spezielle *GetPoint*-Funktion der Klasse *MyUtility* zum Einsatz, damit der Benutzer die Eingabe der einzelnen Punkte korrekt ausführt. Hat er die Auswahl der Punkte abgebrochen, wird diese Prozedur vorzeitig verlassen und überhaupt kein Punkt hinzugefügt. (Sie erinnern sich: Nur eine komplette Reihe kann hinzugefügt werden.)

```
Prompt = "Punkt wählen oder <eXit>"
VertexPoint = Empty
VertexPoint = Util.GetPoint(LastPoint, Prompt, "eXit")
If TypeName(VertexPoint) = "Double()" Then
    Vertex(x) = VertexPoint(0)
    x = x + 1
    Vertex(x) = VertexPoint(1)
    x = x + 1
    Vertex(x) = VertexPoint(2)
    x = x + 1
ElseIf UCase(VertexPoint) = "EXIT" Then
    Me.Show
    Exit Sub
End If
Next
```

Nachdem das Array komplett zusammengestellt wurde, braucht dieses nur noch mit der *AppendVertex*-Anweisung aufgerufen zu werden und die neue Reihe ist erstellt. Damit das Formular auf den neuesten Stand gebracht wird, müssen wir nur noch die *ScrollBar*-Eigenschaft *Max* aktualisieren und die Umringkoordinaten neu ermitteln.

```
Act3DMesh.AppendVertex Vertex
ScrollBar1.Max = (Act3DMesh.NVertexCount - 1) * _
    (Act3DMesh.MVertexCount - 1)
ReadCoordinates
Me.Show
End Sub
```

8.3.23 Rasterbild (AcadRasterImage)

Mithilfe dieser Klasse können Sie die Eigenschaften von Rasterdateien in AutoCAD bearbeiten. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen *image* oder *_image* der Zeichnung hinzufügen. Rasterbilder werden dabei nicht direkt in die Zeichnung eingefügt, sondern lediglich eine Verknüpfung zu den Bilddateien. So können Sie mehrere Kopien desselben Bilds erstellen, die alle die gleiche Quelle für die Anzeige in AutoCAD verwenden.



Abbildung 8.71: Eingefügtes Pixelbild

Sie können einen Verweis auf ein solches Bild erstellen, indem Sie mit der Funktion *AddRasterImage* ein neues Pixelbild dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Pixelbild über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften eines einzelnen Pixelbildobjekts

Der Einfüge- bzw. Ursprungspunkt des Bilds wird in der Eigenschaft *Origin* als 3D-Array verwaltet. Die Verarbeitung dieser Eigenschaft entspricht der Stützpunkteigenschaft anderer Objekte. Die *ImageFile*-Eigenschaft bestimmt den vollständigen Pfad des Pixelbilds, einschließlich Dateiname und Erweiterung. Mit der *Name*-Eigenschaft hingegen kann ein unabhängiger Name für das Bildobjekt festgelegt werden, so dass Sie hier eine genauere Bezeichnung des Pixelbilds festlegen können. Beide Eigenschaften haben globalen Charakter, so dass eine Änderung an einem der Bilder auch Auswirkungen auf eventuelle Kopien hat. Alle anderen Eigenschaften treffen immer nur auf die jeweils angesprochenen Bilddateien zu.

Mit *Brightness* können Sie die Helligkeit des Pixelbilds steuern. *Contrast* bestimmt den Kontrast des Bilds und *Fade* dessen Fade-Wert. Sie können für diese Eigenschaften einen Wert zwischen 0 und 100 einstellen, was den prozentualen Anteil der jeweiligen Eigenschaft angibt.

Die Eigenschaften *ImageHeight* und *ImageWidth* bestimmen die aktuelle Höhe und Breite des Pixelbilds. Diese entsprechen der gesamten Breite bzw. Höhe des Bilds einschließlich des ausgeblendeten Bereichs bei zugeschnittenen Bildern. Das Objekt besitzt aber noch zwei Eigenschaften, die die Bildgröße angeben (*Height* und *Width*). Hierbei handelt es sich um die Originalgrößen des Bilds in Pixel. *Height* gibt die Höhe an und *Width* die Breite. Diese Angaben können nicht geändert werden und besitzen somit rein informativen Charakter.

ClippingEnabled steuert die Anzeige der Zuschneideumgrenzung des Pixelbilds. Setzen Sie diese Eigenschaft auf *FALSE*, wird die Zuschneideumgrenzung deaktiviert und das gesamte Bild angezeigt. Der Wert *TRUE* aktiviert die Umgrenzung wieder (falls vorhanden).

Mit der *ImageVisibility*-Eigenschaft können Sie die Anzeige des jeweiligen Pixelbilds steuern. Je nachdem, welchen Wert Sie für diese Eigenschaft einstellen, wird das gesamte Pixelbild (*TRUE*) oder nur der Rahmen des Bilds (*FALSE*) angezeigt.

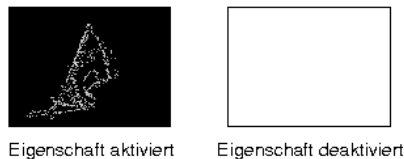


Abbildung 8.72: Auswirkung der Eigenschaft *ImageVisibility*

Mit *ShowRotation* kann ebenfalls die Sichtbarkeit des Rasterbilds bestimmt werden, allerdings in einem anderen Zusammenhang. Ist die Eigenschaft deaktiviert (*FALSE*), wird nur der Rahmen des Bilds angezeigt, wenn das Bild nicht parallel zum Zeichnungsfenster gedreht ist.

Transparency legt fest, ob die Transparenz eines zweifarbigen Bilds aktiviert (*TRUE*) oder deaktiviert (*FALSE*) ist.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Pixelbilds fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Pixelbilds zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Pixelbild ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Pixelbilds zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).

Tabelle 8.80: Weitere Eigenschaften des Rasterbilds

Eigenschaft	Beschreibung
<i>Layer</i>	Zeigt den Layer des Pixelbilds an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Pixelbilds an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Pixelbilds an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für das Pixelbild an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Pixelbilds zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadRasterImage</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Pixelbild zurück oder legt diesen fest.
<i>Rotation</i>	Legt den aktuellen Drehwinkel des Pixelbilds fest.
<i>ScaleFactor</i>	Bestimmt den Skalierfaktor des Pixelbilds.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Pixelbilds steuern.

Tabelle 8.80: Weitere Eigenschaften des Rasterbilds (Forts.)

Methoden des Pixelbildobjekts

Über die *ClipBoundary*-Prozedur bestimmen Sie die Zuschneideumgrenzung des Rasterbilds. Die Umgrenzung wird dabei ein geschlossenes 2D-Polygon als Array an diese Prozedur übergeben.

```
Dim cPts(0 To 13) As Double
cPts(0) = 0: cPts(1) = 0
cPts(2) = 5: cPts(3) = 0
cPts(4) = 5: cPts(5) = 2
cPts(6) = 7: cPts(7) = 2
cPts(8) = 7: cPts(9) = 4
cPts(10) = 3: cPts(11) = 4
cPts(12) = 0: cPts(13) = 0
rasterObj.ClipBoundary cPts
```

Das Besondere an dieser Umgrenzungsdefinition ist, dass Sie eine geschlossene Umgrenzung angeben müssen, was bedeutet, dass Sie immer den ersten Stützpunkt noch mal ans Ende der Liste anhängen müssen.

Prozeduren	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Rasterbilds ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Rasterbilds ab.
<i>Highlight</i>	Hebt das Rasterbild hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Rasterbilds mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Kopie des Rasterbilds als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine Kopie des Rasterbilds als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt das Rasterbild von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht das Rasterbild um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht das Rasterbild um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Rasterbild um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Rasterbilds.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Rasterbild an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Rasterbilds im AutoCAD-Zeichnungsfenster.

Tabelle 8.81: Methoden des Rasterbilds

Anwendungsbeispiel

(Beispielprojekt: 08\AcadRasterImage.dvb)

Im folgenden Anwendungsbeispiel soll dem Benutzer die Möglichkeit gegeben werden, einzelne Pixelbilder in der Zeichnung auszuwählen und anschließend in einem Formular anzupassen. Auch in dieses Projekt wurde die Klasse *MyUtility* implementiert, um die Auswahl der Zuschneideumgrenzung etwas einfacher zu gestalten. Betrachten wir uns zunächst das Formular, in dem die Informationen des Bilds verarbeitet werden sollen.

In diesem Formular gibt es neben den Standardsteuerelementen die *Slider*, zur Regelung von Helligkeit, Kontrast und des Fade-Werts, und das *CommonDialog*-Steuerelement zur Auswahl einer Pixelbilddatei.

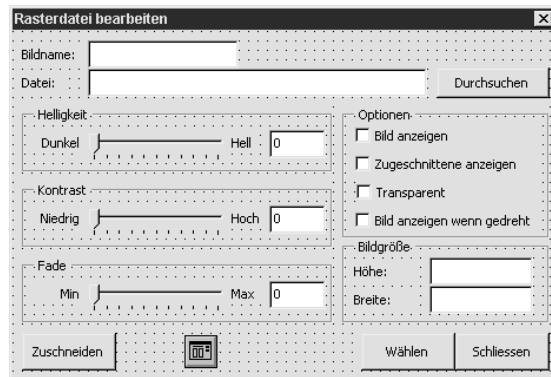


Abbildung 8.73: Formular des Beispielprojekts

Startprozedur

Die Startprozedur ist wieder recht einfach gestrickt. Hier muss der Benutzer nur ein Rasterbild in der Zeichnung auswählen, das dann an die Formularvariable *ActRaster* weitergegeben wird. Danach wird die Initialisierungsprozedur *initRasterFrm* aufgerufen und das Formular gestartet.

```
Public Sub selectRasterImage()
On Error Resume Next
Set Util = New MyUtility
Util.GetEntity ActRaster, PickedPoint, "Bild wählen:"
If TypeName(ActRaster) = "IAcadRasterImage" Then
    Set RasterFrm.ActRaster = ActRaster
    RasterFrm.initRasterFrm
    RasterFrm.show
End If
End Sub
```

Initialisierungsprozedur

Außer der Zuweisung der einzelnen Eigenschaften an die entsprechenden Steuerelemente, hat auch diese Prozedur nicht viel zu bieten. Sie wird immer dann aufgerufen, wenn der Benutzer ein neues Pixelbild in der Zeichnung ausgewählt hat.

```
Public Sub initRasterFrm()
On Error Resume Next
TextBox1.Text = ActRaster.Name
TextBox2.Text = ActRaster.ImageFile
Slider1.Value = ActRaster.Brightness
Slider2.Value = ActRaster.Contrast
Slider3.Value = ActRaster.Fade
TextBox6.Text = Util.RealToString(ActRaster.ImageHeight)
TextBox7.Text = Util.RealToString(ActRaster.ImageWidth)
CheckBox1 = ActRaster.ImageVisibility
CheckBox2 = ActRaster.ClippingEnabled
CheckBox3 = ActRaster.Transparency
CheckBox4 = ActRaster.ShowRotation
End Sub
```

Schaltfläche „Durchsuchen“

Hat der Benutzer die Schaltfläche DURCHSUCHEN betätigt, soll ein Dialog gestartet werden, in dem er die entsprechende Bilddatei auswählen kann. Zunächst übergeben wir den alten Dateinamen des Bilds an die Eigenschaft *FileName*. Anschließend werden alle Bildformate, die AutoCAD unterstützt, als Filter zusammengefasst.

```
Private Sub CommandButton2_Click()
On Error Resume Next
ComDialog.FileName = TextBox2
ComDialog.Filter = "Alle Bilddateien|*.bmp;*.rle;*.dib;*.rst;" & _
    "*.gp4;*.mil;*.cal;*.cg4;*.flc;*.fli;*.bil;*.ig4;*.igs;" & _
    "*.jpg;*.pcx;*.pct;*.png;*.rlc;*.tga;*.tif;*.tiff|BMP|" & _
    "*.bmp;*.rle;*.dib|CALS1|*.rst;*.gp4;*.mil;*.cal;*.cg4" & _
    "|FLIC|*.flc;*.fli|GEOSPOT|*.bil|IG4|*.ig4|IGS|*.igs|" & _
    "JFIF|*.jpg|PCX|*.pcx|PCT|*.pct|PNG|*.png|RLC|*.rlc|" & _
    "TGA|*.tga|TIFF|*.tif;*.tiff|"
Err.Clear
ComDialog.ShowOpen
If Err = 0 Then
```

Nachdem der Benutzer eine neue Bilddatei ausgewählt hat (ein Abbruch würde einen Fehler auslösen), wird der neue Dateiname dem ausgewählten Bild zugeordnet und in dem Textfeld angezeigt. Wichtig hierbei ist noch die Aktualisierung des Bilds in der Zeichnung (*Update*) und die Regenerierung des aktuellen Ansichtsfensters, damit eventuelle Kopien des Bilds ebenfalls neu gezeichnet werden.

```
    ActRaster.ImageFile = ComDialog.FileName
    TextBox2 = ActRaster.ImageFile
    ActRaster.Update
    ThisDrawing.Regen acActiveViewport
End If
End Sub
```

Schaltfläche „Wählen“

Über die Schaltfläche WÄHLEN wird es dem Benutzer ermöglicht, ein anderes Pixelbild in der Zeichnung auszuwählen, ohne das Programm neu zu starten. Die Vorgehensweise hierbei ist dieselbe wie in der Startprozedur, mit dem Unterschied, dass hier zunächst das Formular ausgeblendet werden muss.

```
Private Sub CommandButton3_Click()
On Error Resume Next
Me.Hide
Util.GetEntity Object, PickedPoint, "Bild wählen:"
If TypeName(ActRaster) = "IAcadRasterImage" Then
    Set RasterFrm.ActRaster = Object
    initRasterFrm
End If
Me.show
End Sub
```

Schaltfläche „Zuschneiden“

Das Zuschneiden des Bilds ist auch kein Problem. Nachdem das Formular ausgeblendet wurde, verwenden wir die Funktion *GetPointsArray* der Klasse *MyUtility*, um den Umring auszuwählen. Wichtig ist, dass Sie im Parameter *Closed* den Wert *TRUE* einsetzen, damit der Umring wirklich geschlossen wird, da andernfalls die *ClipBoundary*-Anweisung versagen würde.

```
Private Sub CommandButton4_Click()
On Error Resume Next
Me.Hide
PointsArray = Util.GetPointsArray("Umring wählen:", , True, True)
ActRaster.ClipBoundary PointsArray
ActRaster.Update
Me.show
End Sub
```

Alle anderen Ereignisprozeduren bedürfen keiner weiteren Erklärungen, da hier nur die Eigenschaftswerte gesetzt bzw. gelesen werden.

Änderung der Helligkeit

```
Private Sub Slider1_Change()
On Error Resume Next
TextBox3.Text = Slider1.Value
ActRaster.Brightness = Slider1
ActRaster.Update
End Sub

Private Sub Slider1_Scroll()
Slider1_Change
End Sub

Private Sub TextBox3_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Slider1 = TextBox3
End Sub
```

Änderung des Kontrasts

```
Private Sub Slider2_Change()
On Error Resume Next
TextBox4.Text = Slider2.Value
ActRaster.Contrast = Slider2
ActRaster.Update
End Sub

Private Sub Slider2_Scroll()
Slider2_Change
End Sub

Private Sub TextBox4_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Slider2 = TextBox4
End Sub
```

Änderung des Fade-Werts

```
Private Sub Slider3_Change()  
On Error Resume Next  
TextBox5.Text = Slider3  
ActRaster.Fade = Slider3  
ActRaster.Update  
End Sub  
  
Private Sub Slider3_Scroll()  
Slider3_Change  
End Sub  
  
Private Sub TextBox5_BeforeUpdate(ByVal Cancel As _  
MSForms.ReturnBoolean)  
On Error Resume Next  
Slider3 = TextBox5  
End Sub
```

Änderung der Breite und Höhe des Bilds

```
Private Sub TextBox6_BeforeUpdate(ByVal Cancel As _  
MSForms.ReturnBoolean)  
On Error Resume Next  
ActRaster.ImageHeight = Util.DistanceToReal(TextBox6)  
TextBox6 = Util.RealToString(ActRaster.ImageHeight)  
ActRaster.Update  
End Sub  
  
Private Sub TextBox7_BeforeUpdate(ByVal Cancel As _  
MSForms.ReturnBoolean)  
On Error Resume Next  
ActRaster.ImageWidth = Util.DistanceToReal(TextBox7)  
TextBox7 = Util.RealToString(ActRaster.ImageWidth)  
ActRaster.Update  
End Sub
```

Änderung der Option „Bild anzeigen“

```
Private Sub CheckBox1_Click()  
On Error Resume Next  
ActRaster.ImageVisibility = CheckBox1  
ActRaster.Update  
End Sub
```

Änderung der Option „Zugeschnittene anzeigen“

```
Private Sub CheckBox2_Click()  
On Error Resume Next  
ActRaster.ClippingEnabled = CheckBox2  
ActRaster.Update  
End Sub
```

Änderung der Option „Transparent“

```
Private Sub CheckBox3_Click()
On Error Resume Next
ActRaster.Transparency = CheckBox3
ActRaster.Update
End Sub
```

Änderung der Option „Bild anzeigen wenn gedreht“

```
Private Sub CheckBox4_Click()
On Error Resume Next
ActRaster.ShowRotation = CheckBox4
ActRaster.Update
End Sub
```

8.3.24 Regionen (AcadRegion)

Die Klasse *AcadRegion* stellt Ihnen die Eigenschaften und Methoden einer begrenzten, planaren Fläche, im Folgenden Region genannt, zur Verfügung. Diese Region kann Linien, Kreise, Bögen, elliptische Bögen und Splines enthalten. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen *region* oder *_region* aus den zuvor genannten Objekten erstellen.

Sie können einen Verweis auf eine solche Region erstellen, indem Sie mit der Funktion *AddRegion* eine begrenzte Fläche dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene Region über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften der Region

Die meisten Eigenschaften dieses Objekts sind schreibgeschützt und haben somit nur informativen Charakter.

Mit der *Centroid*-Eigenschaft ermitteln Sie den aktuellen Schwerpunkt der Region. Die Koordinaten des Schwerpunkts werden als 2D-Array (X, Y) ausgegeben.

MomentOfInertia ermittelt den Trägheitsmoment und *PrincipalMoments* den Hauptträgheitsmoment der Region. Beide Momente werden als 2D-Array (X, Y) ausgegeben. Über die *ProductOfInertia*-Eigenschaft können Sie den Deviationsmoment der Region ermitteln.

Der aktuelle Umfang der Region ist in der *Perimeter*-Eigenschaft gespeichert und der Flächeninhalt in der *Area*-Eigenschaft.

Die *RadiiOfGyration*-Eigenschaft ermittelt den Trägheitsradius der Region und gibt ihn als 2D-Array (X, Y) zurück. Die beiden Hauptrichtungen können Sie mit der *PrincipalDirections*-Eigenschaft abrufen. Beide Richtungen werden zusammen in einem 4D-Array zurückgegeben.

```
Dim Direct As Variant
Direct = ActRegion.PrincipalDirections
X1 = Direct(0) 'X-Koordinate der ersten Richtung
Y1 = Direct(1) 'Y-Koordinate der ersten Richtung
X2 = Direct(2) 'X-Koordinate der zweiten Richtung
Y2 = Direct(3) 'Y-Koordinate der zweiten Richtung
```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der Region fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Region zurück. Der Handle ist der eindeutige Identifier eines AutoCAD-Elements.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Region ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Region zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der Region an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Region an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Region an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Region an oder legt diese fest.
<i>Normal</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Region zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadRegion</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Region zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der Region steuern.

Tabelle 8.82: Weitere Eigenschaften der Region

Methoden der Region

Mithilfe der *Boolean*-Anweisung können Sie eine boolesche Operation anhand einer zweiten Region durchführen. (Einen Verweis auf diese zweite Region müssen Sie im zweiten Parameter der Anweisung angeben.) Die beiden Regionen werden dabei miteinander verschnitten und je nach Eingabe des ersten Parameters erhalten Sie eine neue Figur der aufrufenden Region. Sie können dabei folgende Konstanten als ersten Parameter verwenden.

Konstante	Beschreibung	Wert
<i>acUnion</i>	Vereinigt die beiden Regionen zu einer, indem der überlagernde Teil herausgeschnitten wird.	0
<i>acIntersection</i>	Verwenden Sie diese Konstante, um als Ergebnis die aufrufende Region so zu verändern, dass der durch beide Regionen überlagerte Teil erhalten bleibt	1
<i>acSubtraction</i>	Sollten Sie diese Konstante verwenden, so bleibt als Region nur noch der Teil erhalten, der durch die zweite Region nicht überlagert wurde.	2

Tabelle 8.83: Konstanten der Boolean-Anweisung

```
Dim SecRegion As Object
Dim Prompt As String
Dim PPoint As Variant
Prompt = "Wählen Sie eine zweite Region:"
Utility.GetEntity SecRegion, PPoint, Prompt
ActRegion.Boolean acUnion, SecRegion
```

Die *Explode*-Anweisung löst die Region in ihre Bestandteile auf und gibt ein Array mit den Verweisen auf die entstandenen Objekte zurück.

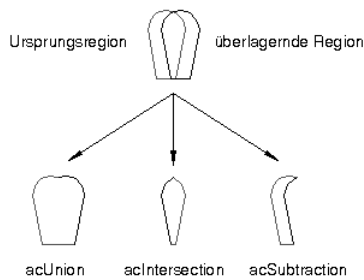


Abbildung 8.74: Ergebnisse der unterschiedlichen Überlagerungen

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Region ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Region ab.
<i>Highlight</i>	Hebt die Region hervor.

Tabelle 8.84: Weitere Methoden der Region

Prozedur	Beschreibung
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Region mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Kopie der Region als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine Kopie der Region als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt die Region von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die Region um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Region um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Region um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Region.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Region an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Region im AutoCAD-Zeichnungsfenster.

Tabelle 8.84: Weitere Methoden der Region (Forts.)

Anwendungsbeispiel

(Beispielprojekt 08\AcadRegion.dvb)

Im Beispielprojekt dieses Objekttyps werden wir die Informationen, die das Objekt liefert, in einem Formular zusammenfassend anzeigen lassen und dem Benutzer die Möglichkeit geben, Regionen zu überlagern und aufzulösen. Das Beispielprojekt enthält wieder ein Formular, in dem ein Strukturansicht-Steuerelement (*TreeView*) platziert wurde, das die einzelnen Informationen zur gewählten Region anzeigen wird.

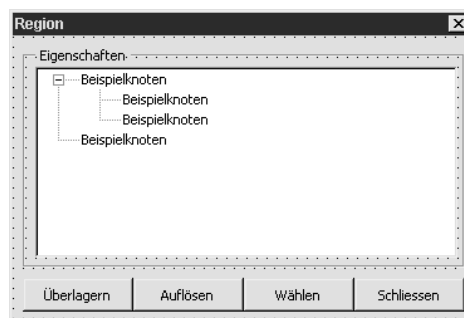


Abbildung 8.75: Hauptformular des Beispielprojekts

Startprozedur

Die Startprozedur des Projekts enthält wiederum keine Neuerungen gegenüber den vorangegangenen Projekten. Auch hier wird der Benutzer zunächst aufgefordert, ein Objekt in der Zeichnung zu wählen, das einer Region entspricht. Anschließend wird die Initialisierungsprozedur des Formulars und danach das Formular selbst gestartet.

```

Public Sub selectRegion()
Dim Object As Object
On Error Resume Next
Utility.GetEntity Object, PickedPoint, "Region wählen:"
If TypeName(Object) = "IAcadRegion" Then
    Set RegionFrm.ActRegion = Object
    RegionFrm.initRegionFrm
    RegionFrm.Show
End If
End Sub

```

Initialisierungsprozedur

In der Initialisierungsprozedur werden die einzelnen Eigenschaften in das *TreeView*-Steuerelement eingetragen, wobei diese hierarchisch angeordnet werden sollen. Dazu unterteilen wir die Prozedur in einzelne Abschnitte.

```

Public Sub initRegionFrm()
Dim xNode As MSComctlLib.Node
Dim yNode As MSComctlLib.Node
On Error Resume Next

```

Als Erstes werden die einzelnen Momentdaten dem Steuerelement hinzugefügt und zwar so, dass als oberste Kategorie der Eintrag „Momente“ erscheint. Der Verweis auf diesen Eintrag wird in der Variablen *xNode* gespeichert.

```

TreeView1.Nodes.Clear
Set xNode = TreeView1.Nodes.Add(, , "Momente")

```

Als nächste Kategorie wird das Trägheitsmoment hinzugefügt und der Verweis auf diesen Eintrag in der Variablen *yNode* gespeichert (*xNode* wird noch für die nächsten *Moment*-Eigenschaften benötigt). Dieser Kategorie werden wiederum die Werte des Moments zugeordnet.

```

Mnt = ActRegion.MomentOfInertia
Set yNode = TreeView1.Nodes.Add(xNode, tvwChild, , _
    "Trägheitsmoment")
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Mnt(0), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Mnt(1), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))

```

Nach dem gleichen Prinzip werden nun auch die Werte des Hauptträgheitsmoments hinzugefügt, zunächst der Kategorieeintrag (*yNode*) und anschließend die Werte des Moments.

```

Mnt = ActRegion.PrincipalMoments
Set yNode = TreeView1.Nodes.Add(xNode, tvwChild, , _
    "Hauptträgheitsmoment")
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Mnt(0), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Mnt(1), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))

```

Der Wert des Deviationsmoments wird über die Kategorien „Momente“ → „Deviationsmoment“ hinzugefügt, zunächst der Kategorieeintrag (*yNode*) und anschließend die Werte des Moments.

```
Set yNode = TreeView1.Nodes.Add(xNode, tvwChild, , _
    "Devationsmoment")
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(ActRegion.ProductOfInertia, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
```

Alle weiteren Eigenschaften der Region werden jeweils als Extrakategorie erzeugt.

Flächeninhalt:

```
Set xNode = TreeView1.Nodes.Add(, , , "Flächeninhalt")
TreeView1.Nodes.Add xNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(ActRegion.Area, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
```

Flächeninhalt:

```
Set xNode = TreeView1.Nodes.Add(, , , "Umfang")
TreeView1.Nodes.Add xNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(ActRegion.Perimeter, _
    acDefaultUnits, ThisDrawing.GetVariable("LUPREC"))
```

Trägheitsradius:

```
Mnt = ActRegion.RadiiOfGyration
Set xNode = TreeView1.Nodes.Add(, , , "Trägheitsradius")
TreeView1.Nodes.Add xNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Mnt(0), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
TreeView1.Nodes.Add xNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Mnt(1), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
```

Die Richtungsvektoren wiederum werden nach dem gleichen Prinzip wie die Momente angezeigt, zunächst die Hauptkategorie.

```
Set xNode = TreeView1.Nodes.Add(, , , "Hauptrichtungen")
Vektor = ActRegion.PrincipalDirections
```

Diese erhalten dann die beiden Unterkategorien „X-Vektor“ und „Y-Vektor“, die dann die jeweiligen X- und Y-Koordinaten aufnehmen.

X-Vektor:

```
Set yNode = TreeView1.Nodes.Add(xNode, tvwChild, , "X-Vektor")
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Vektor(0), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Vektor(1), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
```

Y-Vektor:

```

Set yNode = TreeView1.Nodes.Add(xNode, tvwChild, , "Y-Vektor")
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Vektor(2), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
TreeView1.Nodes.Add yNode, tvwChild, , _
    ThisDrawing.Utility.RealToString(Vektor(3), acDefaultUnits, _
    ThisDrawing.GetVariable("LUPREC"))
End Sub

```

Nachdem alle Werte in das *TreeView*-Steuerelement eingetragen wurden, ergibt sich ungefähr folgendes Bild für das Steuerelement:



Abbildung 8.76: TreeView-Steuerelement

Schaltfläche „Überlagern“

Die Schaltfläche ÜBERLAGERN soll es dem Benutzer ermöglichen, eine weitere Region in der Zeichnung zu wählen, mit der dann die bereits gewählte überlagert wird. Der Modus für diese Überlagerung wird zunächst in einem weiteren Formular gewählt.

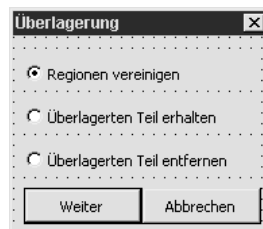


Abbildung 8.77: Auswahl des Überlagerungsmodus

Hat der Benutzer hier die Schaltfläche ABBRECHEN betätigt, wird die Formularvariable OK den Wert FALSE zurückgeben und die Prozedur kann vorzeitig verlassen werden.

```

Private Sub CommandButton1_Click()
Dim Object As Object
On Error Resume Next
BoolFrm.Show
If BoolFrm.OK = False Then Exit Sub

```

Andernfalls setzen wir die Ausführung an dieser Stelle fort. Zunächst wird das Formular ausgeblendet, damit der Benutzer die Region in der Zeichnung wählen kann.

```
Me.Hide
ThisDrawing.Utility.GetEntity Object, PickedPoint, _
    "überlagernde Region wählen:"
If TypeName(Object) = "IAcadRegion" Then
```

Hat er dies getan, müssen wir die Auswahl, die der Benutzer im Formular getätigt hat, überprüfen und die Anweisung *Boolean* mit den entsprechenden Parametern ausführen. Nachdem alles erledigt wurde, muss zunächst die Region aktualisiert (damit der Benutzer das Ergebnis sehen kann) und abschließend das Formular wieder eingeblendet werden.

```
    If BoolFrm.OptionButton1 = True Then
        ActRegion.Boolean acUnion, Object
    ElseIf BoolFrm.OptionButton2 = True Then
        ActRegion.Boolean acIntersection, Object
    Else
        ActRegion.Boolean acSubtraction, Object
    End If
    ActRegion.Update
    initRegionFrm
End If
Me.Show
End Sub
```

Schaltfläche „Auflösen“

Das Auflösen der Region stellt keine große Schwierigkeit dar. Nachdem die *Explode*-Anweisung ausgeführt wurde (die Verweise auf die neuen Objekte sind im Array *ObjList* gespeichert), wird die Region gelöscht und das Formular ausgeblendet, womit das Programm auch beendet wird.

```
Private Sub CommandButton3_Click()
On Error Resume Next
ObjList = ActRegion.Explode
ActRegion.Delete
Me.Hide
End Sub
```

Schaltfläche „Wählen“

Als Letztes bleibt noch der Code, der ausgeführt wird, wenn der Benutzer die Schaltfläche WÄHLEN betätigt. Auch hier gibt es nichts Neues. Der Code entspricht in etwa dem der Startprozedur, außer dass das Formular zunächst ausgeblendet wird.

```
Private Sub CommandButton4_Click()
Dim Object As Object
On Error Resume Next
Me.Hide
ThisDrawing.Utility.GetEntity Object, PickedPoint, _
    "Region wählen:"
If TypeName(Object) = "IAcadRegion" Then
    Set ActRegion = Object
    initRegionFrm
End If
Me.Show
End Sub
```

8.3.25 Spline (AcadSpline)

Splines sind quadratische oder kubische NURBS-Kurven (Non-Uniform Rational B-Spline), die anhand von Stützpunkten, Kontrollpunkten, Tangentenvektoren im Start- und Endsteuertpunkt definiert werden. Die Klasse *AcadSpline* stellt Ihnen die Eigenschaften und Methoden eines einzelnen Spline zur Verfügung. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen `spline` oder `_spline` in der Zeichnung erstellen.

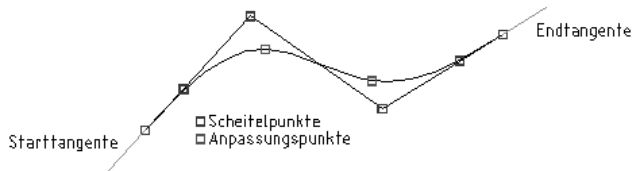


Abbildung 8.78: Definition des Spline-Objekts

Sie können einen Verweis auf einen Spline erstellen, indem Sie mit der Funktion *AddSpline* unter Angabe der Stützpunkte und der Start- sowie Endtangente einen Spline dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch einen vorhandenen Spline über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften des Spline-Objekts

Aufgrund der Anpassungsmöglichkeiten, die ein Spline besitzt, stellt auch die Klasse, mit der ein Spline in VBA abgebildet wird, neben den Standardeigenschaften eine ganze Reihe zusätzlicher Eigenschaften zur Verfügung. Die meisten dieser Eigenschaften dienen der Positionierung des Objekts, wobei hier zwischen den Scheitel- und den Anpassungspunkten zu unterscheiden ist.

Scheitelpunkte verwalten

Die Scheitelpunkte (Abbildung 8.78) des Spline werden in der Eigenschaft *ControlPoints* gespeichert. Wie viele Scheitelpunkte vorhanden sind, gibt die *NumberOfControlPoints*-Eigenschaft zurück.

Die *Degree*-Eigenschaft gibt Ihnen den Grad des Spline zurück. Je höher der Grad eines Spline ist, desto mehr Scheitelpunkte hat er.

Anpassungspunkte verwalten

Die Anpassungspunkte (Abbildung 8.78) werden mit der *FitPoints*-Eigenschaft verwaltet, wobei die Anzahl dieser Punkte die Eigenschaft *NumberOfFitPoints* wiedergibt.

Mit der Eigenschaft *FitFolerance* kann die Anpassungstoleranz des Spline abgerufen bzw. festgelegt werden. Enthält der Spline keine Anpassungspunkte, so enthält diese Eigenschaft den Wert 0. Bei Festlegung eines Werts größer als 0 kann der Spline innerhalb der festgelegten Toleranzgrenzen durch die Anpassungspunkte verlaufen.

Die Start- und Endtangente

Mit den Vektoren für die Start- und Endtangente können Sie die Eingangs- bzw. Ausgangsrichtung des Spline beeinflussen. Diese Vektorangaben sind in den Eigenschaften *StartTangent* und *EndTangent* enthalten. Diese Eigenschaft ist nicht unbedingt bei jedem Spline initialisiert. Falls keine Richtungsvektoren vorhanden sind, löst der Abruf der entsprechenden Eigenschaft einen Fehler aus.

Weitere Eigenschaften

Mit der Eigenschaft *Weights* können Sie die Stärkevektoren des Spline abrufen. Sie erhalten ein Array, das genauso viele Felder enthält, wie der Spline Scheitelpunkte besitzt. Jedem Scheitelpunkt ist somit ein Stärkevektor zugeordnet (siehe Anweisung *SetWeights*). Diese Eigenschaft ist nicht unbedingt bei jedem Spline initialisiert. Falls kein Stärkevektor vorhanden ist, löst der Abruf der Eigenschaft einen Fehler aus. Der Knotenvektor wird durch die *Knots*-Eigenschaft bestimmt. Der Knotenvektor wird dabei als 3D-Array zurückgegeben.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Area</i> (schreibgeschützt)	Diese Eigenschaft liefert den Flächeninhalt des Spline.
<i>Closed</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der entsprechende Spline geschlossen oder geöffnet ist.
<i>Color</i>	Legt die Farbe des Spline fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Spline zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Spline ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Spline zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>IsPeriodic</i> (schreibgeschützt)	Zeigt an, ob der entsprechende Spline periodisch ist oder nicht. Als Ergebnis dieser Abfrage können folgende Werte geliefert werden: <i>TRUE</i> – der Spline ist periodisch <i>FALSE</i> – der Spline ist nicht periodisch
<i>IsPlanar</i> (schreibgeschützt)	Zeigt an, ob der entsprechende Spline planar ist oder nicht. Als Ergebnis dieser Abfrage können folgende Werte geliefert werden: <i>TRUE</i> – der Spline ist planar <i>FALSE</i> – der Spline ist nicht planar
<i>IsRational</i> (schreibgeschützt)	Zeigt an, ob der entsprechende Spline rational ist oder nicht. Als Ergebnis dieser Abfrage können folgende Werte geliefert werden: <i>TRUE</i> – der Spline ist rational <i>FALSE</i> – der Spline ist nicht rational

Tabelle 8.85: Weitere Spline-Eigenschaften

Eigenschaft	Beschreibung
<i>Layer</i>	Zeigt den Layer des Spline an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Spline an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Spline an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für den Spline an oder legt diese fest.
<i>ObjectID</i> (schreibgeschützt)	Gibt die ObjectID des Spline zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadSpline</i> zurück.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für den Spline zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Spline steuern.

Tabelle 8.85: Weitere Spline-Eigenschaften (Forts.)

Methode des Spline-Objekts

Neben der Vielzahl an Eigenschaften zur Anpassung eines Spline verfügt das Objekt auch über eine ganze Reihe von Prozeduren und Funktionen, die die Darstellung des Spline erheblich beeinflussen können.

Anpassungspunkte verarbeiten

Diese *AddFitPoint*-Anweisung fügt einen Anpassungspunkt an einer bestimmten Stelle dem Spline hinzu. Als Parameter müssen Sie der Anweisung nur die Position und die Koordinaten des neuen Anpassungspunkts mitgeben. Legen Sie als *Index* einen negativen Wert fest, so wird der neue Punkt an den Anfang der Punktliste gestellt. Ist der Wert größer als die Anzahl der Anpassungspunkte (*NumberOfFitPoints*), wird der neue Punkt ans Ende der Punktliste gestellt. In allen anderen Fällen wird der Punkt hinter dem mit *Index* angegebenen Punkt gesetzt.

```
Dim KtPoint(2) As Double
Dim Prompt As String
Prompt = "Anpassungspunkt festlegen"
varPoint = Utility.GetPoint(, Prompt)
If TypeName(varPoint) = "Double()" Then
    KtPoint(0) = varPoint(0)
    KtPoint(1) = varPoint(1)
    KtPoint(2) = varPoint(2)
    ActSpline.AddFitPoint -1, KtPoint
End If
```

Die *GetFitPoint*-Anweisung ruft einen Anpassungspunkt an einer bestimmten Position des Spline ab. Diese Funktion gibt den entsprechenden Punkt als 3D-Array zurück. Als Parameter müssen Sie nur die Nummer des entsprechende Punkts dieser Funktion mitgeben. Der Bereich dieses Index liegt zwischen 0 und der Anzahl der Anpassungspunkte – 1. *SetFitPoint* hingegen ersetzt die Koordinaten eines Anpassungspunkts an einer bestimmten Position des Spline. Diese Funktion verlangt neben der Angabe der Position (0 bis Anzahl der Anpassungspunkte – 1) auch die neuen Koordinaten des entsprechende Punkts, die als 3D-Array deklariert werden.


```
Dim ApPoint As Variant
ApPoint = ActSpline.GetFitPoint(0)
ApPoint = Utility.GetPoint(ApPoint, "neue Punktposition")
ActSpline.SetFitPoint 0, ApPoint
```

DeleteFitPoint löscht den über den Parameter *Index* angegebenen Anpassungspunkt.

```
Dim Prompt As String
Dim Index As Long
Prompt = "Nummer des Anpassungspunkts:"
varPoint = Utility.GetInteger(Prompt)
ActSpline.DeleteFitPoint Index
```

Siehe auch *NumberOfFitPoints*.

