Microsoft

VBA-Programmierung

Integrierte Lösungen mit Office 2003

VBA2003 669-10725

INHALTSVERZEICHNIS

1 Über diese Unterlage 6

1.1 Voraussetzungen und Ziele 6
1.2 Aufbau und Konventionen 7

2 Einführung 10

2.1 Was bedeutet VBA? 10
2.2 Was sind integrierte Lösungen? 11
2.3 Einsatzgebiete der Office-Anwendungen 12
2.4 Ein einfaches Beispiel für eine integrierte Lösung 13
2.5 VBA-Prozedur mit Menüleiste verknüpfen 15
2.6 Schnellübersicht 16
2.7 Übung 16

3 Arbeiten mit der VBA-Entwicklungsumgebung 18

3.1 VBA-Entwicklungsumgebung verwenden 18
3.2 Bestandteile der Entwicklungsumgebung 19
3.3 Der Projekt-Explorer 20
3.4 Das Eigenschaftenfenster 21
3.5 Das Modulfenster benutzen 22
3.5.1 Bestandteile des Modulfensters 22
3.5.2 Anweisungen eingeben und ausführen 22
3.5.3 Automatische Eingabehilfen verwenden 23
3.5.4 Neue Prozedur erstellen 26
3.5.5 Prozeduren bearbeiten 27
3.5.6 Tipps zum Arbeiten mit dem Modulfenster 28
3.6 Das Direktfenster 30
3.7 Das Lokalfenster 30
3.8 Schnellübersicht 31
3.9 Übung 31

4 Die Sprachelemente von VBA 32

4.1 Module verwenden 32
4.2 Mit Prozeduren programmieren 34
4.3 Variablen, Konstanten und deren Datentypen 37
4.3.1 Der Begriff Variable 37
4.3.2 Deklaration von Variablen 38
4.3.3 Konstanten deklarieren und verwenden 39
4.3.4 Datentypen von VBA 40
4.3.5 Datenfelder (Arrays) 41
4.4 Programmablauf mit Kontrollstrukturen steuern 43
4.4.1 Verzweigungen 43
4.4.2 Schleifen 45
4.5 Vordefinierte Funktionen 47
4.6 Schnellübersicht 47
4.7 Übung 48

5 Objektorientierte Programmierung mit VBA 50

5.1 Was sind Objekte? 50
5.2 Die Objekthierarchien (Objektmodelle) 51
5.3 Eigenschaften und Methoden 51
5.3.1 Eigenschaften von Objekten 52
5.3.2 Methoden von Objekten 53
5.3.3 Die `With`-Anweisung zum Zugriff auf Eigenschaften und Methoden 54
5.4 Objektvariablen 54
5.5 Auflistungen in der Objekthierarchie 56
5.5.1 Objektauflistungen 56
5.5.2 Auf einzelne Objekte einer Auflistung zugreifen 56
5.5.3 Namen beim Zugriff auf Auflistungen verwenden 57
5.5.4 Auf alle Objekte einer Auflistung nacheinander zugreifen 57
5.6 Den Objektkatalog verwenden 58
5.7 Schnellübersicht 60
5.8 Übung 60

6 Gemeinsam genutzte VBA-Elemente 62

6.1 VBA-Elemente für alle Office-Anwendungen 62
6.2 Programmieren des Anwendungsfensters 62
6.3 Die Dokumentenfenster programmieren 64
6.4 Dateien suchen 67
6.5 Dialogfenster zur Datei- und Ordnerauswahl 68
6.6 Mit dem Dateisystem programmieren 69
6.7 Eingabedialoge und Meldungsfenster 70
6.7.1 Eingabedialoge verwenden 70
6.7.2 Meldungsfenster verwenden 72
6.8 Schnellübersicht 74
6.9 Übung 74

7 Programmieren der Office-Anwendungen 76

7.1 Mit Word programmieren 76
7.1.1 Das Word-Objektmodell 76
7.1.2 Mit Dokumenten arbeiten 76
7.1.3 Auf Dokumenteninhalte zugreifen 78
7.2 Mit Excel programmieren 80
7.2.1 Das Excel-Objektmodell 80
7.2.2 Mit Arbeitsmappen und Tabellenblättern arbeiten 80
7.2.3 Zugriff auf Zellen und Zellbereiche 82
7.3 Mit Access programmieren 84
7.3.1 Das Access-Objektmodell 84
7.3.2 Mit Access-Datenbanken arbeiten 84
7.3.3 Zugriff auf Steuerelemente in Formularen und Berichten 85
7.4 Mit PowerPoint programmieren 87
7.4.1 Das PowerPoint-Objektmodell 87
7.4.2 Mit Präsentationen und Folien arbeiten 87
7.4.3 Zugriff auf den Folieninhalt 88
7.5 Mit Outlook programmieren 90
7.5.1 Das Outlook-Objektmodell 90
7.5.2 Auf Ordner zugreifen 90
7.5.3 Auf Elemente zugreifen 91
7.6 Schnellübersicht 92
7.7 Übung 93

8 Kommunikation zwischen Office-Anwendungen 94
8.1 Integrierte Office-Automatisierung 94
8.2 Technische Grundlagen 95
8.3 Verweis auf eine Objektbibliothek erstellen 96
8.4 Deklarieren von Objektvariablen für die Automation 96
8.5 Erzeugen einer Objektinstanz 98
8.5.1 Grundlagen 98
8.5.2 Objektinstanz mit `CreateObject` erzeugen 99
8.5.3 Objektinstanz mit `GetObject` erzeugen 100
8.5.4 Objektinstanz mit `New` erstellen 100
8.6 Objekt schließen und Arbeitsspeicher freigeben 101
8.7 Meldungen des Automationsobjekts unterdrücken 103
8.8 Schnellübersicht 104
8.9 Übung 105

9 Benutzerdefinierte Dialoge verwenden 106
9.1 Kommunikation mit dem Anwender 106
9.2 UserForm-Dialoge erstellen 107
9.3 UserForm-Dialoge verwenden und programmieren 111
9.4 Steuerelemente programmieren 114
9.5 Schnellübersicht 115
9.6 Übung 116

10 Datenbankzugriff in Office-Anwendungen 118
10.1 Objekte zum Datenbankzugriff 118
10.1.1 Grundlagen 118
10.1.2 Hierarchie des ADO-Objektmodells 119
10.1.3 Ablauf des Datenzugriffs mit ADO 119
10.2 Verbindung zu einer Datenquelle herstellen 120
10.3 Datensatzgruppe auswählen und öffnen 122
10.4 Datensätze und Datenfelder auslesen 123
10.4.1 Datensätze auslesen 123
10.4.2 Zugriff auf Datenfelder 125
10.4.3 Datensätze suchen 126
10.5 Datensätze hinzufügen und bearbeiten 127
10.5.1 Datensätze hinzufügen 127
10.5.2 Datensätze ändern und löschen 128
10.6 Schnellübersicht 129
10.7 Übung 130

11 Integrierte Lösungen mit Word 132
11.1 Automation mit Word 132
11.2 Diagrammfunktion von Excel nutzen 132
11.3 Adressen aus Access übernehmen 135
11.4 Schnellübersicht 138
11.5 Übung 138

12 Integrierte Lösungen mit Excel 140
12.1 Automation mit Excel 140
12.2 Druckfunktion von Word nutzen 141
12.3 Daten aus Outlook und Access importieren 144
12.4 Schnellübersicht 147
12.5 Übung 148

13 Integrierte Lösungen mit Access 150
13.1 Automation mit Access 150
13.2 Datenbank zur Dokumentenverwaltung 151
13.3 Dokumente in die Datenbank einlesen 152
13.3.1 Das Formular DokumenteEinlesen 152
13.3.2 Dokumente suchen 154
13.3.3 Vorschaufunktion für Word-Dokumente 156
13.3.4 Vorschaufunktion für Excel-Dokumente 157
13.3.5 Dokumente in der Datenbank archivieren 158
13.3.6 Liste der gefundenen Dokumente löschen 159
13.3.7 Versteckt geöffnete Word- bzw. Excel-Anwendung schließen 160
13.4 Dokumente verwalten und öffnen 161
13.4.1 Das Formular DokumenteVerwalten 161
13.4.2 Dokumente in der Datenbank suchen 162
13.4.3 Dokumente öffnen 164
13.5 Schnellübersicht 165
13.6 Übung 165

14 Integrierte Lösungen mit PowerPoint 168
14.1 Automation mit PowerPoint 168
14.2 Die Anwendung SCHNELLPRÄSENTATION als integrierte Lösung 169
14.3 Bearbeiten der Titelfolie 170
14.4 Erstellen einer Textfolie mit Word-Unterstützung 172
14.5 Erstellen einer Diagrammfolie mit Excel-Unterstützung 175
14.6 Suchfunktion für Excel-Arbeitsmappen 179
14.7 Schnellübersicht 180
14.8 Übung 180

15 Integrierte Lösungen mit Outlook 182
15.1 Automation mit Outlook 182
15.2 Notizen in Word erstellen 183
15.3 Serien-E-Mails mit Word und Outlook versenden 184
15.3.1 Mit der Anwendung SERIEN-E-MAIL arbeiten 184
15.3.2 Initialisieren der Anwendung und Bestimmen der Kontakteordner 186
15.3.3 Kontaktinformationen auslesen 187
15.3.4 E-Mails an ausgewählte Empfänger senden 188
15.4 Schnellübersicht 191
15.5 Übung 192

Stichwortverzeichnis 194

1 Über diese Unterlage

In diesem Kapitel erfahren Sie

- → wie diese Schulungsunterlage aufgebaut ist
- → wie Sie die Beispiele und Übungen installieren
- → welche Kenntnisse für diese Schulungsunterlage vorausgesetzt werden

1.1 Voraussetzungen und Ziele

Zielgruppe

Diese Schulungsunterlage richtet sich an Office-Anwender, die bereits mit den wichtigsten Office-Anwendungen gearbeitet haben, und die nun bestimmte Tätigkeiten bzw. Aufgaben automatisieren wollen.