Scheitelpunkte anpassen

Mit der *GetControlPoint*-Anweisung können Sie einen Scheitelpunkt an einer bestimmten Position des Spline abrufen. Diese Funktion gibt den entsprechenden Punkt als 3D-Array zurück. Als Parameter müssen Sie die Nummer des gesuchten Punkts dieser Funktion mitgeben. Der Bereich dieses Index liegt zwischen 0 und der Anzahl der Scheitelpunkte – 1.

```
Dim KtPoint As Variant
KtPoint = ActSpline.GetControlPoint(0)
```

Das Gegenstück zu *GetControlPoint* ist logischerweise *SetControlPoint*. Mit dieser Anweisung können Sie die Koordinaten eines Scheitelpunkts an einer bestimmten Position des Spline ersetzen. Als Parameter müssen Sie sowohl die Nummer als auch die neuen Koordinaten des entsprechenden Punkts dieser Funktion mitgeben. Der Bereich dieses Index liegt zwischen 0 und der Anzahl der Scheitelpunkte – 1. Die Koordinaten müssen als 3D-Array deklariert werden.

```
Dim KtPoint As Variant
KtPoint = ActSpline.GetControlPoint(0)
KtPoint = Utility.GetPoint(KtPoint, "neue Punktposition")
ActSpline.SetControlPoint 0, KtPoint
```

Um das Gewicht eines bestimmten Scheitelpunkts abzurufen, müssen Sie die Funktion *GetWeight* verwenden. Als Parameter geben Sie auch hier wieder die Nummer des Scheitelpunkts mit. Der Bereich dieses Index liegt zwischen 0 und der Anzahl der Scheitelpunkte – 1.

```
Dim KtWeight As Double
KtWeight = ActSpline.GetWeight(0)
KtWeight = Utility.GetReal("neue Stärke für den Scheitelpunkt:")
ActSpline.SetWeight 0, KtWeight
```

Mithilfe der *SetWeight*-Anweisung können Sie das Gewicht eines bestimmten Scheitelpunkts verändern. Neben der Nummer, die die Position des gewünschten Scheitelpunkts bestimmt, verlangt die Funktion zusätzlich die Stärke. Der Bereich dieses Index liegt zwischen 0 und der Anzahl der Scheitelpunkte – 1. Ein größerer Wert für das Gewicht zieht den Spline näher an den Kontrollpunkt heran.

Siehe auch *NumberOfControlPoints*, *Weights*, *SetWeight*.

Weitere Methoden

Um die Anpassungsdaten des Spline zu bereinigen, müssen Sie die Anweisung *PurgeFitData* aufrufen. Daraufhin werden alle Anpassungspunkte des Spline entfernt. Außerdem verfügt der Spline dann auch nicht mehr über Start- und Endtangentenvektoren. Er wird dann nur noch durch die Scheitelpunkte definiert. Einen ähnlichen Effekt erzielen Sie mit der *ElevateOrder*-Anweisung. Mit dieser Anweisung kann der Spline-Grad erhöht werden. Nachdem diese Anweisung ausgeführt wurde, wird der Spline von einem *fitSpline* in einen *ControlPointSpline* konvertiert. Das bedeutet, dass der Spline danach keine Anpassungspunkte, keine Anpassungs-, Start- und Endtangente mehr enthält. Die Eigenschaften *FitPoints*, *FitTolerance*, *StartTangent* und *EndTangent* sind somit nicht mehr verfügbar. Im Parameter dieser Anweisung müssen Sie den Wert angeben, um den der Spline-Grad erhöht werden soll. Dieser Wert muss immer größer als der Wert der *Degree*-Eigenschaft sein (zeigt den aktuellen Spline-Grad an). Der maximale Wert dieser Eigenschaft beträgt 26. Wenn der Spline-Grad einmal erhöht wurde, dann können Sie diesen Wert nicht mehr verringern (ausgenommen mit der *undo*-Funktion von AutoCAD)

Eine weitere Methode dieses Objekts ist *Reverse*. Wenn Sie diese Anweisung ausführen, dann wird die Richtung des Spline umgekehrt.

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Spline ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Spline ab.
<i>Highlight</i>	Hebt den Spline in der Zeichnung hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Spline mit einem anderen AutoCAD-Objekt ergeben. <i>Hinweis:</i> Auf Grund der Komplexität eines Spline kann es vorkommen, dass diese Funktion in einigen Fällen keine Ergebnisse liefert, obwohl es offensichtlich ist, dass es Schnittpunkte gibt. In diesem Fall hilft dann nur eine geringfügige Verschiebung der Schnittlinien, um ein korrektes Ergebnis zu erhalten.
<i>Mirror</i>	Erstellt eine Kopie des Spline als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine Kopie des Spline als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt den Spline von den Ursprungs- auf die Zielkoordinaten.

Tabelle 8.86: Weitere Spline-Methoden

Prozedur	Beschreibung
<i>Offset</i>	Diese Anweisung versetzt den Spline um einen bestimmten Betrag. Nach Ausführung der Funktion wird ein neuer Spline erstellt und als Objektvariable zurückgegeben
<i>Rotate</i>	Dreht den Spline um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Spline um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert den Spline um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Spline.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf den Spline an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Spline im AutoCAD-Zeichnungsfenster.

Tabelle 8.86: Weitere Spline-Methoden (Forts.)

Anwendungsbeispiel

(Beispielprojekt: 08\AcadSpline.dvb)

Mit dem folgenden Anwendungsbeispiel erhält der Benutzer die Möglichkeit, die Eigenschaften eines beliebigen Spline in einem Formular zu verarbeiten. Die Koordinaten der Scheitel- und Anpassungspunkte (sofern vorhanden) werden jeweils in einem separaten *ListView*-Steuerelement zur Auswahl angeboten. Zusätzlich wurde wieder die Klasse *MyUtility* in das Projekt implementiert.

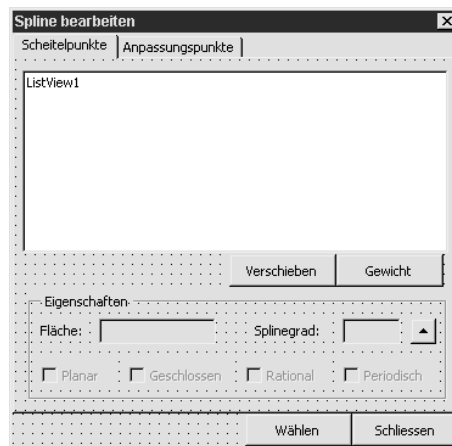


Abbildung 8.79: Formular des Beispielprojekts

Das Projekt selbst unterteilt sich wieder in zwei Teilabschnitte: die Startprozedur (wurde wie immer im Klassenmodul *ThisDrawing* abgelegt) und der Formularcode (übernimmt die Hauptarbeit des Programms). Das Besondere an diesem Projekt ist, dass das Formular ungebunden angezeigt wird und der Benutzer somit in der Zeichnung weiterarbeiten kann. Damit das Programm sofort auf Änderungen des Benutzers am Spline reagieren kann, die nicht über

das Formular durchgeführt wurden, wird der Verweis auf den gewählten Spline diesmal auf eine besondere Weise in das Formular eingebunden. Wir wollen die Eigenschaft grafischer Objekte nutzen, um auf Änderungen zu reagieren und die Ereignisprozedur *Modified* auszulösen, falls eine Änderung vorgenommen wurde. Dazu wird die Objektvariable *ActSpline* (speichert den Verweis auf den gewählten Spline) in dem Formular wie folgt deklariert:

```
Public WithEvents ActSpline As AcadSpline
```

Damit wird die Ereignisprozedur *Modified* im Formular verfügbar und wir können die entsprechende Reaktion auf Änderungen am Spline ausführen. Der einzige Befehl, der hier ausgeführt werden soll, ist der Aufruf der Initialisierungsprozedur, die den Inhalt des Formulars aktualisiert.

```
Private Sub ActSpline_Modified(ByVal pObject As IAcadObject)
    initSplineFrm
End Sub
```

Startprozedur

In der Startprozedur (*ThisDrawing.SplineEdit*) wird der Benutzer aufgefordert, einen Spline in der aktuellen Zeichnung auszuwählen. Hat er dies erfolgreich getan, wird der Verweis auf diesen Spline in der zuvor angesprochenen Objektvariablen *ActSpline* gespeichert, die Initialisierungsprozedur ausgeführt und abschließend das Formular ungebunden eingeblendet.

```
Public Sub SplineEdit()
    Dim Object As Object
    On Error Resume Next
    Set Util = New MyUtility
    Util.GetEntity Object, PickedPoint, "Spline wählen:"
    If TypeName(Object) = "IAcadSpline" Then
        Set SplineFrm.ActSpline = Object
        SplineFrm.initSplineFrm
        SplineFrm.Show 0
    End If
End Sub
```

Initialisierungsprozedur

Da der Spline über ein Reihe von Informationen verfügt, die es zu verarbeiten gilt, ist auch diese Prozedur entsprechend umfangreich. Generell unterteilt sie sich in drei Abschnitte. Der erste füllt das *ListView*-Steuerelement, das die Daten der Scheitelpunkte aufnimmt, der zweite füllt das *ListView*-Steuerelement, das die Daten der Anpassungspunkte enthalten soll, und der letzte überträgt die restlichen Daten des Spline auf die verschiedensten Steuerelemente.

```
Public Sub initSplineFrm()
    Dim LV As MSComctlLib.ListItem
    On Error Resume Next
    If NoChange = True Then Exit Sub
```

Die Scheitelpunkte auszulesen, bereitet keine Probleme. Wir müssen hierfür nur eine Schleife durchlaufen, die vom Index 0 bis zur Anzahl der Scheitelpunkte – 1 zählt. In jedem Durchlauf wird dann der entsprechende Scheitelpunkt ausgelesen und den jeweiligen Spalten hinzugefügt. In die letzte Spalte wird dann noch das Gewicht des Scheitelpunkts eingetragen, das mit der Funktion *GetWeight* abgerufen werden kann.

```

ListView1.ListItems.Clear
For i = 0 To ActSpline.NumberOfControlPoints - 1
    CtlWeight = ActSpline.GetWeight(i)
    CtlPoint = ActSpline.GetControlPoint(i)
    Set LV = ListView1.ListItems.Add(, , CStr(i + 1))
    LV.ListSubItems.Add , , Util.RealToString(CtlPoint(0))
    LV.ListSubItems.Add , , Util.RealToString(CtlPoint(1))
    LV.ListSubItems.Add , , Util.RealToString(CtlPoint(2))
    LV.ListSubItems.Add , , Util.RealToString(CtlWeight)
Next

```

Das Auslesen der Anpassungspunkte funktioniert auf die gleiche Weise. Innerhalb einer Schleife, die von 0 bis zur Anzahl der Anpassungspunkte verläuft, wird bei jedem Durchlauf der entsprechende Punkt ausgelesen und die Position und die Koordinaten werden dem zweiten *ListView*-Steuerelement hinzugefügt. Da es vorkommen kann, dass ein Spline keine Anpassungspunkte enthält, wird dies nach der Schleife noch geprüft und anschließend werden, je nach Ergebnis, die Schaltflächen LÖSCHEN und VERSCHIEBEN aktiviert bzw. deaktiviert.

```

ListView2.ListItems.Clear
For i = 0 To ActSpline.NumberOfFitPoints - 1
    ApPoint = ActSpline.GetFitPoint(i)
    Set LV = ListView2.ListItems.Add(, , CStr(i + 1))
    LV.ListSubItems.Add , , Util.RealToString(ApPoint(X))
    LV.ListSubItems.Add , , Util.RealToString(ApPoint(Y))
    LV.ListSubItems.Add , , Util.RealToString(ApPoint(Z))
Next
If ActSpline.NumberOfFitPoints = 0 Then
    CommandButton3.Enabled = False
    CommandButton4.Enabled = False
Else
    CommandButton3.Enabled = True
    CommandButton4.Enabled = True
End If

```

Im dritten Abschnitt der Prozedur werden nun noch die restlichen Daten an die entsprechenden Steuerelemente übergeben.

```

TngPt = ActSpline.StartTangent
STangentX = "": STangentY = "": STangentZ = ""
If IsEmpty(TngPt) = False Then
    STangentX = Util.RealToString(TngPt(0))
    STangentY = Util.RealToString(TngPt(1))
    STangentZ = Util.RealToString(TngPt(2))
    STangentBtn.Enabled = True
Else
    STangentBtn.Enabled = False
End If
TngPt = Empty
TngPt = ActSpline.EndTangent
ETangentX = "": ETangentY = "": ETangentZ = ""
ETangentBtn.Enabled = True
If IsEmpty(TngPt) = False Then
    ETangentX = Util.RealToString(TngPt(0))
    ETangentY = Util.RealToString(TngPt(1))
    ETangentZ = Util.RealToString(TngPt(2))

```

```

    ETangentBtn.Enabled = True
Else
    ETangentBtn.Enabled = False
End If
Label2 = Util.RealToString(ActSpline.Area)
Label3 = ActSpline.Degree
CheckBox1 = ActSpline.Closed
CheckBox2 = ActSpline.IsPlanar
CheckBox3 = ActSpline.IsRational
End Sub

```

Schaltfläche „Verschieben“

Der Code, der ausgeführt wird, wenn der Benutzer die Schaltfläche VERSCHIEBEN betätigt, egal ob die Schaltfläche für die Scheitel- oder die Anpassungspunkte, gleicht sich vom Grunde her. Zunächst wird die ursprüngliche Koordinate dieses Punkts ermittelt, um diese Position bei der Punktauswahl anzuzeigen. Dann wird der gewählte Punkt an den im entsprechenden *List-View*-Steuerelement selektierten Scheitel- bzw. Anpassungspunkt übergeben.

Scheitelpunkt verschieben

```

Private Sub CommandButton5_Click()
On Error Resume Next
PtPoint = ActSpline.GetControlPoint(ListView1.SelectedItem. _
    Index - 1)
NewPoint = Util.GetPoint(PtPoint, "neue Scheitelpunktposition:")
If TypeName(NewPoint) = "Double()" Then
    ActSpline.SetControlPoint ListView1.SelectedItem.Index - 1, _
        NewPoint
End If
End Sub

```

Anpassungspunkt verschieben

```

Private Sub CommandButton4_Click()
On Error Resume Next
PtPoint = ActSpline.GetFitPoint(ListView2.SelectedItem.Index - 1)
NewPoint = Util.GetPoint(PtPoint, _
    "neue Position des Anpassungspunkts:")
If TypeName(NewPoint) = "Double()" Then
    ActSpline.SetFitPoint ListView2.SelectedItem.Index - 1, _
        NewPoint
End If
End Sub

```

Schaltfläche „Gewicht“

Wenn der Benutzer ein neues Gewicht für die ausgewählten Scheitelpunkte festlegen möchte, dann müssen wir dieses Gewicht zunächst abfragen (*InputBox*). Hat er einen Wert eingegeben, wird das *List-View*-Steuerelement Zeile für Zeile durchlaufen und geprüft, ob der Punkt selektiert wurde. Ist dies der Fall, dann kann das entsprechende Gewicht dem ausgewählten Scheitelpunkt zugeordnet werden. Wichtig in dieser Prozedur ist das Setzen der Variablen *NoChange* auf *TRUE*, bevor die Schleife beginnt. Das hat einen einfachen Grund. Wenn eine Änderung am Spline durchgeführt wird, was das Setzen des Gewichts ja ist, dann löst dies das

Ereignis *Modified* aus, das wiederum die Prozedur *initSplineFrm* ausführen wird. Diese Prozedur wird allerdings vorzeitig verlassen, wenn die Variable *NoChange* auf *TRUE* gesetzt wurde. Dies ist wichtig, da sonst die Liste neu gelesen wird und die Auswahl verloren gehen würde. Nachdem das Setzen des Gewichts abgeschlossen wurde, wird die Variable *NoChange* wieder zurückgesetzt und die Prozedur *initSplineFrm* manuell ausgelöst.

```
Private Sub CommandButton7_Click()
On Error Resume Next
sValue = InputBox("Geben Sie das neue Gewicht für die " & _
    "gewählten Scheitelpunkte an:", "Gewicht", _
    ListView1.SelectedItem.ListSubItems(4))
If sValue = "" Then Exit Sub
NoChange = True
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        ActSpline.SetWeight i - 1, Util.DistanceToReal(CStr(sValue))
    End If
Next
NoChange = False
initSplineFrm
End Sub
```

Schaltfläche „Löschen“

Da es dem Benutzer in diesem Projekt möglich ist, gleichzeitig mehrere Anpassungspunkte in einem Arbeitsgang zu entfernen, muss die Ereignisprozedur entsprechend aufgebaut sein. Nach einer Sicherheitsabfrage wird zunächst die Variable *NoChange* auf *TRUE* gesetzt (soll verhindern, dass die Initialisierungsprozedur automatisch ausgeführt wird), um dann eine Schleife in umgekehrter Reihenfolge (vom letzten zum ersten Eintrag) auszuführen. Die Ausführung in dieser Richtung ist notwendig, da bei einer Mehrfachauswahl sonst die Reihenfolge der Punkte durcheinander kommt. Stellen Sie sich vor, Sie wollen von vier Punkten den ersten und dritten löschen. Wenn Sie nun zunächst den ersten entfernen, dann rücken die anderen alle um eine Position nach vorn und der vormals vierte Punkt steht nun an dritter Position. Wenn Sie das Pferd von hinten aufzäumen, kann dies nicht passieren, da eine Entfernung hinterer Punkte auf die Position der vorderen keinen Einfluss hat.

Die Schleife wird nun also durchlaufen und sobald ein Listeneintrag gefunden wird, der durch den Benutzer selektiert wurde, kann die Anweisung *DeleteFitPoint* unter Angabe der Position aufgerufen und der entsprechende Punkt damit entfernt werden.

```
Private Sub CommandButton3_Click()
On Error Resume Next
If MsgBox("Gewählte Anpassungspunkte entfernen?", vbYesNo, _
    "Löschen") = vbYes Then
    NoChange = True
    For i = ListView2.ListItems.Count To 1 Step -1
        If ListView2.ListItems(i).Selected = True Then
            ActSpline.DeleteFitPoint i - 1
        End If
    Next
    NoChange = False
    initSplineFrm
End If
End Sub
```

Schaltfläche „Hinzufügen“

Diese Schaltfläche wird der Benutzer immer dann betätigen, wenn er einen Anpassungspunkt dem Spline hinzufügen möchte. Zunächst ermitteln wir hier die Koordinaten des Punkts, der im *ListView*-Steuerelement ausgewählt wurde. Diese Punktkoordinaten werden dann der *GetPoint*-Funktion mitgegeben, um ein Gummiband darzustellen, damit der Benutzer eine optische Anzeige bekommt, wo der Punkt eingefügt wird. Mit den ermittelten Koordinaten wird die Anweisung *AddFitPoint* aufgerufen. Als Position für den neuen Anpassungspunkt geben wir einfach den Index des ausgewählten Punkts an, damit der neue Punkt hinter diesem eingefügt wird.

```
Private Sub CommandButton2_Click()
On Error Resume Next
PtPoint = ActSpline.GetFitPoint(ListView2.SelectedItem. _
    Index - 1)
NewPoint = Util.GetPoint(PtPoint, _
    "Position des neuen Anpassungspunkts:")
If TypeName(NewPoint) = "Double()" Then
    ActSpline.AddFitPoint ListView2.SelectedItem.Index, NewPoint
End If
End Sub
```

Schaltfläche „Bereinigen“

Der Code dieser Prozedur ist einfach. Nach einer Sicherheitsabfrage wird nur die Anweisung *PurgeFitData* aufgerufen, den Rest erledigt AutoCAD. (Da eine Änderung am Spline durchgeführt wurde, wird auch die Prozedur *initSplineFrm* automatisch ausgeführt.)

```
Private Sub CommandButton6_Click()
On Error Resume Next
If MsgBox("Sollen die Anpassungspunkte jetzt bereinigt werden?", _
    vbYesNo, "Anpassungspunkte bereinigen") = vbYes Then
    ActSpline.PurgeFitData
End If
End Sub
```

Tangente auswählen

Die Schaltflächen für die Auswahl der Tangenten sind in diesem Fall keine richtigen Schaltflächen, sondern vielmehr *Image*-Steuerelemente, die zur Schaltfläche umfunktioniert wurden. Dies wurde notwendig, da eine Symbolanzeige auf so kleiner Fläche mit einer normalen Schaltfläche nicht möglich wäre. Der Code für die Auswahl der Tangentenrichtungen wird trotzdem in der *Click*-Ereignisprozedur ausgeführt, die ausgelöst wird, wenn der Benutzer das Steuerelement anklickt. Der Benutzer wird also kaum einen Unterschied bemerken.

Der Code für beide Tangenten ist nahezu identisch, nur dass für die Starttangente der erste und für die Endtangente der letzte Scheitelpunkt als Bezug gewählt wird. Hat der Benutzer einen neuen Richtungspunkt gewählt, können die Richtungen bestimmt werden. Für die Eingangstangente werden die Einzelkoordinaten des Neupunkts von den Koordinaten des ersten Scheitelpunkts abgezogen (ergibt einen Eingangsvektor). Der Ausgangsvektor wird genau umgekehrt ermittelt. Die Koordinaten des letzten Scheitelpunkts werden von den Koordinaten des Neupunkts abgezogen.

Starttangente

```

Private Sub STangentBtn_Click()
Dim NewVector(2) As Double
On Error Resume Next
StpPoint = ActSpline.GetControlPoint(0)
VecPt = Util.GetPoint(StpPoint, "neue Tangentenrichtung:")
If TypeName(VecPt) = "Double()" Then
    NewVector(0) = StpPoint(0) - VecPt(0)
    NewVector(1) = StpPoint(1) - VecPt(1)
    NewVector(2) = StpPoint(2) - VecPt(2)
    ActSpline.StartTangent = NewVector
End If
End Sub

```

Endtangente

```

Private Sub ETangentBtn_Click()
Dim NewVector(2) As Double
On Error Resume Next
EndPoint = ActSpline.GetControlPoint( _
    ActSpline.NumberOfControlPoints - 1)
VecPt = Util.GetPoint(EndPoint, "neue Tangentenrichtung:")
If TypeName(VecPt) = "Double()" Then
    NewVector(0) = VecPt(0) - EndPoint(0)
    NewVector(1) = VecPt(1) - EndPoint(1)
    NewVector(2) = VecPt(2) - EndPoint(2)
    ActSpline.EndTangent = NewVector
End If
End Sub

```

Spline-Grad erhöhen

Um den Spline-Grad zu erhöhen, muss der Benutzer lediglich die Schaltfläche neben dem Feld wählen, die den aktuellen Grad anzeigt. Er wird daraufhin aufgefordert, einen Spline-Grad einzugeben, der dann mit der Anweisung *ElevateOrder* dem Spline zugeordnet wird.

```

Private Sub Image1_Click()
On Error Resume Next
sValue = InputBox("Splinegrad erhöhen auf:", "Splinegrad")
If sValue = "" Then Exit Sub
ActSpline.ElevateOrder CLng(sValue)
If Err <> 0 Then
    MsgBox Err.Description, vbCritical
End If
End Sub

```

8.3.26 Geometrisches Toleranzobjekt (AcadTolerance)

Die Klasse *AcadTolerance* stellt Ihnen die Eigenschaften und Methoden eines Toleranzobjekts zur Verfügung. Toleranzobjekte werden in Zusammenhang mit anderen grafischen Objekten verwendet und sollen die maximal zulässige Abweichung von Form oder Profil, Ausrichtung,

Position und Lauf der Objekte von der exakten Geometrie einer Zeichnung definieren und bestehen aus einem Textobjekt (stellt die Toleranzwerte einschließlich der Symbole dar) und einer Bemaßungslinie, die die einzelnen Textabschnitte umrandet. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen `toleranz` oder `_tolerance` in der Zeichnung erstellen.

Sie können einen Verweis auf ein Toleranzobjekt erstellen, indem Sie mit der Funktion `AddTolerance` unter Angabe der Zeichenfolge, des Einfügepunkts und des Ausrichtungsvektors ein Toleranzobjekt dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch ein vorhandenes Toleranzobjekt über die `Item`-Funktion der entsprechenden Auflistung (`ModelSpace`, `PaperSpace`, `Block`, `SelectionSet`, `Group`) ermitteln.

Eigenschaften

Mit der `TextColor`-Eigenschaft können Sie die Farbe des Maßtextes und mit der Eigenschaft `DimensionLineColor` die Farbe der Bemaßungslinie des Toleranzobjekts bestimmen. Es gelten die üblichen Farbwerte des Systems (0–256). Die Höhe des Textes wird in der `TextHeight`-Eigenschaft gespeichert und `TextStyle` bestimmt den Stil des Textes. Der Zugriff auf einzelne Textstile über die `TextStyles`-Auflistung wird in Kapitel 7.7 genauer beschrieben.

Mit der `TextString`-Eigenschaft können Sie die Zeichenfolge des Textes abrufen oder festlegen. Die eingegebene Zeichenfolge bestimmt auch den Aufbau des Toleranzobjekts. Geben Sie hier eine einfache Zeichenfolge ohne den Zusatz von Steuerzeichen an, wird ein umrahmter Text erzeugt. Durch die zusätzliche Angabe bestimmter Steuerzeichen (Tabelle 8.87) können Symbole, Trennungslinien und Zeilenumbrüche in das Toleranzobjekt eingefügt werden.

Definition	Beschreibung
<code>{\Fgdt;}</code>	Bestimmt eine Symboldefinition. Der dem Semikolon folgende Buchstabe legt das verwendete Symbol fest, z.B. <code>{\Fgdt;n}</code> für ein Durchmesserzeichen a b c d e f g h i j           k l m n p r s t u         
<code>%%v</code>	Legt eine senkrechte Trennungslinie fest.
<code>vbLf</code>	Diese Konstante ermöglicht einen Zeilenumbruch, womit die Definition in der nächsten Zeile fortgesetzt wird. erste Zeile zweite Zeile

Tabelle 8.87: Steuerzeichen für geometrische Toleranzen

Die Darstellung der Abbildung 8.80 wird nach der Definition aus der Tabelle 8.87 durch folgende Zeichenfolge bestimmt:

```
"{\Fgdt;j}%%v{\Fgdt;n}0.08{\Fgdt;s}" & vbLf & "1.000{\Fgdt;p}"
```

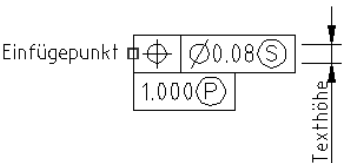


Abbildung 8.80: Geometrische Toleranz

die sich wie folgt zusammensetzt:

{\Fgdt;j}	%%v	{\Fgdt;n}	0.08	{\Fgdt;s}	VbLf	1.000	{\Fgdt;n}
			0.08		Fortsetzung mit der nächsten Zeile	1.000	

In der *StyleName*-Eigenschaft ist der Name des zugeordneten Bemaßungsstils für das Toleranzobjekt gespeichert. Sie können den aktuellen Namen abrufen oder einen neuen Stil zuordnen, indem Sie den Namen des entsprechenden Bemaßungsstils an diese Eigenschaft übergeben. Der aktuelle Bemaßungsstil ist in der Systemvariablen *DIMSTYLE* gespeichert. Der Breitenfaktor des Objekts wird mit der *ScaleFactor*-Eigenschaft festgelegt.

Toleranzposition

Die Position des Toleranzobjekts wird durch zwei Eigenschaften bestimmt. Der Einfügapunkt des Toleranzobjekts wird in *InsertionPoint* gespeichert und der Richtungsvektor wird mit *DirectionVector* bestimmt. Der Richtungsvektor zeigt dabei die Ausrichtung des Objekts unabhängig von den Einheiteneinstellungen des Systems an. Die jeweiligen Vektoren liegen in einem Wertebereich von -1 bis 1.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe des Toleranzobjekts fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Toleranzobjekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Toleranzobjekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Toleranzobjekts zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer des Toleranzobjekts an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Toleranzobjekts an oder legt diesen fest.

Tabelle 8.88: Weitere Eigenschaften des Toleranzobjekts

Eigenschaft	Beschreibung
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Toleranzobjekts an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuellen Linienstärke für das Toleranzobjekt an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Toleranzobjekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadTolerance</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für das Toleranzobjekt zurück oder legt diesen fest.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit des Toleranzobjekts steuern.

Tabelle 8.88: Weitere Eigenschaften des Toleranzobjekts (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechteckige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Toleranzobjekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Toleranzobjekts ab.
<i>Highlight</i>	Hebt das Toleranzobjekt in der AutoCAD-Zeichnung hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Toleranzobjekts mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt ein Toleranzobjekt als Resultat einer Spiegelung
<i>Mirror3D</i>	Erstellt ein neues Toleranzobjekt als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt das Toleranzobjekt von den Ursprungs- auf die Zielkoordinaten.

Tabelle 8.89: Methoden des Toleranzobjekts

Prozedur	Beschreibung
<i>Rotate</i>	Dreht das Toleranzobjekt um einen bestimmten Winkel
<i>Rotate3D</i>	Dreht das Toleranzobjekt um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Toleranzobjekt um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Toleranzobjekts.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Toleranzobjekt an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Toleranzobjekts im AutoCAD-Zeichnungsfenster.

Tabelle 8.89: Methoden des Toleranzobjekts (Forts.)

Anwendungsbeispiel

(Beispielprojekt 08\AcadTolerance.dvb)

Um Ihnen das Prinzip des Aufbaus der *TextString*-Eigenschaft verständlicher zu machen, werden wir im folgenden Anwendungsbeispiel Toleranzobjekte über ein Formular erzeugen lassen. In diesem Formular soll der Benutzer das Aussehen des Toleranzobjekts zusammenstellen können. Es ist ähnlich aufgebaut wie der Dialog des `_TOLERANCE`-Befehls von AutoCAD. Der Unterschied zum AutoCAD-Befehl besteht darin, dass die Symbole, die das Toleranzobjekt unterstützt, komplett über ein Formular (Abbildung 8.82) auswählbar sein sollen. Nachdem der Benutzer dann alle Einstellungen für das neue Toleranzobjekt vorgenommen hat und die Schaltfläche **SPEICHERN** betätigt, werden wir mittels einer Funktion aus den Angaben des Benutzers eine Zeichenfolge zusammenstellen, die dem Toleranzobjekt zu seinem Aussehen verhilft. Anhand dieser Funktion werden Sie feststellen, dass das Prinzip des Aufbaus dieser Zeichenfolge recht einfach zu handhaben ist.

Abbildung 8.81: Formular des Beispielprojekts

Startprozedur

Nachdem der Benutzer das Programm gestartet hat, wird zunächst wieder die Startprozedur des Projekts ausgeführt. Im ersten Teil dieser Prozedur wird, neben der Deklaration einiger Variablen und der Erzeugung einer neuen Instanz der Klasse *MyUtility*, die Initialisierungsprozedur des Hauptformulars ausgeführt und dann das Formular gebunden aufgerufen. Der Code der Prozedur wird erst wieder fortgesetzt, wenn der Benutzer das Formular beendet hat.

```

Public Sub createTolerance()
Dim NewTolerance As AcadTolerance
Dim Direction
On Error Resume Next
Set Util = New MyUtility
ToleranceFrm.initToleranzFrm
ToleranceFrm.Show

```

Die Fortsetzung erfolgt dann an dieser Stelle. Hat der Benutzer im Formular die Schaltfläche **SPEICHERN** betätigt, ist die Formularvariable *OK* auf *TRUE* gesetzt und der Code innerhalb des *If...End If*-Blocks kann ausgeführt werden. Da wir für die Erzeugung des Toleranzobjekts noch zusätzliche Angaben des Benutzers benötigen, wird zunächst der Einfügapunkt und danach der Richtungsvektor mithilfe der *GetPoint*-Funktion der Klasse *MyUtility* abgefragt.

```

If ToleranceFrm.OK = True Then
    InsPoint = Util.GetPoint(, "Einfügapunkt wählen")
    If TypeName(InsPoint) = "Double()" Then
        Direction = Util.GetPoint(InsPoint, "Richtung:")
        If IsEmpty(Direction) = True Then
            ReDim Direction(2) As Double
            Direction(0) = 0
            Direction(1) = 0
            Direction(2) = 0
        Else
            Direction(0) = Direction(0) - InsPoint(0)
            Direction(1) = Direction(1) - InsPoint(1)
            Direction(2) = Direction(2) - InsPoint(2)
        End If
    End If

```

Die Erzeugung des Toleranzobjekts kann nun mit der *AddTolerance*-Funktion erfolgen, die entweder auf das Modellbereichs- oder das Papierbereichsobjekt angewendet wird, je nachdem, in welchem Bereich sich AutoCAD gerade befindet. Interessant ist hierbei die Formularfunktion *GetText*. Diese wird die Zusammenstellung des Benutzers im Formular in eine Zeichenfolge umwandeln, die das Toleranzobjekt darstellen kann.

```

With ToleranceFrm
    If ActiveSpace = acModelSpace Then
        Set NewTolerance = ModelSpace.AddTolerance(.GetText, _
            InsPoint, Direction)
    Else
        Set NewTolerance = PaperSpace.AddTolerance(.GetText, _
            InsPoint, Direction)
    End If

```

Abschließend folgt nur noch die Zuweisung einiger Einstellungen wie Farbe, Texthöhe usw. an die entsprechenden Eigenschaften des entstandenen Toleranzobjekts:

```

        NewTolerance.TextColor = .ActiveColor
        NewTolerance.DimensionLineColor = .ActiveLineColor
        NewTolerance.TextHeight = Util.DistanceToReal(.TextBox1)
        NewTolerance.StyleName = .ComboBox1
        NewTolerance.ScaleFactor = Util.DistanceToReal(.TextBox4)
    End With
End If
End If
End Sub

```

Initialisierungsprozedur

In der Initialisierungsprozedur, die durch die Startprozedur aufgerufen wurde, bevor das Formular gestartet wird, werden die Listenfelder mit den entsprechenden Auswahlwerten gefüllt und einige Voreinstellungen des Systems an die Steuerelemente des Formulars übergeben.

```
Public Sub initToleranzFrm()
On Error Resume Next
```

Zuerst wird die Combobox mit den vorhandenen Textstilen der aktuellen Zeichnung gefüllt. Aus dieser kann der Benutzer anschließend einen Textstil auswählen, der dem neuen Toleranzobjekt zugeordnet werden soll. Der aktuelle Textstil wird danach bereits als aktuelle Auswahl gesetzt.

```
ComboBox1.Clear
For Each TStyle In ThisDrawing.TextStyles
    If InStr(1, TStyle.Name, "|") = 0 And TStyle.Name <> "" Then
        ComboBox1.AddItem TStyle.Name
    End If
Next
ComboBox1.Text = ThisDrawing.GetVariable("DIMTXSTY")
```

Übergabe der als Zeichenfolge formatierten aktuellen Texthöhe und des Breitenfaktors:

```
TextBox1 = Util.RealToString(ThisDrawing.GetVariable("DIMTXT"))
TextBox4 = Util.RealToString(1)
```

Zum Schluss werden noch die beiden *ImageCombo*-Steuerelemente, die der Farbauswahl für die Texte und die Umrahmungslinien dienen, mit den Grundfarben und den Einträgen „von-Layer“, „vonBlock und „andere Farbe“ gefüllt. Dabei können wir die Funktion *GetColorText* der Klasse *MyUtility* verwenden, um die Namen der Grundfarben in einer *For...Next*-Schleife zu übergeben. Als Vorgabe werden dann die Farben der Systemvariablen *DIMCLRT* und *DIMCLRD* gesetzt, die die Farbe des Bemaßungstextes und der Bemaßungslinien steuern.

Auswahlliste der Textfarbe:

```
With ImageCombo1
    Set .ImageList = ImageList2
    .ComboItems.Clear
    .ComboItems.Add , , "vonLayer", 7
    .ComboItems.Add , , "vonBlock", 7
    For i = 1 To 7
        .ComboItems.Add , , Util.GetColorName(i), i
    Next
    .ComboItems.Add , , "andere Farbe", 8
    Set .SelectedItem = .ComboItems(GetColorIndex( _
        ThisDrawing.GetVariable("DIMCLRT")))
    ActiveColor = getColor(.SelectedItem.Index)
End With
```

Auswahlliste der Rahmenfarbe:

```
With ImageCombo2
    Set .ImageList = ImageList2
    .ComboItems.Clear
    .ComboItems.Add , , "vonLayer", 7
```

```

.ComboItems.Add , , "vonBlock", 7
For i = 1 To 7
    .ComboItems.Add , , Util.GetColorName(i), i
Next
.ComboItems.Add , , "andere Farbe", 8
Set .SelectedItem = .ComboItems(GetColorIndex( _
    ThisDrawing.GetVariable("DIMCLRD")))
ActiveLineColor = getColor(.SelectedItem.Index)
End With
End Sub

```

Farbauswahl

Die Farbauswahl des jeweiligen *ImageCombo*-Steuerelements erfolgt in zwei Teilen. Entweder der Benutzer hat eine der Grundfarben bzw. die Farben „vonLayer“ oder „vonBlock“ gewählt oder er möchte über die Auswahl des Punkts „andere Farbe“ ein Formular aufrufen, in dem er dann weitere Farben vorfindet. Der erste Fall ist einfach zu lösen. Hier übergeben wir einfach die gewählte Farbe (ermittelt die Funktion *GetColor* aus dem Index des gewählten Listeneintrags) an die Formularvariablen *ActiveColor* (für Textfarbe) oder *ActiveLineColor* (für Rahmenfarbe). Im zweiten Fall kommen die Funktionen der Klasse *AcadColorDialog* (siehe *Kapitel A.4*) zum Einsatz. Dieser werden die aktuellen Einstellungen übergeben, um anschließend einen Farbauswahldialog mit der Anweisung *ShowColorDialog* zu starten. Auch hier wird nach Beendigung des Dialogs die gewählte Farbe übergeben.

Textfarbe:

```

Private Sub ImageCombo1_Click()
Dim LastColor As Long
On Error Resume Next
On Error Resume Next
If ImageCombo1.SelectedItem.Index = 10 Then
    Set ClrDialog = New AcadDialogs.AcadColorDialog
    ClrDialog.ShowByBlock = True
    ClrDialog.ShowByLayer = True
    ClrDialog.SelectedColor = ActiveColor
    ClrDialog.ShowColorDialog ThisDrawing
    ActiveColor = ClrDialog.SelectedColor
    ImageCombo1.SelectedItem = ImageCombo1.ComboItems( _
        GetColorIndex(ActiveColor))
Else
    ActiveColor = getColor(ImageCombo1.SelectedItem)
End If
End Sub

```

Rahmenfarbe:

```

Private Sub ImageCombo2_Click()
Dim LastColor As Long
On Error Resume Next
On Error Resume Next
If ImageCombo2.SelectedItem.Index = 10 Then
    Set ClrDialog = New AcadDialogs.AcadColorDialog
    ClrDialog.ShowByBlock = True
    ClrDialog.ShowByLayer = True

```



```

ClrDialog.SelectedColor = ActiveLineColor
ClrDialog.ShowColorDialog ThisDrawing
ActiveLineColor = ClrDialog.SelectedColor
ImageCombo2.SelectedItem = ImageCombo2.Items( _
    GetColorIndex(ActiveLineColor))
Else
    ActiveLineColor = getColor(ImageCombo2.SelectedItem)
End If
End Sub

```

Ermittlung des ComboItems-Index aus dem Attribut-Farbwert

Die Ermittlung des korrekten Index ist allerdings nicht allzu schwierig. Mittels einer *Select...Case*-Anweisung wird der übergebene Farbwert überprüft. Wurde die Farbe „VonBlock“ übergeben oder eine der Grundfarben, braucht der Farbe nur der Wert 2 hinzuaddiert werden, um die korrekte Position in der Farbauswahlliste zu bestimmen, da diese Farben in derselben Reihenfolge in der Liste geführt werden wie ihre zugehörigen Farbwerte. Anders ist dies bei der Farbe „VonLayer“. Diese Farbe hat den Wert 256, steht aber an der ersten Stelle in der Auswahlliste. Deshalb gibt die Funktion auch den Wert 1 zurück, wenn die Farbe 256 übergeben wurde. Bei allen anderen Farbwerten (8–255) wird der Index 10 zurückgegeben, der ja den Eintrag „andere Farbe“ widerspiegelt.

```

Public Function GetColorIndex(Color) As Integer
On Error Resume Next
Select Case Color
    Case 0 To 7 'Farbe VonBlock und Grundfarben
        GetColorIndex = Color + 2
    Case 256 'Farbe VonLayer
        GetColorIndex = 1
    Case Else 'alle anderen Farben
        GetColorIndex = 10
End Select
End Function

```

Ermittlung des Farbwerts aus dem ComboItems-Index:

```

Public Function getColor(ColorItem) As Long
On Error Resume Next
Select Case ColorItem.Index
    Case 1
        getColor = 256
    Case 2 To 9
        getColor = ColorItem.Index - 2
End Select
End Function

```

Symbolauswahl

Die Auswahl bzw. Zuordnung eines Symbols zu einem der *Image*-Steuerelemente wurde in einer Funktion zusammengefasst, da der Ablauf immer derselbe ist und nur das Steuerelement jedes Mal ein anderes sein kann. Der Verweis auf das Steuerelement, das der Benutzer angeklickt hat, wird im Argument der Funktion mitgeliefert. Mit diesem Verweis werden wir dann innerhalb der Funktion arbeiten, was das Ganze recht flexibel gestaltet.