Empfohlene Vorkenntnisse

Für das Erstellen von integrierten Lösungen mit VBA sollten Sie über folgende Grundkenntnisse verfügen:

- ☑ Arbeiten mit der VBA-Entwicklungsumgebung
- ☑ Sprachelemente und Kontrollstrukturen von VBA
- ☑ Objektbasierte Programmierung mit VBA
- ☑ Programmieren von Ereignisprozeduren
- ☑ Grundwissen in den benötigten Office-Anwendungen

Verfügen Sie über Kenntnisse der Programmiersprache Visual Basic 6 (VB 6) oder deren Vorgängerversionen, können Sie diese in VBA nutzen. Visual Basic 6 und Visual Basic for Application gehören zur gleichen Technologiefamilie. Der Unterschied zwischen beiden besteht darin, dass VB eine Programmiersprache zur Entwicklung eigenständiger Windows-Anwendungen ist, VBA dagegen der Automatisierung und Anpassung der Anwendungsprogramme dient, in die es eingebettet ist.

Lernziele

Das Studium der Unterlage befähigt den Leser, bestimmte Aufgaben in den Teilanwendungen von Microsoft Office 2003 mithilfe des Objektmodells der jeweiligen Anwendung zu automatisieren.

Hinweise zu Soft- und Hardware

Die Schulungsunterlage bezieht sich auf Microsoft Office Professional Edition 2003. Für die Installation von Microsoft Office 2003 bestehen folgende Voraussetzungen:

- ☑ PC mit einem Intel Pentium Prozessor mit mindestens 233 MHz, Pentium III empfohlen
- ☑ Mindestens 128 MB RAM
- ☑ Mindestens 400 MB freier Speicherplatz auf der Festplatte, zusätzlich temporär ca. 300 MB während der Installation
- ☑ Windows 2000 mit Service Pack 3, Windows XP oder höher

1.2 Aufbau und Konventionen

Aufbau der Unterlage

- ☑ Am Anfang jedes Kapitels finden Sie die Lernziele und am Ende einiger Kapitel eine Schnellübersicht mit den wichtigsten Funktionen im Überblick.
- ☑ Die einzelnen Kapitel enthalten Übungen, mit deren Hilfe Sie die erlernten Kapitelinhalte einüben können.
- ☑ Die Notizfelder in der Unterlage geben Ihnen die Möglichkeit, eigene Anmerkungen und Erkenntnisse sowie praktische Arbeitstechniken einzutragen.

Inhaltliche Gliederung

In den Kapiteln 3 bis 6 werden die notwendigen VBA-Grundkenntnisse wiederholt und für den Einsatz in integrierten Lösungen vorbereitet.

In den Kapiteln 7 und 8 erhalten Sie einen Überblick über die Objektmodelle der wichtigen Office-Anwendungen.

In den Kapiteln 9 und 10 geht es um benutzerdefinierte Dialoge und um Datenbankzugriffe.

Falls Sie nur für bestimmte Office-Anwendungen integrierte Lösungen erstellen möchten, finden Sie in den Kapiteln 11 bis 15 detaillierte Anweisungen für das Erstellen integrierter Lösungen in den jeweiligen Anwendungen.

Typografische Konventionen

Im Text erkennen Sie bestimmte Programmelemente an der Formatierung. So werden beispielsweise Menüpunkte immer in GROSSBUCHSTABEN geschrieben.

KAPITÄLCHEN	kennzeichnen alle vom Programm vorgegebenen Bezeichnungen für Schaltflächen, Dialogfenster, Symbolleisten etc.
GROSSBUCHSTABEN	verweisen auf Menüs bzw. Menüpunkte (z. B. DATEI - SCHLIESSEN).
Kursivschrift	kennzeichnet alle vom Anwender zugewiesenen Namen wie Dateinamen, Ordnernamen, eigene Symbolleisten, Internetadressen etc.
`Courier New`	Programmtext, Programmnamen, Funktionsnamen, Variablennamen, Datentypen, Operatoren, etc.
`Courier New Kursiv`	Zeichenketten, die vom Programm ausgegeben oder ins Programm eingegeben werden
[ ]	Bei Darstellungen der Syntax kennzeichnen eckige Klammern ein optionales Element.
|	Bei Darstellungen der Syntax werden alternative Elemente durch einen senkrechten Strich voneinander getrennt.

Notizen

Symbole

Besondere **Information** zur Programmbedienung

Hinweis auf eine besonders **schnelle Möglichkeit**, eine Arbeit durchzuführen

Warnhinweis bei Aktionen, die **unerwünschte** Wirkungen haben können

Besonders **praktische** und einfache Möglichkeit, eine Aktion durchzuführen;

Tipp, wie eine Arbeit erleichtert werden kann

Übungsdateien

Die Beispiele und Übungen dieser Schulungsunterlage sind mit Microsoft Office Professional Edition 2003 unter Microsoft Windows XP Professional (Service Pack 1) entwickelt und getestet worden. Sie stehen im Internet unter *http://www.herdt.com* zum Download zur Verfügung. Die Beispiele sind in nach Kapiteln benannten Verzeichnissen (*..\Kap02*, *..\Kap03* usw.) abgelegt.

Bitte beachten Sie, dass es bei einem Teil der Beispiele und Übungen zu Problemen mit Zugriffsrechten kommen kann, wenn sie von einem Netzlaufwerk aus gestartet werden. Es ist daher zu empfehlen, die Beispiele und Übungen auf ein Laufwerk Ihres Rechners zu kopieren und sie von dort aus zu starten.

Internetadressen

Die in der Schulungsunterlage aufgeführten Internetadressen waren zum Zeitpunkt der Unterlagenerstellung gültige Adressen. Änderungen zu einem späteren Zeitpunkt können bewirken, dass Sie die Inhalte unter einer neuen Adresse finden. Es ist auch möglich, dass Inhalte komplett aus dem Internet genommen wurden.

Notizen

2 Einführung

In diesem Kapitel erfahren Sie

- → was VBA bedeutet und wozu sie es nutzen können
- → was integrierte Lösungen sind
- → welche speziellen Fähigkeiten die einzelnen Office-Anwendungen besitzen

2.1 Was bedeutet VBA?

Arbeitserleichterung durch Makros

Das Office-Paket enthält Anwendungen, mit denen Sie Aufgaben der Textverarbeitung, Tabellenkalkulation, Datenverarbeitung, Präsentation und Planung erledigen können. Sie können alle diese Anwendungen ohne Programmierkenntnisse einsetzen.

Im Praxiseinsatz der Office-Anwendungen treten häufig immer gleiche Handlungen auf, die auch automatisiert werden könnten, um Zeit zu sparen. Zu diesem Zweck wurden so genannte Makros eingeführt, die eine Folge von Tastaturanschlägen oder Mausbedienungen zu einer Einheit zusammenfassen konnten. Makros waren jedoch keine Entwicklungswerkzeuge für Programmierer, sondern nur Arbeitserleichterungen für die Anwender.

Im Laufe der Zeit wurde die Makro-Sprache der Office-Anwendungen erweitert und immer weiter zu einer relativ umfassenden Programmiersprache entwickelt. Mithilfe dieser Programmiersprache ist es nun auch möglich, die Office-Anwendungen um benutzerdefinierte Funktionen zu erweitern.

Programmierung mit VBA

Die umfangreichen Makro-Sprachen der einzelnen Office-Anwendungen (Excel-Makros, Word-Basic, Access-Basic) erlaubten ursprünglich keine einheitliche Programmierung im Office-Paket. Aus diesem Grund wurde später eine einheitliche Programmiersprache entwickelt und in alle Anwendungen integriert: Visual Basic for Applications (VBA).

VBA ist eine zweckorientierte Programmiersprache, die sich an Visual Basic (bis Version 6) anlehnt. Sie bietet über umfassende Objektmodelle Zugriff auf die Office-Anwendungen und erlaubt eine strukturierte und modulorientierte Programmierung. Auch wenn VBA speziell auf die Eigenschaften und Anforderungen für den Einsatz innerhalb einer Office-Anwendung ausgerichtet ist, verfügt es über viele Fähigkeiten und Merkmale einer allgemeinen Programmiersprache. So ist es z. B. mit VBA möglich, Dateien zu erstellen, zu löschen, zu kopieren, usw.

Um die Programmierung zu erleichtern, verfügt das Office-Paket über eine VBA-Entwicklungsumgebung, die als eigenständige Anwendung ausgeführt wird. Die Entwicklungsumgebung bietet Ihnen eine übersichtliche Arbeitsumgebung zum Programmieren und stellt Ihnen darüber hinaus noch einige Programmierhilfen und Arbeitserleichterungen zur Verfügung. Mit der Entwicklungsumgebung können Sie sowohl aufgezeichnete Makros bearbeiten als auch eigene Automatisierungs- und Erweiterungsroutinen erstellen.

Da VBA Microsoft-Forms und ActiveX-Steuerelemente unterstützt, können Sie eigene Dialogfenster und Benutzeroberflächen erstellen. Damit ist es beispielsweise möglich, die Dateneingabe über einen Dialog durchzuführen und Dokumente mit diesen Daten automatisch zu erzeugen.

Durch den Einsatz von VBA können Sie z. B.:

- ☑ Benutzerdefinierte Fehlermeldungen anzeigen
- ☑ Eigene Funktionen erstellen

- ☑ Daten aus anderen Anwendungen verwenden (z. B. aus Excel)
- ☑ Benutzerdefinierte Dialogfenster zur Abfrage verschiedener Eingabewerte oder zur Auswahl von Optionen erstellen
- ☑ Dialogfenster über Steuerelemente steuern
- ☑ System- bzw. Anwendungsinformationen auswerten (Benutzerdaten, Positionen von Grafiken, Dokumentstatistiken usw.)

2.2 Was sind integrierte Lösungen?

Jede Office-Anwendung ist auf ein bestimmtes Aufgabengebiet spezialisiert. So verfügt Word beispielsweise über weit reichende Funktionen zur Arbeit mit umfangreichen Texten. Excel besitzt dagegen viele Funktionen, um Datenbestände auszuwerten.

Bei der praktischen Arbeit kann es immer wieder vorkommen, dass Sie die Funktionen und Möglichkeiten einer anderen Anwendung benötigen. Möglicherweise wollen Sie Daten aus einer Datenbank in Excel grafisch auswerten oder Serienbriefe basierend auf einem Outlook-Adressbestand erstellen. In diesen Fällen können Sie von der Integrationsfähigkeit der Office-Anwendungen profitieren.

Alle Office-Anwendungen stellen ihre eigenen Funktionen als Objekte für andere Anwendungen zur Verfügung. Dieser Zustand wird als Automation-Server bezeichnet. Gleichzeitig erlauben alle Office-Anwendungen den Zugriff als so genannte Automation-Clients auf fremde Objekte (Automation-Server). Sie können daher beispielsweise in Word auf Excel-Objekte zugreifen und diese bei der Programmierung mit VBA verwenden wie in Excel selbst.

Integrierte Lösungen mit VBA helfen Ihnen, unter Verwendung mehrerer Office-Anwendungen, die Arbeit eines Anwenders zu vereinfachen. Integrierte Lösungen können jedoch auch verwendet werden, um fehlende Funktionen einer Anwendung durch die Funktionalität einer anderen Anwendung nachzurüsten.

Beispiele für integrierte Lösungen

- ☑ Über das Betätigen einer Schaltfläche können Sie eine Adresse aus den Outlook-Kontakten auswählen und in ein Word-Dokument einfügen.
- ☑ Sie importieren Umsatz- und Budgetzahlen aus einer Access-Datenbank, um Sie in Excel detailliert auszuwerten und um anschauliche Diagramme zu erstellen.
- ☑ Sie nutzen die Druckfunktion von Word, um mehrere Seiten einer umfangreichen Excel-Tabelle auf ein einziges Blatt Papier zu drucken.

Notizen

Wann ist eine Integration sinnvoll?

Eine integrierte Lösung ist immer dann sinnvoll und notwendig, wenn innerhalb einer Aufgabe mehrere Teilaufgaben existieren, für die bestimmte Office-Anwendungen durch ihre Spezialisierung besondere Vorteile bieten.

Viele Aufgabenstellungen lassen sich nur mithilfe von integrierten Lösungen in vertretbarem Aufwand erstellen. Sollen beispielsweise komplexe und miteinander verbundene Daten (z. B. Umsatz, Menge, Preis, Kosten) verwaltet und später grafisch aufbereitet werden, ist dies ohne eine Integration von Access und Excel nicht möglich. Access bietet als relationale Datenbank die Möglichkeit, große Mengen anfallender Daten zu speichern, und Excel verfügt über passende Funktionen zum grafischen Aufbereiten der Daten.

Integration mit und ohne VBA

Auch ohne Programmierung bieten die Office-Anwendungen einige Integrationsmöglichkeiten:

- ☑ Datenaustausch über die Zwischenablage
- ☑ Datenaustausch über XML
- ☑ Verknüpfen oder Einbetten von OLE-Objekten
- ☑ Datenzugriff mit MS Query
- ☑ Serienbrieffunktion von Word

Die Integrationsmöglichkeiten ohne Programmierung sind wenig flexibel. Mithilfe von VBA können Sie komplexe und speziell auf Ihre Bedürfnisse abgestimmte integrierte Lösungen erstellen. Einige besondere Möglichkeiten, wie z. B. ActiveX-Automation und ADO-Datenobjekte (vgl. Kapitel 6.3), können Sie nur über VBA-Prozeduren verwenden.

Ebenfalls möglich ist die Office-Automatisierung mithilfe der Visual Studio Tools für Office 2003, die von Microsoft angeboten werden. Diese Tools integrieren sich in Visual Studio .NET und erweitern unter anderem die Auswahl der Projekttypen auf Word-Dokumente und Excel-Arbeitsmappen. Voraussetzung für diese Art der Automatisierung ist außerdem das Beherrschen einer .NET-Programmiersprache (z. B. Visual Basic .NET, Visual C# .NET, Visual J# .NET).

2.3 Einsatzgebiete der Office-Anwendungen

Jede Office-Anwendung besitzt Einsatzgebiete, für die sie aufgrund ihrer Spezialisierung besonders geeignet ist. Mithilfe von integrierten Lösungen können Sie die spezialisierten Funktionen anwendungsübergreifend nutzen. Die folgende Übersicht zeigt Ihnen die besonderen Stärken jedes Office-Programms.

Word

Umfangreiche Dokumente	Das Haupteinsatzgebiet von Word ist das Verarbeiten umfangreicher Texte. Für diesen Zweck verfügt es über zahlreiche Formatierungsmöglichkeiten, z. B. Inhaltsverzeichnisse, Tabellen, Fußnoten, Gliederungen, Wörterbücher.
Seriendruck	Auf der Basis von Datenquellen wird das automatische Erstellen von Dokumenten mit eingefügten Datensätzen, insbesondere Adressen, erlaubt.

Excel

Tabellenkalkulation	Das Haupteinsatzgebiet von Excel ist das Analysieren, Verwalten und Aufbereiten von Daten. Excel erlaubt einen sehr flexiblen Umgang mit Daten.
Formeln	Excel enthält mehr als 500 eingebaute Funktionen und Formeln mit zum Teil sehr hoher Komplexität. Von einfachen mathematischen Berechnungen bis zu aufwändigen finanzmathematischen und statistischen Betrachtungen sind Formeln für viele Einsatzzwecke verfügbar.
Diagramme	Excel erlaubt das Erstellen und Formatieren von sehr vielen unterschiedlichen Diagrammarten.

Access

Datenspeicherung und -verwaltung	Access ist ein relationales Datenbankprogramm, mit dem sich komplexe Datenzusammenhänge erstellen und aktualisieren lassen. Mit einer Datenbank können Daten in mehreren Tabellen verknüpft gespeichert werden.
Programmierbare Formulare	Formulare in Access verfügen über umfangreiche Formatierungs- und Programmiermöglichkeiten. Mit Ereignissen wird der Programmablauf gesteuert.

PowerPoint

Folienpräsentation	Das Hauptaufgabengebiet von PowerPoint ist das Erstellen von Präsentationen. Dafür verwaltet es Folien, die ausgedruckt werden können oder als Bildschirmpräsentation angezeigt werden.
Gestalten von Drucksachen	PowerPoint erlaubt das grafische Gestalten von Drucksachen.

Outlook

Kommunikation	Outlook ermöglicht das Senden und Empfangen von E-Mails.
Planung	Outlook verfügt über viele Funktionen zur Planung der Arbeit. So können Termine und Aufgaben verwaltet werden.
Adressbücher	Outlook erlaubt das Speichern und Verwalten von Adressbeständen.

2.4 Ein einfaches Beispiel für eine integrierte Lösung

In der Buchführung wird häufig von einem Jahr mit 12 Monaten von je 30 Tagen ausgegangen. Beim Schreiben eines kaufmännischen Textes in Word kann es vorkommen, dass Sie die Anzahl von Tagen zwischen zwei Datumswerten basierend auf einem Jahr mit 360 Tagen benötigen. Word verfügt nicht über eine Funktion, mit der Sie diesen Wert direkt ermitteln können. Abhilfe schafft eine VBA-Prozedur, die eine geeignete Excel-Funktion aufruft.

Die Prozedur `AnzahlTage360` liest ein im Dokument ausgewähltes Datum aus und übergibt es zusammen mit dem aktuellen Datum an die Excel-Funktion `Day360`. Die ermittelte Anzahl von Tagen zwischen diesen beiden Daten wird danach in das Word-Dokument eingefügt und ersetzt das ausgewählte Datum.

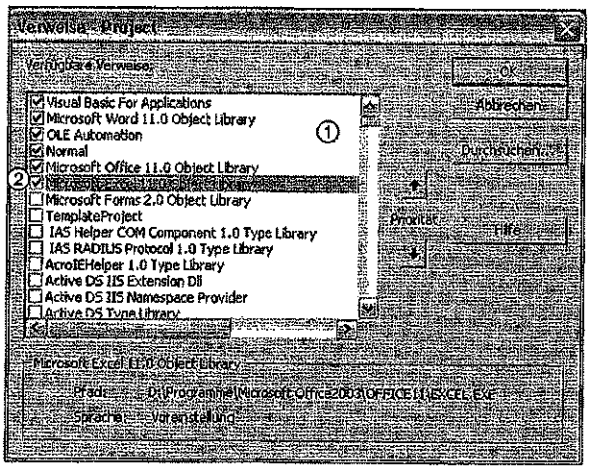
Für den Zugriff auf die Excel-Objekte benötigt VBA einen Verweis auf die Excel-Objektbibliothek:

⇒ Öffnen Sie in Word die VBA-Entwicklungsumgebung mit der Tastenkombination ALT F11.