Das Symbolformular selbst enthält nur ein *Toolbar*-Steuerelement (siehe *Kapitel 3.3.7*), dem 27 Schaltflächen zugeordnet wurden, wobei jede Schaltfläche ein Symbol darstellt. Da alle Schaltflächen vom Typ *tbrButtonGroup* sind, kann jeweils nur eine Schaltfläche heruntergedrückt sein. Diese heruntergedrückte Schaltfläche zeigt dann auch das aktuell gewählte Symbol an.

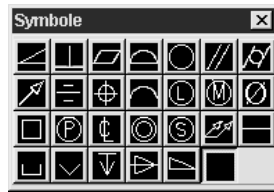


Abbildung 8.82: Auswahlformular der Toleranzsymbole

```
Public Sub SelectSymbol(ToleranceSymbol As MSForms.Image)
On Error Resume Next
```

Im ersten Schritt müssen wir im Symbolformular zunächst die Schaltfläche aktivieren, die dem Symbol des *Image*-Steuerelements entspricht, das der Benutzer angeklickt hat. Die Nummer dieses Symbols speichern wir in der *Tag*-Eigenschaft des Steuerelements. Obwohl bereits zur Entwurfszeit der Wert 21 (letztes Symbol der Symbolliste) zugeordnet wurde, ist es sicherer, die Zuordnung mit einer *IIF()*-Funktion vorzunehmen. Mit dieser Funktion werden wir den Index der Schaltfläche ermitteln, die heruntergedrückt werden muss. Anschließend rufen wir die Schaltfläche aus der *Buttons*-Auflistung des *Toolbar*-Steuerelements ab und weisen deren *Value*-Eigenschaft den Wert *tbrPressed* zu. Danach brauchen wir dann nur noch das Formular mit der *Show*-Anweisung zu starten. Wichtig hierbei ist, dass das Formular gebunden angezeigt wird, damit der Code erst fortgesetzt wird, nachdem das Formular beendet wurde.

```
Idx = IIf(ToleranceSymbol.Tag = "", 21, CLng(ToleranceSymbol.Tag))
With SymbolFrm
    .Toolbar1.Buttons(Idx).Value = tbrPressed
    .Show
```

Hat der Benutzer eine Auswahl getroffen, können wir den Wert der Variablen *SelectedImage* des Symbolformulars an die *Tag*-Eigenschaft des *Image*-Steuerelements weitergeben. Außerdem weisen wir dem Steuerelement noch das entsprechende Symbol zu, damit das ausgewählte Symbol sichtbar ist. Da alle Symbole in der *ListImages*-Auflistung des Steuerelements *ImageList1* des Symbolformulars in der gleichen Reihenfolge verwaltet werden, wie sie in den Schaltflächen dargestellt werden, muss als Symbolindex nur die Nummer des gewählten Symbols angegeben werden. Das Bild selbst ist dann in der *Picture*-Eigenschaft des jeweiligen Symbols zu finden.

```
If .OK = True Then
    ToleranceSymbol.Tag = .SelectedImage
    Set ToleranceSymbol.Picture = .ImageList1.ListImages( _
        .SelectedImage).Picture
End If
End Sub
```

Der Aufruf dieser Funktion erfolgt immer über das *Click*-Ereignis des jeweiligen *Image*-Steuerelements. Als Parameter wird der Funktion immer der Verweis auf das Steuerelement mitgegeben, womit diese dann den Bezug zum gewählten Steuerelement herstellen kann.

```
Private Sub Image10_Click()
SelectSymbol Image10
End Sub
Private Sub Image11_Click()
SelectSymbol Image11
End Sub
```

Textumwandlung

Die Umwandlung der Benutzereinstellungen in eine Zeichenfolge ist nicht so kompliziert, wie es im ersten Moment erscheint. Sie müssen sich die einzelnen Formulareinträge nur als einzelne Felder vorstellen, die zusammengefügt eine Zeichenfolge ergeben. Genauso ist auch die Prozedur aufgebaut. Hier werden nacheinander die Werte der Steuerelemente abgefragt und dem Funktionsnamen *GetText* angefügt.

```
Public Function GetText() As String
On Error Resume Next
```

Die erste Zeile des Toleranzobjekts setzt sich aus den Einstellungen der ersten Rahmenreihe des Formulars zusammen. Wir gehen also die Steuerelemente dieser Reihe einzeln von links nach rechts durch und werten den Inhalt dieser Steuerelemente aus. Handelt es sich um ein Symbol, wird die Symbolnummer an die Funktion *GetSymbolText* weitergegeben. Diese wandelt dann sozusagen diese Nummer in eine Zeichenfolge um. Den Inhalt der Textfelder können wir direkt anfügen. Die Zeichenfolge „%%v“ zieht dann zwischen den einzelnen Feldern eine vertikale Trennlinie ein.

```
GetText = GetSymbolText(CLng(Sym1.Tag))
GetText = GetText & "%%v" & GetSymbolText(CLng(TolSym1_1.Tag))
GetText = GetText & Toleranz1_1
GetText = GetText & GetSymbolText(CLng(TolSym1_2.Tag))
GetText = GetText & "%%v"
GetText = GetText & GetSymbolText(CLng(TolSym1_3.Tag))
GetText = GetText & Toleranz1_2
GetText = GetText & GetSymbolText(CLng(TolSym1_4.Tag))
GetText = GetText & "%%v"
GetText = GetText & Daten1_1
GetText = GetText & GetSymbolText(CLng(DatenSym1_1.Tag))
GetText = GetText & "%%v"
GetText = GetText & Daten1_2
GetText = GetText & GetSymbolText(CLng(DatenSym1_2.Tag))
GetText = GetText & "%%v"
GetText = GetText & Daten1_3
GetText = GetText & GetSymbolText(CLng(DatenSym1_3.Tag))
```

Nachdem die einzelnen Felder ausgewertet wurden, muss gegebenenfalls ein Zeilenumbruch (*vbLf*) erfolgen. Hat der Benutzer in der ersten Datenzeile keine Werte und Symbole festgelegt, kann dieser Zeilenumbruch wegfallen, da in der ersten Zeile ja keine Daten angezeigt werden müssen.

```

If GetText <> "%%v%%v%%v%%v%%v" Then
    GetText = GetText & vbCrLf
Else
    GetText = ""
End If

```

Die zweite Zeile des Toleranzobjekts setzt sich aus den Einstellungen der zweiten Rahmenreihe des Formulars zusammen. Die Auswertung der einzelnen Felder funktioniert genauso wie bei den ersten Reihen. Die Zeichenfolge für die Symboldarstellung wird über die Funktion *GetSymbolText* ermittelt, der Inhalt der Textboxen kann direkt übergeben werden und die Zeichenfolge „%%v“ stellt wieder eine vertikale Trennungslinie dar.

```

GetText = GetText & GetSymbolText(CLng(Sym2.Tag))
GetText = GetText & "%%v" & GetSymbolText(CLng(TolSym2_1.Tag))
GetText = GetText & Toleranz2_1
GetText = GetText & GetSymbolText(CLng(TolSym2_2.Tag))
GetText = GetText & "%%v"
GetText = GetText & GetSymbolText(CLng(TolSym2_3.Tag))
GetText = GetText & Toleranz2_2
GetText = GetText & GetSymbolText(CLng(TolSym2_4.Tag))
GetText = GetText & "%%v"
GetText = GetText & Toleranz2_3
GetText = GetText & GetSymbolText(CLng(DatenSym2_1.Tag))
GetText = GetText & "%%v"
GetText = GetText & Daten2_2
GetText = GetText & GetSymbolText(CLng(DatenSym2_2.Tag))
GetText = GetText & "%%v"
GetText = GetText & Daten2_3
GetText = GetText & GetSymbolText(CLng(DatenSym2_3.Tag))

```

Damit die Toleranz in der nächsten Zeile fortgesetzt wird, dürfen Sie am Ende des zweiten Abschnitts natürlich den Zeilenumbruch *vbLf* nicht vergessen.

```

If Right(GetText, 15) = "%%v%%v%%v%%v%%v" Then
    GetText = Left(GetText, Len(GetText) - 16)
End If

```

Die dritte Datenzeile besteht nun nur noch aus einem Text und einem Symbol und die vierte dann nur noch aus einem Text.

```

GetText = GetText & Toleranz3_1
GetText = GetText & GetSymbolText(CLng(TolSym3_1.Tag))
If Right(GetText, 1) = vbCrLf Then
    GetText = Left(GetText, Len(GetText) - 1)
End If
GetText = GetText & vbCrLf
GetText = GetText & Toleranz4_1
If Right(GetText, 1) = vbCrLf Then
    GetText = Left(GetText, Len(GetText) - 1)
End If
End Function

```

Bleibt noch die Umwandlung der Symbolnummer in eine dem Toleranzobjekt verständliche Zeichenfolge. Auch dieser Vorgang ist recht einfach. Bis auf den Buchstaben, der das Symbol bestimmt, das angezeigt werden soll, ist der Rest der Symboldefinition "{\Fgdt;}" fest vorge-

geben. Jetzt brauchen wir also nur noch den Buchstaben zu ermitteln (a–z), der zwischen Semikolon und geschweifter Klammer platziert werden soll. Da der ASCII-Code des Buchstaben a 97 beträgt, das Symbol, mit dem das zugehörige Zeichen im Formular angezeigt wird, mit dem Index abgerufen wurde, müssen wir diesem Index 96 hinzuaddieren, damit wir den richtigen ASCII-Code erhalten. Das funktioniert mit allen anderen Symbolen genauso:

Symbol 2 + 96 → ASCII-Code 98 entspricht dem Buchstaben b

Ausgenommen von dieser Umwandlung wird nur das letzte Symbol (*SNumber* = 27), das ja nur den Umstand keines gewählten Symbols darstellt:

```
Public Function GetSymbolText(SNumber As Long) As String
On Error Resume Next
If SNumber = 27 Then
    GetSymbolText = ""
Else
    GetSymbolText = "{\Fgdt;" & Chr(CLng(SNumber) + 96) & "}"
End If
End Function
```

8.3.27 Attributdefinitionen (AcadAttribute)

Attributdefinitionen, die später von Blöcken als Attributreferenzen verwendet werden, werden mit der Klasse *AcadAttribute* abgebildet. Dieses Objekt stellt also die Definition einer Attributreferenz dar. In AutoCAD kann der Benutzer dieses Objekt mit den Befehlen *attdef* oder *_attdef* in der Zeichnung erstellen.

Sie können einen Verweis auf ein Attribut erstellen, indem Sie die Funktion *AddAttribute* im Modellbereich, Papierbereich oder auf einen Block anwenden. Sie können auch ein vorhandenes Attribut über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften der Attributdefinition

Die Eigenschaften entsprechen im Großen und Ganzen denen des Textobjekts (*AcadText* Kapitel 8.3.16). Zusätzlich zu diesen Standardeigenschaften besitzt das Objekt aber noch ganz spezielle Eigenschaften, die als Vorlage für die Attributreferenzen definiert werden können (diese entstehen, wenn eine Blockreferenz eingefügt wird, deren Blockvorlage Attributdefinitionen enthält).

Verify legt den Prüfmodus des Attributs fest. Wenn Sie den Prüfmodus aktivieren (*Verify* = *TRUE*), dann fordert AutoCAD den Benutzer beim Einfügen des Blocks auf, die Richtigkeit des Attributwerts zu prüfen. Die *Constant*-Eigenschaft vergibt an Attribute einen festen Wert für das Einfügen von Blöcken. Ist diese Eigenschaft auf *TRUE* gesetzt, wird der Vorgabewert beim Einfügen des Blocks verwendet. Ob der Attributtext beim Einfügen des Blocks unsichtbar ist (*TRUE*) oder nicht (*FALSE*), bestimmt die Eigenschaft *Invisible*. Ist die *Preset*-Eigenschaft auf den Wert *TRUE* eingestellt, wird das Attribut auf den Vorgabewert gesetzt, wenn der Benutzer einen Block einfügt, der ein voreingestelltes Attribut enthält.

Alle vier Eigenschaften können auch mit *Mode* zusammengefasst werden, indem Sie folgende Konstanten einzeln zuweisen oder sie kombinieren, indem Sie sie addieren. Durch die Kombination der Konstanten werden die entsprechenden Eigenschaften aktiviert. Mit der Codezeile

```
ActAttrib.Mode = acAttributeModeInvisible + acAttributeModeVerify
```

werden die Eigenschaften *Invisible* und *Verify* jeweils auf den Wert *TRUE* gesetzt. Die beiden anderen Eigenschaften (*Preset*, *Constant*) werden automatisch auf *FALSE* eingestellt. Wenn Sie der *Mode*-Eigenschaft die Konstante *acAttributeModeNormal* zuweisen, werden alle vier Eigenschaften deaktiviert.

Konstante	Beschreibung	Wert
<i>acAttributeModeNormal</i>	Das Attribut wird normal dargestellt.	0
<i>acAttributeModeInvisible</i>	Der Attributtext ist unsichtbar.	1
<i>acAttributeModeConstant</i>	Die Eigenschaft <i>Constant</i> wird aktiviert.	2
<i>acAttributeModeVerify</i>	Die Eigenschaft <i>Verify</i> wird gesetzt.	4
<i>acAttributeModePreset</i>	Die Eigenschaft <i>Preset</i> wird gesetzt.	8

Tabelle 8.90: Konstanten der Mode-Eigenschaft

Mit der *TextString*-Eigenschaft können Sie die Zeichenfolge festlegen, die als Vorgabetext für das Attribut verwendet werden soll.

Den Bezeichner des Attributs bestimmt die *TagString*-Eigenschaft. Der *TagString* kann alle Zeichen außer Leerzeichen oder Ausrufezeichen enthalten. Kleinbuchstaben werden automatisch von AutoCAD in Großbuchstaben geändert. *PromptString* legt die Attributanfrage fest, die beim Einfügen eines Blocks mit dieser Attributdefinition angezeigt wird. Enthält diese Eigenschaft eine leere Zeichenfolge, wird der Wert der *TagString*-Eigenschaft als Eingabeaufforderung angezeigt. Ist die *Constant*-Eigenschaft aktiviert, wird diese Eigenschaft beim Einfügen des Blocks nicht berücksichtigt.

Eigenschaft	Beschreibung
<i>Alignment</i>	Bestimmt die Ausrichtung des Textes.
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Backward</i>	Bestimmt, ob der Attributtext rückwärts oder vorwärts geschrieben wird oder normal.
<i>Color</i>	Legt die Farbe des Attributtextes fest.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>FieldLength</i>	Bestimmt die Feldlänge des Attributtextes.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Attributtextes zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Attributtext ein Erweiterungswörterbuch zugeordnet wurde.
<i>Height</i>	Bestimmt die Höhe des Attributtextes.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Attributtextes zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).

Tabelle 8.91: Weitere Eigenschaften der Attributdefinition

Eigenschaft	Beschreibung
<i>InsertionPoint</i>	Legt den Einfügepunkt des Attributtextes fest.
<i>Layer</i>	Zeigt den Layer des Attributtextes an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp des Attributtextes an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor des Attributtextes an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für den Attributtext an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Attributs zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadAttribute</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>ObliqueAngle</i>	Bestimmt den Neigungswinkel des Attributtextes.
<i>OwnerId</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für den Attributtext zurück oder legt diesen fest.
<i>Rotation</i>	Legt den aktuellen Drehwinkel des Attributtextes fest oder gibt diesen zurück.
<i>ScaleFactor</i>	Bestimmt den Breitenfaktor des Attributtextes.
<i>StyleName</i>	Bestimmt den Textstil des Attributtextes.
<i>TextAlignmentPoint</i>	Diese Eigenschaft legt den Ausrichtungspunkt des Attributtextes fest oder ruft diesen ab.
<i>Thickness</i>	Diese Eigenschaft speichert die Objekthöhe des Attributtextes.
<i>UpsideDown</i>	Bestimmt, ob der Attributtext auf dem Kopf oder normal dargestellt wird.
<i>Visible</i>	Steuert die Sichtbarkeit des Attributtextes.

Tabelle 8.91: Weitere Eigenschaften der Attributdefinition (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.

Tabelle 8.92: Methoden der Attributdefinition

Prozedur	Beschreibung
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Attributs ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Attributs ab.
<i>Highlight</i>	Hebt das Attribut hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Attributs mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Kopie des Attributs als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt ein neues Attribut als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt das Attribut von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht das Attribut um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Attribut um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert das Attribut um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Attributs.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf das Attribut an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Attributs im AutoCAD-Zeichnungsfenster.

Tabelle 8.92: Methoden der Attributdefinition (Forts.)

Anwendungsbeispiel

(Beispielprojekt: 08\AcadAttribute.dvb)

In diesem Anwendungsbeispiel werden wir auf die Attributdefinitionen der Blockdefinitionen der aktuellen Zeichnung zugreifen. Dabei sollen diese Attributdefinitionen in einem Formular angezeigt und dessen Eigenschaften zur Bearbeitung angeboten werden.

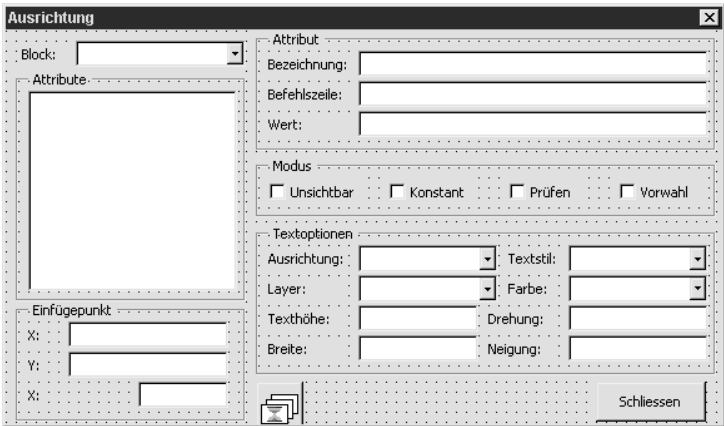


Abbildung 8.83: Formular des Beispielprojekts

Die Arbeitsweise durch den Benutzer soll dabei wie folgt ablaufen. Zunächst wählt er eine Blockdefinition aus der Combobox aus, die mit „Block“ beschriftet ist. Daraufhin werden alle Attributdefinitionen dieses Blocks im Listenfeld „Attribute“ zusammengestellt. Wenn der Benutzer nun ein bestimmtes Attribut aus dieser Liste wählt, werden dessen aktuelle Einstellungen in den entsprechenden Steuerelementen eingetragen und können dann durch den Benutzer angepasst werden.

Startprozedur

Die Startprozedur ist in diesem Projekt recht simpel aufgebaut. Nachdem der Benutzer das Programm gestartet hat, wird zunächst wieder eine neue Instanz der Klasse *MyUtility* erzeugt, danach die Initialisierungsprozedur des Formulars gestartet, die das Formular mit den entsprechenden Einträgen füllen wird. Abschließend braucht nun nur noch das Formular gestartet zu werden.

```
Public Sub editAttribut()
On Error Resume Next
Set Util = New MyUtility
AttributFrm.InitAttributFrm
AttributFrm.Show
End Sub
```

Initialisierungsprozedur

In dieser Prozedur, die ja durch die Startprozedur ausgeführt wurde, werden die Listen mit Werten gefüllt, deren Inhalt sich im Nachhinein nicht mehr ändert. Dazu gehört zunächst einmal das Füllen der *ComboBox*, in der der Benutzer später den entsprechenden Blocknamen wählen kann.

```
Public Sub InitAttributFrm()
On Error Resume Next
ComboBox1.Clear
For Each Blk In ThisDrawing.Blocks
    If InStr(1, Blk.Name, "|") = 0 And Left(Blk.Name, 1) <> "*" _
        Then
        ComboBox1.AddItem Blk.Name
    End If
Next
```

Anschließend folgt das Füllen der *ComboBox*, in der der Benutzer die Ausrichtung des Attributs festlegen kann. Sie müssen bei der Zusammenstellung dieser Liste lediglich darauf achten, dass die jeweiligen Werte in der Reihenfolge des dazugehörigen Konstantenwerts hinzugefügt werden müssen, damit später mit der *ListIndex*-Eigenschaft der Wert gesetzt bzw. dargestellt werden kann. Die Konstante *acAlignmentLeft* hat den Wert 0 und entspricht der Ausrichtung „Links“, also muss „Links“ an die Position gesetzt werden, die den *ListIndex* 0 wiedergibt oder *acAlignmentMiddleCenter*, diese Konstante hat den Wert 10 und entspricht dem Eintrag „Mitte zentriert“. Damit später die Zuweisung dieses Werts über die *ListIndex*-Eigenschaft erfolgen kann, muss der Eintrag an elfter Stelle also an *ListIndex*-Position 10 hinzugefügt werden.

```
ComboBox2.Clear
ComboBox2.AddItem "Links"
ComboBox2.AddItem "Zentriert"
```

```

ComboBox2.AddItem "Rechts"
ComboBox2.AddItem "Ausgerichtet"
ComboBox2.AddItem "Mitte"
ComboBox2.AddItem "Einpassen"
ComboBox2.AddItem "Oben links"
ComboBox2.AddItem "Oben zentriert"
ComboBox2.AddItem "Oben rechts"
ComboBox2.AddItem "Mitte links"
ComboBox2.AddItem "Mitte zentriert"
ComboBox2.AddItem "Mitte rechts"
ComboBox2.AddItem "Unten links"
ComboBox2.AddItem "Unten zentriert"
ComboBox2.AddItem "Unten rechts"

```

Als Nächstes folgen nun die Comboboxen, die die vorhandenen Textstile

```

ComboBox3.Clear
For Each TStyle In ThisDrawing.TextStyles
    If InStr(1, TStyle.Name, "|") = 0 And TStyle.Name <> "" Then
        ComboBox3.AddItem TStyle.Name
    End If
Next

```

und die vorhandenen Layer der Zeichnung aufnehmen und somit zur Auswahl anbieten sollen:

```

ComboBox4.Clear
For Each Layer In ThisDrawing.Layers
    If InStr(1, Layer.Name, "|") = 0 Then
        ComboBox4.AddItem Layer.Name
    End If
Next

```

Bleibt zum Schluss noch das Füllen der Farbauswahlliste. Diese wird wieder im Steuerelement *ImageCombo* zusammengestellt und beinhaltet neben den Grundfarben auch die Einträge „vonLayer“, „vonBlock“ und „andere Farbe“. Die entsprechenden Symbole, die die Farbe optisch anzeigen, sind wieder im Symbolauflistungssteuerelement *ImageList* gespeichert, die der *ImageList*-Eigenschaft der *ImageCombo*-Liste zugewiesen wird.

```

Set ImageCombo1.ImageList = ImageList1
ImageCombo1.ComboItems.Clear
ImageCombo1.ComboItems.Add , , "vonLayer", 7
ImageCombo1.ComboItems.Add , , "vonBlock", 7
ImageCombo1.ComboItems.Add , , "Rot", 1
ImageCombo1.ComboItems.Add , , "Gelb", 2
ImageCombo1.ComboItems.Add , , "Grün", 3
ImageCombo1.ComboItems.Add , , "Cyan", 4
ImageCombo1.ComboItems.Add , , "Blau", 5
ImageCombo1.ComboItems.Add , , "Magenta", 6
ImageCombo1.ComboItems.Add , , "Schwarz/Weiss", 7
ImageCombo1.ComboItems.Add , , "andere Farbe", 8
End Sub

```

Auswahl einer Blockdefinition

Nachdem die einzelnen Comboboxen mit den entsprechenden Einträgen gefüllt wurden, ist der Benutzer nun gefordert. Das Erste, was er nun machen wird, ist die Auswahl eines Blocknamens in der *Block-ComboBox*. Darauf werden wir nun reagieren und die zu dem gewählten Block gehörigen Attributdefinitionen ermitteln.

```
Private Sub ComboBox1_Click()  
On Error Resume Next
```

Dafür müssen wir nur den gewählten Blocknamen mit der *Item*-Funktion der *Blocks*-Auflistung ausführen. Als Ergebnis erhalten wir einen Verweis auf das Blockobjekt, den wir in der Objektvariablen *ActBlk* speichern werden.

```
Set ActBlk = ThisDrawing.Blocks(CStr(ComboBox1))
```

Die Verweise auf die Attribute dieses Blocks werden wir anschließend in der Auflistung *mCol* speichern, womit wir zunächst eine neue Instanz dieser Auflistung erstellen müssen:

```
Set mCol = New Collection
```

Im nächsten Schritt werden alle notwendigen Schritte unternommen, um den Ursprung des Formulars wieder herzustellen. Dazu gehört die Entfernung des Verweises aus einem vorherigen Durchlauf auf ein bestimmtes Attribut, die Leerung der Liste, die die vorhandenen Attribute aufnimmt, und das Setzen der *Enabled*-Eigenschaft der einzelnen Rahmen.

```
Set ActAttribut = Nothing  
ListBox1.Clear  
Frame1.Enabled = False  
Frame3.Enabled = False  
Frame4.Enabled = False  
Frame5.Enabled = False
```

Der letzte Schritt dieser Prozedur besteht darin, die Attributliste (*ListBox1*) mit den Attributnamen des ausgewählten Blocks zu füllen, damit der Benutzer dann ein bestimmtes Attribut in der Liste zur Bearbeitung auswählen kann. Hierfür gehen wir den Block in einer *For...Each*-Schleife Element für Element durch. Wird ein Objekt gefunden, das der Klasse *IAcadAttribute* entspricht, wird der Attributbezeichner (*TagString*-Eigenschaft) der Liste hinzugefügt und der Verweis auf das Attribut in der Auflistung *mCol* gespeichert.

```
For Each BlkItem In ActBlk  
    If TypeName(BlkItem) = "IAcadAttribute" Then  
        ListBox1.AddItem BlkItem.TagString  
        mCol.Add BlkItem  
    End If  
Next  
End Sub
```

Nachdem diese Prozedur erfolgreich ausgeführt wurde, könnte sich dem Benutzer folgendes Bild im Formular bieten:

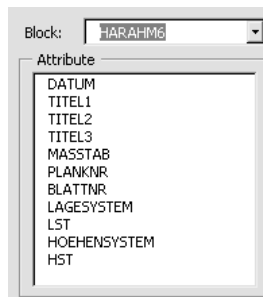


Abbildung 8.84: Ausgewählter Block mit den darin enthaltenen Attributen

Auswahl einer Attributbezeichnung:

Als Nächstes könnte der Benutzer eine Attributbezeichnung in der entstandenen Auflistung auswählen, um die Eigenschaften dieses Attributs anpassen zu können. Aus diesem Grund müssen wir auf das *Click*-Ereignis des Listenfelds reagieren und dort den entsprechenden Code platzieren, der die einzelnen Werte der Eigenschaften des Attributs in die dazugehörigen Steuerelemente einträgt.

```
Private Sub ListBox1_Click()  
On Error Resume Next
```

Als Erstes erfolgt die Zuordnung des Verweises des gewählten Attributs an die Objektvariable *ActAttribut*. Da die Reihenfolge der Bezeichner mit der Reihenfolge der Verweise in der *mCol*-Auflistung übereinstimmt, können wir die *ListIndex*-Eigenschaft des Listenfelds verwenden, um das richtige Attribut anzusprechen. Zu beachten gilt hier nur, dass die Auflistung mit dem Index 1 beginnt, die Liste aber mit dem Wert 0, weshalb das zum Listeneintrag gehörige Attribut in der Auflistung immer an der Position *ListIndex + 1* zu finden sein wird.

```
Set ActAttribut = mCol.Item(ListBox1.ListIndex + 1)
```

Nachdem das Attribut gefunden wurde, erfolgt nun nur noch die Übergabe der Werte der einzelnen Eigenschaften an die entsprechenden Steuerelemente:

```
TextBox1 = ActAttribut.TagString 'Attributbezeichner  
TextBox2 = ActAttribut.PromptString 'Eingabeaufforderung  
TextBox3 = ActAttribut.TextString 'Attributwert  
CheckBox1 = ActAttribut.Invisible 'Attributsichtbarkeit  
CheckBox2 = ActAttribut.Constant 'Konstant-Eigenschaft  
CheckBox3 = ActAttribut.Verify 'Prüfen-Eigenschaft  
CheckBox4 = ActAttribut.Preset 'Vorgabe-Eigenschaft  
ComboBox2.ListIndex = ActAttribut.Alignment 'Textausrichtung  
ComboBox3 = ActAttribut.StyleName 'Textstilname  
ComboBox4 = ActAttribut.Layer 'Layer  
TextBox7 = Util.RealToString(ActAttribut.Height) 'Texthöhe  
TextBox8 = Util.RealToString(ActAttribut.ScaleFactor) 'Textbreite  
TextBox9 = Util.AngleToString(ActAttribut.Rotation) 'Drehwinkel  
TextBox10 = Util.AngleToString(ActAttribut.ObliqueAngle) 'Neigung
```

Bei der Ermittlung des Einfügepunkts müssen Sie berücksichtigen, dass es zwei Eigenschaften gibt, die diesen enthalten können. Das hängt ganz von den Einstellungen der *Alignment*-Eigenschaft ab (siehe auch *Kapitel 8.3.16*). Ist diese auf den Wert *acAlignmentLeft* eingestellt,

wird der Einfügapunkt in der Eigenschaft *InsertionPoint* verwaltet. Im anderen Fall finden wir die verwendeten Koordinaten in *TextAlignmentPoint*. In jedem Fall übergeben wir den Inhalt dieser Eigenschaft an die Variable *InsPoint*.

```
If ActAttribut.Alignment = acAlignmentLeft Then
    InsPoint = ActAttribut.InsertionPoint
Else
    InsPoint = ActAttribut.TextAlignmentPoint
End If
```

Die Werte der einzelnen Felder der so entstandenen 3D-Arrays können nun verwendet werden, um die entsprechenden Textfelder zu füllen.

```
TextBox4 = Util.RealToString(InsPoint(0))
TextBox5 = Util.RealToString(InsPoint(1))
TextBox6 = Util.RealToString(InsPoint(2))
```

Die Zuordnung des korrekten Eintrags der Farbauswahlliste zur aktuellen Farbe des Attributs ist schon etwas komplexer. Für diesen Fall benötigen wir eine Funktion, die den richtigen Index aus dem Farbwert ermittelt. Gehen Sie zunächst einmal davon aus, dass *GetColorIndex* diese Aufgabe übernimmt und den korrekten Eintrag in der Farbauswahlliste auswählt.

```
ImageCombo1.SelectedItem = ImageCombo1.Items(GetColorIndex( _
    ActAttribut.Color))
Frame1.Enabled = True
Frame3.Enabled = True
Frame4.Enabled = True
Frame5.Enabled = True
End Sub
```

Ermittlung des ComboItems-Index aus dem Attributfarbwert

Die Ermittlung des korrekten Index ist allerdings nicht allzu schwierig. Mittels einer *Select...Case*-Anweisung wird der übergebene Farbwert überprüft. Wurde die Farbe „VonBlock“ übergeben oder eine der Grundfarben braucht der Farbe nur der Wert 2 hinzuaddiert werden, um die korrekte Position in der Farbauswahlliste zu bestimmen, da diese Farben in derselben Reihenfolge in der Liste geführt werden wie ihre zugehörigen Farbwerte. Anders ist dies bei der Farbe „VonLayer“. Diese Farbe hat den Wert 256, steht aber an der ersten Stelle in der Auswahlliste. Deshalb gibt die Funktion auch den Wert 1 zurück, wenn die Farbe 256 übergeben wurde. Bei allen anderen Farbwerten (8–255) wird der Index 10 zurückgegeben, der ja den Eintrag „andere Farbe“ widerspiegelt.

```
Public Function GetColorIndex(Color) As Integer
On Error Resume Next
Select Case Color
    Case 0 To 7 'Farbe VonBlock und Grundfarben
        GetColorIndex = Color + 2
    Case 256 'Farbe VonLayer
        GetColorIndex = 1
    Case Else 'alle anderen Farben
        GetColorIndex = 10
End Select
End Function
```

Auswahl einer Farbe aus der ImageCombo-Liste

Wo wir schon bei der Farbauswahl sind, können wir gleich die Funktionen betrachten, die die Auswahl der Farbe steuern – zunächst die *Click*-Ereignisprozedur des *ImageCombo*-Steuerelements. Diese wird ausgelöst, wenn der Benutzer eine Farbe aus der Liste auswählt. Dabei müssen wir zwei Farbkategorien unterscheiden. Zum einen gibt es die direkten Farbeinträge („VonLayer“, „VonBlock“ und die Grundfarben), die mit der Funktion *getColor* unter Angabe des Index des ausgewählten Eintrags ermittelt werden können. Die zweite Kategorie, die gesondert behandelt werden muss, ist die Auswahl des Eintrags „andere Farbe“. Hat der Benutzer diesen Eintrag gewählt, kommt die Klasse *AcadColorDialog* (siehe Kapitel A.4) zum Einsatz.

```
Private Sub ImageCombo1_Click()
On Error Resume Next
If ImageCombo1.SelectedItem.Index = 10 Then
```

Zunächst wird dann eine neue Instanz dieser Klasse in der Objektvariablen *ClrDialog* erzeugt. Diese Variable besitzt nun die entsprechenden Funktionen, um einen Farbauswahldialog anzuzeigen, in dem der Benutzer dann die gewünschte Farbe auswählen kann. Nach der Einstellung gewisser Eigenschaften wird der Farbdialog mit der Anweisung *ShowColorDialog* unter Angabe des Zeichnungsobjekts gestartet. Hier übernimmt dann *AcadColorDialog* das Kommando, bis der Benutzer den Farbdialog beendet.

```
Set ClrDialog = New AcadDialogs.AcadColorDialog
ClrDialog.ShowByBlock = True
ClrDialog.ShowByLayer = True
ClrDialog.SelectedColor = ActAttribut.Color
ClrDialog.ShowColorDialog ThisDrawing
```

Die in diesem Dialog gewählte Farbe kann nun über die Eigenschaft *SelectedColor* abgerufen und dem ausgewählten Attribut zugewiesen werden. Anschließend muss diese Farbe wieder im *ImageCombo*-Steuerelement angezeigt werden. (Falls der Benutzer den Vorgang abgebrochen hat, enthält *SelectedColor* die Ursprungsfarbe des Attributs.)

```
ActAttribut.Color = ClrDialog.SelectedColor
ImageCombo1.SelectedItem = ImageCombo1.Items( _
    GetColorIndex(ActAttribut.Color))
Else
    ActAttribut.Color = getColor(ImageCombo1.SelectedItem)
End If
End Sub

Public Function getColor(ColorItem) As Long
On Error Resume Next
Select Case ColorItem.Index
    Case 1
        getColor = 256
    Case 2 To 9
        getColor = ColorItem.Index - 2
End Select
End Function
```

Die Zuweisung der restlichen Eigenschaften stellt uns vor keine großen Probleme. Sie müssen dann lediglich auf das entsprechende Ereignis des Steuerelements reagieren und den dort eingetragenen bzw. ausgewählten Wert an die jeweilige Eigenschaft des Attributs weitergeben.

Änderung der Attributbezeichnung (TagString)

```
Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Err.Clear
ActAttribut.TagString = TextBox1
If Err <> 0 Then
    MsgBox Err.Description
    Cancel = True
End If
End Sub
```

Änderung der Attributeingabeaufforderung (PromptString)

```
Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Err.Clear
ActAttribut.PromptString = TextBox2
If Err <> 0 Then
    MsgBox Err.Description
    Cancel = True
End If
End Sub
```

Änderung des Attributwerts (TextString)

```
Private Sub TextBox3_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Err.Clear
ActAttribut.TextString = TextBox3
If Err <> 0 Then
    MsgBox Err.Description
    Cancel = True
End If
End Sub
```

Änderung der Attributmodi Sichtbar, Konstant, Prüfen und Vorwahl

```
Private Sub CheckBox1_Click()
ActAttribut.Invisible = CheckBox1
End Sub

Private Sub CheckBox2_Click()
ActAttribut.Constant = CheckBox2
End Sub

Private Sub CheckBox3_Click()
ActAttribut.Verify = CheckBox3
End Sub

Private Sub CheckBox4_Click()
ActAttribut.Preset = CheckBox4
End Sub
```

Änderung der Attributeigenschaften Ausrichtung, Textstil und Layer

```

Private Sub ComboBox2_Click()
On Error Resume Next
ActAttribut.Alignment = ComboBox2.ListIndex
End Sub

Private Sub ComboBox3_Change()
On Error Resume Next
ActAttribut.StyleName = ComboBox3
End Sub

Private Sub ComboBox4_Click()
On Error Resume Next
ActAttribut.Layer = ComboBox4
End Sub

```

Änderung der Texthöhe des Attributs

Die Übergabe der Texthöhe erfordert noch einige zusätzliche Schritte. Zunächst wird erst einmal die eingegebene Höhe in eine reelle Zahl umgewandelt und der *Height*-Eigenschaft übergeben. Falls der Benutzer eine ungültige Höhe eingegeben hat (≤ 0), behält die *Height*-Eigenschaft den ursprünglichen Wert und AutoCAD löst einen Fehler aus. In jedem Fall wird der Wert der *Height*-Eigenschaft ins Textfeld formatiert zurückgeschrieben. Für den Fall eines Fehlers wird dessen Beschreibung in einem Meldungsfeld angezeigt.

```

Private Sub TextBox7_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Err.Clear
ActAttribut.Height = Util.DistanceToReal(TextBox7)
TextBox7 = Util.RealToString(ActAttribut.Height)
If Err <> 0 Then
    MsgBox Err.Clear
    Cancel = True
End If
End Sub

```

Ähnlich funktioniert dies auch bei den Eigenschaften für die Textbreite, den Drehwinkel und der Neigung des Attributs.

Änderung der Textbreite des Attributs

```

Private Sub TextBox8_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Err.Clear
ActAttribut.ScaleFactor = Util.DistanceToReal(TextBox8)
TextBox8 = Util.RealToString(ActAttribut.ScaleFactor)
If Err <> 0 Then
    MsgBox Err.Clear
    Cancel = True
End If
End Sub

```


Änderung des Drehwinkels des Attributs

```
Private Sub TextBox9_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Err.Clear
ActAttribut.Rotation = Util.AngleToReal(TextBox9)
TextBox9 = Util.AngleToString(ActAttribut.Rotation)
If Err <> 0 Then
    MsgBox Err.Clear
    Cancel = True
End If
End Sub
```

Änderung des Neigungswinkels des Attributs

```
Private Sub TextBox10_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
Err.Clear
ActAttribut.ObliqueAngle = Util.AngleToReal(TextBox10)
TextBox10 = Util.AngleToString(ActAttribut.ObliqueAngle)
If Err <> 0 Then
    MsgBox Err.Clear
    Cancel = True
End If
End Sub
```

Änderung des Einfügepunkts des Attributs

Da zwar die Koordinaten des Einfügepunkts auf drei verschiedene Steuerelemente verteilt sind, wir aber auf jede Änderung in den jeweiligen Textfeldern reagieren müssen, kann sozusagen die Anpassung des Einfügepunkts in einer Prozedur zusammengefasst werden. Die Ereignisprozeduren der anderen beiden Textfelder brauchen dann nur noch diese Prozedur aufzurufen.

```
Private Sub TextBox4_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
Dim InsPoint(2) As Double
On Error Resume Next
Err.Clear
```

Als Erstes werden die einzelnen Koordinaten aus den drei verschiedenen Textfeldern an die Felder des *InsPoint*-Arrays übergeben.

```
InsPoint(0) = Util.DistanceToReal(TextBox4)
InsPoint(1) = Util.DistanceToReal(TextBox5)
InsPoint(2) = Util.DistanceToReal(TextBox6)
```

Je nachdem, welchen Wert die *Alignment*-Eigenschaft des Attributs besitzt, wird dieses Array an die Eigenschaften *InsertionPoint* oder *TextAlignmentPoint* weitergegeben.

```
If ActAttribut.Alignment = acAlignmentLeft Then
    ActAttribut.InsertionPoint = InsPoint
Else
    ActAttribut.TextAlignmentPoint = InsPoint
End If
```

Danach werden dann nur noch die formatierten Werte des Einfügekpunkts in die einzelnen Textfelder zurückgeschrieben.

```
TextBox4 = Util.RealToString(InsPoint(0))
TextBox5 = Util.RealToString(InsPoint(1))
TextBox6 = Util.RealToString(InsPoint(2))
End Sub

Private Sub TextBox5_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
TextBox4_BeforeUpdate Cancel
End Sub

Private Sub TextBox6_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
TextBox4_BeforeUpdate Cancel
End Sub
```

8.3.28 Attributreferenzen (AcadAttributeReference)

Bei der Klasse *AcadAttributeReference* handelt es sich um eine besondere Objektklasse, da Sie dieses Objekt nicht direkt erstellen, sondern nur darauf zugreifen können. Attributreferenzen werden immer erzeugt, wenn in AutoCAD eine Blockreferenz eingefügt wird, deren Blockdefinition Attribute enthält.

Sie können die einzelnen Attributreferenzen mit der Methode *GetAttributes* der jeweiligen Blockreferenz abrufen (mehr dazu aber im nächsten Abschnitt). Da diese Klasse auf der Klasse *AcadAttribute* basiert, sind die Eigenschaften dieses Objekts dieselben wie die der Attributdefinitionen.

Eigenschaft	Beschreibung
<i>Alignment</i>	Bestimmt die Ausrichtung des Textes.
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Backward</i>	Bestimmt, ob der Attributtext rückwärts oder vorwärts geschrieben wird oder normal.
<i>Color</i>	Legt die Farbe des Attributtextes fest.
<i>Constant</i> (schreibgeschützt)	Diese Eigenschaft gibt den aktuellen Wert der <i>Constant</i> -Eigenschaft des Vorlageattributs zurück. Ist diese Eigenschaft auf <i>TRUE</i> gesetzt, wurde der Vorgabewert verwendet.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>FieldLength</i>	Bestimmt die Feldlänge des Attributtextes.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Attributtextes zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Attributtext ein Erweiterungswörterbuch zugeordnet wurde.

Tabelle 8.93: Eigenschaften der Klasse Attributreferenz

Eigenschaft	Beschreibung
<i>Height</i>	Bestimmt die Höhe des Textes der Attributreferenz.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Attributtextes zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>InsertionPoint</i>	Bestimmt den Einfügepunkt des Textes der Attributreferenz.
<i>Invisible</i>	Bestimmt, ob der Text der Attributreferenz unsichtbar ist oder nicht.
<i>Layer</i>	Zeigt den Layer der Attributreferenz an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Attributreferenz an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Attributreferenz an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Attributreferenz an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Attributreferenz zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadAttributeReference</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>ObliqueAngle</i>	Bestimmt den Neigungswinkel des Attributtextes.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnete Blockreferenz) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für der Attributreferenz zurück oder legt diesen fest.
<i>Rotation</i>	Legt den aktuellen Drehwinkel des Attributtextes fest oder gibt diesen zurück.
<i>ScaleFactor</i>	Bestimmt den Breitenfaktor des Attributtextes oder gibt den aktuellen Wert zurück.
<i>StyleName</i>	Diese Eigenschaft legt den Textstil des Textes der Attributreferenz fest oder ruft diesen ab.
<i>TagString</i>	Mit dieser Eigenschaft bestimmen Sie den Bezeichner der Attributreferenz oder Sie können ihn hiermit ermitteln.
<i>TextAlignmentPoint</i>	Diese Eigenschaft legt den Ausrichtungspunkt des Textes der Attributreferenz fest oder ruft diesen ab.
<i>TextString</i>	Bestimmt die Zeichenfolge des Vorgabetextes der Attributreferenz.
<i>Thickness</i>	Diese Eigenschaft speichert die Objekthöhe der Attributreferenz.
<i>UpsideDown</i>	Diese Eigenschaft gibt zurück, ob der Text der Attributreferenz auf dem Kopf oder normal dargestellt wird, oder legt dies fest.
<i>Visible</i>	Steuert die Sichtbarkeit der Attributreferenz.

Tabelle 8.93: Eigenschaften der Klasse *Attributreferenz* (Forts.)

Prozedur	Beschreibung
Delete	Löscht das Objekt aus der AutoCAD-Zeichnung.
GetBoundingBox	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
GetExtensionDictionary	Ruft das erweiterte Wörterbuch der Attributreferenz ab.
GetXData	Ruft die erweiterten Daten der Attributreferenz ab.
Highlight	Hebt den Text der Attributreferenz hervor.
IntersectWith	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Attributreferenz mit einem anderen AutoCAD-Objekt ergeben.
Move	Verschiebt den Text der Attributreferenz von Ursprungs- auf Zielkoordinaten.
Rotate	Dreht den Text der Attributreferenz um einen bestimmten Winkel.
Rotate3D	Dreht den Text der Attributreferenz um eine räumlich definierte Achse.
ScaleEntity	Skaliert den Text der Attributreferenz um einen bestimmten Faktor.
SetXData	Speichert die erweiterten Daten der Attributreferenz.
TransformBy	Wendet die angegebene Transformationsmatrix auf den Text der Attributreferenz an.
Update	Diese Funktion aktualisiert die Darstellung des Textes der Attributreferenz im AutoCAD-Zeichnungsfenster.

Tabelle 8.94: Verwendete Methoden der Klasse Attributreferenz

Da Attributreferenzen immer in Blockreferenzen vorliegen und niemals als alleinige Objekte in AutoCAD eingefügt werden, fassen wir das Anwendungsbeispiel zu diesem Thema mit dem nächsten Kapitel zusammen.

8.3.29 Blockreferenzen (AcadBlockReference)

Mit der Klasse *AcadBlockReference* werden in der Zeichnung eingefügte Blockdefinitionen (*AcadBlock*) abgebildet. Diese Klasse ist also abhängig von den Einträgen der Blockauflistung (*Kapitel 7.11*). Bevor Sie eine Blockreferenz in AutoCAD einfügen können, muss diese als Block definiert worden sein. Liegt die Definition des Blocks vor, können Sie eine Referenz des jeweiligen Blocks mit der Anweisung *InsertBlock* in AutoCAD erstellen. Diese Anweisung können Sie im Modellbereich, im Papierbereich sowie auf einen vorhandenen Block anwenden.

```
Dim CrBlock As AcadBlockReference
Dim InsPnt(2) As Double
Set CrBlock = ModelSpace.InsertBlock(InsPnt, "Block1", 1, 1, 1, 0)
```

Wenn Sie eine Blockreferenz aus einem Block erstellen möchten, der nicht in der Zeichnung, sondern als Datei (*dwg*) extern vorliegt, brauchen Sie statt des Namens nur den Dateinamen einschließlich Verzeichnis anzugeben.

```
Set CrBlock = ModelSpace.InsertBlock(InsPnt, _
    "C:\Block1.dwg", 1, 1, 1, 0)
```

Der Block wird automatisch geladen und eine Referenz wird erstellt. Diese Vorgehensweise gilt auch für Blöcke, die Sie durch externe Blockdefinitionen ersetzen möchten. Einfach Dateiname mit Verzeichnis angeben und der Block wird getauscht.

Bevor wir nun einige Beispiele zu diesem Thema durchgehen, folgen wie immer erst einmal die Eigenschaften und Methoden, die Sie mit einer Blockreferenz anwenden können.

Eigenschaften der Blockreferenz

Mit der *Name*-Eigenschaft stellen Sie eine Verbindung zu den Blockdefinitionen der Blockauflistung her. Die hier festgelegte Zeichenfolge muss dem Namen einer vorhandenen Blockdefinition entsprechen und definiert somit das Aussehen der Blockreferenz. Wenn Sie der Referenz einen anderen Namen zuweisen, ändert sich ebenfalls deren Darstellung. Den Einfügepunkt der Blockreferenz bestimmt die Eigenschaft *InsertionPoint*. Der Drehwinkel der Blockreferenz wird in der Eigenschaft *Rotation* gespeichert. Der Wert des Drehwinkels ist dabei von der Systemvariablen *ANGBASE* abhängig. Die Skalierung wird auf drei Eigenschaften verteilt: *XScaleFactor* speichert den Wert der X-Skalierung, *YScaleFactor* gilt der Y-Skalierung und *ZScaleFactor* schließlich der Z-Skalierung. Bei der Skalierung wird dabei nicht die Ausrichtung der Blockreferenz in der Zeichnung berücksichtigt, vielmehr bezieht sich diese auf das Koordinatensystem der Blockdefinition selbst.

Sollte die Blockreferenz Attributreferenzen enthalten, so können Sie dies mit der Eigenschaft *HasAttributes* kontrollieren. Gibt diese Eigenschaft den Wert *TRUE* zurück, enthält sie ein oder mehr Attribute, andernfalls gibt *HasAttributes* den Wert *FALSE* zurück.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der Blockreferenz fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Blockreferenz zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Blockreferenz ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Blockreferenz zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der Blockreferenz an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Blockreferenz an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Blockreferenz an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Blockreferenz an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.

Tabelle 8.95: Weitere Eigenschaften der Blockreferenz

Eigenschaft	Beschreibung
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Blockreferenz zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadBlockReference</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Blockreferenz zurück oder legt diesen fest.
<i>Visible</i>	Steuert die Sichtbarkeit der Blockreferenz.

Tabelle 8.95: Weitere Eigenschaften der Blockreferenz (Forts.)

Methoden der Blockreferenz

Sollte der Block Attribute enthalten (*HasAttributes*), können Sie ein Array mit den Verweisen auf die einzelnen Attribute mit der Funktion *GetAttributes* abrufen. Wie viele Attribute enthalten sind, kann anschließend mit der *UBound()*-Funktion geprüft werden.

```
Dim BlkAtts As Variant
Dim BlkAtt As AcadAttributReference
BlkAtts = ActBlkRef.GetAttributes
For i = 0 To UBound(BlkAtts)
    Set BlkAtt = BlkAtts(i)
    MsgBox BlkAtt.TagString
Next
```

Mit der *GetConstantAttributes*-Anweisung rufen Sie die konstanten Attribute der Blockreferenz ab, also alle Attribute, in denen die Eigenschaft *Constant* auf *TRUE* gesetzt ist. Sie erhalten als Ergebnis ein Array, das alle Attribute auflistet. Im Gegensatz zu *GetAttributes* werden mit dieser Funktion nicht die Attributreferenzen zurückgegeben. Bevor Sie diese Funktion verwenden, sollten Sie die Blockreferenz auf Attribute prüfen (*HasAttributes*).

```
Dim BlkAtts As Variant
Dim BlkAtt As AcadAttributReference
BlkAtts = ActBlkRef.GetAttributes
For i = 0 To UBound(BlkAtts)
    Set BlkAtt = BlkAtts(i)
    MsgBox BlkAtt.TagString
Next
```

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.

Tabelle 8.96: Weitere Methoden der Blockreferenz

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>Explode</i>	Diese Anweisung löst die Blockreferenz in ihre Bestandteile auf und gibt die daraus entstandenen Elemente als Array zurück.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Blockreferenz ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Blockreferenz ab.
<i>Highlight</i>	Hebt die Blockreferenz hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Blockreferenz mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Blockreferenz als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt eine neue Blockreferenz als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt die Blockreferenz von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die Blockreferenz um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Blockreferenz um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Blockreferenz um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Blockreferenz.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Blockreferenz an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Blockreferenz im AutoCAD-Zeichnungsfenster.

Tabelle 8.96: Weitere Methoden der Blockreferenz (Forts.)

Anwendungsbeispiele

(Beispielprojekt: 08\AcadBlockReference.dvb)

Diese Projektdatei enthält insgesamt drei Anwendungsbeispiele für Blockreferenzen.

Im ersten Beispiel lesen wir den Namen eines Blocks und die Koordinaten aus einer Datei ein, um mit diesen Informationen Blockreferenzen in einer Zeichnung einzufügen. Die Auswahl der Datei, die diese Daten enthält, soll der Benutzer in einem Formular vornehmen können. Dieses Formular wird in der Startprozedur aufgerufen und enthält selbst nicht allzu viele Codezeilen, da der Hauptteil des Programms in der Startprozedur abläuft.

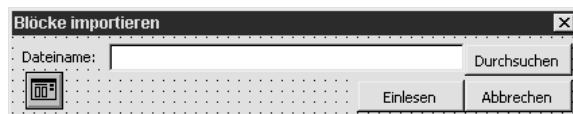


Abbildung 8.85: Formular zur Auswahl eines Dateinamens

Schaltfläche „Durchsuchen“

Über diese Schaltfläche kann der Benutzer einen Dateinamen in einem beliebigen Windows-Verzeichnis auswählen. Dazu nutzen wir wieder die Funktionalität des Windows-Standarddialogs *CommonDialog*. Als Filter wird hier, neben der Anzeige aller Dateien, das Format *asc* eingestellt. Bei diesem Format handelt es sich um eine Eigenproduktion, die weiter unten genauer erklärt wird. Als Vorgabedateinamen (*FileName*-Eigenschaft) können wir den Inhalt des Textfelds *TextBox1* übergeben, in das der ausgewählte Dateiname zurückgeschrieben wird, wenn die Auswahl erfolgreich war.

```
Private Sub CommandButton1_Click()
On Local Error Resume Next
With ComDialog
.CancelError = True
.FileName = TextBox1.Text
.DialogTitle = "Datei einlesen"
.Filter = "Asc-Dateien (*.asc)|*.asc|Alle Dateien (*.*)|*.*|"
.Err.Clear
ComDialog.ShowOpen
If Err = 0 Then
    TextBox1.Text = .FileName
End If
End With
End Sub
```

ModulCode

Der Rest dieses Programms läuft im Klassenmodul *ThisDrawing* ab. Dieser Rest besteht aus zwei Prozeduren: die Startprozedur und eine Funktion, die die eingelesenen Daten auswertet und in ein Array umwandelt, welches folgendes Format hat:

Daten(0) = BlockName, Daten(1) = X, Daten(2) = Y, Daten(3) = Z

Hier die Funktion:

```
Public Function KonvertData(ByVal Eingabe As String) As Variant
Dim OutField(3) As String
On Local Error Resume Next
Eingabe = Trim(Eingabe) & " "
x = 0
Do
    OutField(x) = Trim(Left(Eingabe, InStr(1, Eingabe, " ")))
    Eingabe = LTrim(Mid(Eingabe, InStr(1, Eingabe, " ")))
    x = x + 1
    If x = 4 Then Exit Do
Loop
KonvertData = OutField
End Function
```

Ich fasse kurz zusammen: Zuerst wird ein Array deklariert, das die einzelnen Daten beinhalten soll. Die Anweisung *Trim(Eingabe) & " "* kürzt die Leerzeichen der Eingabezeichenfolge und hängt ein Leerzeichen hinten dran. Dies ist notwendig, damit die Schleife richtig abgeschlossen werden kann. Die Variable *X* wird zum Zähler für die einzelnen Datenfelder *OutField(0)....OutField(3)*. An das jeweilige Datenfeld *OutField(x)* wird immer der erste Teil der Eingabezeichenfolge bis zum ersten Vorkommen eines Leerzeichens übergeben.

Kürze alle Leerzeichen auf der linken und rechten Seite weg.	Gib den linken Teil der Zeichenkette zurück, ...	... bis zu der Stelle, an der ein Leerzeichen gefunden wird.
<i>OutField(x) = Trim(</i>	<i>Left(Eingabe,</i>	<i>Instr(1, Eingabe, " "))</i>)
<i>OutField(x) = „Schacht 4475388,8869 5997768,3955 0,0000“</i>		

Anschließend wird der Teil der Eingabezeichenfolge abgeschnitten, der an das Array übergeben wurde. Die Variable *X* wird um einen Zähler erhöht und mit dem Rest der Zeichenfolge wird weitergearbeitet bis *X* = 4.

Sie können auch weitere Spalten zur Datei hinzufügen (z.B. für Layer, Linientyp, Skalierung usw.). Sie müssen dann nur das Array und den maximalen Wert für *X* entsprechend erhöhen.

Zum Schluss wird das erstellte Datenfeld noch an die Funktion übergeben und das Programm kann mit dem Einlesen der Blöcke fortfahren.

Startprozedur

Im oberen Teil werden, wie gehabt, die einzelnen Variablen deklariert und anschließend wird das Formular aufgerufen. Wenn dieses Formular mit EINLESEN geschlossen wurde, wird der Importabschnitt ausgeführt. Zuerst wird geprüft, ob die Datei wirklich vorhanden ist. Dazu nutzen wir die Funktion *Dir()*. Diese Funktion liefert den Dateinamen zurück, wenn die Datei gefunden wurde. Andernfalls wird eine Fehlermeldung ausgegeben und die Prozedur verlassen.

```
Public Sub ImportBlocks()
Dim Daten As String
Dim DField As Variant
Dim BlkName As String
Dim InsPoint(2) As Double
On Local Error Resume Next
With ImportFrm
.Show
If .OK = True Then
If Dir(.TextBox1.Text, vbNormal) = "" Then
MsgBox "Kann Datei " & .TextBox1.Text & _
" nicht öffnen", vbCritical, "Datei einlesen"
Exit Sub
End If
```

Die Datei war also vorhanden. Nun wird die Datei für das Einlesen geöffnet und an eine Dateinummer (#1) übergeben. Die Datei wird anschließend in der Schleife Zeile für Zeile durchlaufen, so lange, bis das Dateiende erreicht ist. Die jeweils eingelesene Zeile (Line Input #1, Daten), also die Variable *Daten*, wird an die Funktion *KonvertDaten* übergeben. Diese Funktion wandelt, wie oben beschrieben, die Zeile in ein Array um und übergibt dieses Array an die Variable *DField*. Da im ersten Datenfeld der Blockname steht, wird dieser Name in die Variable *BlkName* geschrieben, um mit der Anweisung *Blocks.Item(BlkName)* zu prüfen, ob der Block in der Zeichnung definiert wurde (mehr dazu in Kapitel 7.11). Ist der Block vorhanden, werden die Datenfelder eins bis drei des Arrays *DField* in eine Zahl umgewandelt (ist ja bis jetzt noch eine Zeichenkette) und an das 3D-Array, das den Einfügepunkt definiert, übergeben.

Zum Schluss wird die Blockreferenz mit der Anweisung *ModelSpace.InsertBlock()* in den Modellbereich der Zeichnung eingefügt. War das Einfügen erfolgreich, wird ein Verweis auf die Blockreferenz in die Objektvariable *NewBlk* zurückgegeben. Sie können nun mithilfe dieser Objektvariablen weitere Änderungen an den Eigenschaften der Blockreferenz vornehmen (z.B. *NewBlk.Layer = LayerName* usw.), wenn Sie möchten.

```
Open .TextBox1.Text For Input As #1
Do While Not EOF(1)
  Line Input #1, Daten
  DField = KonvertData(Daten)
  BlkName = DField(0)
  If TypeName(Blocks.Item(BlkName)) <> "Nothing" Then
    InsPoint(0) = Cdbl(DField(1))
    InsPoint(1) = Cdbl(DField(2))
    InsPoint(2) = Cdbl(DField(3))
    Set NewBlk = ModelSpace.InsertBlock(InsPoint, _
      BlkName, 1, 1, 1, 0)
  End If
Loop
Close
End If
End With
Application.ZoomExtents
End Sub
```

Die Eingabedatei, die wir für diese Prozedur verwenden, hat folgendes Format.

BlockName	X	Y	Z
Schacht	4475388,8869	5997768,3955	0,0000

Achten Sie bei den Trennern für die Dezimalstellen auf die Einstellungen des Systems (SYSTEM-
STEUERUNG → LÄNDEREINSTELLUNGEN → ZAHLEN).

Zweites Anwendungsbeispiel

Im zweiten Beispiel setzen wir uns mit Attributreferenzen auseinander. Wir werden diese Referenzen aus den jeweiligen Blöcken auslesen und in einem Dialogfenster auflisten und somit dem Benutzer die Möglichkeit der Anpassung dieser Daten geben.

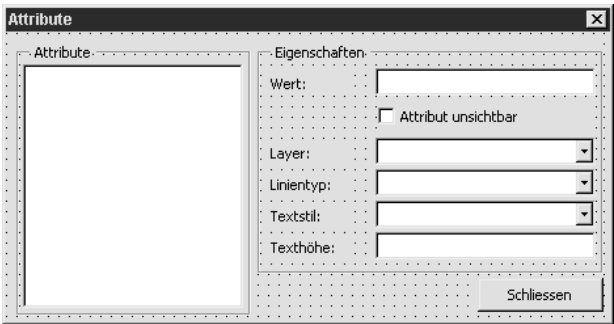


Abbildung 8.86: Formular zur Attributbearbeitung

Dieses Beispiel soll nur als Einstieg zum Anpassen der Eigenschaften dienen. Sie können dieses Programm auch problemlos erweitern und z.B. Textbreite und Neigungswinkel einarbeiten. Ich habe es bei den oben aufgeführten Eigenschaften belassen, da das Prinzip danach klar sein dürfte.

Startprozedur

In der Startprozedur soll der Benutzer eine Blockreferenz auswählen, deren Attributreferenzen (falls vorhanden) dem Formular (Abbildung 8.86) hinzugefügt werden. Gehen wir diese Prozedur mal im Einzelnen durch. Nach der Variablendeklaration beginnt zunächst eine *Do..Loop*-Schleife, die erst am Ende der Prozedur geschlossen wird. Innerhalb der Schleife läuft folgender Vorgang ab. Zuerst wird wieder das Objekt abgefragt (*GetEntity*), danach wird geprüft, ob es sich bei dem gewählten Objekt wirklich um eine Blockreferenz handelt. Ist dies nicht der Fall, wird die Schleife mit der Anweisung *Exit Do* verlassen und das Programm beendet. Andernfalls kann jetzt geprüft werden, ob der entsprechende Block Attribute enthält (*HasAttributes*). Ist dies der Fall, kommt schon der letzte Teil der Prozedur zum Einsatz.

```
Public Sub SelectBlkAttributes()
Dim VarPoint As Variant
Dim Object As Object
Dim Prompt As String, AttList
On Local Error Resume Next
Set Util = New MyUtility
Do
    Prompt = "Wählen Sie eine Blockreferenz:"
    Set Object = Nothing
    Util.GetEntity Object, PickedPoint, Prompt
    If TypeName(Object) = "IAcadBlockReference" Then
        If Object.HasAttributes = True Then
```

Zunächst werden der Variablen *AttList* des Formulars die Attribute des Blocks als Array übergeben (*GetAttributes*). In *AttList(0)* befindet sich dann das erste Attribut, in *AttList(1)* das zweite und so weiter. Die Obergrenze des Arrays liefert die *UBound()*-Funktion, womit eine Schleife geöffnet werden kann, die das Array vom ersten bis zum letzten Feld durchläuft. Ein Beispiel für den Aufbau dieses Arrays sehen Sie in der Abbildung 8.87. Da eine Blockreferenz auch konstante Attribute enthalten kann (also Attribute, deren Wert beim Einfügen des Blocks nicht durch den Benutzer festgelegt wird), müssen diese zunächst mit der Anweisung *GetConstantAttributes* abgerufen werden, um anschließend an das Array mit den „normalen“ Attributen angefügt zu werden.

```
With AttributFrm
    AttList = Object.GetAttributes
    ConstAttList = Object.GetConstantAttributes
    If UBound(ConstAttList) > -1 Then
        x = UBound(AttList)
        ReDim Preserve AttList(UBound(AttList) + _
            UBound(ConstAttList) + 1)
        For i = 0 To UBound(ConstAttList)
            Set AttList(x + i + 1) = ConstAttList(i)
        Next
    End If
    .AttList = AttList
```

Bei jedem Durchlauf wird nun der Name des Attributs dem Listenfeld des Formulars hinzugefügt. Sie müssen hierfür nicht extra den Verweis auf das Attribut an eine weitere Objektvariable übergeben, sondern können auch direkt mit dem Feld arbeiten.

```

        .ListBox1.Clear
    For i = 0 To UBound(.AttList)
        .ListBox1.AddItem .AttList(i).TagString
    Next
    .Show
End With
Else
    MsgBox "Blockreferenz enthält keine Attribute", _
        vbCritical, "Attributedit"
End If
ElseIf TypeName(Object) = "Nothing" Then
    Exit Do
End If
Loop
End Sub

```

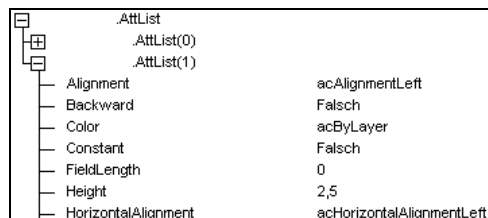


Abbildung 8.87: Beispiel für den Aufbau eines Arrays, das mit Attributreferenzen gefüllt ist

Formularcode

In der Initialisierungsprozedur des Formulars werden zunächst die einzelnen Listenfelder gefüllt, mit denen später der Benutzer die Auswahl des Layers, des Linientyps und/oder des Textstils vornehmen kann. Dabei ist es wichtig, dass die Elemente der jeweiligen Auflistung nicht berücksichtigt werden, die durch externe Referenzen definiert werden. Solche Einträge sind an dem Pipe-Zeichen (|) zu erkennen und können problemlos herausgefiltert werden.

```

Private Sub UserForm_Initialize()
    On Local Error Resume Next
    Set AllLayers = ThisDrawing.Layers
    For Each l In AllLayers
        If InStr(1, l.Name, "|") = 0 Then
            ComboBox1.AddItem l.Name
        End If
    Next
    Set AllLTypes = ThisDrawing.Linetypes
    For Each l In AllLTypes
        If InStr(1, l.Name, "|") = 0 Then
            ComboBox2.AddItem l.Name
        End If
    Next
    Set AllTStyles = ThisDrawing.TextStyles

```

```

For Each l In AllTStyles
    If InStr(1, l.Name, "|") = 0 And l.Name <> "" Then
        ComboBox3.AddItem l.Name
    End If
Next
End Sub

```

Die Ereignisprozedur der Listbox wird immer dann ausgelöst, wenn Sie ein bestimmtes Attribut der Liste auswählen. Wir können also die Eigenschaften des jeweils gewählten Attributs an das zur entsprechenden Eigenschaft zugehörige Steuerelement weiterreichen.

```

Private Sub ListBox1_Click()
    On Local Error Resume Next
    TextBox1.Text = AttList(ListBox1.ListIndex).TextString
    CheckBox1.Value = AttList(ListBox1.ListIndex).Invisible
    ComboBox1.Text = AttList(ListBox1.ListIndex).Layer
    ComboBox2.Text = AttList(ListBox1.ListIndex).Linetype
    ComboBox3.Text = AttList(ListBox1.ListIndex).StyleName
    TextBox2.Text = Util.RealToString(AttList( _
        ListBox1.ListIndex).Height)
End Sub

```

Die einzelnen *Click*-Ereignisprozeduren ändern jeweils die Eigenschaft, die mit dem dazugehörigen Steuerelement verwaltet wird. Die einzelnen Ereignisprozeduren zur Änderung der Eigenschaften der gewählten Attributreferenz sind vom Prinzip her alle gleich aufgebaut, nur dass eben immer eine andere Eigenschaft angesprochen wird. Nehmen wir beispielsweise die Änderung der Sichtbarkeit des Attributs. Diese wird in der Eigenschaft *Invisible* verwaltet, weshalb der Wert des Steuerelements *CheckBox1* direkt übergeben werden kann. Welches Attribut angesprochen werden muss, bestimmt die Auswahl in der Listbox.

```

Private Sub CheckBox1_Click()
    On Local Error Resume Next
    AttList(ListBox1.ListIndex).Invisible = CheckBox1.Value
End Sub

```

Änderung des Layers:

```

Private Sub ComboBox1_Click()
    On Local Error Resume Next
    AttList(ListBox1.ListIndex).Layer = ComboBox1.Text
End Sub

```

Änderung des Linientyps:

```

Private Sub ComboBox2_Click()
    On Local Error Resume Next
    AttList(ListBox1.ListIndex).Linetype = ComboBox2.Text
End Sub

```

Änderung des Textstils:

```

Private Sub ComboBox3_Click()
    On Local Error Resume Next
    AttList(ListBox1.ListIndex).StyleName = ComboBox3.Text
End Sub

```

Änderung der Zeichenfolge:

In den beiden Ereignisprozeduren der Textboxen wird ebenfalls die Eigenschaft, die zur jeweiligen Textbox gehört, angepasst. Die erste ändert die Zeichenfolge des Textes.

```
Private Sub TextBox1_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Local Error Resume Next
AttList(ListBox1.ListIndex).TextString = TextBox1.Text
End Sub
```

Änderung des Texthöhe:

Die zweite ändert die Höhe des Textes. Allerdings wird zusätzlich die Texthöhe kontrolliert und gegebenenfalls eine Fehlermeldung angezeigt.

```
Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Local Error Resume Next
If Util.DistanceToReal(TextBox2.Text) <= 0 Then
    MsgBox "Die Texthöhe muss einen Wert grösser 0 erhalten!", _
        vbCritical, "Texthöhe"
    Cancel = True
Else
    AttList(ListBox1.ListIndex).Height = Util.DistanceToReal( _
        TextBox2.Text)
    TextBox2.Text = Util.RealToString(AttList( _
        ListBox1.ListIndex).Height)
End If
End Sub
```

Im abschließenden Beispiel zu den Objekten Block- und Attributreferenz wollen wir uns noch einmal mit den Attributreferenzen auseinander setzen. Der Benutzer soll wiederum eine Blockreferenz in der Zeichnung auswählen, deren Attributtexte (*TextString*-Eigenschaft) in einem Formular aufgelistet werden. Das Besondere an diesem Projekt ist, dass die Steuerelemente, in denen die Attributtexte angezeigt werden, erst zur Laufzeit erstellt werden.

Als Erstes benötigen wir also ein Formular, in dem eine Combobox platziert ist, die die Namen der vorhandenen Blockdefinitionen aufnehmen wird. In diesem Feld wird außerdem der Name der gewählten Blockreferenz angezeigt. Im Rahmenbereich des Formulars werden dann die einzelnen Texte der Attribute in Textboxen eingetragen, die zur Laufzeit hinzugefügt werden.

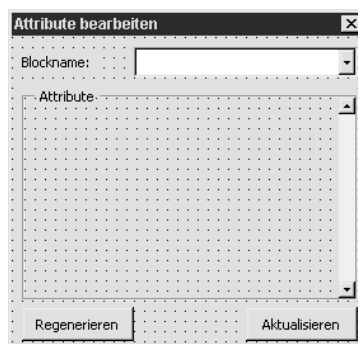


Abbildung 8.88: Projektformular

Startprozedur

Schauen wir uns zunächst die Startprozedur des Projekts an. In dieser Prozedur kann der Benutzer wieder eine Blockreferenz wählen, die dann an die Formularvariable *BlockRef* weitergegeben wird. Diesmal wird das Formular allerdings ungebunden angezeigt, so dass der Benutzer wieder eine Blockreferenz auswählen kann, ohne das Formular ausblenden zu müssen. Der ganze Vorgang läuft dabei in einer *Do...Loop*-Schleife ab, die erst verlassen wird, wenn der Benutzer kein Objekt mehr ausgewählt hat.

```
Public Sub editBlockref()
Dim Object As Object
On Error Resume Next
Set Util = New MyUtility
Do
    Set Object = Nothing
    Util.GetEntity Object, PickedPoint, "Blockreferenz wählen:"
    If TypeName(Object) = "IAcadBlockReference" Then
        If Object.HasAttributes = True Then
```

Nachdem die Auswahl erfolgreich ausgeführt wurde, wird der Verweis auf die gewählte Blockreferenz an die Formularvariable *BlockRef* übergeben. Anschließend wird die Initialisierungsprozedur *initAttributFrm* aufgerufen und dann, falls noch nicht geschehen, das Formular ungebunden eingeblendet. Da das Formular nicht immer in der linken oberen Ecke erscheinen soll, wird zusätzlich die letzte gespeicherte Position gesetzt und nach Verlassen der Schleife geschrieben.

```
        Set AttribEditFrm.BlockRef = Object
        AttribEditFrm.initAttributFrm
        If AttribEditFrm.Visible = False Then
            AttribEditFrm.Show 0
            AttribEditFrm.Left = GetSetting("BspBlockReference", _
                "Position", "Left", AttribEditFrm.Left)
            AttribEditFrm.top = GetSetting("BspBlockReference", _
                "Position", "Top", AttribEditFrm.top)
        End If
    Else
        MsgBox "Blockreferenz enthält keine Attribute", _
            vbCritical, "Attributedit"
    End If
    ElseIf TypeName(Object) = "Nothing" Then
        Exit Do
    End If
Loop
SaveSetting "BspBlockReference", "Position", "Left", _
    AttribEditFrm.Left
SaveSetting "BspBlockReference", "Position", "Top", _
    AttribEditFrm.top
Unload AttribEditFrm
End Sub
```

Initialisierungsprozedur

Diese Prozedur stellt wieder den umfangreichsten Teil des Projekts dar. Sie wird immer dann ausgeführt, wenn der Benutzer eine neue Blockreferenz in der Zeichnung ausgewählt hat.

```
Public Sub initAttributFrm()
On Error Resume Next
NoChange = True
```

Im ersten Teil dieser Prozedur wird die Combobox mit den Namen der Blockdefinitionen aus der Blockauflistung gefüllt. Dabei ist es wichtig, unbenannte Blockdefinitionen (beginnen mit *) und Blockreferenzen, die durch externe Referenzen definiert werden, nicht mit in die Auswahlliste zu übernehmen.

```
ComboBox1.Clear
Set AllBlocks = ThisDrawing.Blocks
For Each Blk In AllBlocks
    If InStr(1, Blk.Name, "|") = 0 And Left(Blk.Name, 1) <> "*" _
        Then
        ComboBox1.AddItem Blk.Name
    End If
Next
ComboBox1.Text = BlockRef.Name
```

Im zweiten Teil der Prozedur werden dem Rahmen nun für die einzelnen Attribute der Blockreferenz *Label*- und *TextBox*-Steuerelemente hinzugefügt. Zunächst werden aber wieder die normalen und konstanten Attribute (falls beide Formen existieren) zu einem Array zusammengefasst.

```
Frame1.Controls.Clear
Frame1.ScrollHeight = 0
If BlockRef.HasAttributes = True Then
    AttList = BlockRef.GetAttributes
    ConstAttList = BlockRef.GetConstantAttributes
    If UBound(ConstAttList) > -1 Then
        x = UBound(AttList)
        ReDim Preserve AttList(UBound(AttList) + _
            UBound(ConstAttList) + 1)
        For i = 0 To UBound(ConstAttList)
            Set AttList(x + i + 1) = ConstAttList(i)
        Next
    End If
```

Anschließend folgt die Schleife, die die Steuerelemente erstellen und mit den Attributwerten füllen wird. Pro Attribut ist immer ein *Label* und eine *TextBox* erforderlich. Da sich diese nicht überlagern sollen, muss ein gewisses System in die Positionierung der Steuerelemente gebracht werden. Das bedeutet, dass wir eine Variable benötigen, die den aktuellen Abstand vom oberen Rand (*Top*) des zuletzt eingefügten Steuerelements speichert. Den Anfangswert dieses Abstands setzen wir einfach mal auf den Wert 10, damit das erste Steuerelement nicht genau am oberen Rand des Rahmens eingefügt wird.

```
LastTop = 10
For i = 0 To UBound(AttList)
```


Als Erstes wird das Label eingefügt. Das passiert mit der *Add*-Anweisung der *Controls*-Auflistung unter Angabe des Klassennamens „Forms.Label.1“ (siehe *Kapitel 3.4.2*). Da als Nächstes bereits alle Eigenschaften zugewiesen werden, brauchen Sie hier nicht unbedingt einen Namen für das Steuerelement zu vergeben.

```
Set NewLb = Frame1.Controls.Add("Forms.Label.1", _
    "AttLabel" & CStr(i), True)
NewLb.Left = 5
NewLb.Top = LastTop + 2
NewLb.Width = 80
NewLb.Caption = AttList(i).TagString
```

Im zweiten Schritt wird das Textfeld hinzugefügt. Hier ist es wichtig, den Namen festzulegen, da im späteren Verlauf wieder auf die einzelnen Textfelder zugegriffen wird und somit ein bekannter Identifier festgelegt wurde. Der Name setzt sich dabei aus der Zeichenfolge „AttBox“ und dem Wert der Variablen *i* zusammen. Auf diese Weise entstehen Textfelder, deren Namen wie folgt lauten werden: AttBox0, AttBox1, AttBox2, AttBox3 usw.

```
Set NewTB = Frame1.Controls.Add("Forms.TextBox.1", _
    "AttBox" & CStr(i), True)
NewTB.Left = 90
NewTB.Top = LastTop
NewTB.Width = 90
NewTB.Text = AttList(i).TextString
LastTop = LastTop + NewTB.Height
```

Falls die Blockreferenz mehr Attributinformationen enthält, als im sichtbaren Bereich des Rahmens dargestellt werden können, muss die *ScrollHeight*-Eigenschaft entsprechend angepasst werden, damit der Benutzer auch das unterste Textfeld erreichen kann.

```
If LastTop > Frame1.Height Then
    Frame1.ScrollHeight = LastTop + 11
End If
Next
End If
NoChange = False
End Sub
```

Auswahl eines neuen Blocknamens

Wenn der Benutzer einen Blocknamen aus dem Auswahlfeld der Blockdefinitionen auswählt, wird dieser der *Name*-Eigenschaft der Blockreferenz zugewiesen, womit sich nicht nur eine vollkommen neue Blockdarstellung in der Zeichnung ergibt, sondern auch die Attributreferenzen des Blocks andere sind. Das gilt nur, wenn es sich um konstante Attribute handelt. Diese werden ja bekanntlich nicht auf die neue Blockdefinition übertragen, normale schon. In jedem Fall muss die Initialisierungsprozedur erneut aufgerufen werden, damit die Anzeige der Attributwerte aktualisiert wird.

```
Private Sub ComboBox1_Click()
On Error Resume Next
If NoChange = True Then Exit Sub
BlockRef.Name = ComboBox1.Text
initAttributFrm
End Sub
```

Schaltfläche „Aktualisieren“

Da es äußerst schwierig ist, Ereignisprozeduren einer unbekannten Anzahl von Steuerelementen zu aktivieren, ist es notwendig, die Aktualisierung der Attributwerte über eine Schaltfläche auszuführen. Diese Aktualisierung läuft in einer *For...Next*-Schleife ab. Bei jedem Durchlauf wird der Name des Steuerelements wieder mit „AttBox“ + Index zusammengesetzt und so das entsprechende Steuerelement über die *Controls*-Auflistung des Rahmens abgerufen. Da die Nummer des Textfelds mit der Position des Attributs im Array *AttList* übereinstimmt, kann die Zeichenfolge der jeweiligen Textbox an die *TextString*-Eigenschaft des entsprechenden Attributs weitergegeben werden.

```
Private Sub CommandButton1_Click()
On Error Resume Next
For i = 0 To UBound(AttList)
    Err.Clear
    AttList(i).TextString = Frame1.Controls("AttBox" & _
        CStr(i)).Text
    If Err = 0 Then
        AttList(i).Update
    Else
        Frame1.Controls("AttBox" & CStr(i)).Text = _
            AttList(i).TextString
    End If
    AttList(i).Update
Next
End Sub
```

8.3.30 Mehrfachblockreferenz (AcadMInsertBlock)

Bei dieser Klasse handelt es sich um eine Blockreferenz, mit der Sie eine rechtwinklige Anordnung von Blockreferenzen erzeugen können, ohne mehrere Blöcke in der Zeichnung zu erzeugen. Blockreferenzen der Klasse *AcadMInsertBlock* werden in AutoCAD mit dem Befehl *_minsert* eingefügt.