⇒ Wählen Sie den Menüpunkt EXTRAS – VERWEISE.

⇒ Suchen Sie im Listenfeld ① den Eintrag MICROSOFT EXCEL 11.0 OBJECT LIBRARY und aktivieren Sie das Kontrollfeld ② in dieser Zeile.

⇒ Bestätigen Sie mit OK.



Der Verweis bezieht sich immer auf das aktuelle Dokument.

Beispiel: *AnzahlTage360.doc*, Modul `ExcelZugriff`

Die Prozedur ermittelt mithilfe von Excel den Wochentag für ein im Word-Dokument markiertes Datum.

```
Sub AnzahlTage360()
  Dim AppXL As Excel.Application
  Dim Datum As Date, Heute As Date, AnzahlTage As Integer
  Heute = Date
  Set AppXL = CreateObject("Excel.Application")
  Datum = DateValue(Selection.Text)
  AnzahlTage = AppXL.WorksheetFunction.Days360(Datum, Heute)
  Selection.TypeText AnzahlTage
  Set AppXL = Nothing
End Sub
```

① Für den Zugriff auf Excel wird eine Objektvariable benötigt. Über diese Variable kann auf die Excel-Objekte zugegriffen werden wie bei der Programmierung in Excel selbst.

② Die VBA-Funktion `Date` liefert das aktuelle Datum.

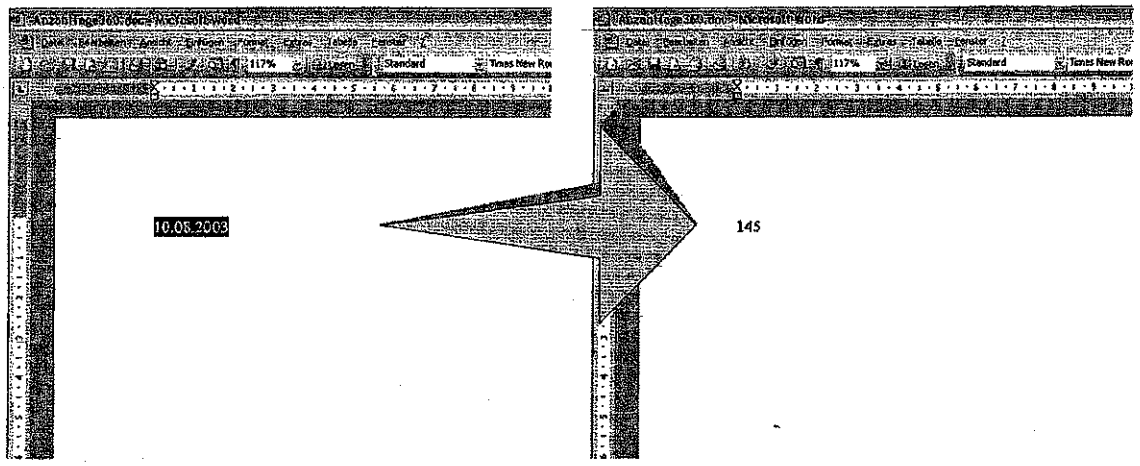
③ Mit der `CreateObject`-Funktion wird die Excel-Anwendung im Hintergrund gestartet. Danach sind alle Excel-Objekte verfügbar.

④ Der ausgewählte Text im Word-Dokument wird hier ausgelesen und in ein korrektes Datumsformat umgewandelt.

⑤ Die Objektvariable `AppXL` repräsentiert die Excel-Anwendung. Über das `WorksheetFunctions`-Objekt kann auf die Excel-Funktionen zugegriffen werden. Hier erfolgt der Aufruf der Funktion `Day360` mit dem im Dokument ausgelesenen Datum und dem aktuellen Datum als Argument.

⑥ Die `Day360`-Funktion liefert die Anzahl der Tage zwischen den beiden Daten. Dieser Wert wird an der aktuellen Cursorposition im Dokument eingefügt.

⑦ Die Excel-Anwendung wird nicht mehr benötigt. Durch das Zuweisen von `Nothing` wird die Anwendung geschlossen und der Speicherplatz freigegeben.



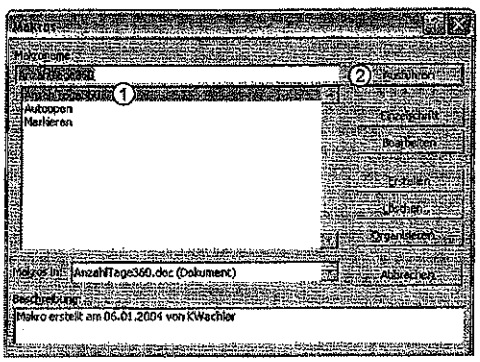
Die VBA-Prozedur liest ein markiertes Datum aus und fügt die Anzahl der Tage bis zum aktuellen Datum in kaufmännischer Form ein

⇒ Achten Sie darauf, dass die Datumsangabe markiert ist.

⇒ Starten Sie nun die Prozedur in Word über den Menüpunkt EXTRAS - MAKRO - MAKROS oder über die Tastenkombination ALT F8.

⇒ Markieren Sie im Dialog MAKROS die Prozedur `AnzahlTage360` ①.

⇒ Betätigen Sie die Schaltfläche AUSFÜHREN ②.



Das Ausführen von Makros in nicht selbst erstellten Dokumenten ist aus Sicherheitsgründen standardmäßig deaktiviert. Um dies zu ändern, wählen Sie den Menüpunkt EXTRAS - OPTIONEN - SICHERHEIT und aktivieren Sie im Dialogfeld MAKROSICHERHEIT die Option NIEDRIG. Um das Ausführen von Makro-Viren zu unterbinden, sollten Sie die Makro-Sicherheit danach wieder auf HOCH bzw. MITTEL einstellen.

2.5 VBA-Prozedur mit Menüleiste verknüpfen

Für einen schnellen Zugriff auf ein Makro kann es nützlich sein, die entsprechende VBA-Prozedur über eine Schaltfläche in der Menüleiste aufzurufen. So verknüpfen Sie die Prozedur mit einer Schaltfläche in der Menüleiste:

⇒ Wählen Sie den Menüpunkt EXTRAS - ANPASSEN. Das Dialogfenster ANPASSEN wird geöffnet.

oder Klicken Sie mit der rechten Maustaste auf die Symbolleiste. Wählen Sie den Kontextmenüpunkt ANPASSEN.

⇒ Wählen Sie im Listenfeld ③ den Titel des Dokuments oder der Vorlage, in dem oder in der das Makro enthalten ist.

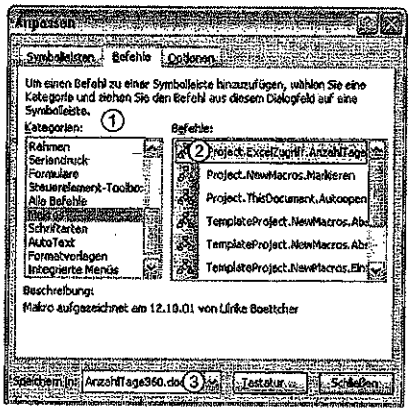
⇒ Wählen Sie im Listenfeld KATEGORIEN ① den Eintrag MAKROS. Im Listenfeld BEFEHLE ② werden die verfügbaren Prozeduren angezeigt.

⇒ Wählen Sie im Listenfeld BEFEHLE ② die Prozedur, die das Hauptmenü einblendet.

⇒ Ziehen Sie den Eintrag mit gedrückter linker Maustaste an die gewünschte Stelle in der Menüleiste.

⇒ Lassen Sie die linke Maustaste los, um den Eintrag in der Menüleiste zu erzeugen.

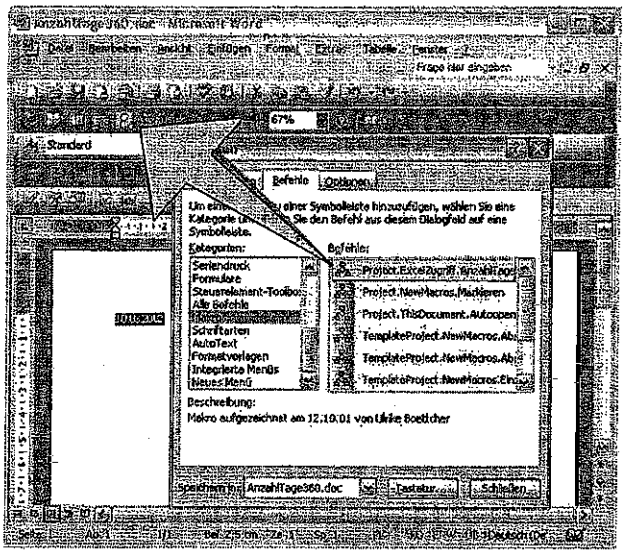
⇒ Lassen Sie das Dialogfenster ANPASSEN geöffnet, und klicken Sie mit der rechten Maustaste auf den neuen Eintrag in der Menüleiste. Das Kontextmenü des Menübefehls wurd geöffnet.