Sie können einen Verweis auf eine Mehrfachblockreferenz erstellen, indem Sie mit der Funktion *AddMInsertBlock* eine neue Mehrfachblockreferenz dem Modellbereich, Papierbereich oder einem Block hinzufügen. Sie können auch eine vorhandene Mehrfachblockreferenz über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften des Mehrfachblocks

Die Mehrfachblockreferenz ist eine Erweiterung der normalen Blockreferenz, die somit auch über die Eigenschaften normaler Blockreferenzen, wie Skalierung, Drehwinkel, Einfügepunkt usw. verfügt. Darüber hinaus besitzt dieses Objekt aber noch ganz spezielle Eigenschaften, die nur in diesem Objekttyp vorkommen. Mit diesen speziellen Eigenschaften wird die rechtwinklige Anordnung des Blocks definiert.

Die Spalten der Anordnung werden durch die Eigenschaften *Columns* und *ColumnSpacing* bestimmt und die Zeilen durch *Rows* und *RowSpacing*. Die Anzahl der Spalten bzw. Zeilen werden mit den Eigenschaften *Columns* und *Rows* festgelegt, die anderen beiden bestimmen jeweils den Abstand zwischen den einzelnen Objekten.

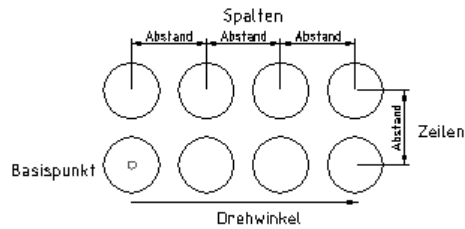


Abbildung 8.89: Mehrfachblock

Für die Anzahl der Reihen bzw. Spalten müssen Sie Ganzzahlen, die größer 0 sind, verwenden. Als Abstandswerte sind allerdings auch negative Werte möglich. In diesem Fall wird die Reihe und/oder Spalte in die entgegengesetzte Richtung erstellt.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der Blockreferenz fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der Blockreferenz zurück.
<i>HasAttributes</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie feststellen, ob die entsprechende Blockreferenz Attributdefinitionen enthält. Siehe auch Blockreferenz.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der Blockreferenz ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der Blockreferenz zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>InsertionPoint</i>	Diese Eigenschaft legt den Einfügepunkt der Blockreferenz fest oder ruft diesen ab. Siehe auch Blockreferenz.
<i>Layer</i>	Zeigt den Layer der Blockreferenz an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der Blockreferenz an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der Blockreferenz an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuelle Linienstärke für die Blockreferenz an oder legt diese fest.
<i>Name</i>	Mit dieser Eigenschaft können Sie den Namen der aktuellen Blockreferenz abrufen oder durch Zuweisung eines neuen Namens die Darstellung der Blockreferenz ändern. Siehe auch Blockreferenz.

Tabelle 8.97: Weitere Eigenschaften des Mehrfachblocks

Eigenschaft	Beschreibung
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der Blockreferenz zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadMInsertBlock</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die Blockreferenz zurück oder legt diesen fest.
<i>Rotation</i>	Diese Eigenschaft legt den aktuellen Drehwinkel der Blockreferenz fest oder gibt diesen zurück. Siehe auch Blockreferenz.
<i>Visible</i>	Mit dieser Eigenschaft können Sie die Sichtbarkeit der Blockreferenz steuern.
<i>XScaleFactor</i>	Zeigt den aktuellen Wert des X-Skalierungsfaktors der Blockreferenz an oder legt diesen fest. Siehe auch Blockreferenz.
<i>YScaleFactor</i>	Zeigt den aktuellen Wert des Y-Skalierungsfaktors der Blockreferenz an oder legt diesen fest. Siehe auch Blockreferenz.
<i>ZScaleFactor</i>	Zeigt den aktuellen Wert des Z-Skalierungsfaktors der Blockreferenz an oder legt diesen fest. Siehe auch Blockreferenz.

Tabelle 8.97: Weitere Eigenschaften des Mehrfachblocks (Forts.)

Methoden des Mehrfachblocks

Eine Mehrfachblockreferenz enthält dieselben Methoden wie eine einfache Blockreferenz. Sie können also ebenfalls Attribute mit der *GetAttributes*- oder *GetConstantAttributes*-Anweisung abrufen oder die Blockreferenz mit *Explode* auflösen.

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.

Tabelle 8.98: Methoden des Mehrfachblocks

Prozedur	Beschreibung
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>Explode</i>	Diese Anweisung löst die Blockreferenz auf und gibt die daraus entstandenen Objekte als Array zurück. Siehe auch Blockreferenz.
<i>GetAttributes</i>	Mit dieser Anweisung rufen Sie die Attribute der Blockreferenz ab. Sie erhalten als Ergebnis ein Array, das alle Attributreferenzen auflistet. Siehe auch Blockreferenz.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetConstantAttributes</i>	Mit dieser Anweisung rufen Sie die konstanten Attribute, also alle Attribute, in denen die Eigenschaft <i>Constant</i> auf <i>TRUE</i> gesetzt ist, der externen Referenz ab. Sie erhalten als Ergebnis ein Array, das alle Attribute auflistet. Im Gegensatz zu <i>GetAttributes</i> werden mit dieser Funktion nicht die Attributreferenzen zurückgegeben. Siehe auch Blockreferenz.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der Blockreferenz ab.
<i>GetXData</i>	Ruft die erweiterten Daten der Blockreferenz ab.
<i>Highlight</i>	Hebt die Blockreferenz hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der Blockreferenz mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine Blockreferenz als Resultat einer Spiegelung mit dieser Blockreferenz.
<i>Mirror3D</i>	Erstellt eine neue Blockreferenz als Resultat einer räumlichen Spiegelung dieser Blockreferenz.
<i>Move</i>	Verschiebt die Blockreferenz von Ursprungs- auf Zielkoordinaten.
<i>Rotate</i>	Dreht die Blockreferenz um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die Blockreferenz um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die Blockreferenz um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der Blockreferenz.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die Blockreferenz an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der Blockreferenz im AutoCAD-Zeichnungsfenster.

Tabelle 8.98: Methoden des Mehrfachblocks (Forts.)

Anwendungsbeispiel

(Beispielprojekt: 08\AcadMInsertBlock.dvb)

Dieses Beispielprojekt enthält zwei verschiedene Startprozeduren, die beide unterschiedliche Funktionen ausführen, aber auf das gleiche Formular zugreifen. Mit der ersten Prozedur kann der Benutzer über eben dieses Formular einige Einstellungen zur Einfügung eines Mehrfachblocks vornehmen und parallel dazu einen Einfügekpunkt im Zeichnungsfenster wählen. Hat er dies getan, wird der Mehrfachblock an der gewählten Position und mit den im Formular eingestellten Eigenschaften eingefügt.

Die zweite Prozedur ermöglicht es dem Benutzer, einen Auswahlatz mit Objekten zusammenzustellen, von dem dann die Eigenschaften der gewählten Mehrfachblöcke mit den Einstellungen des parallel aufgerufenen Formulars angepasst werden.

1. Prozedur: Mehrfachblöcke erzeugen

Mit dieser Prozedur kann der Benutzer Mehrfachblöcke in der Zeichnung einfügen. Dazu wird ein Formular ungebunden gestartet, so dass die Auswahl des Einfügekpunkts parallel erfolgen kann. Der gesamte Abfragecode läuft dabei in einer *Do...Loop*-Schleife ab, die erst verlassen wird, wenn der Benutzer keinen Punkt in der Zeichnung mehr wählt.

```
Public Sub insertMBlockref()
Dim Object As Object
Const PI = 3.14159265358979
On Error Resume Next
Set Util = New MyUtility
Do
```

Zunächst wird geprüft, ob das Formular sichtbar ist. Ist dies nicht der Fall, wird es an der letzten gespeicherten Position ungebunden eingeblendet. Danach wird der Benutzer zur Wahl eines Einfügekpunkts aufgefordert.

```
If MBlockFrm.Visible = False Then
    MBlockFrm.initMBlockFrm
    MBlockFrm.Show 0
    MBlockFrm.Left = GetSetting("BspMBlockReference", _
        "Position", "Left", MBlockFrm.Left)
    MBlockFrm.top = GetSetting("BspMBlockReference", _
        "Position", "Top", MBlockFrm.top)
End If
InsPoint = Empty
MBlockFrm.LastPrompt = "Einfügekpunkt wählen:"
InsPoint = Util.GetPoint(, "Einfügekpunkt wählen:")
If TypeName(InsPoint) = "Double()" Then
```

Hat er dies getan, werden die Einstellungen, die der Benutzer im Formular vorgenommen hat, für die Einfügung des Mehrfachblocks verwendet.

```
With MBlockFrm
    MName = .ComboBox1
    XScale = Util.DistanceToReal(.TextBox5)
    YScale = Util.DistanceToReal(.TextBox6)
    ZScale = Util.DistanceToReal(.TextBox7)
    If .CheckBox2 = True Then
        Rotation = 2 * PI - GetVariable("VIEWTWIST")
```

```

Else
    Rotation = Util.AngleToReal(.TextBox8)
End If
Rows = CLng(.TextBox1)
Cols = CLng(.TextBox3)
RSpace = Util.DistanceToReal(.TextBox2)
CSpace = Util.DistanceToReal(.TextBox4)
Err.Clear

```

Die Blockreferenz selbst wird entweder in den Modell- oder in den Papierbereich eingefügt, je nachdem, welcher gerade aktiv ist (*ActiveSpace*-Eigenschaft).

```

If ActiveSpace = acModelSpace Then
    Set .BlockRef = ModelSpace.AddInsertBlock( _
        InsPoint, MName, XScale, YScale, ZScale, _
        Rotation, Rows, Cols, RSpace, CSpace)
Else
    Set .BlockRef = PaperSpace.AddInsertBlock( _
        InsPoint, MName, XScale, YScale, ZScale, _
        Rotation, Rows, Cols, RSpace, CSpace)
End If
If Err <> 0 Then
    MsgBox Err.Description
End If
End With
ElseIf TypeName(Object) = "Nothing" Then
    Exit Do
End If
Loop
With MBlockFrm
End With
Unload MBlockFrm
End Sub

```

2. Prozedur: Eigenschaften von Mehrfachblöcken anpassen

Mit der zweiten Prozedur kann der Benutzer vorhandene Objekte in der Zeichnung mittels eines Auswahlsatzes zusammenstellen, von denen dann die Mehrfachblöcke mit den Einstellungen des Formulars angepasst werden.

```

Public Sub editMBlockref()
Dim sSet As AcadSelectionSet
Const PI = 3.14159265358979
On Error Resume Next
Set Util = New MyUtility

```

Der erste Schritt in dieser Prozedur ist die Initialisierung eines Auswahlsatzes mit dem Namen „Auswahl“. Sollte dieser noch nicht in der *SelectionSets*-Auflistung existieren, wird er dieser hinzugefügt. Anschließend wird ein Verweis auf diesen Auswahlatz in der Variablen *sSet* gespeichert.

```

If TypeName(SelectionSets("AUSWAHL")) = "Nothing" Then
    SelectionSets.Add "AUSWAHL"
End If
Set sSet = SelectionSets("AUSWAHL")

```

Innerhalb der *Do..Loop*-Schleife kann der Benutzer nun bei jedem Durchlauf einen Auswahl-satz zusammenstellen, der beim nächsten Durchlauf wieder geleert wird, bevor erneut eine Auswahl durch den Benutzer erfolgt. Hat der Benutzer keine Objekte mehr gewählt, wird die Schleife mit *Exit Do* verlassen und das Programm beendet.

Do

```

    If MBlockFrm.Visible = False Then
        MBlockFrm.initMBlockFrm
        MBlockFrm.Show 0
        MBlockFrm.Left = GetSetting("BspMBlockReference", _
            "Position", "Left", MBlockFrm.Left)
        MBlockFrm.top = GetSetting("BspMBlockReference", _
            "Position", "Top", MBlockFrm.top)
    End If
    sSet.Clear
    sSet.SelectOnScreen
    If sSet.Count = 0 Then Exit Do

```

Hat der Benutzer hingegen Objekte in der Zeichnung ausgewählt, können wir die gewählten Objekte auswählen, indem wir den Auswahl-satz Objekt für Objekt durchlaufen. Falls es sich bei dem gewählten Objekt um einen Mehrfachblock handelt (*TypeName()*-Funktion), können die Eigenschaften dieses Blocks mit den Einstellungen, die der Benutzer im Formular vorgenommen hat, angepasst werden. Sie können statt der Prüfung des Objekttyps auch einen Filter bei der Auswahl der Objekte verwenden, so dass dieser nur Blockreferenzen selektiert, zu denen auch Mehrfachblöcke zählen.

```

    For i = 0 To sSet.Count - 1
        If TypeName(sSet.Item(i)) = "IAcadMInsertBlock" Then
            Set MBlockFrm.BlockRef = sSet.Item(i)
            With MBlockFrm
                .BlockRef.Name = .ComboBox1
                .BlockRef.XScale = Util.DistanceToReal(.TextBox5)
                .BlockRef.YScale = Util.DistanceToReal(.TextBox6)
                .BlockRef.ZScale = Util.DistanceToReal(.TextBox7)
                If .CheckBox2 = True Then
                    .BlockRef.Rotation = 2 * PI - GetVariable( _
                        "VIEWTWIST")
                Else
                    .BlockRef.Rotation = Util.AngleToReal(.TextBox8)
                End If
                .BlockRef.Rows = CLng(.TextBox1)
                .BlockRef.Columns = CLng(.TextBox3)
                .BlockRef.RowSpacing = Util.DistanceToReal(.TextBox2)
                .BlockRef.ColumnSpacing = Util.DistanceToReal( _
                    .TextBox4)
                Err.Clear
                If Err <> 0 Then
                    MsgBox Err.Description
                End If
            End With
        End If
    Next
Loop

```



```

SaveSetting "BspMBlockReference", "Position", "Left", MBlockFrm.Left
SaveSetting "BspMBlockReference", "Position", "Top", MBlockFrm.top
Unload MBlockFrm
End Sub

```

Formularcode

In der Initialisierungsprozedur, die durch die jeweilige Startprozedur aufgerufen wird, werden zunächst die Namen aller vorhandenen Blockdefinitionen der Blockauflistung der *ComboBox1* hinzugefügt. Aus dieser Liste kann später der Benutzer den Blocknamen auswählen, mit dem der Mehrfachblock eingefügt oder die ausgewählten geändert werden sollen.

```

Public Sub initMBlockFrm()
On Error Resume Next
NoChange = True
ComboBox1.Clear
Set AllBlocks = ThisDrawing.Blocks
For Each Blk In AllBlocks
    If InStr(1, Blk.Name, "|") = 0 And Left(Blk.Name, 1) _
        <> "*" Then
        ComboBox1.AddItem Blk.Name
    End If
Next

```

Die anderen Einstellungen werden aus der Windows-Registrierungsdatenbank gelesen, falls bereits eine Speicherung durchgeführt wurde. Andernfalls werden einfach Vorgabewerte vergeben. Wenn der Benutzer dann das Programm beendet und das Formular entladen wird, werden die letzten Einstellungen des Formulars in die Registrierung zurückgeschrieben.

```

ComboBox1.ListIndex = 0
TextBox1 = GetSetting("BspMBlockReference", "Setup", _
    "TextBox1", 1)
TextBox2 = GetSetting("BspMBlockReference", "Setup", _
    "TextBox2", Util.RealToString(1))
TextBox3 = GetSetting("BspMBlockReference", "Setup", _
    "TextBox3", 1)
TextBox4 = GetSetting("BspMBlockReference", "Setup", _
    "TextBox4", Util.RealToString(1))
TextBox5 = GetSetting("BspMBlockReference", "Setup", _
    "TextBox5", Util.RealToString(1))
TextBox6 = GetSetting("BspMBlockReference", "Setup", _
    "TextBox6", Util.RealToString(1))
TextBox7 = GetSetting("BspMBlockReference", "Setup", _
    "TextBox7", Util.RealToString(1))
TextBox8 = GetSetting("BspMBlockReference", "Setup", _
    "TextBox8", Util.AngleToString(0))
CheckBox1 = CBool(GetSetting("BspMBlockReference", _
    "Setup", "CheckBox1", CheckBox1))
CheckBox2 = CBool(GetSetting("BspMBlockReference", _
    "Setup", "CheckBox2", CheckBox2))
End Sub

```

Ereignisprozeduren des Formulars

Das Formular stellt wieder einige Funktionen zur Verfügung, mit denen der Benutzer die entsprechenden Einstellungen im Formular vornehmen kann. Dazu zählen neben den Formatierungen der eingegebenen Werte (Skalierung, Abstandswerte, Winkel usw.) auch die Ereignisprozeduren einiger Schaltflächen. Diese Schaltflächen sind wieder keine Schaltflächen im herkömmlichen Sinn (*CommandButton*), sondern *Image*-Steuerelemente, die zu Schaltflächen umfunktioniert wurden. Das wurde notwendig, da sie ein Symbol zur Darstellung der Funktion aufnehmen sollen. Diese Darstellung wäre mit *CommandButton* bei der geringen Größe der Schaltflächen nicht möglich gewesen. Um nun wirklich eine Schaltfläche zu simulieren, muss der Benutzer den Eindruck erhalten, dass diese beim Betätigen der Maustaste eingedrückt und beim Loslassen wieder erhaben dargestellt wird. Dazu nutzen wir die Ereignisse *MouseDown* für das Drücken der Maustaste und *MouseUp* für das Loslassen. Innerhalb dieser Prozeduren muss nun nur die *SpecialEffect*-Eigenschaft des entsprechenden Image auf *fmSpecialEffectSunken*, für eine heruntergedrückte Maustaste oder *fmSpecialEffectRaised*, wenn die Maustaste wieder losgelassen wird.

```
Private Sub OpenBtn_MouseDown(ByVal Button As Integer, ByVal _
Shift As Integer, ByVal X As Single, ByVal Y As Single)
OpenBtn.SpecialEffect = fmSpecialEffectSunken
End Sub
Private Sub OpenBtn_MouseUp(ByVal Button As Integer, ByVal _
Shift As Integer, ByVal X As Single, ByVal Y As Single)
OpenBtn.SpecialEffect = fmSpecialEffectRaised
End Sub
```

Die eigentliche Aufgabe der Schaltfläche wird in der *Click*-Ereignisprozedur des jeweiligen Image ausgeführt.

Blockdefinition einlesen

Über diese Schaltfläche soll es dem Benutzer ermöglicht werden, eine Zeichnungsdatei über den ÖFFNEN-Standarddialog auszuwählen, damit diese dann in die Zeichnung als Blockdefinition eingelesen werden kann. Da die Eigenschaften dieses Steuerelements (*CommonDialog*) bereits zur Entwurfszeit angepasst wurden (*DialogTitle*, *Filter*, *CancelError*), muss hier nur der entsprechende Dialog mit der *ShowOpen*-Anweisung aufgerufen werden.

```
Private Sub OpenBtn_Click()
On Error Resume Next
Err.Clear
ComDialog.ShowOpen
If Err = 0 Then
```

Hat der Benutzer eine Datei ausgewählt (die Betätigung der ABBRUCH-Schaltfläche löst ja einen Fehler aus, da *CancelError* auf *TRUE* gesetzt wurde), kann die gewählte Zeichnung in das aktuelle Dokument als Blockreferenz eingefügt werden. Dazu muss zunächst eine Blockreferenz mit der *InsertBlock*-Anweisung erzeugt werden. Damit die gewählte Zeichnung eingefügt wird, ist es wichtig, den Blocknamen einschließlich der Verzeichnisangabe als Name des Blocks anzugeben. Als Basispunkt dieser Einfügung nehmen wir einfach das Zentrum des aktuellen Zeichnungsausschnitts.

```
InsPoint = ThisDrawing.GetVariable("VIEWCTR")
Set newBlk = ThisDrawing.ModelSpace.InsertBlock(InsPoint, _
ComDialog.FileName, 1, 1, 1, 0)
```

Wenn der Block erfolgreich in die Zeichnung eingefügt wurde, enthält die Variable *newBlk* einen Verweis auf die neue Blockreferenz. Da die Zeichnung nun eventuell eine weitere Blockdefinition enthält (falls keine vorhandene Blockdefinition ersetzt wurde), müssen wir mit *initBlockFrm* die Einträge dieser Liste aktualisieren. Anschließend wird der Name der Blockreferenz als aktuelle Auswahl in der Blockauswahlliste (*ComboBox1*) gesetzt. Da nun die erzeugte Blockreferenz nicht mehr benötigt wird, kann diese mit der *Delete*-Anweisung wieder aus der Zeichnung entfernt werden.

```

    If TypeName(newBlk) = "IAcadBlockReference" Then
        initMBlockFrm
        ComboBox1.Text = newBlk.Name
        newBlk.Delete
    End If
End If
End Sub

```

Reihen- und Zeilenabstände in der Zeichnung wählen

Damit der Benutzer den Abstand der Reihen bzw. Zeilen des Mehrfachblocks nicht nur im Textfeld eingeben, sondern auch in der Zeichnung wählen kann, müssen wir eine Funktion zur Verfügung stellen, die diese Aufgabe übernimmt.

Für die Auswahl des Zeilenabstands ist die *Click*-Ereignisprozedur der Schaltfläche *ColBtn* zuständig. Die Prozedur selbst unterteilt sich in zwei Abschnitte. Im ersten wird der Benutzer aufgefordert, einen Abstand in der Zeichnung mithilfe der *GetDistance*-Funktion der Klasse *MyUtility* zu wählen. Zusätzlich werden die Schlüsselwörter „Von“ und „Objekt“ der Funktion im Parameter *KeyWordList* mitgegeben. Dadurch kann der Benutzer in der Eingabeaufforderung die Option „von Objekt“ eingeben, um die nächste Möglichkeit zur Auswahl eines Abstands zu nutzen.

```

Private Sub ColBtn_Click()
    On Error Resume Next
    Dist = Util.GetDistance(, "Abstand wählen oder <Von Objekt>:", _
        "Von Objekt")
    ThisDrawing.Utility.Prompt LastPrompt
    If Dist = "Von" Or Dist = "Objekt" Then

```

Hat er also diese Option gewählt, wird er nun aufgefordert, einen Mehrfachblock in der Zeichnung zu bestimmen, von dem der entsprechende Zeilenabstand übernommen werden soll. Dazu nutzen wir die *GetEntity*-Funktion der Klasse *MyUtility*. Hat der Benutzer einen Mehrfachblock gewählt, übergeben wir den formatierten Wert der *ColumnSpacing*-Eigenschaft an das Textfeld, das den aktuellen Zeilenabstand enthält.

```

        Util.GetEntity Object, PickedPoint, "Mehrfachblock wählen:"
        If TypeName(Object) = "IAcadMInsertBlock" Then
            TextBox4 = Util.RealToString(Object.ColumnSpacing)
        End If
        ThisDrawing.Utility.Prompt LastPrompt
    ElseIf TypeName(Dist) = "Double" Then

```

Hat der Benutzer hingegen mit der ersten Funktion (*GetDistance*) einen Abstand bestimmt, wird dieser Wert formatiert an das Textfeld übergeben.

```

        TextBox4 = Util.RealToString(Dist)
    End If
End Sub

```

Die Ereignisprozedur der Schaltfläche für den Reihenabstand funktioniert ähnlich. Der Unterschied zur vorherigen Prozedur besteht lediglich darin, bei der Auswahl eines Mehrfachblocks die Eigenschaft *RowSpacing* statt *ColumnSpacing* anzusprechen.

```
Private Sub RowBtn_Click()
On Error Resume Next
Dist = Util.GetDistance(, "Abstand wählen oder <Von Objekt>:", _
    "Von Objekt")
ThisDrawing.Utility.Prompt LastPrompt
If Dist = "Von" Or Dist = "Objekt" Then
    Util.GetEntity Object, PickedPoint, "Mehrfachblock wählen:"
    If TypeName(Object) = "IAcadMinsertBlock" Then
        TextBox2 = Util.RealToString(Object.RowSpacing)
    End If
    ThisDrawing.Utility.Prompt LastPrompt
ElseIf TypeName(Dist) = "Double" Then
    TextBox2 = Util.RealToString(Dist)
End If
End Sub
```

Drehwinkel in der Zeichnung wählen

Auch die Auswahl eines Drehwinkels für den Block läuft in zwei Schritten ab. Zunächst kann der Benutzer einfach einen Winkel in der Zeichnung mit der *GetOrientation*-Anweisung ermitteln. Auch dieser Funktion werden im Parameter *KeyWordList* die Schlüsselwörter „Von“ und „Objekt“ mitgegeben, damit der Benutzer den Drehwinkel eines grafischen Objekts übernehmen kann.

```
Private Sub AngleBtn_Click()
On Error Resume Next
Angle = Util.GetOrientation(, "Drehwinkel wählen oder " & _
    "<vonObjekt>:", "Von Objekt")
ThisDrawing.Utility.Prompt LastPrompt
If Angle = "Von" Or Angle = "Objekt" Then
```

Hat er diese Auswahl getroffen, wird er wiederum aufgefordert, ein Objekt in der Zeichnung zu wählen. Dabei kann es sich um eine Linie oder ein Objekt handeln, das die Eigenschaft *Rotation* besitzt. Ist das gewählte Objekt eine Linie, übernehmen wir den Winkel von dessen *Angle*-Eigenschaft. Ist es ein anderes Objekt, versuchen wir, die *Rotation*-Eigenschaft des Objekts zu verwenden. Besitzt das Objekt diese Eigenschaft nicht, löst AutoCAD einen Fehler aus, was aber nichts macht, da das Programm in diesem Fall mit der nächsten Codezeile fortgesetzt wird. Diese Vorgehensweise ist zwar nicht sehr elegant, reicht aber für unsere Belange vollkommen aus.

```
    Util.GetEntity Object, PickedPoint, "Objekt wählen:"
    If TypeName(Object) = "IAcadLine" Then
        TextBox8 = Util.AngleToString(Object.Angle)
    Else
        TextBox8 = Util.AngleToString(Object.Rotation)
    End If
    ThisDrawing.Utility.Prompt LastPrompt
ElseIf TypeName(Angle) = "Double" Then
```

Sollte der Benutzer allerdings mit der *GetOrientation*-Funktion bereits einen Winkel gewählt haben, können wir den formatierten Wert in dem Textfeld eintragen:

```

    TextBox8 = Util.AngleToString(Angle)
End If
End Sub

```

8.3.31 Externe Referenz (AcadExternalReference)

Externe Referenzen werden in VBA durch die Klasse *AcadExternalReference* dargestellt. Sie können über diese Klasse auf die Eigenschaften und Methoden des Objekts zugreifen und somit gewisse Anpassungen an der externen Referenz vornehmen.

Sie können einen Verweis auf eine externe Referenz erstellen, indem Sie mit der Funktion *AttachExternalReference* eine Referenz in den Modellbereich, den Papierbereich oder einen Block einfügen. Sie können auch eine vorhandene externe Referenz über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln.

Eigenschaften der externen Referenz

Da dieses Objekt nichts anderes als ein eingefügter Block ist, dessen Daten sich in einer externen Datei befinden, enthält diese Klasse auch dieselben Eigenschaften wie eine Blockreferenz. *InsertionPoint* beispielsweise speichert den Einfügepunkt, *Name* bestimmt den Namen oder *XScaleFactor*, *YScaleFactor* und *ZScaleFactor* die Skalierungen der Referenz. Zusätzlich besitzt das Objekt die Eigenschaft *Path*, mit der der Name der Datei gespeichert wird, auf der die externe Referenz basiert. Wenn Sie dieser Eigenschaft einen neuen Dateinamen zuordnen, dann ändert sich ebenfalls der Wert der *Path*-Eigenschaft eventueller Kopien der externen Referenz. Damit die Änderungen in der Zeichnung sichtbar werden, müssen Sie die *Reload*-Anweisung der Referenzdefinition ausführen. Diese Definition befindet sich in der Blockauflistung der Zeichnung und kann mithilfe der *Name*-Eigenschaft abgerufen werden:

```

Set XRefBlock = Blocks(XRefObject.Name)
XRefObject.Path = FileName
XRefBlock.Reload

```

Auch die Änderung der *Name*-Eigenschaft wirkt sich auf den Inhalt der *Path*-Eigenschaft aus, was logisch ist, da der Name einen Bezug auf die Definition der externen Referenz in der Blockauflistung darstellt. Bevor Sie den Namen der externen Referenz ändern, sollten Sie allerdings kontrollieren, ob es sich bei dem angegebenen Block auch wirklich um eine externe Referenz handelt. Dazu müssen Sie nur die zugehörige Blockdefinition aus der Blockauflistung auslesen und die *IsXRef*-Eigenschaft prüfen.

```

Set XRefBlock = Blocks(BlkName)
If XRefBlock.IsXRef = True Then
    XRefObject.Name = BlkName
    XRefObject.Update
End If

```

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Color</i>	Legt die Farbe der externen Referenz fest oder gibt diese zurück.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle der externen Referenz zurück.
<i>HasAttributes</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie feststellen, ob die entsprechende externe Referenz Attributdefinitionen enthält. siehe auch Blockreferenz.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob der externen Referenz ein Erweiterungswörterbuch zugeordnet wurde.
<i>Hyperlinks</i> (schreibgeschützt)	Gibt die Auflistung der Hyperlinks der externen Referenz zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
<i>Layer</i>	Zeigt den Layer der externen Referenz an oder legt diesen fest.
<i>Linetype</i>	Zeigt den Linientyp der externen Referenz an oder legt diesen fest.
<i>LinetypeScale</i>	Zeigt den aktuellen Linientypfaktor der externen Referenz an oder legt diesen fest.
<i>Lineweight</i>	Zeigt die aktuellen Linienstärke der externen Referenz an oder legt diese fest.
<i>Normal</i>	Mit dieser Eigenschaft können Sie die X-, Y- und Z-Koordinaten des normalen Richtungsvektors abrufen oder festlegen.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> der externen Referenz zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse <i>AcadExternalReference</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.
<i>PlotStyleName</i>	Gibt den Plotstilnamen für die externe Referenz zurück oder legt diesen fest.
<i>Rotation</i>	Diese Eigenschaft bestimmt den aktuellen Drehwinkel der externen Referenz oder gibt diesen zurück. Der Wert des Drehwinkels ist dabei von der Systemvariablen <i>ANGBASE</i> abhängig. Siehe auch Blockreferenz.
<i>Visible</i>	Mit dieser Eigenschaft steuern Sie die Sichtbarkeit des Objekts.
<i>XScaleFactor</i>	Zeigt den aktuellen Wert des X-Skalierungsfaktors der externen Referenz an oder legt diesen fest. Siehe auch Blockreferenz.

Tabelle 8.99: Weitere Eigenschaften der externen Referenz

Eigenschaft	Beschreibung
<i>YscaleFactor</i>	Zeigt den aktuellen Wert des Y-Skalierungsfaktors der externen Referenz an oder legt diesen fest. Siehe auch Blockreferenz.
<i>ZscaleFactor</i>	Zeigt den aktuellen Wert des Z-Skalierungsfaktors der externen Referenz an oder legt diesen fest. Siehe auch Blockreferenz.

Tabelle 8.99: Weitere Eigenschaften der externen Referenz (Forts.)

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>Explode</i>	Diese Methode ist zwar in dieser Klasse vorhanden, wird aber nicht verwendet.
<i>GetAttributes</i>	Mit dieser Anweisung rufen Sie die Attribute der externen Referenz ab. Sie erhalten als Ergebnis ein Array, das alle Attributreferenzen auflistet. Siehe auch Blockreferenz.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetConstantAttributes</i>	Mit dieser Anweisung rufen Sie die konstanten Attribute der externen Referenz ab – also alle Attribute, in denen die Eigenschaft <i>Constant</i> auf <i>TRUE</i> gesetzt ist. Sie erhalten als Ergebnis ein Array, das alle Attribute auflistet. Im Gegensatz zu <i>GetAttributes</i> werden mit dieser Funktion nicht die Attributreferenzen zurückgegeben. Siehe auch Blockreferenz.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch der externen Referenz ab.
<i>GetXData</i>	Ruft die erweiterten Daten der externen Referenz ab.
<i>Highlight</i>	Hebt die externen Referenz hervor.
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt der externen Referenz mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt eine externe Referenz als Resultat einer Spiegelung mit dieser externen Referenz.
<i>Mirror3D</i>	Erstellt eine neue externe Referenz als Resultat einer räumlichen Spiegelung dieser externen Referenz.
<i>Move</i>	Verschiebt die externe Referenz von den Ursprungs- auf die Zielkoordinaten.

Tabelle 8.100: Methoden der externen Referenz

Prozedur	Beschreibung
<i>Rotate</i>	Dreht die externe Referenz um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht die externe Referenz um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert die externe Referenz um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten der externen Referenz.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf die externe Referenz an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung der externen Referenz im AutoCAD-Zeichnungsfenster.

Tabelle 8.100: Methoden der externen Referenz (Forts.)

Anwendungsbeispiel

(Beispielprojekt: 08\AcadExternalReference.dvb)

Im nachfolgenden Beispielprojekt werden wir den Benutzer eine externe Referenz in der Zeichnung wählen lassen und anschließend die Informationen, die die externe Referenz bietet, in einem Formular anzeigen.

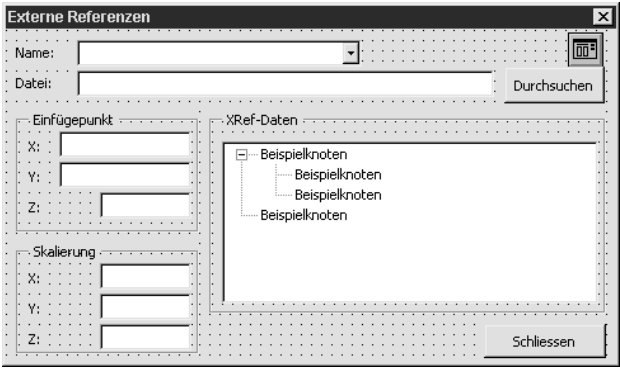


Abbildung 8.90: Formular des Beispielprojekts

Neben den Informationen, die das gewählte Objekt direkt anbietet, sollen auch die Objektdaten, wie Layer, Linientypen, Anzahl grafischer Objekte usw., die das Definitionsobjekt der Referenz enthält, in einer hierarchischen Auflistung (*TreeView*) angezeigt werden. Dazu ist es notwendig, dass dieses Definitionsobjekt mithilfe des Namens der externen Referenz aus der Blockauflistung ermittelt wird.

Startprozedur

In der Startprozedur des Projekts wird der Benutzer zunächst aufgefordert, eine externe Referenz in der Zeichnung auszuwählen. Hat er dies getan, wird der Verweis auf das Objekt an die Formularvariable *ActXRef* übergeben. Anschließend wird die Initialisierungsprozedur *InitXRefFrm* des Formulars gestartet, um danach das Formular einzublenden. Da das Formular gebunden angezeigt wird, wird der nachfolgende Code erst ausgeführt, wenn das Formular beendet wurde.


```

Public Sub editXRef()
Dim Object As Object
On Error Resume Next
Set Util = New MyUtility
Do
    Set Object = Nothing
    Util.GetEntity Object, PickedPoint, "externe Referenz wählen:"
    If TypeName(Object) = "IAcadExternalReference" Then
        Set XRefFrm.ActXRef = Object
        XRefFrm.InitXRefFrm
        XRefFrm.Show
    ElseIf TypeName(Object) = "Nothing" Then
        Exit Do
    End If
Loop
End Sub

```

Initialisierungsprozedur

Die Hauptarbeit des Projekts fällt auch hier wieder der Initialisierungsprozedur zu. In dieser Prozedur werden sowohl die Eigenschaften der gewählten externen Referenz an die entsprechenden Steuerelemente übergeben als auch das Definitionsobjekt der Referenz ermittelt und ausgewählte Daten dieses Objekts in dem *TreeView*-Steuerelement dargestellt.

```

Public Sub InitXRefFrm()
Dim xNode As MSComctlLib.Node
On Error Resume Next
NoChange = True

```

Die Ermittlung dieses Definitionsobjekts ist relativ einfach. Sie müssen nur den Namen der externen Referenz als Index der Blockauflistung angeben und den zurückgegebenen Verweis in der Objektvariablen *XRefBlock* speichern.

```
Set XRefBlock = ThisDrawing.Blocks(ActXRef.Name)
```

Im nächsten Schritt werden die Namen aller in der Blockauflistung vorhandenen Blockdefinitionen ermittelt, die als externe Referenz verwendet werden können.

```

ComboBox1.Clear
For Each Block In ThisDrawing.Blocks
    If Block.IsXRef = True Then
        ComboBox1.AddItem Block.Name
    End If
Next

```

Als Nächstes werden einige Eigenschaften der externen Referenz in den entsprechenden Steuerelementen dargestellt.

```

ComboBox1 = ActXRef.Name
TextBox2 = ActXRef.Path
InsPoint = ActXRef.InsertionPoint
TextBox3 = Util.RealToString(InsPoint(0))
TextBox4 = Util.RealToString(InsPoint(1))
TextBox5 = Util.RealToString(InsPoint(2))
TextBox6 = Util.RealToString(ActXRef.XScaleFactor)
TextBox7 = Util.RealToString(ActXRef.YScaleFactor)
TextBox8 = Util.RealToString(ActXRef.YScaleFactor)

```

Im letzten Teil der Prozedur werden noch die Informationen, die die ermittelte Referenzdefinition (*XRefBlock*) enthält, in dem *TreeView*-Steuerelement zusammengestellt. Dabei soll eine hierarchische Darstellung entstehen, in der jeder Zweig gewisse Informationen zusammenfasst.

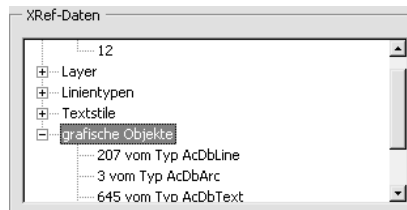


Abbildung 8.91: Informationen über die externe Referenz angezeigt im *TreeView*-Steuerelement

Als Erstes werden die einzelnen Blocknamen ausgelesen, die die externe Referenz enthält. Dabei werden wieder die anonymen Blöcke (beginnen mit *) ausgeschlossen.

```
TreeView1.Nodes.Clear
ThisDrawing.Utility.Prompt "Lese Blöcke ..." & vbCrLf
Set xNode = TreeView1.Nodes.Add(, , "Blöcke")
For Each Item In XRefBlock.XRefDatabase.Blocks
    If Left(Item.Name, 1) <> "*" Then
        TreeView1.Nodes.Add xNode, tvwChild, , Mid(Item.Name, _
            InStr(1, Item.Name, "|") + 1)
    End If
Next
```

Als Nächstes folgen die Layer, die in der Kategorie „Layer“ abgelegt werden.

```
ThisDrawing.Utility.Prompt "Lese Layer ..." & vbCrLf
Set xNode = TreeView1.Nodes.Add(, , "Layer")
For Each Item In XRefBlock.XRefDatabase.Layers
    If Left(Item.Name, 1) <> "*" Then
        TreeView1.Nodes.Add xNode, tvwChild, , Mid(Item.Name, _
            InStr(1, Item.Name, "|") + 1)
    End If
Next
```

Anschließend kommen noch die Linientypen und Textstile, wobei jeder seinen eigenen Zweig erhält.

```
ThisDrawing.Utility.Prompt "Lese Linientypen ..." & vbCrLf
Set xNode = TreeView1.Nodes.Add(, , "Linientypen")
For Each Item In XRefBlock.XRefDatabase.Linetypes
    If Left(Item.Name, 1) <> "*" Then
        TreeView1.Nodes.Add xNode, tvwChild, , Mid(Item.Name, _
            InStr(1, Item.Name, "|") + 1)
    End If
Next
ThisDrawing.Utility.Prompt "Lese Textstile ..." & vbCrLf
Set xNode = TreeView1.Nodes.Add(, , "Textstile")
```

```

For Each Item In XRefBlock.XRefDatabase.TextStyles
  If Left(Item.Name, 1) <> "*" Then
    TreeView1.Nodes.Add xNode, tvwChild, , Mid(Item.Name, _
      InStr(1, Item.Name, "|") + 1)
  End If
Next

```

Als Abschluss wird noch ein zusätzlicher Zweig in dem *TreeView*-Steuerelement erstellt. In diesem Zweig sollen die grafischen Objekte, die in der externen Referenz gespeichert sind, gezählt und für jeden Objekttyp ein eigener Eintrag erstellt werden.

```

ThisDrawing.Utility.Prompt "Lese grafische Objekte ..." & vbCrLf
Set xNode = TreeView1.Nodes.Add(, , "grafische Objekte")
x = 0
For Each Item In XRefBlock
  x = x + 1
  ThisDrawing.Utility.Prompt CStr(x) & " Objekte gelesen ..." & _
    vbCrLf

```

Um zu garantieren, dass jeder Objekttyp auch nur einmal angezeigt wird, verwenden wir den Objektnamen (*AcDbLine*, *AcDbText*, *AcDbArc* usw.) des jeweiligen Objekts als Schlüsselwert (*Key*-Parameter). Damit kann bei weiteren Durchläufen geprüft werden, ob bereits ein Eintrag des aktuellen Typnamens existiert. Ist dies nicht der Fall, dann wird der *TreeView*-Liste ein weiterer Eintrag hinzugefügt.

```

If TypeName(TreeView1.Nodes(Item.ObjectName)) = "Nothing" Then
  TreeView1.Nodes.Add xNode, tvwChild, Item.ObjectName, _
    "1 vom Typ " & Item.ObjectName
Else

```

Wurde der Eintrag gefunden, wird die aktuelle Anzahl des entsprechenden Objekttyps ermittelt, dieser Wert anschließend um eins erhöht und in die *Text*-Eigenschaft des Node-Objekts zurückgeschrieben:

```

  Count = CInt(Left(TreeView1.Nodes(Item.ObjectName).Text, _
    InStr(1, TreeView1.Nodes(Item.ObjectName).Text, " ")))
  Count = Count + 1
  TreeView1.Nodes(Item.ObjectName).Text = CStr(Count) & _
    " vom Typ " & Item.ObjectName
End If
Next
NoChange = False
End Sub

```

Änderung der *Path*-Eigenschaft

Die Änderung der *Path*-Eigenschaft erfolgt entweder durch die direkte Eingabe in dem Textfeld oder durch Auswahl einer Datei über den Standarddialog. Hat der Benutzer den Dateinamen direkt eingegeben, wird das Ereignis *BeforeUpdate* ausgelöst und es kann entsprechend reagiert werden. Mit der *Dir()*-Funktion prüfen wir zunächst, ob die Datei wirklich im System vorhanden ist. Konnte die Datei nicht gefunden werden, wird eine Fehlermeldung ausgegeben und der Vorgang durch Setzen des *Cancel*-Parameters auf *TRUE* abgebrochen. Andernfalls

wird der *Path*-Eigenschaft die neue Datei zugeordnet und mit der *Reload*-Anweisung, die auf das Definitionsobjekt der Referenz angewendet wird, werden die externen Referenzen, die auf diesem Definitionsobjekt aufbauen, aktualisiert.

```
Private Sub TextBox2_BeforeUpdate(ByVal Cancel As _
MSForms.ReturnBoolean)
On Error Resume Next
If Dir(TextBox2) = "" Then
    MsgBox "Zeichnungsdatei " & TextBox2 & _
        " konnte nicht gefunden werden!", vbCritical
    Cancel = True
Else
    ActXRef.Path = TextBox2
    XRefBlock.Reload
    TextBox2 = ActXRef.Path
    InitXRefFrm
End If
End Sub
```

Das Gleiche trifft auf die Auswahl der Datei über den Standarddialog zu. Nur dass hier nicht erst geprüft werden muss, ob die Datei vorhanden ist, da diese Aufgabe der Standarddialog übernimmt.

```
Private Sub CommandButton1_Click()
On Error Resume Next
ComDialog.FileName = TextBox2
Err.Clear
ComDialog.ShowOpen
If Err = 0 Then
    ActXRef.Path = ComDialog.FileName
    XRefBlock.Reload
    TextBox2 = ActXRef.Path
    InitXRefFrm
End If
End Sub
```

Änderung der Name-Eigenschaft

Bei der Änderung der *Name*-Eigenschaft ist es eigentlich nur wichtig, dass abschließend die Initialisierungsprozedur ausgeführt wird, damit der Inhalt des *TreeView*-Steuerelements aktualisiert wird, da sich durch Zuweisung eines neuen Namens auch die *Path*-Eigenschaft und damit auch die Datenquelle der externen Referenz ändert.

```
Private Sub ComboBox1_Click()
On Error Resume Next
If NoChange = True Then Exit Sub
ActXRef.Name = ComboBox1
ActXRef.Update
InitXRefFrm
End Sub
```

8.3.32 Volumenkörperobjekt (Acad3DSolid)

Eine Objektvariable, die mit dieser Klasse deklariert wurde, stellt jeweils einen einzelnen 3D-Volumenkörper dar. Sie erstellen einen Verweis auf einen Volumenkörper, indem Sie eine der folgenden Funktionen im Modellbereich, Papierbereich oder auf einen Block anwenden:

AddBox, *AddCone*, *AddCylinder*, *AddEllipticalCone*, *AddEllipticalCylinder*, *AddTorus*, *AddExtrudedSolid*, *AddExtrudedSolidAlongPath*, *AddRevolvedSolid*, *AddSphere*, *AddWedge*

Sie können auch ein vorhandenes Element über die *Item*-Funktion der entsprechenden Auflistung (*ModelSpace*, *PaperSpace*, *Block*, *SelectionSet*, *Group*) ermitteln..

Eigenschaften des Volumenkörpers

Die wichtigsten Eigenschaften dieses Objekts sind schreibgeschützt und dienen somit rein informativen Zwecken. Die *Centroid*-Eigenschaft ruft den Schwerpunkt eines Bereichs oder einer Masse des Volumenkörpers ab. Es werden die X- und Y-Koordinate als zweidimensionales Array (*Point(1)*) zurückgegeben.

```
Dim VarPoint As Variant
VarPoint = Act3dSolid.Centroid
```

Das Trägheitsmoment des Volumenkörpers können Sie mit der Eigenschaft *MomentOfInertia* ermitteln. Die *PrincipalMoments*-Eigenschaft enthält die Hauptträgheitsmomente und *ProductOfInertia* das Deviationsmoment des Volumenkörpers. Alle drei Eigenschaften geben ein 3D-Array zurück, das die X-, Y- und Z-Koordinate des jeweiligen Moments enthält.

Die Hauptrichtungen des Volumenkörpers sind in der Eigenschaft *PrincipalDirections* gespeichert. Diese Eigenschaft *RadiiOfGyration* gibt den Trägheitsradius des Volumenkörpers zurück.

Zusätzlich zu den eben genannten Eigenschaften, die Sie auch schon von der Klasse *AcadRegion* kennen, besitzt dieses Objekt noch die *Volume*-Eigenschaft, die das Volumen des Körpers zurückgibt.

Eigenschaften	Beschreibung
Application (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
Color	Legt die Farbe des Volumenkörpers fest oder gibt diese zurück.
Document (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
Handle (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Volumenkörpers zurück. Der Handle ist der eindeutige Identifier eines AutoCAD-Objekts.
HasExtensionDictionary (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Volumenkörper ein Erweiterungswörterbuch zugeordnet wurde.
Hyperlinks (schreibgeschützt)	Gibt die Auflistung der Hyperlinks des Volumenkörpers zurück und zeigt den Namen bzw. die Beschreibung des Hyperlinks an (falls angegeben).
Layer	Zeigt den Layer des Volumenkörpers an oder legt diesen fest.

Tabelle 8.101: Weitere Eigenschaften des Volumenkörpers

Eigenschaften	Beschreibung
Linetype	Zeigt den Linientyp des Volumenkörpers an oder legt diesen fest.
LinetypeScale	Zeigt den aktuellen Linientypfaktor des Volumenkörpers an oder legt diesen fest.
Lineweight	Zeigt die aktuelle Linienstärke für den Volumenkörper an oder legt diese fest.
ObjectID (schreibgeschützt)	Gibt die ObjectID des Volumenkörpers zurück.
ObjectName (schreibgeschützt)	Gibt den Namen der Klasse <i>Acad3DSolid</i> zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren.
OwnerId (schreibgeschützt)	Gibt die Eigentümer-ObjectID (übergeordnetes Objekt) zurück.
PlotStyleName	Gibt den Plotstilnamen für den Volumenkörper zurück oder legt diesen fest.
Visible	Steuert die Sichtbarkeit des Volumenkörpers.

Tabelle 8.101: Weitere Eigenschaften des Volumenkörpers (Forts.)

Methoden des Volumenkörpers

Diese *Boolean*-Anweisung führt eine boolesche Operation anhand eines anderen Volumenkörpers durch. Als Parameter muss die entsprechende Operation als Konstante sowie der Volumenkörper, mit dem diese Operation durchgeführt werden soll, der Funktion mitgegeben werden.

Konstante	Beschreibung	Wert
<i>acUnion</i>	Vereinigt die beiden Volumenkörper, indem der überlagernde Teil herausgeschnitten wird.	0
<i>acIntersection</i>	Verwenden Sie diese Konstante, um als Ergebnis den aufrufenden Körper so zu verändern, dass der durch beide Körper überlagerte Teil erhalten bleibt	1
<i>acSubtraction</i>	Sollten Sie diese Konstante verwenden, so bleibt als Körper nur noch der Teil erhalten, der durch den zweiten Körper nicht überlagert wurde.	2

Tabelle 8.102: Konstanten der Boolean-Anweisung

```
Dim FSolid As Acad3DSolid
Dim SSolid As Acad3DSolid
'Objektvariablen müssen hier initialisiert 'werden
FSolid.Boolean acUnion, SSolid
```

Mit der *CheckInterference*-Funktion können Sie eine Kollisionskontrolle des Volumenkörpers mit einem anderen Volumenkörper durchführen. Wenn Sie den Parameter *CreateInterferenceSolid* auf *TRUE* setzen, wird ein Volumenkörper der Kollision erzeugt und als Objektvariable zurückgegeben. Als ersten Parameter müssen Sie einen Verweis auf den zweiten Volumenkörper angeben.

```
Dim FSolid As Acad3DSolid
Dim SSolid As Acad3DSolid
Dim NSolid As Acad3DSolid
'Objektvariablen müssen hier initialisiert werden
FSolid.CheckInterference SSolid, False
```

oder

```
Set NSolid = FSolid.CheckInterference(SSolid, True)
```

Um einen Querschnitt des Volumenkörpers zu erstellen, müssen Sie die *SectionSolid*-Funktion ausführen. Als Parameter verlangt diese Funktion drei Arrays, die die Ebene definieren, durch die der Querschnitt verlaufen soll. Der entstandene Querschnitt wird als *Region*-Objekt (*AcadRegion*) zurückgegeben.

```
Point1 = Util.GetPoint(, "erster Punkt")
Point2 = Util.GetPoint(Point1, "zweiter Punkt")
Point3 = Util.GetPoint(Point1, "dritter Punkt")
Set NewRegion = FSolid.SectionSolid(Point1, Point2, Point3)
```

Mit der *SliceSolid*-Anweisung können Sie den Volumenkörper anhand einer Ebene, die durch drei Punkte definiert wird, kappen. Als Ergebnis erhalten Sie einen aus der Anweisung hervorgehenden Volumenkörper. Zusätzlich zur Ebene können Sie noch den Parameter *Negative* festlegen. Der Parameter bestimmt, welche Seite des Körpers gekappt werden soll. Mögliche Wert für diesen Parameter sind:

- TRUE – gibt den Volumenkörper auf der negativen Seite der Fläche zurück.
- FALSE – gibt den Volumenkörper auf der positiven Seite der Fläche zurück.

```
Point1 = Util.GetPoint(, "erster Punkt")
Point2 = Util.GetPoint(Point1, "zweiter Punkt")
Point3 = Util.GetPoint(Point1, "dritter Punkt")
Set NewSolid = FSolid.SliceSolid(Point1, Point2, Point3, True)
```

Prozedur	Beschreibung
<i>ArrayPolar</i>	Erstellt eine polare Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück.
<i>ArrayRectangular</i>	Erstellt eine rechtwinklige Anordnung des Objekts und gibt die erstellten Objekte in einem Array zurück
<i>Copy</i>	Erzeugt eine Kopie des Objekts und gibt die Kopie als Objektvariable zurück.
<i>Delete</i>	Löscht das Objekt aus der AutoCAD-Zeichnung.
<i>GetBoundingBox</i>	Berechnet die Extrempunkte eines Umgrenzungsrahmens des Objekts.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Volumenkörpers ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Volumenkörpers ab.
<i>Highlight</i>	Hebt den Volumenkörper hervor.

Tabelle 8.103: Methoden des Volumenkörpers

Prozedur	Beschreibung
<i>IntersectWith</i>	Ermittelt die Schnittpunkte, die sich aus dem Schnitt des Volumenkörpers mit einem anderen AutoCAD-Objekt ergeben.
<i>Mirror</i>	Erstellt einen Volumenkörper als Resultat einer Spiegelung.
<i>Mirror3D</i>	Erstellt einen Volumenkörper als Resultat einer räumlichen Spiegelung.
<i>Move</i>	Verschiebt den Volumenkörper von den Ursprungs- auf die Zielkoordinaten.
<i>Rotate</i>	Dreht den Volumenkörper um einen bestimmten Winkel.
<i>Rotate3D</i>	Dreht den Volumenkörper um eine räumlich definierte Achse.
<i>ScaleEntity</i>	Skaliert den Volumenkörper um einen bestimmten Faktor.
<i>SetXData</i>	Speichert die erweiterten Daten des Volumenkörpers.
<i>TransformBy</i>	Wendet die angegebene Transformationsmatrix auf den Volumenkörper an.
<i>Update</i>	Diese Funktion aktualisiert die Darstellung des Volumenkörpers im AutoCAD-Zeichnungsfenster.

Tabelle 8.103: Methoden des Volumenkörpers (Forts.)

Anwendungsbeispiele

(Beispielprojekt: 08\Acad3DSolid.dvb)

Abschließend folgen wieder wie gewohnt einige Beispiele zur Steuerung der Eigenschaften eines Volumenkörpers.

Im ersten Beispiel werden zunächst ein Quader und ein Zylinder mithilfe der Funktionen *AddBox* und *AddCylinder*, die auf den Modellbereich angewendet werden, erstellt. Hierzu werden die jeweiligen Parameter wie Länge, Breite, Höhe usw. festgelegt. Als Ergebnis dieser Funktionen wird jeweils ein Objekt vom Typ *Acad3DSolid* erstellt. Nachdem die Zeichnung in eine dreidimensionale Ansicht umgestellt wurde, wird der Quader mit dem Zylinder verschnitten.

```
Public Sub SolidBoolean()  
Dim FSolid As Acad3DSolid  
Dim sLength As Double, sWidth As Double, sHeight As Double  
Dim Center(2) As Double  
Dim cObj As Acad3DSolid  
Dim cCenter(2) As Double  
Dim cRadius As Double  
Dim cHeight As Double  
Dim NewDirection(2) As Double  
On Local Error Resume Next  
Center(0) = 5: Center(1) = 5: Center(2) = 0  
sLength = 10: sWidth = 7: sHeight = 10  
Set FSolid = ModelSpace.AddBox(Center, sLength, sWidth, sHeight)  
cCenter(0) = 0: cCenter(1) = 0: cCenter(2) = 0  
cRadius = 5  
cHeight = 20  
Set cObj = ModelSpace.AddCylinder(cCenter, cRadius, cHeight)  
NewDirection(0) = -1: NewDirection(1) = -1: NewDirection(2) = 1
```



```

ThisDrawing.ActiveViewport.Direction = NewDirection
ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
ThisDrawing.Application.ZoomAll
FSolid.Boolean acIntersection, cObj
ThisDrawing.Regen True
End Sub

```

Im zweiten Beispiel wird ebenfalls ein Quader erstellt. Wieder wird der Quader mit der Funktion *AddBox*, unter Verwendung von Länge, Breite und Höhe, erstellt und in die Objektvariable *FSolid* zurückgegeben. Anschließend werden zwei Achsen definiert, die die Schnittebene darstellen. Mithilfe der Funktion *SectionSolid* erstellen wir dann die Region, die sich den beiden definierten Achsen angleicht, und übergeben diese an die Objektvariable *FRegion*.

```

Public Sub SectionSolid()
Dim FSolid As Acad3DSolid
Dim sLength As Double, sWidth As Double, sHeight As Double
Dim Center(2) As Double
Dim sPnt1(2) As Double
Dim sPnt2(2) As Double
Dim sPnt3(2) As Double
Dim FRegion As AcadRegion
Dim NewDirection(2) As Double
On Local Error Resume Next
Center(0) = 5: Center(1) = 5: Center(2) = 0
sLength = 5: sWidth = 7: sHeight = 10
Set FSolid = ModelSpace.AddBox(Center, sLength, sWidth, sHeight)
sPnt1(0) = 1.5: sPnt1(1) = 7.5: sPnt1(2) = 0
sPnt2(0) = 1.5: sPnt2(1) = 7.5: sPnt2(2) = 10
sPnt3(0) = 8.5: sPnt3(1) = 2.5: sPnt3(2) = 10
Set FRegion = FSolid.SectionSolid(sPnt1, sPnt2, sPnt3)
NewDirection(0) = -1: NewDirection(1) = -1: NewDirection(2) = 1
ThisDrawing.ActiveViewport.Direction = NewDirection
ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
ThisDrawing.Application.ZoomAll
End Sub

```


9 Plotten/Drucken mit VBA in AutoCAD

Kommen wir nun zu dem Programmteil, der im großen und ganzen am Ende der Erstellung einer Zeichnung ausgeführt werden wird: die Ausgabe einer Zeichnung auf einen Plotter. Auch für diesen Vorgang stellt VBA ein Objekt zur Verfügung. Dabei unterteilt sich dieser Vorgang in zwei Bereiche: zum einen die Zusammenstellung der Plotkonfiguration, ausschlaggebend für die Qualität der Plotausgabe, und zum anderen der Plotvorgang selbst.

Kapitelübersicht

9.1 Ploteinstellungen

9.2 Zeichnungen plotten

9.3 Drucken mit VBA

9.1 Ploteinstellungen

Vor Beginn des eigentlichen Plotvorgangs, bedarf es natürlich noch einiger Konfigurationen. Diese betreffen z.B. die Festlegung des Plotbereichs oder den Plotter selbst. Normalerweise werden diese Einstellungen durch den Benutzer persönlich vorgenommen, so dass für den Entwickler die Verwaltung der Einstellungen bleibt. Damit man nicht dieselben Einstellungen immer wieder vornehmen muss, werden einzelne Plotkonfigurationen in einer Auflistung gespeichert, die dann nur noch dem jeweiligen Layout, das geplottet werden soll, zugewiesen werden. Zugriff auf dieses Auflistungsobjekt erhalten Sie über die *PlotConfigurations*-Eigenschaft des *Document*-Objekts.

```
Set PlotCfgs = ThisDrawing.PlotConfigurations
```

oder

```
Set PlotCfgs = Documents(n).PlotConfigurations
```

Die Verwaltung der Plotkonfigurationen läuft nun genauso ab, wie Sie es bereits von den anderen Auflistungen her kennen. Mit der *Add*-Anweisung fügen Sie der Auflistung ein neues Konfigurationsobjekt hinzu. Dieses enthält zunächst die Einstellungen, die im aktuellen Layout verwendet werden. Eine vorhandene Konfiguration kann mit der *Item*-Funktion abgerufen werden.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>Count</i> (schreibgeschützt)	Ermittelt die Anzahl der Objekte, die in der Auflistung enthalten sind.
<i>Document</i> (schreibgeschützt)	Mit dieser Eigenschaft können Sie das Dokument abrufen, dem dieses Objekt zugeordnet ist.
<i>Handle</i> (schreibgeschützt)	Diese Eigenschaft gibt den Handle des Objekts zurück.
<i>HasExtensionDictionary</i> (schreibgeschützt)	Diese Eigenschaft zeigt an, ob dem Objekt ein Erweiterungswörterbuch zugeordnet wurde.
<i>ObjectID</i> (schreibgeschützt)	Gibt die <i>ObjectID</i> des Objekts zurück.
<i>ObjectName</i> (schreibgeschützt)	Gibt den Namen der Klasse zurück. Sie können auch die Anweisung <i>TypeName()</i> verwenden, um den Typ der Klasse zu identifizieren. Eine Auflistung und Zuordnung aller Objektnamen finden Sie im Anhang.
<i>OwnerID</i> (schreibgeschützt)	Gibt die Eigentümer- <i>ObjectID</i> (übergeordnetes Objekt) zurück.

Tabelle 9.1: Eigenschaften der Konfigurationsauflistung

Prozedur	Beschreibung
<i>Add</i>	Fügt eine Ansichtsfensterkonfiguration der Auflistung hinzu.
<i>GetExtensionDictionary</i>	Ruft das erweiterte Wörterbuch des Objekts ab.
<i>GetXData</i>	Ruft die erweiterten Daten des Objekts ab.
<i>Item</i>	Ruft das Objekt an der angegebenen Position aus der Auflistung ab.
<i>SetXData</i>	Speichert die erweiterten Daten des Objekts.

Tabelle 9.2: Prozeduren der Konfigurationsauflistung

9.1.1 Einzelne Ploteinstellungen verarbeiten

Nachdem Sie nun ein einzelnes Konfigurationsobjekt aufgerufen haben, sei es durch Hinzufügen zur Konfigurationsauflistung oder durch Abrufen einer vorhandenen Konfiguration, können Sie durch Anpassung der Eigenschaften und Anwendung der Prozeduren die Einstellungen der einzelnen Konfigurationen ändern. Diese Eigenschaften und Prozeduren entsprechen in ihrer Bedeutung denen der *Layout*-Objekte. Deshalb kann ich mir an dieser Stelle umfangreiche Erklärungen zu den einzelnen Eigenschaften und Methoden sparen. Wenn Sie mehr darüber erfahren möchten, können Sie dies im *Kapitel 7.6.2* nachlesen.

9.2 Zeichnungen plotten

Der Plotvorgang an sich wird in VBA über das Objekt *Plot* des jeweiligen *Document*-Objekts gesteuert. Dieses Objekt besitzt hierfür spezielle Eigenschaften und Methoden.

Nachdem Sie alle Einstellungen am Layout vorgenommen haben, können Sie mit der Anweisung *DisplayPlotPreview* eine Vorschau von dem zu plottenden Layout erstellen. Dabei können Sie wie im AutoCAD-Plotmanager festlegen, welche Art von Vorschau angezeigt werden soll. Dieser Modus wird im Parameter der Anweisung mitgegeben. Um eine vollständige Vorschau zu erhalten, also eine, in der auch der Inhalt des Layouts zu sehen ist, müssen Sie folgende Anweisung ausführen.

```
ThisDrawing.Plot.DisplayPlotPreview acFullPreview
```

Für eine partielle Ansicht, in der nur der plotbare Bereich angezeigt wird, wäre dies

```
ThisDrawing.Plot.DisplayPlotPreview acPartialPreview
```

Um nun das Layout an ein Ausgabegerät zu senden, stehen Ihnen zwei Anweisungen zur Verfügung. Mit *PlotToDevice* senden Sie den Plot an das im Layout (*ConfigName*-Eigenschaft) festgelegte Ausgabegerät.

```
ThisDrawing.Plot.PlotToDevice
```

Sie könnten aber auch ein anderes Gerät beim Plotten festlegen, indem Sie im optionalen Parameter dieser Anweisung eine *pc3*-Konfigurationsdatei angeben.

```
ThisDrawing.Plot.PlotToDevice pc3FileName
```

Eine Liste mit allen verfügbaren *pc3*-Dateien können Sie mittels der *GetPlotDeviceNames*-Funktion ermitteln, die Sie entweder auf ein *Layout*- oder ein *PlotConfiguration*-Objekt anwenden müssen.

Die zweite Anweisung, mit der Sie die Ausgabe eines Plots auf ein Ausgabegerät aktivieren, ist die *PlotToFile*-Anweisung. Im Gegensatz zur *PlotToDevice*-Anweisung gibt diese allerdings den Plot nicht auf dem Gerät, sondern in die im ersten Parameter angegebene Datei aus. Im zweiten optionalen Parameter können Sie wieder eine *pc3*-Datei festlegen, wenn Sie mit einer anderen als der im Layout festgelegten Konfigurationsdatei plotten möchten.

Sollte der Plotvorgang erfolgreich ausgeführt worden sein, geben die Anweisungen *PlotToFile* und *PlotToDevice* beide den Wert *TRUE* zurück. Ist hingegen der Plot gescheitert, erhalten Sie den Wert *FALSE* als Ergebnis des Plotvorgangs.

Nun kann es aber auch häufig vorkommen, dass Sie nicht nur ein Layout, sondern gleichzeitig mehrere auf dem Ausgabegerät ausgeben wollen. Auch hierfür stellt Ihnen das *Plot*-Objekt einige Eigenschaften und Methoden zur Verfügung.

Als Erstes müssen Sie ein Array mit den zu plottenden Layouts zusammenstellen. Der Weg, den Sie hierbei beschreiten müssen, ist ein wenig gewöhnungsbedürftig. Als Erstes wird das Array wie gewohnt zusammengestellt, so weit ist auch alles normal. Anschließend müssen Sie dieses Array an eine Variable vom Datentyp *Variant* (unbedingt beachten) übergeben. Diese Variable wird dann als Parameter der *SetLayoutsToPlot*-Anweisung verwendet. Sie können nicht das zuvor zusammengestellte Array direkt verwenden, da dies einen Fehler auslösen würde.

```
Dim LayoutList As Variant, X As Integer
Dim LList() As String
X = 1
For Each Layout In ThisDrawing.Layouts
  If MsgBox("Layout " & Layout.name & " plotten?", vbYesNo + _
    vbQuestion) = vbYes Then
    X = X + 1
    ReDim Preserve LList(X)
    LList(X) = Layout.name
  End If
Next
LayoutList = LList
```

Anschließend wird die *StartBatchMode*-Anweisung unter Angabe der Anzahl der auszuführenden Plots aufgerufen. In unserem Fall entspricht dies der oberen Grenze des Arrays *LayoutList*.

```
Plot.StartBatchMode UBound(LayoutList)
```

Außerdem sollten Sie die *QuietErrorMode*-Eigenschaft auf den Wert *TRUE* setzen, damit eventuell beim Plotten auftretende Fehler nicht durch eine AutoCAD-Meldung angezeigt werden. Die Fehlermeldungen würden stattdessen in einer Log-Datei ausgegeben werden. Wenn Sie allerdings nur ein Layout plotten wollen, sollten Sie diese Eigenschaft wieder deaktivieren, damit der normale Anzeigemodus für Fehler (Anzeige von Fehlermeldungen) aktiviert wird.

```
Plot.QuietErrorMode = True
```

Nachdem alle Voreinstellungen abgeschlossen wurden, legen Sie die zu plottenden Layouts mit der Anweisung *SetLayoutsToPlot* fest. Im Parameter erwartet diese Anweisung ein Array mit den Namen der Layouts, die geplottet werden sollen: Anschließend wird dann der Ausgabebefehl für den Plot ausgeführt. Dies kann dann entweder *PlotToDevice* oder *PlotToFile* sein.

```
Plot.SetLayoutsToPlot LayoutList
Plot.PlotToFile "D:\Plot\"
```

Wenn Sie die Anweisung *PlotToFile* anwenden, legen Sie als Parameter der Anweisung das Verzeichnis fest, in dem die Plotdateien abgelegt werden sollen. Die Dateinamen erstellt der Stapelplot automatisch aus den Layoutnamen.

Eigenschaft	Beschreibung
<i>Application</i> (schreibgeschützt)	Gibt das Anwendungsobjekt (Klasse <i>AcadApplication</i>) zurück.
<i>BatchPlotProgress</i>	Ruft den aktuellen Status für das Stapelplotten ab oder beendet das Stapelplotten.
<i>NumberOfCopies</i>	Gibt die Anzahl der zu plottenden Exemplare an.
<i>QuietErrorMode</i>	Aktiviert den stillen Fehlermodus für Plot-Fehlermeldungen.

Tabelle 9.3: *Eigenschaften des Plot-Objekts*

Prozedur	Beschreibung
<i>DisplayPlotPreview</i>	Startet das Dialogfeld <i>PLOT-VORANSICHT</i> mit der partiellen oder vollständigen Voransicht.
<i>PlotToDevice</i>	Startet die Plotausgabe eines Layouts an ein Gerät.
<i>PlotToFile</i>	Startet die Plotausgabe eines Layouts in eine Datei.
<i>SetLayoutsToPlot</i>	Bestimmt das oder die zu plottenden Layouts.
<i>StartBatchMode</i>	Ruft den Stapelmodusdruck auf.

Tabelle 9.4: Prozeduren des Plot-Objekts

9.3 Drucken mit VBA

Eine Schnittstelle zur direkten Ausgabe auf einen Systemdrucker ist leider in VBA nicht integriert. Um nun trotzdem auf die Systemdrucker zugreifen zu können und einzelne Zeilen oder Dokumente auf einem dieser Druckern auszugeben, bedarf es eines Tricks. Da eine direkte Einbindung der VB-Laufzeitbibliothek VB6.OLB leider häufig zu Abstürzen führt, war es erforderlich, eine Laufzeitbibliothek in VB zu erstellen und über diese sowohl auf die *Printers*-Auflistung als auch auf das *Printer*-Objekt zuzugreifen. Dabei werden diese Objekte im Prinzip nur durch einen indirekten Zugriff angesprochen. Diese Laufzeitbibliothek finden Sie ebenfalls auf der Buch-CD. Um sie zu verwenden, müssen Sie nur das Installationsprogramm des VBAPrinter-Pakets ausführen. Die im Windows-Systemverzeichnis registrierte Bibliothek kann anschließend über EXTRAS → VERWEISE in das jeweilige VBA-Projekt eingebunden werden. Um das *Printer*-Objekt zu aktivieren, müssen Sie eine Objektvariable mit dem Typ *AcadVBAPrinter* deklarieren. Anschließend erstellen Sie eine neue Instanz der Klasse *AcadVBAPrinters*. Diese Klasse enthält nun zwei Eigenschaften. Die *VBAPrinter*-Eigenschaft gibt einen Verweis auf das *Printer*-Objekt (Standarddrucker) zurück und *VBAPrinters* einen Verweis auf die *Printers*-Auflistung (im System verfügbare Drucker).

```
Dim MyPrinter As VBAPrinter.AcadVBAPrinter
Set MyPrinter = New VBAPrinter.AcadVBAPrinter
```

Da die Klasse *Printer* nicht in VBA eingebunden ist, können Sie sich auch nicht die Eigenschaften dieser Objekte im Überwachungsfenster anzeigen lassen. Obwohl auch keine Liste mit den Informationen angezeigt wird, die die Anweisung an der aktuellen Einfügemarke logisch vervollständigen würden (VBAPrinter wird als Objekt deklariert), verfügt dieses Objekt dennoch über eine ganze Reihe von Eigenschaften und Methoden, die der Ausgabe von Daten auf den jeweiligen Drucker dienen.

Die Ausgabe auf das Gerät erfolgt mit der *Print*-Anweisung unter Angabe der auszugebenden Daten.

```
MyPrinter.Print "Ausgabe auf den Drucker"
```

Diese Datenzeile wird nun im Druckerspooler gelagert. Erst durch die *EndDoc*-Anweisung wird das *Printer*-Objekt angewiesen, den Druck tatsächlich auszuführen.

```
MyPrinter.EndDoc
```

Neben diesen beiden Funktionen, die im Prinzip das Grundgerüst der Ausgabe auf einen Drucker bilden, besitzt das *Printer*-Objekt noch einige Eigenschaften und Methoden, die der Anpassung der Ausgabe dienen. Dazu gehört beispielsweise die *Font*-Eigenschaft, mit der Sie den Schriftstil des Plotters festlegen können. Welche Schriftarten der jeweilige Drucker unterstützt, können Sie über die *Fonts*-Eigenschaft ermitteln. Die maximale Anzahl an verfügbaren Schriften ist in der *FontCount*-Eigenschaft gespeichert.

```
PrintFrm.ComboBox2.Clear
For i = 0 To MyPrinter.VBAPrinter.fontCount - 1
    PrintFrm.ComboBox2.AddItem MyPrinter.VBAPrinter.fonts(i)
Next
```

Die Druckqualität können Sie mit den Eigenschaften *ColorMode* und *PrintQuality* bestimmen. *ColorMode* legt fest, ob die Ausgabe in Graustufen oder in Farbe (falls durch den Drucker unterstützt) erfolgen soll.

Konstante	Beschreibung	Wert
<i>vbPRCMMonochrome</i>	Monochrome Druckausgabe (normalerweise Graustufen)	1
<i>vbPRCMColor</i>	Farbige Druckausgabe	2

Tabelle 9.5: Konstanten der *ColorMode*-Eigenschaft

Mit der *PrintQuality*-Eigenschaft bestimmen Sie die Auflösung des Drucks. Dazu müssen Sie der Eigenschaft nur den entsprechenden Wert zuweisen.

Konstante	Beschreibung	Wert
<i>vbPRPQDraft</i>	Entwurfsauflösung	-1
<i>vbPRPQLow</i>	Geringe Auflösung	-2
<i>vbPRPQMedium</i>	Mittlere Auflösung	-3
<i>vbPRPQHigh</i>	Hohe Auflösung	-4

Tabelle 9.6: Konstanten der *PrintQuality*-Eigenschaft

Die Position, an der der nächste Druckvorgang auf dem Papier ausgeführt werden soll, bestimmen die Eigenschaften *CurrentX* und *CurrentY*. Diese Eigenschaften sind besonders dann interessant, wenn Sie Grafiken (beispielsweise Kreise oder Linien) mit den Anweisungen *Circle*, *Line* und/oder *PSet* ausgeben wollen. *CurrentX* bestimmt dabei die horizontalen und *CurrentY* die vertikalen Koordinaten. Die Koordinaten werden von der linken oberen Ecke eines Objekts aus berechnet. Die Einstellung der *CurrentX*-Eigenschaft ist 0 am linken Rand eines Objekts und die Einstellung der *CurrentY*-Eigenschaft ist 0 am oberen Rand. Koordinaten werden in Twips oder in der durch die Eigenschaften *ScaleHeight*, *ScaleWidth*, *ScaleLeft*, *ScaleTop* und *ScaleMode* angegebenen Maßeinheit ausgedrückt.

Die Höhe und Breite des Papiers können Sie auf zweierlei Arten vorgeben. Zum einen gibt es die *PaperSize*-Eigenschaft, der Sie eine der Konstanten der Auflistung *vbPRPaperSize* zuweisen, um das entsprechende Papierformat einzustellen. Zum anderen können Sie die Höhe und Breite des Blatts auch direkt über die *Height*- und *Width*-Eigenschaften anpassen. In diesem Fall werden die Einstellungen der *PaperSize*-Eigenschaft ignoriert.

Mit der *Orientation*-Eigenschaft legen Sie fest, ob das Dokument in Hoch- oder Querformat gedruckt werden soll.

Konstante	Beschreibung	Wert
<i>vbPRORPortrait</i>	Hochformat	1
<i>vbPRORLandscape</i>	Querformat	2

Tabelle 9.7: Konstanten der *Orientation*-Eigenschaft

Die Anzahl der gedruckten Seiten können Sie mit der *Page*-Eigenschaft abrufen. Dabei zählt VB die Seiten beginnend mit eins hoch, wenn eine neue Seite begonnen werden muss, da der auszudruckende Text nicht mehr auf die aktuelle Seite passt oder weil Sie mittels der *NewPage*-Anweisung einen Seitenvorschub auslösen und somit eine neue Seite beginnen. Sobald eine *EndDoc*-Anweisung mit dem *Printer*-Objekt ausgelöst wurde, beginnt die Zählung der Seiten bei der nächsten *Print*-Anweisung wieder von vorn. Zusätzlich können Sie mit der *Duplex*-Eigenschaft bestimmen, ob das Papier ein- oder beidseitig bedruckt werden soll.

Konstante	Beschreibung	Wert
<i>VbPRDPSimplex</i>	Einseitiges Drucken mit der eingestellten Ausrichtung	1
<i>vbPRDPHorizontal</i>	Beidseitiges Drucken mit horizontalem Seitenwechsel	2
<i>vbPRDPVertical</i>	Beidseitiges Drucken mit vertikalem Seitenwechsel	3

Tabelle 9.8: Konstanten der *Duplex*-Eigenschaft

Wenn Sie den horizontalen Seitenwechsel eingestellt haben, befindet sich der obere Rand der beiden Seiten am selben Blattende. Bei vertikalem Seitenwechsel befindet sich der untere Rand einer Seite am selben Blattende wie der obere Rand der nächsten Seite.

Falls Sie Drucker mit mehreren Papierbehältern verwenden, können Sie über die *PaperBin*-Eigenschaft steuern, aus welchem Behälter der Drucker das Papier anfordern soll. Sie müssen hierfür nur eine der Konstanten der *vbPRPaperBin*-Auflistung der Eigenschaft zuweisen.

Eigenschaft	Beschreibung
<i>Copies</i>	Legt die Anzahl der zu druckenden Exemplare fest.
<i>DeviceName</i>	Gibt den Namen des Geräts zurück.
<i>DrawMode</i>	Legt das Erscheinungsbild der Ausgabe von Grafikmethoden oder eines Figur- oder Liniensteuerelements fest.
<i>DrawStyle</i>	Bestimmt den Liniestil für die Ausgabe von Grafikmethoden.
<i>DrawWidth</i>	Bestimmt die Linienbreite für die Ausgabe von Grafikmethoden.
<i>DriverName</i>	Gibt den Treibernamen des <i>Printer</i> -Objekts zurück.
<i>FillColor</i>	Bestimmt die Farbe, die zum Auffüllen von Figuren, Kreisen und Feldern verwendet wird.
<i>FillStyle</i>	Bestimmt den Füllstil einer Figur.
<i>ForeColor</i>	Bestimmt die Vordergrundfarbe, die zum Drucken von Text und Grafiken verwendet wird.

Tabelle 9.9: Weitere Eigenschaften des *Printer*-Objekts

Eigenschaft	Beschreibung
<i>HDC</i>	Gibt eine Zugriffsnummer für den Gerätekontext des Objekts zurück.
<i>Port</i>	Gibt den Anschlussnamen zurück, durch den das Dokument an den Drucker gesendet wird.
<i>RightToLeft</i>	Steuert die Darstellung des Textes auf einem bidirektionalen System.
<i>TrackDefault</i>	Bestimmt, ob das <i>Printer</i> -Objekt die Einstellung für den Standarddrucker in der Systemsteuerung verwendet.
<i>TwipsPerPixelX</i>	Gibt die Anzahl der Twips pro Pixel für ein horizontal gemessenes Objekt zurück.
<i>TwipsPerPixelY</i>	Gibt die Anzahl der Twips pro Pixel für ein vertikal gemessenes Objekt zurück.
<i>Zoom</i>	Bestimmt den Prozentsatz, mit dem die Druckausgabe aufwärts oder abwärts skaliert wird.

Tabelle 9.9: Weitere Eigenschaften des *Printer*-Objekts (Forts.)