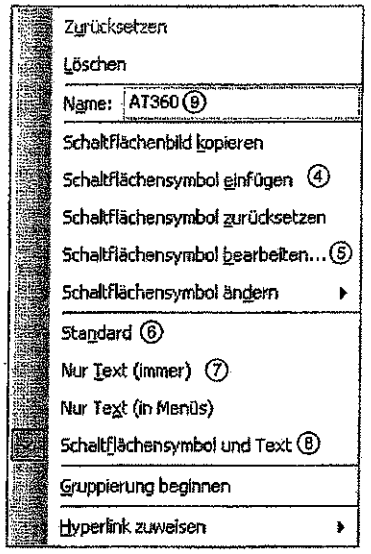
Das Dialogfenster ANPASSEN

Notizen

⇒ Im Kontextmenü können Sie ein Symbol einfügen ④ bzw. bearbeiten ⑤ und festlegen, ob nur das Symbol ⑥, nur der Text ⑦ oder ob beides ⑧ angezeigt werden soll. Außerdem können Sie im Eingabefeld NAME des Kontextmenüs die Beschriftung des Menübefehls ändern ⑨.



Menübefehl durch Ziehen mit der Maus einfügen



Ändern der Beschriftung im Kontextmenü

Das beschriebene Verfahren kann in dieser Form in allen Office-Anwendungen verwendet werden.

2.6 Schnellübersicht

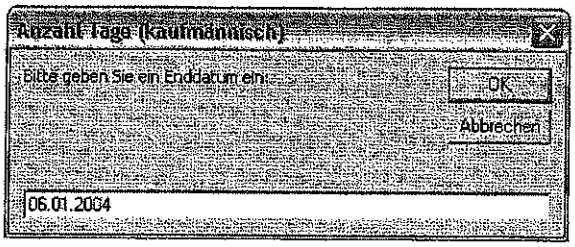
Was bedeutet ...	
VBA	Visual Basic for Applications; mit Visual Basic (bis Version 6) verwandte Programmiersprache der Office-Anwendungen
Prozeduren	Bestandteile eines VBA-Programms, die eine bestimmte Aufgabenstellung bearbeiten
Entwicklungsumgebung	Arbeitsfläche, in der (VBA-) Programme entwickelt werden; stellt Programmierhilfen und Programmierunterstützung zur Verfügung
integrierte Lösung	Eine oder mehrere VBA-Prozeduren, die auf fremde Office-Anwendungen zugreifen und deren Funktionen nutzen

2.7 Übung

Übungsdatei: *Übung1.doc* **Ergebnisdatei:** *Übung1E.doc*

① Öffnen Sie die Datei *Übung1.doc* und starten Sie die VBA-Entwicklungsumgebung.

② Basierend auf der Beispiel-Prozedur Anzahl Tage360 aus diesem Kapitel soll eine gleichnamige Prozedur entwickelt werden, die den Anwender in einem Eingabedialog nach einem Enddatum und gegebenenfalls nach einem Startdatum fragt, bevor sie die kaufmännische Datumsdifferenz ermittelt.



Der Eingabedialog zum Abfragen des Enddatums

③ Benennen Sie die Variablen `Datum` und `Heute` in `StartDatum` und `EndDatum` um.

④ Fügen Sie an geeigneter Stelle eine `If`-Abfrage ein, die mithilfe der Funktion `IsDate` prüft, ob es sich bei dem ausgewählten Text im Dokument um ein Datum handelt. Ist dies der Fall, weisen Sie es der Variablen `StartDatum` zu. Handelt es sich nicht um ein Datum, blenden Sie mithilfe der Funktion `InputBox` einen Eingabedialog für die Eingabe eines Startdatums ein.

⑤ Fügen Sie einen weiteren Eingabedialog für das Enddatum ein. Als Vorgabe verwenden Sie das aktuelle Datum, das Sie mit dem Befehl `Date` erhalten.

⑥ Fügen Sie wie in der Originalprozedur die Anzahl der Tage zwischen beiden Datumsangaben an der aktuellen Cursorposition ein.

Notizen

3 Arbeiten mit der VBA-Entwicklungsumgebung

In diesem Kapitel erfahren Sie

→ wie die VBA-Entwicklungsumgebung aufgebaut ist

→ wie Sie mit dem Modulfenster arbeiten

→ wie Sie Eingabehilfen beim Programmieren verwenden

Voraussetzungen

✓ Kenntnisse im Umgang mit den Office-Programmen

3.1 VBA-Entwicklungsumgebung verwenden

Die VBA-Programmierung erfolgt in allen Office-Anwendungen im Visual Basic-Editor. Dieser Editor ermöglicht nicht nur die Eingabe des Programmcodes, sondern stellt eine Vielzahl von Hilfsmitteln zur Verfügung (z. B. zum Ermitteln von Fehlern oder zum Finden verschiedener Objekte). Daher wird auch der Begriff VBA-Entwicklungsumgebung verwendet.

Öffnen der VBA-Entwicklungsumgebung

⇒ Starten Sie z. B. Word.

⇒ Wählen Sie den Menüpunkt EXTRAS - MAKROS - VISUAL BASIC-EDITOR.

Alternative: ALT + F11

Die VBA-Entwicklungsumgebung wird als eigenständige Anwendung geöffnet und erscheint damit nicht innerhalb der Office-Anwendungen.

Besonderheiten in Access

Zusätzlich zu den bereits beschriebenen Varianten wird die VBA-Entwicklungsumgebung automatisch geöffnet, wenn Sie im Access-Datenbankfenster ein Modul neu erstellen oder im Entwurfsmodus öffnen.

Access-Modul erstellen

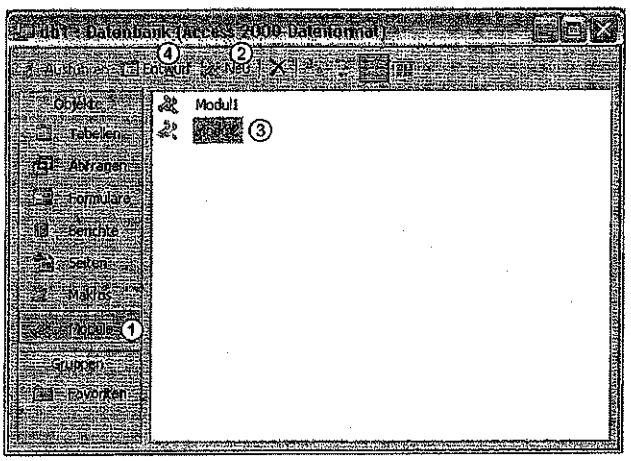
⇒ Wählen Sie im Datenbankfenster in der Auflistung OBJEKTE die Kategorie MODULE ①.

⇒ Betätigen Sie die Schaltfläche NEU ②.

Access-Modul bearbeiten

⇒ Klicken Sie im Datenbankfenster in der Auflistung OBJEKTE auf die Kategorie MODULE ①.

⇒ Markieren Sie in der Auflistung das Modul, das Sie bearbeiten möchten ③.



Das Datenbankfenster in Access

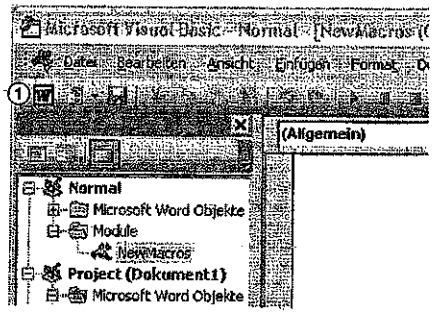
⇒ Betätigen Sie die Schaltfläche ENTWURF ④.
Alternativen: ANSICHT - CODE oder

oder

⇒ Klicken Sie doppelt auf das Modul, das Sie bearbeiten möchten.

Die VBA-Entwicklungsumgebung verlassen

⇒ Wählen Sie in der Symbolleiste VOREINSTELLUNG das Symbol ①. Das Symbol zeigt die Office-Anwendung, aus der Sie die VBA-Entwicklungsumgebung aufgerufen haben.
Alternative: ALT F11