Ausgabe grafischer Objekte

Neben dem Drucken von normalem Text stellt das *Printer*-Objekt auch Befehle zur Ausgabe von grafischen Elementen zur Verfügung. Die einfachste dieser Anweisungen ist die *PSet*-Funktion. Diese gibt einen Punkt an der angegebenen Position in der entsprechenden Farbe auf dem Papier aus. Im ersten Parameter der Anweisung legen Sie die Koordinaten der Ausgabeposition des Punkts fest.

```
MyPrinter.VBAPrinter.PSet (10, 10)
```

Geben Sie zusätzlich das Schlüsselwort *Step* an, beziehen sich die nachfolgenden Koordinatenangaben auf die letzte Ausgabeposition (*CurrentX*, *CurrentY*).

```
MyPrinter.VBAPrinter.PSet Step (10, 10)
```

Im zweiten Parameter der Anweisung können Sie noch einen Farbwert festlegen, mit dem der Punkt gedruckt werden soll. Lassen Sie diesen Parameter weg, so wird die Farbe der *ForeColor*-Eigenschaft verwendet. Zur Ausgabe eines roten Punkts an den Koordinaten 10 und 10 müsste folgende Codezeile ausgeführt werden:

```
MyPrinter.VBAPrinter.PSet (10, 10), RGB(255, 0, 0)
```

Zur Ausgabe einer Linie auf den Drucker können Sie die *Line*-Anweisung verwenden. Die Parameter der Anweisung haben eine ähnliche Wirkungsweise wie bei den Punkten, nur dass Sie hier neben dem Startpunkt der Linie auch deren Endpunkt angeben müssen.

```
MyPrinter.VBAPrinter.Line (10, 10)-(20, 20)
```

Auch hier gilt: Legen Sie für einen der beiden Punkte das Schlüsselwort *Step* fest, beziehen sich die angegebenen Koordinaten auf die letzte Ausgabeposition. So haben die folgenden Zeilen die gleiche Wirkung:

```
MyPrinter.VBAPrinter.Line (10, 10)-(20, 20)
```

und mit Schlüsselwort *Step*:

```
MyPrinter.VBAPrinter.Line (10, 10)-Step(10, 10)
```

Im zweiten Parameter können Sie wieder die Farbe festlegen, mit der die Linie auf dem Drucker ausgegeben werden soll. Lassen Sie diesen Parameter weg, wird wieder die Farbe der *ForeColor*-Eigenschaft verwendet. Im letzten Parameter der Anweisung können Sie noch einige Einstellungen zur Zeichnung eines Rechtecks vornehmen. Wenn Sie für diesen Parameter den Wert *B* festlegen, wird statt einer Linie ein Rechteck gezeichnet. Die beiden Koordinaten, die Sie zuvor festgelegt haben, bestimmen dann die gegenüberliegenden Eckpunkte des Rechtecks. Das Rechteck wird dabei mit der Farbe und dem Stil ausgefüllt, die in den Eigenschaften *FillColor* und *FillStyle* festgelegt wurden.

```
MyPrinter.VBAPrinter.Line (10, 10)-Step(10, 10), ,B
```

Geben Sie neben *B* zusätzlich noch den Wert *F* an, wird das Rechteck mit der Farbe gefüllt, die für den Rand des Rechtecks verwendet wird.

```
MyPrinter.VBAPrinter.Line (10, 10)-Step(10, 10), ,BF
```

Mit dem dritten grafischen Ausgabebefehl können Sie einen Kreis, einen Bogen oder eine Ellipse zeichnen. Die *Circle*-Anweisung stellt hierfür einige Parameter zur Verfügung, mit denen Sie das jeweilige Objekt ausdrucken können. Im ersten Parameter der Anweisung müssen Sie wie gehabt wieder die Koordinaten angeben, auf die der Mittelpunkt des Kreises, des Bogens oder der Ellipse positioniert werden soll. Die Angabe des Schlüsselworts *Step* aktiviert wieder den relativen Bezug zur letzten Ausgabeposition. Im zweiten Parameter müssen Sie den Radius angeben, mit dem der Kreis, der Bogen oder die Ellipse um den Mittelpunkt gezogen werden soll. Mit dem dritten Parameter kann wieder die Farbe des Objekts festgelegt werden. Lassen Sie diesen Parameter weg, kommt wieder die Farbe der *ForeColor*-Eigenschaft zum Einsatz. Ob nun ein Kreis, ein Bogen oder eine Ellipse gezeichnet wird, legen die letzten drei Parameter der *Circle*-Anweisung fest. Wenn Sie einen Kreis zeichnen wollen, können Sie diese einfach weglassen.

```
MyPrinter.VBAPrinter.Circle (10, 10), 5.5, RGB(255, 0, 0)
```

Wenn Sie hingegen einen Kreisbogen zeichnen möchten, müssen Sie im vierten und fünften Parameter der Anweisung den Start- und Endwinkel des Bogens in Radiant angeben (siehe auch Kapitel 7.3.5).

```
MyPrinter.VBAPrinter.Circle (10, 10), 5.5, RGB(255, 0, 0), 0, PI
```

Der letzte Parameter bestimmt das Verhältnis von Hauptradius zu Nebenradius einer Ellipse (siehe auch Kapitel 8.3.7). Ein Verhältnis von 1.0 würde demnach einen Kreis definieren. Alle Werte darüber oder darunter erzeugen bei der Ausgabe auf den Drucker eine Ellipse. Legen Sie zusätzlich die Parameter für Start- und Endwinkel fest, wird ein elliptischer Bogen gezeichnet.

Um einen Kreis, einen Bogen oder eine Ellipse auszufüllen, müssen Sie die Eigenschaften *FillColor* und *FillStyle* auf den entsprechenden Wert einstellen.

Die letzte Anweisung, mit der Sie ein grafisches Objekt auf dem Drucker ausgeben können, ist die *PaintPicture*-Anweisung. Diese gibt die im ersten Parameter festgelegte Grafik auf dem Drucker aus. Dies würde sich beispielsweise für die Ausgabe des Status der Layer anbieten. Sie könnten dann, statt der Bezeichnungen *Ein*, *Frieren* usw. die Symbole, die den jeweiligen Status anzeigen, auf den Drucker ausgeben.

Das Bild, das ausgedruckt werden soll, wird als Objekt vom Datentyp *IPictureDisp* im ersten Parameter der Anweisung festgelegt. Die Koordinaten der Position, wo das Bild ausgegeben werden soll, werden im zweiten und dritten Parameter festgelegt.

```
MyPrinter.VBAPrinter.PaintPicture Pict, 10, 10
```

Alle weiteren Parameter dieser Anweisung sind optional und steuern die Größe und die Ausrichtung des auf dem Drucker ausgegebenen Bilds. Die *ScaleMode*-Eigenschaft bestimmt dabei die verwendete Maßeinheit für die Skalierung des Bilds.

Parameter	Beschreibung
<i>Width1</i>	Bestimmt die Zielbreite des Bilds. Wenn die Zielbreite größer oder kleiner als die Quellbreite (<i>Width2</i>) ist, wird die Breite des Bilds entsprechend angepasst.
<i>Height1</i>	Bestimmt die Zielhöhe des Bilds. Wenn die Zielhöhe größer oder kleiner als die Quellhöhe (<i>Height2</i>) ist, wird die Höhe des Bilds entsprechend angepasst.
<i>X2</i>	Bestimmt die X-Koordinate eines Bildausschnitts des Bilds. Wird das Argument nicht angegeben, so wird 0 angenommen.
<i>Y2</i>	Bestimmt die Y-Koordinate eines Bildausschnitts des Bilds. Wird das Argument nicht angegeben, so wird 0 angenommen.
<i>Width2</i>	Bestimmt die Quellbreite eines Bildausschnitts. Wenn das Argument nicht angegeben wird, wird die gesamte Quellbreite des Bilds verwendet.
<i>Height2</i>	Bestimmt die Quellhöhe eines Bildausschnitts. Wenn das Argument nicht angegeben wird, wird die gesamte Quellhöhe des Bilds verwendet.
<i>OpCode</i>	Legt die bitweisen Operatoren fest, die auf das Bild angewendet werden. Die Konstanten, die Sie für dieses Argument verwenden können, sind in der <i>vbPROpCode</i> -Auflistung zusammengefasst.

Tabelle 9.10: Weitere Parameter der *PaintPicture*-Anweisung

Um ein Bild gedreht auszudrucken, müssen Sie bei der Angabe der Höhe und Breite negative Werte verwenden.

Steuerung der Ausgabe

Sobald Sie mit der *Print*-Anweisung oder einer der grafischen Ausgabeanweisungen einen Drucker angesprochen haben, können Sie die Ausgabe mit einigen Anweisungen des *Printer*-Objekts steuern. Die *EndDoc*-Anweisung müssen Sie wie bereits erwähnt immer dann ausführen, wenn Sie die Ausgabe auf den Drucker abschließen wollen. Führen Sie beispielsweise in einer Prozedur nur die *Print*-Anweisung aus und beenden danach das Programm, wird zwar der Text an die Warteschlange des Druckers geschickt, der Drucker wartet dann aber noch auf weitere Eingaben. Erst die *EndDoc*-Anweisung veranlasst die Warteschlange dazu, die Daten auf den Drucker auszugeben.

Die *NewPage*-Anweisung schließt die aktuelle Seite ab und löst einen Seitenvorschub aus. Mit Beginn der neuen Seite wird die aktuelle Druckposition auf die linke obere Ecke zurückgesetzt. Außerdem wird die *Page*-Eigenschaft um 1 erhöht.

Um den aktuellen Druckauftrag vorzeitig zu beenden, müssen Sie die *KillDoc*-Anweisung ausführen. Wenn der Druck-Manager des Betriebssystems den Druckauftrag verarbeitet (d.h. der Druck-Manager wurde gestartet und kann Druckaufträge im Hintergrund verarbeiten), löscht *KillDoc* den aktuellen Druckauftrag, so dass keine Daten an den Drucker gesendet werden. Verarbeitet der Druck-Manager hingegen keinen Druckauftrag (d.h. Druckaufträge können nicht im Hintergrund verarbeitet werden), können einige oder alle Daten an den Drucker gesendet werden, bevor *KillDoc* wirksam werden kann. In diesem Fall setzt der Druckertreiber den Drucker zum frühestmöglichen Zeitpunkt zurück und beendet den Druckauftrag.

Prozedur	Beschreibung
<i>Scale</i>	Definiert das Koordinatensystem für den Drucker.
<i>ScaleX</i>	Konvertiert den Wert für die Breite der Druckerausgabe von einer Maßeinheit in eine andere.
<i>ScaleY</i>	Konvertiert den Wert für die Höhe der Druckerausgabe von einer Maßeinheit in eine andere.
<i>TextHeight</i>	Gibt zurück, mit welcher Höhe die im Parameter angegebene Zeichenfolge in der aktuellen Schriftart ausgegeben würde.
<i>TextWidth</i>	Gibt zurück, mit welcher Breite die im Parameter angegebene Zeichenfolge in der aktuellen Schriftart ausgegeben würde.

Tabelle 9.11: Weitere Prozeduren des Printer-Objekts

Anwendungsbeispiel

(Beispielprojekt: PrintLayerState.dvb)

Im Anwendungsbeispiel zu diesem Thema werden wir uns mit dem zuvor schon angedeuteten Ausdrucken des Status der Layer einer Zeichnung beschäftigen. Ziel dieser Anwendung wird es sein, den Status (*Ein*, *Frieren*, *Sperren*, *Plotten*) und die Farbe der Layer der aktuellen Zeichnung mit Symbolen darzustellen und die restlichen Informationen als Text auszudrucken, so dass ein Druckbild wie in Abbildung 9.1 entsteht.

Name	Farbe	Linientyp	Plotsil	Linienstärke
0	          Schwarz/Weiss	CONTINUOUS	Color_7	Vorgabe
UNVERGRENZPUNKT	         Gelb	CONTINUOUS	Color_2	Vorgabe
VERGRENZPUNKT	         Gelb	CONTINUOUS	Color_2	Vorgabe
PP-AP	         Gelb	CONTINUOUS	Color_2	Vorgabe
LAMPE	         Gelb	CONTINUOUS	Color_2	Vorgabe
POLLER	         Gelb	CONTINUOUS	Color_2	Vorgabe
GULLY	         Gelb	CONTINUOUS	Color_2	Vorgabe
STEIGLEITER	         Gelb	CONTINUOUS	Color_2	Vorgabe
HYDRANTUNTERFLUR	         Gelb	CONTINUOUS	Color_2	Vorgabe
RETTUNGSRING	         Gelb	CONTINUOUS	Color_2	Vorgabe
STROMKASTEN	         Gelb	CONTINUOUS	Color_2	Vorgabe
WASSERSCHIEBER	         Gelb	CONTINUOUS	Color_2	Vorgabe

Abbildung 9.1: Ausdruck des Layer-Status

Die Symbole, die für den Ausdruck verwendet werden, sind in dem *ImageList*-Steuerelement des Formulars *PrintFrm* abgelegt. Außerdem kann der Benutzer in diesem Formular den Drucker, auf dem die Ausgabe erfolgen soll, und die Schriftart, in der die Texte gedruckt werden sollen, einstellen.

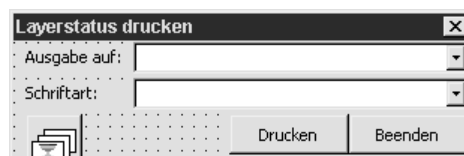


Abbildung 9.2: Formular des Beispielprojekts

Kommen wir nun zum Quellcode des Projekts. Im Klassenmodul *ThisDrawing* werden zunächst folgende Variablen deklariert:

```
Public MyPrinter As VBAPrinter.AcadVBAPrinter
Dim Util As MyUtility
```

Die Variable *MyPrinter* wird den Verweis auf das *Printer*-Objekt erhalten und *Util* wie gehabt eine Instanz der Klasse *MyUtility* aufnehmen.

Startprozedur

Die Startprozedur ist zugleich auch die Hauptprozedur des Programms, übernimmt also den Hauptteil der Arbeit.

```
Public Sub PrintLayerState()
Dim Pict(4) As IPictureDisp
Dim ActLayer As AcadLayer
On Error Resume Next
```

Nach der Deklaration einiger Variablen erstellen wir als Erstes zwei neue Instanzen, eine der Klasse *MyUtility* und eine der Klasse *AcadVBAPrinter* der Bibliothek *VBAPrinter*.

```
Set Util = New MyUtility
Set MyPrinter = New VBAPrinter.AcadVBAPrinter
```

Danach kommt erst einmal das Formular zum Einsatz, in dem der Benutzer den Drucker und die Schriftart auswählen kann. Dafür werden die verfügbaren Drucker des Systems ermittelt und der *ComboBox1* des Formulars hinzugefügt. Als ausgewählten Drucker lassen wir den Standarddrucker anzeigen, der ja in dem *Printer*-Objekt gespeichert ist. Die Schriftarten werden im Formular der *ComboBox2* hinzugefügt, da diese bei den unterschiedlichen Druckern variieren können. Das bedeutet, dass die Auswahlliste der Schriftarten aktualisiert wird, wenn der entsprechende Drucker ausgewählt wurde.

```
PrintFrm.ComboBox1.Clear
For i = 0 To MyPrinter.VBAPrinters.Count - 1
    PrintFrm.ComboBox1.AddItem MyPrinter.VBAPrinters(i).DeviceName
Next
PrintFrm.ComboBox1.Text = MyPrinter.VBAPrinter.DeviceName
PrintFrm.Show
If PrintFrm.OK = False Then Exit Sub
```

Hat der Benutzer im Formular die Schaltfläche *DRUCKEN* betätigt, wird der Code an dieser Stelle fortgesetzt, da die Formularvariable *OK* in diesem Fall den Wert *TRUE* erhalten hat. Als Nächstes folgen einige Einstellungen am *Printer*-Objekt. Die erste Aktion ist dabei die Änderung des Standarddruckers auf den durch den Benutzer im Formular ausgewählten Drucker. Da wir keinen direkten Zugriff auf die *Printer*-Objekte haben, muss dies das Objekt *MyPrinter* erledigen. Zu diesem Zweck enthält dieses die *SetNewStandardPrinter*-Anweisung. Sie müssen dieser Anweisung nur den Index des Druckers aus der *Printers*-Auflistung übergeben und der entsprechende Drucker wird dann eingestellt.

```
MyPrinter.SetNewStandardPrinter PrintFrm.ComboBox1.ListIndex
```

Jetzt folgen die Einstellungen der Papiereinheiten auf Millimeter:

```
MyPrinter.VBAPrinter.ScaleMode = vbMillimeters
```

der Schriftart und -höhe:

```
MyPrinter.VBAPrinter.Font.Name = PrintFrm.ComboBox2.Text
MyPrinter.VBAPrinter.Font.Size = 8
```

und der Ausgabequalität:

```
MyPrinter.VBAPrinter.PrintQuality = PrintFrm.Quality
```

Im Anschluss erfolgt die Zusammenstellung der einzelnen Zeilen und deren Ausdruck auf den eingestellten Drucker. Die Variable *acYPosition* speichert beim Durchlauf immer die aktuelle vertikale Position der Druckausgabe. Den Anfangswert dieser Variablen werden wir auf 20 einstellen, was einer Höhe von 20mm vom oberen Blattrand entspricht. Bevor der Durchlauf beginnt, wird zunächst die *PrintHead*-Anweisung ausgeführt. In dieser Prozedur, die weiter unten beschrieben ist, wird die Titelzeile des Blatts ausgedruckt.

```
acYPosition = 20
PrintHead
For Each ActLayer In Layers
```

Innerhalb der Schleife, die nacheinander alle verfügbaren Layer der Zeichnung abrufen, werden die einzelnen Zeilen mit den Werten des jeweiligen Layers ausgedruckt. Bei jedem Durchlauf wird zunächst geprüft, ob durch den letzten Ausdruck bereits das Blattende erreicht wurde. Ist dies der Fall, wird ein Seitenvorschub ausgelöst und erneut die Titelzeile mit der *PrintHead*-Anweisung ausgedruckt. Danach wird die Variable *acYPosition* wieder auf den Ausgangswert zurückgesetzt.

```
    If acYPosition > MyPrinter.VBAPrinter.scaleY(MyPrinter. _
        VBAPrinter.Height, vbTwips, vbMillimeters) - 20 Then
        MyPrinter.VBAPrinter.NewPage
        PrintHead
        acYPosition = 20
    End If
```

Bei der Zusammenstellung der auszudruckenden Zeile werden als Erstes die Symbole ermittelt, die den jeweiligen Status des Layers darstellen. Die Verweise auf diese Symbole werden im Array *Pict* gespeichert, wobei jedes Feld jeweils ein Symbol aufnehmen wird.

Layer ein- oder ausgeschaltet:

```
    Set Pict(0) = IIf(ActLayer.LayerOn = True, _
        PrintFrm.ImageList1.ListImages(1).Picture, _
        PrintFrm.ImageList1.ListImages(2).Picture)
```

Layer gefroren oder getaut:

```
    Set Pict(1) = IIf(ActLayer.Freeze = True, _
        PrintFrm.ImageList1.ListImages(4).Picture, _
        PrintFrm.ImageList1.ListImages(3).Picture)
```

Layer gesperrt oder entsperrt:

```
    Set Pict(2) = IIf(ActLayer.Lock = True, _
        PrintFrm.ImageList1.ListImages(6).Picture, _
        PrintFrm.ImageList1.ListImages(5).Picture)
```

Layer plotten oder fürs Plotten gesperrt:

```
Set Pict(3) = IIf(ActLayer.Plottable = True, _
    PrintFrm.ImageList1.ListImages(7).Picture, _
    PrintFrm.ImageList1.ListImages(8).Picture)
```

Anschließend wird dann die Zeile, die die Informationen zum jeweiligen Layer wiedergibt, ausgedruckt. Dazu setzen wir den jeweiligen Spaltenwert, der fest definiert ist, und die Zeile, die in der Variablen *acYPosition* gespeichert ist.

Ausdruck des Layer-Namens 1cm vom linken Rand:

```
MyPrinter.VBAPrinter.CurrentX = 10
MyPrinter.VBAPrinter.CurrentY = acYPosition
MyPrinter.VBAPrinter.Print ActLayer.Name
```

Ausdruck des Symbols für die *LayerOn*-Eigenschaft 5cm vom linken Rand:

```
MyPrinter.VBAPrinter.PaintPicture Pict(0), 50, acYPosition
```

Ausdruck des Symbols für die *Freeze*-Eigenschaft 5,5cm vom linken Rand:

```
MyPrinter.VBAPrinter.PaintPicture Pict(1), 55, acYPosition
```

Ausdruck des Symbols für die *Lock*-Eigenschaft 6cm vom linken Rand:

```
MyPrinter.VBAPrinter.PaintPicture Pict(2), 60, acYPosition
```

Ausdruck des Symbols für die *Plottable*-Eigenschaft 6,5cm vom linken Rand:

```
MyPrinter.VBAPrinter.PaintPicture Pict(3), 65, acYPosition
```

Ausdruck des Rechtecks für die *Farbe*-Eigenschaft 7cm vom linken Rand:

```
MyPrinter.VBAPrinter.fillcolor = MyPrinter.GetAcadColorValue( _
    ActLayer.Color)
MyPrinter.VBAPrinter.fillstyle = vbFSSolid
MyPrinter.VBAPrinter.Line (70, acYPosition + 1)-Step(3, 3), , B
```

und des Namens der Farbe 7,5cm vom linken Rand:

```
MyPrinter.VBAPrinter.CurrentY = acYPosition
MyPrinter.VBAPrinter.CurrentX = 75
MyPrinter.VBAPrinter.Print GetColorName(ActLayer.Color)
```

Ausdruck des Linientyps 11cm vom linken Rand:

```
MyPrinter.VBAPrinter.CurrentY = acYPosition
MyPrinter.VBAPrinter.CurrentX = 110
MyPrinter.VBAPrinter.Print ActLayer.Linetype
```

Ausdruck des Plotstilnamens 14cm vom linken Rand:

```
MyPrinter.VBAPrinter.CurrentY = acYPosition
MyPrinter.VBAPrinter.CurrentX = 140
MyPrinter.VBAPrinter.Print ActLayer.PlotStyleName
```


Und schließlich Ausdruck der Linienstärke 17cm vom linken Rand:

```
MyPrinter.VBAPrinter.CurrentY = acYPosition
MyPrinter.VBAPrinter.CurrentX = 170
MyPrinter.VBAPrinter.Print GetLineWeightText( _
    ActLayer.LineWeight)
```

Danach wird dann die nächste Zeile 4mm tiefer als die neue Zeilenposition gesetzt, damit ein entsprechender Zwischenraum zwischen den Symbolen entsteht. Anschließend beginnt der Ausdruck der Informationen des nächsten Layers.

```
acYPosition = acYPosition + 4
Next
```

Nachdem alle Zeilen an die Druckerwarteschlange geschickt wurden, brauchen wir nur noch den Ausdruck mit der *EndDoc*-Anweisung auszulösen.

```
MyPrinter.VBAPrinter.EndDoc
End Sub
```

Ausdruck der Titelzeile

Auf jeder neuen Seite soll beim Ausdruck eine Titelzeile erscheinen, die eine Überschrift der einzelnen Spalten darstellen soll. Diese Titelzeile wird mit der *PrintHead*-Prozedur ausgedruckt. Der Vorgang an sich ist nicht kompliziert. Wichtig hierbei ist nur die genaue Positionierung der Texte auf dem Blatt, so dass diese am Spaltenanfang der jeweiligen LayerEigenschaft erscheinen. Außerdem muss nach jeder *Print*-Anweisung die vertikale Position korrigiert werden, da durch *Print* ein Zeilenvorschub erfolgt.

```
Function PrintHead()
On Local Error Resume Next
With MyPrinter
    .VBAPrinter.CurrentX = 10
    .VBAPrinter.CurrentY = 10
    .VBAPrinter.Print "Name"
    .VBAPrinter.CurrentX = 70
    .VBAPrinter.CurrentY = 10
    .VBAPrinter.Print "Farbe"
    .VBAPrinter.CurrentX = 110
    .VBAPrinter.CurrentY = 10
    .VBAPrinter.Print "Linientyp"
    .VBAPrinter.CurrentX = 140
    .VBAPrinter.CurrentY = 10
    .VBAPrinter.Print "Plotstil"
    .VBAPrinter.CurrentX = 170
    .VBAPrinter.CurrentY = 10
    .VBAPrinter.Print "Linienstärke"
```

Abschließend wird noch eine durchgezogene Linie unterhalb der Titelzeile erzeugt, damit eine optische Trennung zwischen Titel und Layer-Ausdruck zu erkennen ist.

```
.VBAPrinter.Line (10, 15)-Step(190, 0)
End With
End Function
```

Die folgenden Funktionen werden innerhalb der Startprozedur zur Ermittlung der Farb- und Linienstärkebezeichnung verwendet.

```
Private Function GetColorName(Color As Integer) As String
On Local Error Resume Next
Select Case Color
    Case 1
        GetColorName = "Rot"
    Case 2
        GetColorName = "Gelb"
    Case 3
        GetColorName = "Grün"
    Case 4
        GetColorName = "Cyan"
    Case 5
        GetColorName = "Blau"
    Case 6
        GetColorName = "Magenta"
    Case 7
        GetColorName = "Schwarz/Weiss"
    Case Else
        GetColorName = "Farbe " & CStr(Color)
End Select
End Function

Private Function GetLineWeightText(LineWeight As Integer) As _
String
On Local Error Resume Next
Select Case LineWeight
    Case -3
        GetLineWeightText = "Vorgabe"
    Case Else
        GetLineWeightText = Util.RealToString(LineWeight / 100) & _
            " mm"
End Select
End Function
```

A Anhang

Im Folgenden habe ich noch einige Informationen zusammengetragen, die für die Entwicklung von AutoCAD-Anwendungen noch recht interessant erscheinen. Neben einem kurzen Überblick über die Fehlerbehandlung, erhalten Sie einen Einblick in die wichtigste Neuerung der VBA-Schnittstelle in AutoCAD 2004 und einige Informationen zu interessanten Zusatzprogrammen, die Ihnen die Arbeit mit AutoCAD erleichtern werden.

A.1 Fehlerbehandlung

Neben der Fehlersuche innerhalb des Programmcodes mithilfe des Überwachungsfensters, der Haltepunkte und des Debug-Modus kommt es vor, dass Fehler auftreten, die Sie während der Programmausführung abfangen und auswerten müssen. Angenommen, Sie lassen den Benutzer eine Datei mittels des Standarddialogs auswählen, in die dann Daten geschrieben werden sollen. Hat der Benutzer nun eine Datei ausgewählt, die auf einem schreibgeschützten Datenträger (z.B. CD oder schreibgeschützte Diskette) erzeugt werden soll, löst VBA einen Fehler aus. Damit das Programm nun nicht in den Debug-Modus wechselt oder gar beendet wird, müssen Sie diesen Fehler abfangen, auswerten und dem Benutzer eine entsprechende Alternative anbieten. Dies ist nur ein Beispiel für das Auftreten von Fehlern während der Programmausführung. Wenn Sie Code zur Fehlerbehandlung für Ihre Anwendungen entwickeln, werden Sie schnell feststellen, dass Sie immer wieder die gleichen Fehler behandeln. Durch sorgfältige Planung können Sie den Umfang des Codes reduzieren, indem Sie einige Prozeduren schreiben, die Ihr Fehlerbehandlungscode aufrufen kann, um häufig vorkommende Fehlersituationen zu verarbeiten.

A.1.1 Laufzeitfehler abfangen

Bevor überhaupt ein Fehler in einer Prozedur auftritt, muss diese entsprechende Anweisungen darüber erhalten, wie auf das Auftreten eines Fehlers reagiert werden soll. Eine solche Anweisung wird mit *On Error* bzw. *On Local Error* eingeleitet. Als Nächstes müssen Sie in dieser Zeile festlegen, was geschehen soll, wenn ein Fehler auftritt. Mit *Resume Next* wird die Ausführung des Codes an der Zeile fortgesetzt, die der Zeile folgt, die den Fehler ausgelöst hat.

`On Error Resume Next`

Für sich genommen kann diese Anweisung zu ungewollten Ergebnissen bei der Ausführung des Programms führen, besonders dann, wenn Sie *Do...Loop*-Schleifen verwenden und diese aufgrund des Fehlers nicht abgeschlossen werden können. Deshalb sollte diese Anweisung nicht allein verwendet werden, sondern immer in Zusammenhang mit der Auswertung des *Err*-Objekts an noralgischen Punkten innerhalb der Prozedur.

Neben der Syntax *Resume Next* können Sie aber auch eine genaue Sprungmarke festlegen, an der die Fehlerbehandlungsroutine der Prozedur geschrieben wurde. *On Error* wird durch die Schlüsselwörter *GoTo* oder *GoSub* angewiesen, an die folgende Marke zu verzweigen, egal wo innerhalb der Prozedur der Fehler aufgetreten ist.

```
On Error GoTo ErrorHandler
```

Die angesprochene Fehlerbehandlungsroutine ist keine *Sub*- oder *Function*-Prozedur, sondern ein durch eine Zeilenmarke oder Zeilennummer gekennzeichneten *Code*-Bereich. Diese Zeilenmarke wird durch den Namen gefolgt durch einen Doppelpunkt in der Prozedur gekennzeichnet und sollte immer am Ende der Prozedur positioniert werden. Außerdem müssen Sie die Prozedur vor der Zeilenmarke verlassen, damit diese nicht am Ende zusätzlich ausgeführt wird. Dementsprechend ergibt sich folgende Struktur innerhalb einer Prozedur:

```
Sub InitialisiereMatrix()
    On Error GoTo ErrorHandler
    . . .
    Exit Sub
ErrorHandler:
    'Fehlerbehandlung
    Resume Next
End Sub
```

Die *Resume Next*-Anweisung am Ende der Fehlerbehandlung legt fest, dass die Ausführung unmittelbar an der Zeile fortgesetzt wird, die der fehlerverursachenden Programmzeile folgt. Wenn Sie nur die *Resume*-Anweisung verwenden, setzt das Programm die Ausführung mit der Zeile fort, die den Fehler ausgelöst hat. Sie können aber auch eine Zeilenmarke angeben, zu der verzweigt werden soll.

A.1.2 Das Err-Objekt

Das *Err*-Objekt ist das wohl wichtigste Werkzeug bei der Analyse von Fehlern. Sollte ein Laufzeitfehler während des Programmlaufs auftreten, werden die Eigenschaften des *Err*-Objekts mit Informationen gefüllt, die sowohl die Fehlernummer als auch einige zusätzliche Informationen zum Fehler enthalten. Die Standardeigenschaft des Objekts ist *Number*, welche die Nummer des Laufzeitfehlers speichert. Eine Beschreibung des Fehlers können Sie über die *Description*-Eigenschaft abrufen. In den meisten Fällen reicht es aus, den Inhalt dieser Eigenschaft in einem Meldungsfeld anzuzeigen, um den Benutzer über die Art des aufgetretenen Fehlers zu informieren.

```
MsgBox Err.Description
```

Die *Source*-Eigenschaft gibt eine Zeichenfolge zurück, die das Objekt oder die Prozedur darstellt, in der der Fehler ursprünglich ausgelöst wurde. Wurde der Fehler durch ein Objekt ausgelöst, gibt die Zeichenfolge normalerweise den Namen der Klasse des Objekts oder die Ressource-ID wieder. Wurde der Code durch eine Aktion in einer Prozedur ausgelöst, enthält *Source* den Namen dieser Prozedur. Bei Klassenmodulen sollte *Source* einen Namen in der Form *Projekt.Klasse* enthalten. Wenn ein unerwarteter Fehler in Ihrem Code auftritt, wird die *Source*-Eigenschaft automatisch gesetzt.

Die *HelpFile*-Eigenschaft enthält den Pfad und den Namen der Hilfedatei, in der der Fehler beschrieben ist. Die *HelpContext*-Eigenschaft gibt das Hilfethema wieder. Sind beide Eigenschaften belegt, wird das Hilfethema der Hilfedatei aufgerufen, wenn der Benutzer im Dialogfeld mit der Fehlermeldung die Schaltfläche HILFE wählt.

Mit der *Clear*-Anweisung können Sie den Inhalt des *Err*-Objekts zurücksetzen. Die *Clear*-Methode wird automatisch aufgerufen, sobald eine der folgenden Anweisungen ausgeführt wird:

- Alle Arten von *Resume*-Anweisungen
- *Exit Sub*, *Exit Function*, *Exit Property*
- Jede *On Error*-Anweisung

Das *Err*-Objekt ist ein integriertes Objekt mit globaler Gültigkeit, was bedeutet, dass Sie keine Instanz dieses Objekts in Ihrem Code erstellen müssen.

A.1.3 Auswertung von Laufzeitfehlern

Für die Auswertung von Laufzeitfehlern bieten sich generell zwei Möglichkeiten an. Entweder Sie verarbeiten den Fehler in einer Fehlerbehandlungsroutine oder Sie werten das *Err*-Objekt an ausgesuchten Punkten in den Prozeduren aus. In beiden Fällen bietet sich die *MsgBox*-Funktion zur Kommunikation mit dem Benutzer an, damit dieser eine Entscheidung treffen kann, wie in den einzelnen Fällen weiter zu verfahren ist. Dafür bietet die Funktion verschiedene Auswahl Schaltflächen, die je nach Fehlercode zur Auswahl angeboten werden können. Nehmen wir das Beispiel mit dem Schreiben von Dateien zu Beginn dieses Kapitels. Da beim Zugriff auf Dateien ein ausgelöster Fehler verschiedene Ursachen haben kann, sollte dieser Fehler zunächst in einer *Select...Case*-Anweisung separiert und je nach Nummer eine entsprechende Benutzerauswahl angezeigt werden.

```
On Error GoTo ErrorHandler
Open FileName For Input As #1
Close
Exit Sub
ErrorHandler:
Select Case Err.Number
    Case 53, 55
        '53 - Datei nicht gefunden
        '55 - Datei bereits geöffnet
        MsgBox Err.Description, vbCritical
        Exit Sub
    Case 68, 70, 71, 75
        '68 - Gerät nicht verfügbar
        '70 - Zugriff verweigert
        '71 - Datenträger nicht bereit
        '75 - Fehler beim Zugriff auf Pfad/Datei
        Select Case MsgBox(Err.Description, vbAbortRetryIgnore)
            Case vbAbort
                Exit Sub
            Case vbRetry
                Resume
            Case vbIgnore
                Resume Next
        End Select
    End Select
End Select
```

A.2 Zugriff auf AutoCAD mit VB6

Als Alternative zu VBA bietet sich in der Regel die Entwicklungsumgebung VB an. Die Interaktion mit AutoCAD basiert dabei auf derselben Grundlage, der AutoCAD-Objektbibliothek. Nachdem Sie ein neues Projekt in VB erzeugt haben, müssen Sie diese Bibliothek in das Projekt einbinden. Rufen Sie hierfür den Verweisdiallog über PROJEKT→VERWEISE auf und selektieren Sie die Bibliothek „AutoCAD2000x-Typenbibliothek“.

Als Nächstes muss eine Verbindung mit AutoCAD hergestellt werden. Dies kann bereits beim Start des Projekts oder als Reaktion auf ein Ereignis beispielsweise über eine Schaltfläche erfolgen. Egal, wann und wo Sie die Verbindung zu AutoCAD herstellen wollen, in jedem Fall stehen Ihnen hierfür zwei Varianten zur Verfügung.

Neue AutoCAD-Sitzung starten

Die normale Vorgehensweise zur Herstellung einer Verbindung zu AutoCAD ist, die Anwendung mit der *CreateObjekt()*-Funktion zu starten und den zurückgegebenen Verweis auf die Sitzung in einer Objektvariablen zu speichern. Als Klassennamen legen Sie im Parameter der Funktion die Zeichenfolge „Autocad.Application“ fest. Dieser Klassenname entspricht der Klasse *AcadApplication* der AutoCAD-Typenbibliothek, womit auch die Funktion des in der Objektvariablen gespeicherten Verweises klar sein dürfte.

```
Set AcadApp = CreateObject("Autocad.Application")
```

Statt der *CreateObjekt()*-Funktion können Sie auch das Schlüsselwort *New* verwenden, um eine AutoCAD-Sitzung zu starten. Das Ergebnis, das Sie erhalten, ist in beiden Fällen dasselbe.

```
Set AcadApp = New Autocad.AcadApplication
```

Nachdem AutoCAD gestartet wurde, können Sie über die Objektvariable, in der Sie den Verweis auf die AutoCAD-Sitzung gespeichert haben, die gleichen Aktionen ausführen, wie wenn Sie in AutoCAD-VBA agieren würden.

Verbindung zu vorhandener AutoCAD-Sitzung aufnehmen

Eine zweite Methode, eine Verbindung zu AutoCAD herzustellen, ist der Zugriff auf eine bestehende AutoCAD-Sitzung mit der *GetObjekt()*-Funktion. Auch hier geben Sie im Parameter der Funktion die Zeichenfolge „Autocad.Application“ an, was die Funktion anweist, eine Verbindung mit dieser Klasse aufzunehmen. Wichtig hierbei ist natürlich, dass AutoCAD vor Ausführung der Funktion bereits gestartet wurde.

```
Set AcadApp = GetObject(, "Autocad.Application")
```

Egal, wie Sie den Zugriff auf eine AutoCAD-Sitzung initialisieren, alles, was danach folgt, ist bei beiden Methoden gleich.

Verbindung verarbeiten

Um ein wenig Struktur in den Programmcode zu bringen, sollten Sie diesen ähnlich aufbauen, wie es auch in der VBA-Umgebung der Fall ist. Dafür sollte zunächst eine globale Objektvariable deklariert werden, die einen Verweis auf das aktive Dokument erhält. Logischerweise wird diese dann den Namen *ThisDrawing* erhalten und im Modul deklariert werden, damit sie global verfügbar ist.

```
Public ThisDrawing As AcadDocument
Sub Main
Set AcadApp = GetObject(, "Autocad.Application")
Set ThisDrawing = AcadApp.ActiveDocument
End Sub
```

ThisDrawing besitzt nun alle Eigenschaften und Methoden eines AutoCAD-Dokuments und enthält zudem auch immer einen Verweis auf die aktuelle Zeichnung der AutoCAD-Sitzung.

Wenn Sie neben den Eigenschaften und Methoden auch auf die Ereignisprozeduren zugreifen wollen, müssen Sie die Variable in einem Klassenmodul unter Verwendung des Schlüsselworts *WithEvents* deklarieren. Daraufhin wird das Objekt dem Klassenmodul hinzugefügt und Sie können die Ereignisprozeduren der Dokumentklasse verwenden.

A.2.1 Das Zusatzprogramm AutoCAD Dockable ActiveX Control Container (acctont)

Dieses Zusatzprogramm wurde entwickelt, um ActiveX-Steuerelemente in einem dockbaren Fenster innerhalb einer AutoCAD-Sitzung einzubinden. Hauptbestandteil dieser Anwendung sind die Dateien *acctont.arx* und *acctont.tlb*. Mit dem Programm *acctont* binden Sie ein beliebiges ActiveX-Steuerelement in die AutoCAD-Sitzung – ähnlich dem Eigenschaftsfenster oder dem DesignCenter – in AutoCAD ein. Dazu müssen Sie zunächst das Programm in AutoCAD laden und dann in der AutoCAD-Befehlszeile den Befehl *dc* ausführen. Das Programm fordert Sie nun auf, die Programm-ID eines Steuerelements einzugeben. Hier müssen Sie nun den Namen Ihres Steuerelements angeben, so wie es in der Registrierung angemeldet wurde. Nehmen wir einmal an, Sie haben ein Steuerelementprojekt in VB erstellt, das mit *MyUserControls* bezeichnet wurde. In diesem Projekt sind nun zwei Steuerelemente enthalten. Das erste wurde beispielsweise mit *UserCtrl1* bezeichnet und das zweite mit *UserCtrl2*. Um nun das erste Steuerelement mit *acctont* in AutoCAD einzubinden, müssen Sie in der Befehlszeile *MyUserControls.UserCtrl1* eingeben und die Eingabe mit Return bestätigen. (Das Steuerelement muss natürlich in Windows registriert sein, was bereits durch Erstellung einer *ocx*-Datei durch VB erledigt wird.) Das zweite Steuerelement würde demnach durch Eingabe der Zeichenfolge *MyUserControls.UserCtrl2* gestartet.