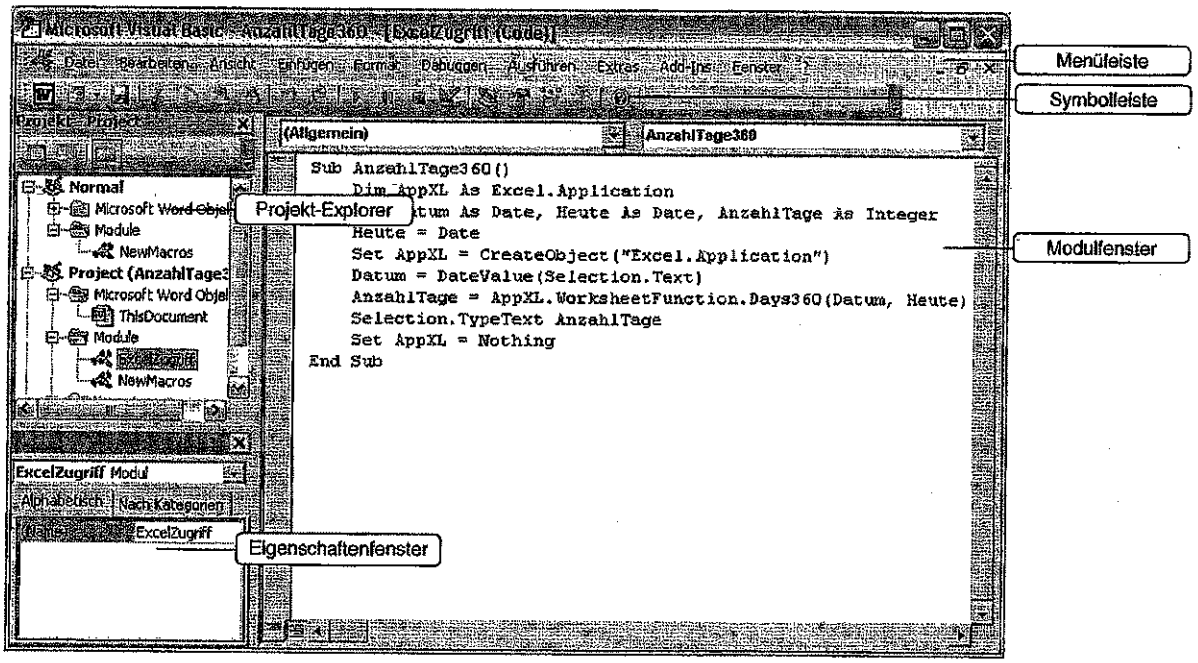
VBA-Entwicklungsumgebung in Word

Mit dem Symbol bzw. der Tastenkombination wechseln Sie zur Ausgangsanwendung. Die VBA-Entwicklungsumgebung bleibt dabei jedoch im Hintergrund geöffnet. Um die Entwicklungsumgebung zu schließen, gehen sie folgendermaßen vor:

⇒ Wählen Sie in der Entwicklungsumgebung den Menüpunkt DATEI - SCHLIESSEN UND ZURÜCK ZU MICROSOFT WORD.
Alternative: ALT Q

3.2 Bestandteile der Entwicklungsumgebung

Die VBA-Entwicklungsumgebung stellt Ihnen zahlreiche Funktionen zur Verfügung, die Sie bei der Programmierung unterstützen. Die Entwicklungsumgebung besteht aus einer Anzahl von Fenstern, die zum Teil standardmäßig geöffnet sind (z. B. das Modulfenster). Einige Fenster und Hilfsmittel erreichen Sie erst über Menübefehle (z. B. den Objektkatalog).

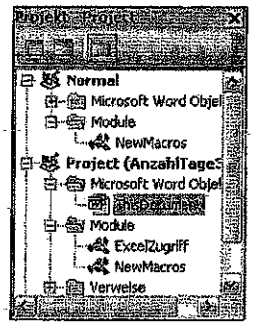


Die VBA-Entwicklungsumgebung (Visual-Basic-Editor)

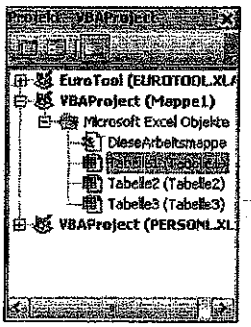
3.3 Der Projekt-Explorer

Im Projekt-Explorer werden alle Bestandteile des gerade bearbeiteten Projekts hierarchisch geordnet (Baumform) dargestellt.

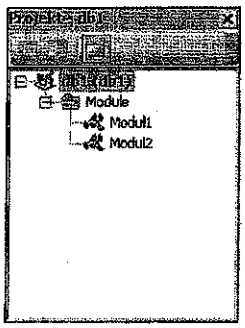
Je nach Office-Anwendung werden unterschiedliche Projektbestandteile angezeigt. Dazu gehören alle geöffneten Dokumente bzw. Arbeitsmappen sowie Module, die Prozeduren aufnehmen. Auch die in Excel und Word automatisch geladenen Add-Ins und Vorlagen gehören zu einem Projekt.



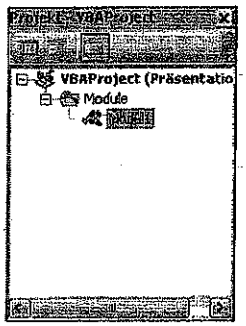
Projekt-Explorer in Word



... in Excel



... in Access



... in PowerPoint

Den Projekt-Explorer einblenden

⇒ Falls der Projekt-Explorer nicht angezeigt wird, können Sie ihn über den Menüpunkt ANSICHT - PROJEKT-EXPLORER einblenden.
Alternativen: STRG R oder

Möglichkeiten im Projekt-Explorer

	Code anzeigen	Zeigt im Modulfenster den Programmcode des ausgewählten Elements an Alternativen: ANSICHT - CODE oder F7
	Objekt anzeigen	Öffnet das zum ausgewählten Element gehörige Objekt (z. B. UserForm-Dialog, Formular, Word-Dokument, Excel-Arbeitsmappe). Bei Modulen ist der Befehl nicht verfügbar. Alternativen: ANSICHT - OBJEKT oder ⇧ UMSCHALTEN F7
	Ordner wechseln	Blendet die Objektordner ein bzw. aus. Die Ordneransicht verbessert besonders bei umfangreichen Projekten die Übersicht.

Um schnell den Programmcode eines im Projekt-Explorer angezeigten Objekts (z. B. eines Moduls) zu bearbeiten, klicken Sie doppelt auf das gewünschte Objekt in der Liste.

Elemente zum Speichern von Prozeduren

Folgende Elemente eines Projekts können VBA-Anweisungen enthalten:

Anwendung	Element	Geltungsbereich der VBA-Anweisungen
Word	Dokument	dieses Dokument
	Dokumentvorlage	alle Dokumente, die auf dieser Vorlage basieren bzw. die mit dieser Vorlage verknüpft sind
	Module	dieses Modul; die Prozeduren können vom aktuellen Dokument aus aufgerufen werden

Excel	Arbeitsmappe	gesamte Arbeitsmappe mit enthalten Tabellenblättern
	Tabellenblatt	nur dieses Tabellenblatt
	Diagrammblatt	nur dieses Diagrammblatt (wenn das Diagramm einzeln angelegt ist)
	Modul	dieses Modul; die Prozeduren können von anderen Stellen aus aufgerufen werden
Access	Module	dieses Modul; die Prozeduren können von anderen Stellen aus aufgerufen werden
	Formulare, Berichte	nur in diesem Formular bzw. Bericht
PowerPoint	Präsentation	die gesamte Präsentation
	Module	dieses Modul; die Prozeduren können von anderen Stellen aus aufgerufen werden

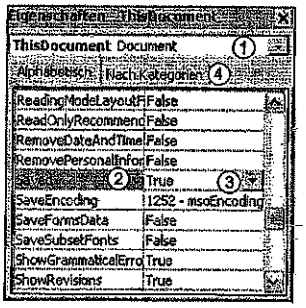
Der VBA-Programmcode kann an ganze Dateien (z. B. Dokumente, Arbeitsblätter) oder an Dateielemente (z. B. Tabellenblätter) gebunden werden. Dabei wird er in Modulen gespeichert, die nur für das jeweilige Element zur Verfügung stehen und von VBA automatisch verwaltet werden.

3.4 Das Eigenschaftenfenster

Dieses Fenster zeigt alle zur Entwurfszeit einstellbaren Eigenschaften des im Projekt-Explorer ausgewählten Objekts bzw. die Eigenschaften der Prozedur im Modulfenster an. Bei Auswahl mehrerer Objekte werden im Eigenschaftenfenster diejenigen Eigenschaften angezeigt, die alle ausgewählten Elemente aufweisen.

Im Listenfeld ① wird der Name des ausgewählten Objekts angezeigt. In der linken Spalte ② sind die Eigenschaften aufgelistet, in der rechten Spalte ③ können die zugehörigen Werte angegeben bzw. verändert werden.

Über das Register ④ können Sie die Eigenschaften entweder nach Kategorien oder alphabetisch sortiert anzeigen lassen.



Eigenschaftenfenster

Das Eigenschaftenfenster einblenden

Falls das Eigenschaftenfenster nicht angezeigt wird, können Sie es einblenden, indem Sie den Menüpunkt ANSICHT - EIGENSCHAFTENFENSTER wählen.

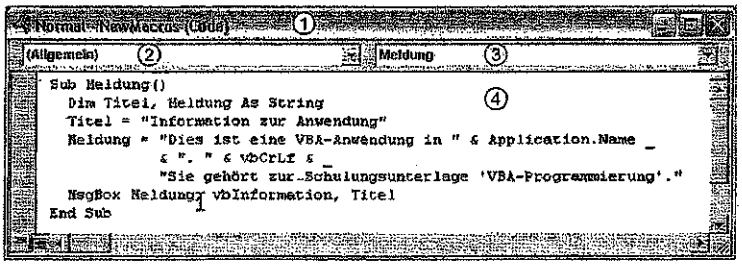
Alternativen: F4 oder

Notizen

3.5 Das Modulfenster benutzen

3.5.1 Bestandteile des Modulfensters

Im Modulfenster (auch Codefenster genannt) erfolgt die eigentliche Programmierung. Hier geben Sie die Anweisungen ein, die von Ihrer Anwendung ausgeführt werden sollen. Falls Sie ein bestehendes Modul geöffnet haben, wird der vollständige Programmcode angezeigt.



Das Modulfenster

Das Meldungsfenster der Beispielprozedur in Word

In der oben stehenden Abbildung wurde das Modul MODUL1 geöffnet. Darin ist die angezeigte Prozedur `InfoProzedur()` enthalten, die Informationen zur aktuellen Anwendung anzeigt.