Die Bibliothek *acctont.tlb* wird in das VB-Benutzersteuerelement-Projekt eingebunden und stellt die Verbindung zum Formular her, in dem das Steuerelement später dargestellt wird. Mithilfe der Eigenschaften und Methoden, die die Klassen dieser Bibliothek enthalten, können Sie später die Position oder das Verhalten des Formulars über den Code des Steuerelements beeinflussen.

Die Einbindung erfolgt nach folgender Vorgehensweise. Zunächst müssen Sie natürlich ein Benutzersteuerelement-Projekt in VB erstellen bzw. ein vorhandenes laden. Anschließend binden Sie die Bibliothek *acctont.tlb* über den Verweisdialog (EXTRAS → VERWEISE) in das Projekt ein. Danach müssen Sie dem Quellcode des Steuerelements folgende Zeilen hinzufügen:

```
Dim dContainer As AcadDockableContainer
Implements IRetrieveDockableContainer
```

Da mit der Anweisung *Implements* die Klasse *IRetrieveDockableContainer* in das Benutzersteuerelement eingebunden wurde, müssen Sie dann auch die Prozedur *SetDockContainer* implementieren. Diese Prozedur wird durch die Anwendung *acctont.arx* ausgeführt, wenn das

Steuerelement akzeptiert wurde, aber noch bevor das Formular, in dem eine Instanz des Steuerelements erstellt wird, angezeigt wird. Durch diese Prozedur wird ein Verweis mitgeliefert, mit dem Sie dann auf die Eigenschaften und Methoden des Containers zugreifen können. Sie können diesen Verweis nun in der Variablen *dContainer* speichern, um auch später noch auf den Container zugreifen zu können.

```
Private Sub IRetrieveDockableContainer_SetDockContainer(ByVal
newVal As AC_CONT.IAcadDockableContainer)
Set dContainer = newVal
End Sub
```

Haben Sie nun Verbindung zum Container aufgenommen, können Sie mit der Eigenschaft *Caption* den Dialogtitel des Containers abrufen oder anpassen. Das Setzen dieser Eigenschaft hat allerdings einen Nebeneffekt, es wird auch der Titel der AutoCAD-Sitzung in den Wert der *Caption*-Eigenschaft umbenannt.

Mit der *AllowDocking*-Eigenschaft legen Sie fest, ob der Container angedockt werden kann (*TRUE*) oder nicht (*FALSE*). Die Information darüber, ob der Container derzeit angedockt ist oder nicht, liefert die Eigenschaft *IsFloating*. Gibt diese den Wert *TRUE* zurück, ist das Containerfenster verschiebbar, andernfalls ist es an das AutoCAD-Sitzungsfenster angedockt.

An welchem Rand des AutoCAD-Fensters der Container angedockt werden kann oder derzeit angedockt ist, bestimmt *EnableDockPositions*. Die Konstanten, die mit dieser Eigenschaft verarbeitet werden, können Sie der Tabelle A.1 entnehmen. Die gleichen Konstanten verwenden Sie ebenfalls mit der *SetPreferredDockPosition*-Anweisung. Der erste Parameter dieser Anweisung setzt den Docking-Modus anhand der Konstanten. Im zweiten Parameter können Sie die Größe und Position des Containers festlegen.

```
Dim rect(0 To 3) As Integer
rect(0) = 50      ' left
rect(1) = 50      ' top
rect(2) = 300     ' right
rect(3) = 300     ' bottom
dContainer.SetPreferredDockPosition acFloating, rect
```

Konstante	Beschreibung	Wert
<i>acFloating</i>	Der Dialogcontainer wird nicht gedockt angezeigt (nur Methode <i>SetPreferredDockPosition</i>).	0xFFFFFFFF
<i>acDockAny</i>	Das Fenster kann an jedem Rand des AutoCAD-Fensters angedockt werden.	0x0000F000
<i>acDockLeft</i>	Das Fenster kann am linken Rand angedockt werden.	0x00001000
<i>acDockTop</i>	Das Fenster kann am oberen Rand angedockt werden.	0x00002000
<i>acDockRight</i>	Das Fenster kann am rechten Rand angedockt werden.	0x00004000
<i>acDockBottom</i>	Das Fenster kann am unteren Rand angedockt werden.	0x00008000
<i>Null</i>	Das Fenster ist nicht gedockt.	Null

Tabelle 1.1: Docking-Konstanten

AutoCAD Dockable ActiveX Control Container (account)

Jorge Lopez
ObjectDBX Development
ADF / Engineering
Autodesk, Inc.
E-mail: jorge.lopez@autodesk.com

A.3 Die Klasse MyUtility

Kommen wir nun zu der Klasse, die in fast jedem Beispielprojekt des Buchs verwendet wurde. Wie Sie bereits bemerkt haben dürften, enthält diese Klasse unter anderem die Funktionen des *Utility*-Objekts der Klasse *AcadDocument*. Diese Funktionen wurden allerdings um einige zusätzliche Funktionalitäten erweitert, so dass Sie nun beispielsweise bei der Verwendung der Funktionen zur Umwandlung von Zahlenwerten (*RealToString*, *DistanceToReal*, *AngleToString* und *AngleToReal*) nicht mehr unbedingt die Einheiten und Kommastellen mit angeben müssen. Lassen Sie diese Parameter weg, verwendet die entsprechende Prozedur die Systemeinstellung für den jeweiligen Parameter.

Außerdem wurden die Auswahlfunktionen für Koordinaten, Winkel, Strecke, Objekte usw. so modifiziert, dass die Funktionen nicht durch einen transparenten Befehl (z.B. einen der Zoom-Befehle) abgebrochen werden.

Des Weiteren wurden auch einige zusätzliche Funktionen integriert, die in den einzelnen Kapiteln bereits beschrieben wurden. Dazu zählen beispielsweise die Funktionen zum Einfügen von Polylinienstützpunkten an beliebigen Positionen oder zur Umkehrung der Richtung von Polylinien.

Da alle Funktionen ausführlich im Quellcode kommentiert wurden, soll an dieser Stelle nur eine Zusammenfassung der einzelnen Funktionen erfolgen. Das Klassenmodul selbst finden Sie auf der Buch-CD unter *05/MyUtility.cls*. Sie brauchen diese Quelldatei nur in das jeweilige VBA-Projekt einzubinden, indem Sie die Datei über den Menüpunkt **DATEI/DATEI IMPORTIEREN** importieren. Anschließend deklarieren Sie eine Objektvariable mit dem Datentyp *MyUtility* – entweder global, um von überall her im Code auf die Funktionen der Klasse zugreifen zu können, oder innerhalb eines Klassenmoduls (z.B. *ThisDrawing*). In diesem Fall können Sie dann auf die Variable über dessen Container zugreifen.

```
Public Util As MyUtility
```

Danach müssen Sie nur eine neue Instanz der Klasse erstellen und diese in der deklarierten Variable speichern.

```
Set Util = New MyUtility
```

Bei der Initialisierung der Klasse wird automatisch eine Verbindung zum Klassenmodul *ThisDrawing* hergestellt, so dass die Funktionen immer mit der aktuellen Zeichnung verarbeitet werden.

Kommen wir nun zu den einzelnen Methoden dieser Klasse.

Die folgenden Funktionen wandeln reelle Zahlen in Zeichenfolgen und umgekehrt um, genau wie die gleichnamige Funktion der Klasse *AcadUtility*, nur dass Sie hier nicht die Parameter für Einheit und Genauigkeit angeben müssen. In diesem Fall werden die Systemeinstellungen verwendet.

AngleToReal: Wandelt eine Zeichenfolge in einen Winkel um.

AngleToString: Wandelt einen Winkel in eine Zeichenfolge um.

DistanceToReal: Wandelt eine Zeichenfolge in eine Strecke um.

RealToString: Wandelt eine reelle Zahl in eine Zeichenfolge um.

Die nun folgenden Funktionen entsprechen in ihrer Funktionsweise zwar ihren gleichnamigen Ursprungsfunktionen, wurden allerdings um einige Features erweitert. So unterstützen alle diese Funktionen die Angabe einer Schlüsselwortzeichenfolge, so dass Sie die Funktionen *InitializeUserInput* und *GetInput* nicht explizit ausführen müssen. Haben Sie den Schlüsselwortparameter der Funktion belegt, wird dieser Schritt automatisch durch die Funktion ausgeführt. Gibt der Benutzer dann ein Schlüsselwort ein, liefert die Funktion dieses als Ergebnis zurück. Ein zweiter Punkt bei der Verwendung dieser Funktionen ist, dass Sie hiermit transparente Befehle ausführen können, ohne dass die Funktion vorzeitig beendet wird. Der dritte Punkt, der für die Verwendung spricht, ist, dass eine Konvertierung von Koordinaten des aktuellen BKS in das WKS vorgenommen wird, wenn Sie den entsprechenden Parameter belegen. Damit erübrigen sich weitere Umrechnungsfunktionen, die Sie ausführen müssten, wenn Sie die Ursprungsfunktionen der Klasse *AcadUtility* verwenden würden (siehe **Kapitel 5.3**).

GetAngle: Ruft den angegebenen Winkel unter Berücksichtigung der Einstellung der Systemvariablen *ANGBASE* ab.

GetCorner: Ruft den zweiten Eckpunkt eines Rechtecks ab.

GetDistance: Ruft einen Abstand ab, den der Benutzer entweder in der Befehlszeile eingeben oder über zwei Punkte auswählen kann.

GetOrientation: Ruft den angegebenen Winkel ohne Berücksichtigung der Einstellung der Systemvariablen *ANGBASE* ab.

GetPoint: Ruft einen Punkt in der AutoCAD-Befehlszeile ab.

GetReal: Ruft einen reellen Wert (Double) des Benutzers ab.

GetString: Ruft eine Zeichenfolge des Benutzers ab.

GetEntity: Ruft ein Objekt interaktiv ab. Im Gegensatz zur gleichnamigen Funktion des *Utility*-Objekts müssen Sie hier die Variable, in der der Verweis auf das Objekt zurückgegeben wird, nicht als Datentyp *Object* deklarieren.

Array-Funktionen

ChangeVertex: Tauscht die Position zweier Stützpunkte des angegebenen Stützpunkt-Arrays. Die zu tauschenden Punkte legen Sie mit dem zweiten und dritten Argument der Funktion fest.

GetVertexCount: Ermittelt die Anzahl der Stützpunkte eines Stützpunkt-Arrays. Im Parameter der Funktion müssen Sie lediglich das Objekt übergeben, dessen Stützpunktanzahl Sie ermitteln möchten.

InsertVertex: Fügt einen Stützpunkt an einer beliebigen Stelle in das im ersten Parameter angegebene Stützpunkt-Array ein.

RemoveVertex: Entfernt einen beliebigen Stützpunkt aus dem angegebenen Stützpunkt-Array.

Weitere Funktionen

ReverseCoordinates: Diese Funktion kehrt die Richtung der Stützpunkte eines Arrays um. Im ersten Parameter wird das Array angegeben, im zweiten können Sie festlegen, ob es sich bei den Stützpunkten um 2D- oder 3D-Koordinaten handelt. Lassen Sie diesen Parameter weg, wird angenommen, dass es sich um 3D-Koordinaten handelt.

AngleFromXAxis: Berechnet den Richtungswinkel zwischen den zwei angegebenen Punkten.

CalcDistance: Berechnet den Abstand zwischen den zwei, in den Parametern angegebenen, Punkten. Im dritten Parameter können Sie festlegen, ob die Funktion den 2D- oder 3D-Abstand zurückgeben soll. Lassen Sie diesen Parameter weg, wird der 2D-Abstand berechnet.

GetDocName: Liefert den Namen inklusive der Verzeichnisangabe des im ersten Parameter angegebenen *Document*-Objekts. Wurde eine neue Zeichnung erstellt (*Zeichng1*), diese aber noch nicht gespeichert, wird als Verzeichnis der Pfad der Datei *acad.exe* verwendet. Im zweiten Parameter können Sie festlegen, ob die Dateierweiterung (*dwg/dxf*) angehängt werden soll oder nicht. Lassen Sie diesen Parameter aus, wird auch die Dateierweiterung weggelassen.

PolarPoint: Berechnet die Koordinaten eines Punkts aus den polaren Werten, ausgehend vom Punkt des ersten Parameters.

Get3DArray: Wandelt ein 2D-Array in ein 3D-Array um.

GetColorName: Liefert den Namen der Farbe.

GetColorValue: Mit dieser Funktion können Sie die RGB-Farbe einer AutoCAD-Farbe abrufen.

A.4 Neuerungen in AutoCAD 2004

Gegenüber den Vorgängerversionen enthält die Klassenbibliothek der Version 2004 einige Neuerungen. Die wichtigste dieser Neuerungen möchte ich Ihnen natürlich nicht vorenthalten. Hierbei handelt es sich um eine Eigenschaft, auf die die AutoCAD-Gemeinde lange hat warten müssen und deren Verwendung in anderen Windows-Anwendungen gang und gäbe ist. Damit meine ich die Verwendung von RGB-Farben in AutoCAD 2004. Sicher wird dem Anwender dieser Version bereits aufgefallen sein, dass die Anzahl der Farben nicht mehr auf die 255 Standardfarben beschränkt wurde. Vielmehr hat man sich dazu entschlossen, die Farben aus den RGB-Werten zusammenzusetzen, was eine Gesamtfarbanzahl von mehreren Millionen ermöglicht. Das erforderte natürlich auch eine Überarbeitung der Klassen der Objektbibliothek von AutoCAD. So wurde jedem grafischen Objekt die Eigenschaft *TrueColor* hinzugefügt, die auf der neuen Klasse *AcadAcCmColor* basiert. In dieser Eigenschaft werden von nun an die Farbeinstellungen der einzelnen Objekte verwaltet. Aber keine Angst, die *Color*-Eigenschaft blieb aus Kompatibilitätsgründen erhalten, ist allerdings jetzt eine verborgene Eigenschaft. Aus diesem Grund blieben die Anwendungen, die Sie in Vorgängerversionen entwickelt haben, auch aufwärtskompatibel.

Aber schauen wir uns doch einfach einmal die Verwendung der Eigenschaft *TrueColor* an. Da diese jetzt auf einer Klasse basiert und somit ein komplettes Objekt verwaltet, reicht es nicht einfach, nur einen Farbwert wie bei der *Color*-Eigenschaft zuzuweisen. Vielmehr müssen Sie hier die Farbwerte an die entsprechende Eigenschaft oder Methode des *TrueColor*-Objekts übergeben. Den Weg, den man hier eingeschlagen hat, ist etwas umständlich. Sie müssen zunächst die Farbeinstellungen an einer Objektvariablen vom Typ *AcadAcCmColor* vornehmen und diese anschließend der *TrueColor*-Eigenschaft des jeweiligen Objekts übergeben.

```
Dim MyCol As AcadAcCmColor
```

Entweder Sie übergeben nun den Inhalt der *TrueColor*-Eigenschaft des Ursprungsobjekts an die Objektvariable:

```
Set MyCol = Object.TrueColor
```

Oder Sie erstellen eine neue Instanz der Klasse *AcadAcCmColor*:

```
Set MyCol = New AcadAcCmColor
```

Jetzt können Sie die Farbeigenschaften dieses Objekts anpassen. Danach übergeben Sie die Objektvariable an die *TrueColor*-Eigenschaft und schon wurde die Farbe des Objekts angepasst.

```
Object.TrueColor = MyCol
```

Eigenschaften und Methoden des Farbbobjekts

Da Sie unterschiedliche Methoden zur Definition der Farben verwenden können, müssen Sie über die *ColorMethod*-Eigenschaft festlegen, welche dieser Methoden auf das entsprechende Objekt angewendet werden soll. Egal, welcher Modus hier eingestellt wurde, es werden immer die Werte der Eigenschaften *Blue*, *Green* und *Red* verwendet, da natürlich auch die AutoCAD-Standardfarben auf diesen Werten basieren. Wenn Sie beispielsweise dem Objekt eine Standardfarbe über dessen *ColorIndex*-Eigenschaft zuweisen, ändert AutoCAD automatisch den Inhalt dieser Eigenschaften auf den dem Farbindex entsprechenden Wert.

Konstante	Beschreibung	Wert
<i>acColorMethodByLayer</i>	Verwendung der Farbe vonLayer (Standard)	192
<i>acColorMethodByBlock</i>	Verwendung der Farbe vonBlock	193
<i>acColorMethodByRGB</i>	Verwendung der RGB-Farbwerte	194
<i>AcColorMethodByACI</i>	Verwendung der 255 Grundfarben	195
<i>acColorMethodForeground</i>	Verwendung der Vordergrundfarbe	197

Tabelle 1.2: Konstanten der *ColorMethod*-Eigenschaft

Was nun die Verwendung der RGB-Farben betrifft, können Sie diese mit der *SetRGB*-Anweisung dem Objekt zuordnen. Sie müssen lediglich die entsprechenden Farbwerte (Rot, Grün Blau), aus denen die gewünschte Farbe zusammengesetzt ist, in den Argumenten der Anweisung übergeben.

```
MyCol.TrueColor.SetRGB 255, 0, 0
```

Eine weitere Möglichkeit, eine RGB-Farbe einem Objekt zuzuordnen, besteht in der Nutzung von so genannten Farbbüchern. Diese Methode wurde entwickelt, um benutzerspezifischen Farben zusammenzufassen und für eine spätere Verwendung in einer Vorlage abzuspeichern. Auf diese Weise bekommen Sie Ordnung in die verschiedenen Farbdefinitionen und brauchen später nur noch den Namen, unter der die bestimmte Farbe im Farbbuch gespeichert wurde, und nicht mehr dessen RGB-Werte anzugeben. Der Befehl, mit dem Sie dem Objekt eine Farbe aus einem der Bücher zuweisen, ist *SetColorBookColor*. Im ersten Argument der Anweisung legen Sie einfach den Namen der Farbbuchs und im zweiten den Namen der Farbe aus diesem Buch fest.

```
MyCol.SetColorBookColor "ral classic", "RAL 1000"
```

Wenn alles korrekt abgelaufen ist, werden die RGB-Werte auf die der angegebenen Farbe eingestellt und die *ColorMethod*-Eigenschaft automatisch auf *acColorMethodByRGB* gesetzt. Außerdem enthalten die Eigenschaften *BookName* den Namen des Farbbuchs und *ColorName* den Namen der verwendeten Farbe des Buchs.

Um eine RGB-Farbe mit einem Namen zu versehen, müssen Sie den Namen des Farbbuchs mit der *SetNames*-Anweisung aufrufen. Der Name, unter dem die Farben gespeichert wurden, wird dann in die Farbauswahlliste übernommen.

```
MyCol.SetRGB 242, 124, 33
```

```
MyCol.SetNames "Benutzerfarbe1", "MyColorBook"
```

Die Farbbücher selbst sind in Dateien mit der Endung *acb* gespeichert. Auf welche Bücher AutoCAD zugreift, hängt von der Einstellung der Eigenschaft *ColorBookPath* ab. AutoCAD lädt alle *acb*-Dateien, die in diesem Verzeichnis abgelegt wurden. Die *ColorBookPath*-Eigenschaft ist Bestandteil des *Preferences*-Objekts von *Application* und kann über dessen *Files*-Eigenschaft angesprochen werden.

```
Application.Preferences.Files.ColorBookPath = "C:\Farbbücher"
```

A.5 Infos zu AutoCAD-VBA im Internet

Das Internet stellt eine schier unerschöpfliche Quelle dar, wenn es darum geht, Informationen zum Thema Visual Basic zusammenzutragen. Ein Hauptaugenmerk sei hierbei auf die deutschen Foren gelegt, in denen die Benutzer von AutoCAD Erfahrungen austauschen können. Zu diesen Foren zählen zweifellos www.induma.com, neuerdings auch unter der Adresse www.industrie24.com zu erreichen, und www.cad.de/foren.shtml. Beide Foren zeichnen sich durch ihre hohe Mitgliederzahl aus, womit Sie auf ein umfangreiches Wissen zurückgreifen können, sollten Sie einmal Fragen zum Thema AutoCAD-VBA haben.

Nicht außer Acht zu lassen ist auch die Möglichkeit, AutoCAD-VBA um eine Vielzahl an Funktionen zu erweitern. Auch hier können Sie auf die Arbeit von Profis zurückgreifen, indem Sie sich auf deren Internetadressen begeben und die Programmiererweiterungen aus dem Download-Bereich herunterladen. Leider sind diese Seiten zumeist in Englisch verfasst, was aber auch hier kein Problem darstellen sollte, da die meisten dieser Bibliotheken nach dem gleichen Prinzip arbeiten wie die Objektbibliothek von AutoCAD. Erst kürzlich entdeckte ich wieder eine Seite (www.contractcaddgroup.com/download), die einige Zusatzbibliotheken für VBA anbietet, die besonders hilfreich sein können. Ein großes Potenzial an Downloads bietet auch die Seite www.caddepot.com. Auf dieser Seite werden Sie bestimmt eine Vielzahl an Freeware-Downloads finden, die Ihnen als Beispielanwendungen dienen könnten.

Stichwortverzeichnis

!

2D-Flächen 765
3D-Arrays 394
3D-Flächenobjekt 635
3D-Polylinie 640
3D-Solid 387

A

Abbildungsliste 108, 111
Abfragen 56
Abstandsermittlung 324
AcadApplication 68
AcadDocument, *Prozeduren* 311
AcadEntity 419, 591
AcadGroups 418
AcadHyperlinks 426
AcadPopupMenu 231, 234
AcadProject 15
Activate 161
ActiveDocument 217
Add 65, 72
Aktueller Layer 437
Alternative Bemaßungseinheiten 669
Ansichten 549
Ansichtsfenster 559
 konfigurieren 560
Anwendungsaufflistung 528
Anwendungsobjekt 211
 WindowMovedOrResized 217
Anzeigefelder 104
Anzeigenreihenfolge 22
Application 217
Arithmetische Operatoren 79
Array 38
 dynamisch 39
ARX 212
Attributdefinition 821
Attributreferenz 834, 842
Auflistungen 415
Auflösen 644
 3D-Netz 780
Ausgabeeigenschaften 273
Ausgeben, Zeichenfolge 332
Auswahlsatz 604
AutoCAD, Typenbibliothek 46
AutoCAD-Befehle
 vbaausdr 14
 vbaausf 14
 vbaentf 14
 vbaide 13
 vbalad 13
 vbaman 13
 vbarun 13
AutoCAD-Objekte, Auswahl
 *einzel*n 603
 mehrere 604
 select 606
 SelectAtPoint 607
 SelectByPolygon 607
 SelectOnScreen 608
 Eigenschaften 407
 Ereignisprozeduren 413
 Farbe 616
 GetBoundingBox 624
 hervorheben 624
 Hyperlinks 616
 kopieren, verschieben, drehen 621
 Layer 616
 Linienstärke 617
 Linientypen 617
 Linientypfaktor 617
 löschen 623
 Methoden 409
 Plotstilname 618
 polare Anordnung 619
 räumlich drehen 628
 rechteckige Anordnung 620
 Schnittpunkte ermitteln 625
 Sichtbarkeit 618
 skalieren 629
 spiegeln 2D 626
 spiegeln 3D 627
 TransformBy 630
AutoCAD-Sitzung 13
AutoTrack 260

B

Band 763
Basisobjekt 393
Befehlsschaltflächen 97
Befehlszeilenmodus 13f.
Beispielprojekt 238

Bemaßungslinien 668
Bemaßungsobjekte 660
Bemaßungstext 661
Benutzer 73
Benutzerformular 157
 Ereignisse 161
 Methoden 161
Benutzergruppe 73
Benutzerkommunikation 320
Benutzerkoordinatensysteme 516
Benutzerzeichenfolge 328
Berechnen, Koordinaten 332
Bereinigen 808
Beschreibungstext 545
Bezeichnungsfeld-Eigenschaften 88
Bezeichnungsfeld-Ereignisse 88
Bibliothek 21
Bilddaten 106
Bilder 107
Bildlaufleisten 102
BKS 519
Blockauflistung 532
Blockauswahl 701
Blockdefinition 470, 532f., 546
Blockname 544
Blockreferenz 56, 341, 592, 836
Blockvorschau 548
Breakpoint 28
ByVal 53

C

Caption 160, 217
CheckBox 94
CheckBox-Eigenschaften 94
Child 144f.
Code 58
Collection 71
ComboBox 93, 125
CommandButton 97, 858
CommandButton-Eigenschaften 97
Container 48
Count 65, 72

D

Darstellung 85
Datei
 acad.dvb 18
 dxf-Files 15
 herunterladen 330
 öffnen 269
 speichern 268
 übertragen 331

Dateiauswahldialog 147
Dateiname 265
Datenbankobjekt 305
Datensicherheit 269
Datentyp 34, 36f., 43, 408
 benutzerdefiniert 44
 Boolean 36
 Double 36
 konvertieren 42
 prüfen 41
 Variant 36
Deactivate 161
Debugging 30
Deklaration 34
Delete 448
Display-Objekt 251
do...loop 60
Documents 218
DoEvents 61
Dokumentaflistung 289
 Eigenschaft 290
 Prozeduren 290
Dokumentobjekt 68
Drafting-Objekt 258
Drehfeldsteuerelement 103
Drehwinkel 860
Drittanbieterobjekte 271
Drop-down-Liste 20
Drucken 879
Drucker 156
Druckerwarteschlange 889
Duplex 881
Durchmesserbemaßung 680
DXF-Gruppencode 610
DXF-Schnittstelle 609

E

Eigenschaftenfenster 27
Eigentümerobjekt 409
Einbetten 15
Einfügen
 Blockreferenz 399
 Externe Referenzen 401
 Pixelbilder 374
Einstellungen 211, 250
Ellipse 655
Elliptischer Bogen 656
Empty 41
End Function 53
End Sub 53
End With 67

- Enter 88
- Entwicklungsumgebung 13
- Enum 46
- Erase 40
- Ereignisprozedur 68, 212
 - Activate* 292
 - Deactivate* 292
- Ereignisse 76
- Erstellen
 - 2D-Polylinien 345
 - 2D-Volumenkörperpolygon 379
 - 3D-Flächen 342
 - 3D-Netz 343
 - 3D-Polylinien 344
 - 3D-Volumenkörper 387
 - Attributdefinition* 349
 - Ausgerichtetes Bemaßungsobjekt* 353
 - Benutzerdefiniertes Objekt* 351
 - Durchmesserbemaßung* 353
 - Ellipse* 359
 - Führungslinie* 362
 - Gedrehte lineare Bemaßung* 358
 - Geometrische Toleranz* 383
 - Instanzen eines Blocks* 365
 - Koordinatenbemaßung* 355
 - Kreis* 350
 - Kreisbogen* 348
 - Linien* 364
 - Mehrzeiligen Text* 369
 - Multilinen* 367
 - Optimierte Polylinien* 347
 - Papierbereichsansichtsfenster* 372
 - Polylinien* 344
 - Punktobjekt* 370
 - Radiusbemaßung* 356
 - Region* 377
 - Schraffur* 360
 - Strahl* 376
 - Symbolobjekt* 378
 - Vielflächennetz* 370
 - Winkelbemaßung* 351
- Erweiterte Daten 409
- Erweiterungswörterbuch 409, 578
- Erzeugen
 - Einzeiligen Text* 382
 - Konstruktionslinie* 387
 - Splines* 381
 - Spurobjekt* 385
- Exit 88
- Exit Function 53
- Explode 643
- Export 448
- Externe Referenz 270, 539
- Externe Referenz siehe XRef
- F**
- Farbe 154
- Farbeigenschaften 84
- Files-Objekt 262
- Flags-Eigenschaft 153
- FM20.DLL 83
- Font-Eigenschaft 86
- Fontnamen 509
- For...Each...Next 59
- for...next 58
- Formular 81
 - schließen 158
 - starten 158
- Formular-Eigenschaften 160
- Formularposition 159
- Formularraster 25
- Fortschrittsanzeige 117
- Frames 98
- Frames-Eigenschaften 99
- Frames-Ereignisse 99
- Frames-Methoden 99
- Führungslinien 696
- FullName 218
- Function 48
- Funktion 42
 - IsMissing 55
 - UBound 55
- Fußgesteuert 60
- G**
- Ganzzahlen 44, 328
- GetEntity 603
- GetFont 502
- GetPoint 158
- Gewicht siehe Splines
- Grafische AutoCAD-Objekte 600
- Grafische Objekte 324
 - drucken 882
- Gruppen 417
- Gruppenelemente 419
- Gültigkeitsbereich 32
- H**
- Handle 408
- HasExtensionDictionary 408
- Hatch 707
- Height 18, 218
- HelpContextID 86

Hide 159
Hilfdatei 151
Hilfslinien 668
Hochformat 492
Hyperlink 282, 426

I

if...then 56
Image 104
Image-Eigenschaften 105
ImageList 106
Image-Steuerelemente 858
Implements 77
Import 448
Initialisierungsroutine 422
IsEmpty 41
IsNull 42
ISO-Schraffurmuster 705
IsRemoteFile 331
IsURL 331
Item 65, 72

K

kanonisch 472
Kapitel 13
Klasse
 AcadAttributeReference 834
 AcadBlockReference 836
 AcadCircle 650
 AcadDimension 660
 AcadEntity 633
 AcadObject 632
 AcadPopupMenuItem 236
 AcadRegion 790
 AcadShape 745
 AcadToolBar 243
 Linie 718
 Printer 879
Klassenmodul 20, 70
Kollisionskontrolle 870
Kombinationsfeld 125
Komponentenfenster 13
Konstanten 46
Konstruktionslinien 761
Kontextmenü 22
Kontrollkästchen 94
Koordinaten 593
Koordinatenbemaßung 688
Koordinatenliste 40
Koordinatenumrechnung 333
Kopfgesteuert 60
Kreis 650
Kreisbogen 646, 727

L

Label 88
Laufzeitbibliothek 879
Layer 434
Layer-Auflistung 63, 434
Layermanager 113
Layerstatusmanager 446
Layout 273, 468
Layoutobjekte 469
Lebensdauer 50
Left 18
Lightwight-Polylinie 724
Lineare Bemaßung 676
LineTypes 457
Linie 718
Linientypen 456
ListBox 90
Listeneinträge 91, 113
 bearbeiten 114
Listenfeld 93
ListImages 107
ListSubItems 113
ListView 108, 506
LocaleId 218
Logische Operatoren 79

M

Makro
 ausführen 16
 bearbeiten 16
 dxf-Files 16
 Einzelschritt 16
 erstellen 17
 löschen 17
 Optionen 17
 schließen 16
Maßlinien 668
Mauszeiger 85
Me 30
Medien verwalten 471
Mehrdimensional 39
Mehrfachauswahl 92
Mehrfachauswahl-Eigenschaften 92
Mehrfachblockreferenz 850
MenuBar 219
Menü
 Ansicht 19
 Ansicht-Direktfenster 30
 Extras/Optionen 25
 Extras/Verweise 24
Menüeinträge
 entfernen 235
 hinzufügen 233

Menügruppen 226
 bearbeiten 228
MenuGroups 219
Methoden 63
Microsoft Word 18
Modellbereich 335, 403
 Eigenschaften 403
 Methoden 404
ModelSpace 335
MouseDown 87
MouseMove 87
MouseUp 87
Move 87
MSForms-Bibliothek 82
MText 498, 754
Multilinie 737
MultiPage 99
Multiseiten 99
 verwalten 100
Multiseiten-Eigenschaften 101f.
Multiseiten-Ereignisse 101

N

Name 218
Neigungswinkel 515
Node-Objekte 141
NURBS-Kurven 798

O

ObjectID 408, 427
Objektauflistungen 64
Objektauswahl 600
Objekte 61, 66
Objekteigenschaft 62
Objekthierarchie 211
Objektkatalog 20
Objektprozeduren 63
Objektvariable 61, 63
OLE-Objekte 273
OpenSave-Objekt 267
Operatoren 77
Option Explicit 25, 34, 41
Optional 54
OptionButton 95
Optionsfelder 95
Output-Objekt 272

P

Panels 122
PaperSize 880
Papierausrichtung 480

Papierbereich 335, 403
Papierformat 491
ParamArray, IsMissing 55
Parameterübergabe 53
Parent 137, 145
Passwort 26
Path 218
Pattern 705
pc3-Datei 471, 877
Pfeilspitzen 667
Picture 108, 160
Pipe-Zeichen 844
Pixelbild 782
Plotbereich 475, 875
Ploteinstellungen 473
Plotkonfiguration 875
Plotoptionen 474
PlotOrigin 490
Plotposition 479
Plotschnittstelle 471
Plotskalierung 477
Plotstil 272
Plotstiltabellen 479
Plotten 875
Polygonnetz 770
Polylinie 724
Polyliniensegmente 727
Position 592
Preferences 219
Private 49
Profiles-Objekt 284
ProgID 162
Programmiersprache 33
ProgressBar 117
Projektdaten, dxf-Files 15
Projektexplorer 19
Projektliste 15
Projekt-Manager 15
Property 75
Prozedur 47
 BeginCommand 212
 BeginFileDrop 213
 des Anwendungsobjekts 221
 EndCommand 212
 Lisp 213
 Modal 214
 NewDrawing 216
 Open 214
 Plot 215
 Quit 215

Save 215
SysVarChanged 216
WindowChanged 216
WindowMovedOrResized 216

Prüfmodus 821
 Public 49
 Pull-down-Menü 230
 Punkt 743
 Punktkoordinaten ermitteln 327

R

Radiusbemaßung 685
 Rahmenfelder 98
 Random-Modus 45
 Rasterbilder 782
 Reagieren 69
 Rechteck 324
 ReDim 40
 Reelle Werte 328
 Referenz 408
 Region 790
 Registrieren 528
 Reichweite 49
 Rekursionen 47
 Rename 448
 Restore 448
 Richtungsvektor 518
 Root 145
 Rotationsvolumenkörpers 394

S

Save 448
 ScaleFactor 745
 Schaltflächen 93, 131, 247
 Schaltflächenauflistung 134
 Schaltflächenmenü 133
 Schieberegler 118
 Schleifen 56
 Schlüsselwort 34, 325

- Dim 34
- Lock 34
- Public 34

 Schraffur 704
 Schriftart 152
 ScrollBar 102
 ScrollBar-Eigenschaften 103
 ScrollBar-Ereignisse 103
 Select 606
 Select...Case 57
 Selection-Objekt 275
 Selektionsmodus 608
 Set 61

SetDataBase 448
 SetFocus 86
 SetFont 502
 Sichtbarkeit 618

- der Kanten 637

 Slider 118
 Sonderzeichen 33
 Sortierung 111
 Spaltenköpfe 111
 Sperren 86
 SpinButton 103
 SpinButton-Eigenschaften 104
 SpinButton-Ereignisse 104
 Splines 798
 Spoolen 267
 Stärkevektoren 799
 Standardlinienart 456
 Standardskalierung 493
 Standardsteuerelemente 21
 Startprozedur 69, 414
 StatusBar 121
 StatusId 219
 Statusleiste 121
 Steuerelementbibliothek 83
 Steuerelemente 82, 106

- hinzufügen
 - Ereignisse* 162

 Steuerelementereignisse 146
 Steuerelementgröße 85
 Steuerelementposition 85
 Strahlen 759
 Stützpunkte 592, 595

- verwalten* 636

 Sub 48
 Subelemente 329
 Sub-Routinen 48
 Suchpfad 262
 Symbolleiste 128
 Symbolleisten-Ereignisse 136
 Symbolleisten-Methoden 138
 System-Objekt 277
 Systemvariable

- ACADLSPASDOC 278
- ANGBASE 745, 837
- Bemaßung 694
- GRIPBLOCK 276
- GRIPCOLOR 277
- GRIPHOT 277
- GRIPS 276
- GRIPSIZE 277
- HPANG 712
- INSUNITS 283

LWUNITS 444
OSNAPCOORD 280
PICKADD 275
PICKAUTO 275
PICKBOX 276
PICKDRAG 276
PICKFIRST 275
PICKSTYLE 276
SDI 278
SHORTCUTMENU 281
SPLFRAME 635
VIEWTWIST 753

T

Tabulatoreigenschaften 85
Tag 86
Tangenten 808
Tastaturverhalten 280
Tastenkombinationen 31
Text, Alignment 748
TextBox 90
TextBox-Eigenschaften 90
Textbreite 515
Texteingabe 702
Textfeld 89
Texthöhe 514
Textobjekte 748
Textstil 498, 512
Textstilname 751
TextStyles 498
Textumwandlung 819
ThisDrawing 49, 212
Titelzeile drucken 889
ToggleButton 96
ToggleButton-Eigenschaften 96
ToggleButton-Ereignisse 96
Toleranzobjekt 809
Toleranzwerte 664
Toolbar 128
Tooltip-Text 29
Top 18
TreeView 138
Typenbibliothek 591

U

UCS 518
Überlagern 796
Überwachungsfenster 32
Umringpunkt 775
Umschaltfelder 96

Umwandeln
 Dezimalzahlen in Zeichen 333
 Winkel in Zeichen 321
 Zeichen in Strecke 322
 Zeichen in Winkel 321
Unload 158
URL 427
Ursprungskoordinaten 526
User-Objekt 279

V

Variable 32f.
 Variant 34
Variablennamen 33, 35
VBA-Befehle
 acadvba.arx 18
 -vbarun 17
VBA-Entwicklungsumgebung 14
VBA-Manager 13
VBA-Projektdatei 13
VBA-Projekte
 einbetten 15
 entfernen 15
 extrahieren 15
 laden 15
VBA-Umgebung 13
VBE 219
vbModal 158
Vektoren 598
Vergleichsoperatoren 78, 611
Verkettungsoperatoren 79
Verlassen 53
Verschachtelung 59
Version 220
Verwaltung der Menüstruktur 225
Verzweigung 56
Vielflächennetz 767
Viewports 559
Visible 86, 220
Visual Basic 18
Visuelle Entwicklungsumgebung 81
Volumenkörper 869
Vorgabewerte 500

W

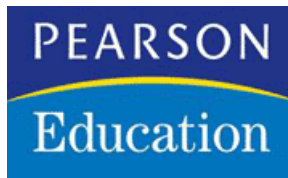
Webbrowser 331
Weltkoordinatensystem 592
Werkzeugfenster 246
Werkzeugkästen 242
Werkzeugsammlung 82

Width 18, 218
Wildcards 613
WindowLeft 220
Windows-Standarddialog 147
WindowState 220, 297
WindowTop 220
Winkel 321, 599
Winkelbemaßung 663, 670
Winkleingabe 323
With 67
WithEvents 69
WKS 518
Wörterbücher 579
 abrufen 579
 erstellen 579
 Zugriff 580

X
XRef 270
X-Vektoren 527

Y
Y-Vektoren 527

Z
Zählschleifen 58
Zeichnung
 Eigenschaften 298
 Ereignisprozedur 292
Zeichnungsausschnitt 551
Zeichnungselemente 591
Zeichnungsstruktur 138, 407
Zeichnungsverwaltung 289
Zentrumsmarke 681
Zoomfunktionen des Anwendungsobjekts 223
Zugreifen auf Zeichnungen 292
Z-Vektoren 527



Copyright

Daten, Texte, Design und Grafiken dieses eBooks, sowie die eventuell angebotenen eBook-Zusatzdaten sind urheberrechtlich geschützt.

Dieses eBook stellen wir lediglich als **Einzelplatz-Lizenz** zur Verfügung!

Jede andere Verwendung dieses eBooks oder zugehöriger Materialien und Informationen, einschliesslich der Reproduktion, der Weitergabe, des Weitervertriebs, der Platzierung im Internet, in Intranets, in Extranets anderen Websites, der Veränderung, des Weiterverkaufs und der Veröffentlichung bedarf der schriftlichen Genehmigung des Verlags.

Bei Fragen zu diesem Thema wenden Sie sich bitte an:

<mailto:info@pearson.de>

Zusatzdaten

Möglicherweise liegt dem gedruckten Buch eine CD-ROM mit Zusatzdaten bei. Die Zurverfügungstellung dieser Daten auf der Website ist eine freiwillige Leistung des Verlags. Der Rechtsweg ist ausgeschlossen.

Hinweis

Dieses und andere eBooks können Sie rund um die Uhr und legal auf unserer Website



(<http://www.informit.de>)

herunterladen