Modulname ①	Der Modulname bezeichnet jedes Modul eindeutig. Bei einem Standardmodul können Sie ihn frei wählen, bei einem Klassenmodul wird er von der jeweiligen Office-Anwendung vorgegeben.
Listenfeld OBJEKT ② und **Listenfeld PROZEDUR** ③	Diese Listenfelder sind nur von Bedeutung, wenn Sie Ereignisprozeduren bearbeiten. Dann wählen Sie im Listenfeld ② das Objekt, dessen Ereignisse dann im Listenfeld ③ angezeigt werden. Ist im Listenfeld OBJEKT ② der Eintrag ALLGEMEIN ausgewählt, werden alle bisher erstellten Prozeduren in diesem Modul im Listenfeld ③ angezeigt.
Editor-Bereich ④	Hier schreiben Sie Ihre Anweisungen, die der Editor nach vorgegebenen Regeln automatisch formatiert.

Da sich der VBA-Programmcode meist auf verschiedene Module verteilt (z. B. durch Anweisungen im Klassenmodul eines Dokuments, eigenständige Module usw.), ist es schwierig herauszufinden, in welchem Modul Sie sich gerade befinden. In der Titelzeile des Modulfensters bzw. der VBA-Entwicklungsumgebung können Sie zu Ihrer Unterstützung ablesen, um was für ein Modul es sich handelt.

MODUL1 (CODE)	Normales Modul mit dem Namen MODUL1 (in allen Office-Anwendungen)
USERFORM1 (CODE)	Klassenmodul eines UserForm-Dialogs mit dem Namen USERFORM1
THISDOCUMENT (CODE)	Klassenmodul eines Word-Dokuments
FORM_FORMULAR1 (CODE)	Klassenmodul eines Access-Formulars mit dem Namen FORMULAR1
DIESEARBEITSMAPPE (CODE)	Klassenmodul einer Excel-Arbeitsmappe
TABELLE1 (CODE)	Klassenmodul eines Excel-Tabellenblatts mit dem Namen TABELLE1

3.5.2 Anweisungen eingeben und ausführen

Anweisungen eingeben

Mit dem Modulfenster können Sie wie mit einem normalen Texteditor arbeiten. Wie unter Windows gewohnt, haben Sie die Möglichkeit, die Zwischenablage zu benutzen. Die üblichen Tastenkombinationen (z. B. STRG C zum Kopieren) stehen ebenfalls zur Verfügung. Auch mit der Maus können Sie wie üblich Textabschnitte markieren und verschieben. Einige der in den Office-Anwendungen verwendeten Tastenkombinationen führen im Modulfenster jedoch zu einer anderen Aktion. So löschen Sie z. B. mit der Tastenkombination STRG Y eine Zeile, statt eine rückgängig gemachte Aktion zu wiederholen.

Um den Programmcode übersichtlich zu gestalten, können Sie Programmzeilen nach rechts einrücken. Verwenden Sie dazu die Tabulatortaste ⇥. Wenn Sie am Zeilenende die Eingabetaste RETURN ↵ betätigen, wird der Cursor automatisch an die eingerückte Position gesetzt.

Möchten Sie gleichzeitig mehrere Zeilen einrücken, markieren Sie zuerst die betreffenden Zeilen und betätigen Sie danach die Taste ⇥.

Die nachfolgende Tabelle zeigt weitere Bearbeitungsmöglichkeiten:

Letzte Aktion rückgängig machen	⇒ Wählen Sie den Menüpunkt BEARBEITEN - RÜCKGÄNGIG. Alternativen: ↶ oder STRG Z oder ALT ← KORREKTUR Sie können bis zu 20 Arbeitsschritte rückgängig machen.
Letzte Aktion wiederholen	⇒ Rufen Sie den Menüpunkt BEARBEITEN - WIEDERHOLEN auf. Der zuvor rückgängig gemachte Arbeitsschritt wird wiederholt. Alternative: ↷
Wörter suchen bzw. ersetzen	⇒ Wählen Sie den Menüpunkt BEARBEITEN - SUCHEN bzw. BEARBEITEN - ERSETZEN. Alternativen: STRG F bzw. STRG H
Programmcode drucken	⇒ Rufen Sie den Menüpunkt DATEI - DRUCKEN auf. Alternative: STRG P

Prozedur starten

Um eine Prozedur zu testen, können Sie diese direkt aus der Entwicklungsumgebung heraus ausführen.

⇒ Klicken Sie mit der Maus auf eine Stelle innerhalb der Prozedur, z. B. auf das Schlüsselwort Sub.
⇒ Wählen Sie den Menüpunkt AUSFÜHREN - SUB/USERFORM AUSFÜHREN.
Alternativen: F5 oder ▷

3.5.3 Automatische Eingabehilfen verwenden

Sobald Sie eine Zeile mit RETURN ↵ abschließen oder den Cursor in eine andere Zeile setzen, überprüft VBA automatisch die Anweisungen dieser Zeile. Folgende Arbeiten führt VBA dabei automatisch durch:

- ☑ Schlüsselwörter beginnen mit einem Großbuchstaben, gefolgt von Kleinbuchstaben.
- ☑ Zusätzliche Leerzeichen werden eingefügt (z. B. vor und nach dem Gleichheitszeichen), um die Lesbarkeit zu verbessern.
- ☑ Verschiedene Sprachelemente werden farbig markiert.
- ☑ Die Syntax der Anweisungen wird überprüft und gegebenenfalls eine Fehlermeldung eingeblendet.

Notizen

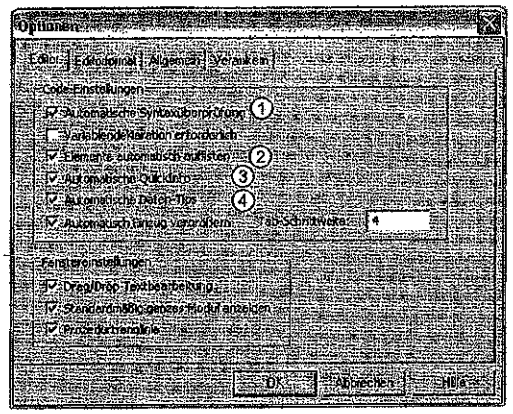
Die folgenden Eingabehilfen werden bereits während der Eingabe aktiv:

- ☑ Eine automatische Direkthilfe zur eingegebenen Anweisung wird eingeblendet (falls verfügbar).
- ☑ Verfügbare Elemente eines Objekts werden zur einfachen Auswahl per Listenfeld eingeblendet.

Über einige dieser Eingabehilfen erfahren Sie im Folgenden mehr.

Sie können die einzelnen Eingabehilfen auch gezielt ein- oder ausschalten:

⇨ Wählen Sie in der VBA-Entwicklungsumgebung den Menüpunkt EXTRAS - OPTIONEN. Das Dialogfenster OPTIONEN wird eingeblendet.



Einstellen der Editor-Optionen

- ☑ Mit dem Kontrollfeld ① schalten Sie die automatische Syntaxüberprüfung aus oder ein. Sie sollte eingeschaltet bleiben, denn Sie werden so schnell über kleine Tippfehler informiert.
- ☑ Bei ② wählen Sie aus, ob während des Schreibens automatisch eine Liste der verfügbaren Elemente einer Anweisung eingeblendet wird.
- ☑ Das Kontrollfeld ③ legt fest, ob Sie eine kurze Syntaxinformation zur eingegebenen Anweisung (QuickInfo) erhalten.
- ☑ Um während des Testvorgangs den Wert einer Variablen zu überprüfen, aktivieren Sie das Kontrollfeld ④.

Farbige Formatierung des Programmcodes

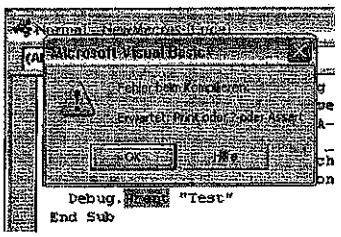
Im Modulfenster werden die VBA-Anweisungen standardmäßig wie folgt formatiert:

Kommentare	Kommentare werden durch ein vorangestelltes einfaches Anführungszeichen (') oder durch das Wort `Rem` gekennzeichnet. Dies kann am Anfang einer Zeile geschehen oder auch nach einer Anweisung. Der nachfolgende Text in der Zeile wird nicht beachtet. Kommentare werden automatisch grün formatiert.
Schlüsselwörter	Schlüsselwörter sind Bestandteile der VBA-Programmiersprache und dürfen nicht als Variablennamen verwendet werden (z. B. `Sub` und `Dim`). Sie werden blau formatiert.
Anweisungen	Befehle, Variablen, Konstanten usw. werden schwarz formatiert.
Haltepunkte	Codezeilen, an denen Haltepunkte zur Fehlersuche gesetzt wurden, sind braun unterlegt und zusätzlich durch einen Punkt am Rand gekennzeichnet.
Fehlerhafter Code	Syntaktisch fehlerhafter Code, z. B. `Debug.Prant` statt `Debug.Print`, wird rot dargestellt.

Automatische Syntaxüberprüfung

Durch die Syntaxüberprüfung werden Sie schon bei der Eingabe auf Fehler im Programmcode hingewiesen. VBA kann auf diese Weise allerdings nur Syntaxfehler ausfindig machen, wie z. B. fehlende schließende Klammern oder falsch geschriebene Anweisungen. Laufzeitfehler, die sich erst bei der Ausführung ergeben (z. B. durch unverträgliche Datentypen), zeigt VBA hier nicht an.

Wenn ein Fehler entdeckt wird, formatiert VBA die betreffenden Anweisungen rot und blendet ein Informationsfenster ein.



Die Syntaxüberprüfung in Aktion

Automatische Direkthilfe (QuickInfo)

Wenn Sie den Namen einer Prozedur, Funktion, Methode oder Eigenschaft, gefolgt von einem Punkt, einer Klammer oder einem Leerzeichen eingeben, wird die automatische Direkthilfe aktiviert. VBA zeigt dann unterhalb der aktuellen Zeile die vollständige Syntax des Elements als QuickInfo an.

Die Direkthilfe in Aktion

Sie erhalten so eine schnelle Information zur Programmierung und erfahren die Reihenfolge der einzelnen Übergabeparameter.

Die Direkthilfe funktioniert auch für selbsterstellte Prozeduren. Auch zu diesen zeigt VBA die Definition an. Falls die Direkthilfe ausbleibt, wissen Sie sofort, dass Sie eine Anweisung nicht richtig geschrieben haben.

Elemente automatisch auflisten

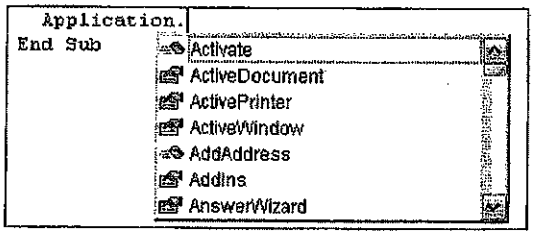
Diese Funktion listet zu einem Parameter oder einem Objekt, falls möglich, alle verfügbaren Elemente auf. Dies sind z. B. alle passenden Konstanten oder bei einem Objekt die Namen aller Eigenschaften und Methoden. Dadurch bleibt Ihnen das Lernen der vielen Elementnamen erspart.

⇒ Schreiben Sie den Namen eines Objekts in das Modulfenster (z. B. `Application`).

⇒ Geben Sie nun als Trennungszeichen für das nachfolgende Element einen Punkt ein.

VBA erkennt das Objekt `Application` und öffnet unterhalb der Zeile ein Listenfeld, das alle für dieses Objekt verfügbaren Methoden und Eigenschaften enthält.

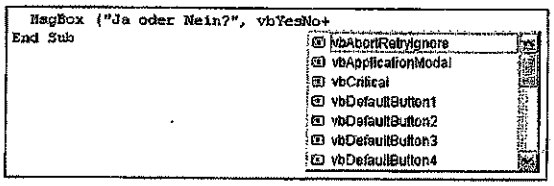
⇒ Wählen Sie das gewünschte Element mit den Pfeiltasten (↑ ↓), gefolgt von der Tabulatortaste (⇆), aus.



Die Funktion ELEMENTE AUTOMATISCH AUFLISTEN in Aktion

Die Werte einiger Parameter setzen sich aus mehreren Konstanten zusammen. Dies kann beispielsweise bei der Funktion `MsgBox` genutzt werden, um die verfügbaren Schaltflächen des Meldungsfensters und das angezeigte Symbol zu bestimmen.

Zur Eingabe solcher Konstanten wählen Sie zunächst die erste Konstante aus dem Listenfeld und betätigen anschließend die Taste [+]. VBA öffnet das Listenfeld nun erneut, und Sie können die zweite Konstante auswählen. Haben Sie die Eingabe des gesamten Parameters beendet, betätigen Sie zum Abschluss die Taste ⇆.



Eingeben zusammengesetzter Parameter

Notizen

Weitere Eingabehilfen

Im Menü BEARBEITEN finden Sie noch weitere Hilfen, die Ihnen das Eingeben von Programmcode erleichtern. Die folgende Tabelle erklärt diese Menüpunkte, die sich immer auf den Begriff an der aktuellen Cursorposition beziehen. Dazu müssen Sie nicht einen kompletten Begriff eingeben, einige Buchstaben genügen.

Eigenschaften/Methoden anzeigen	Öffnet ein Listenfeld mit alternativen Elementen zu dem Begriff; dies können Eigenschaften, Methoden oder Konstanten sein, die sich aus dem Programmkontext ergeben. Alternative: STRG J
Konstanten anzeigen	Zeigt alle passenden Konstanten zu dem Begriff Alternative: STRG ⇧ UMSCHALTEN J
QuickInfo	Zeigt die Syntax für Variablen, Funktionen, Prozeduren, Anweisungen und Methoden Alternative: STRG I
ParameterInfo	Zeigt im Gegensatz zur QuickInfo nur Informationen zu den Parametern einer Funktion oder Anweisung Alternative: STRG ⇧ UMSCHALTEN I
Wort vervollständigen	Falls VBA den gewünschten Bezeichner erkennen kann, ergänzt es automatisch den Rest des eingegebenen Begriffes. Anderenfalls wird ein Listenfeld mit den möglichen Bezeichnern geöffnet. Alternative: STRG LEERTASTE

3.5.4 Neue Prozedur erstellen

Im Modulfenster können Sie den Programmcode direkt eingeben, beispielsweise, um eine Lösung aus der Fachliteratur zu erproben. Die VBA-Entwicklungsumgebung kann zur Vereinfachung jedoch auch das Codegerüst für eine neue Prozedur erzeugen:

⇒ Wählen Sie den Menüpunkt EINFÜGEN PROZEDUR.

Die Entwicklungsumgebung blendet das Dialogfenster PROZEDUR HINZUFÜGEN ein.

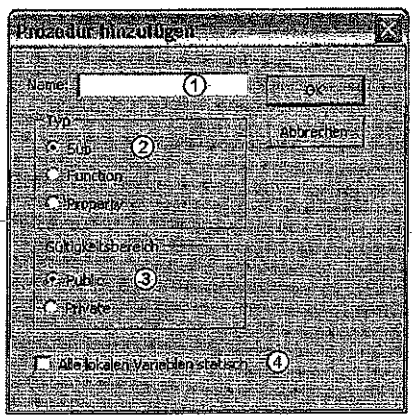
⇒ Geben Sie in das Feld ① einen Namen für die neue Prozedur bzw. Funktion ein.

⇒ Mit dem Optionsfeld ② legen Sie fest, ob Sie eine Sub-, eine Function- oder eine Property-Prozedur erstellen möchten.

⇒ Das Optionsfeld ③ legt den Gültigkeitsbereich der Prozedur fest. Dabei steht PUBLIC für eine öffentliche Gültigkeit der Prozedur, d. h., sie ist in allen Modulen bekannt. Wählen Sie dagegen PRIVATE, kann die Prozedur nur im aktuellen Modul angesprochen werden.

⇒ Aktivieren Sie das Kontrollfeld ④, falls Sie die Variablen der Prozedur als statische Variablen deklarieren möchten. Diese Variablen behalten ihren Wert über verschiedene Aufrufe der Prozedur hinweg.

⇒ Bestätigen Sie die Einstellungen mit Ok.



Dialogfenster PROZEDUR HINZUFÜGEN

Die Entwicklungsumgebung fügt nun die benötigten Anweisungen für eine neue Prozedur ein. Geben Sie jetzt die geplanten Programmanweisungen zwischen `Sub` und `End Sub` ein.

Beispiel: *Meldung.doc*, Modul `ThisDocument`

Es soll eine Prozedur erstellt werden, die Informationen über die aktuelle VBA-Anwendung anzeigt. Sie soll dabei auch anzeigen, in welchem Office-Programm sie aufgerufen wurde.

Ausgabe der Beispielprozedur in Word

Erstellen Sie die folgende Prozedur, und führen Sie diese aus.

```
Sub Meldung()
  Dim Titel, Meldung As String
  Titel = "Information zur Anwendung"
  Meldung = "Dies ist eine VBA-Anwendung in " & Application.Name & ". " & _
            "Sie gehört zur Schulungsunterlage 'VBA-Programmierung'."
  MsgBox Meldung, vbInformation, Titel
End Sub
```

① Für die Variablen `Titel` und `Meldung` wird der Datentyp `String` festgelegt, um jeweils eine Zeichenkette aufzunehmen.

② Der Variablen `Titel` wird eine Zeichenkette mit dem Titel des Meldungsfensters zugewiesen.

③ Die Variable `Meldung` enthält den anzuzeigenden Meldungstext. Mit der Eigenschaft `Name` des Objekts `Application` wird die aktuelle Office-Anwendung ermittelt.

④ Diese Anweisung bewirkt das Anzeigen des Meldungsfensters mit einem Informationssymbol und mit der OK-Schaltfläche.

3.5.5 Prozeduren bearbeiten

Prozeduren ändern

⇒ Wählen Sie in der Office-Anwendung den Menüpunkt EXTRAS - MAKRO - VISUAL BASIC EDITOR, um die VBA-Entwicklungsumgebung zu öffnen.
Alternative: ALT F11

⇒ Das aktuelle Modul wird im Modulfenster angezeigt. Wenn Sie ein anderes bearbeiten möchten, klicken Sie im Projekt-Explorer doppelt auf das gewünschte Element.

⇒ Nehmen Sie im Modulfenster die gewünschten Änderungen vor.

Prozeduren testen

Im Normalfall werden die Prozeduren ausgeführt, wenn sie von einer anderen Prozedur oder einem Formular (UserForm) aus aufgerufen werden. Dies erfolgt beispielsweise bei der ereignisgesteuerten Programmierung durch das Klicken auf eine Schaltfläche.

Notizen

Sie können die Prozedur jedoch zu Testzwecken auch innerhalb der VBA-Entwicklungsumgebung starten.

⇒ Stellen Sie sicher, dass sich der Cursor innerhalb der Prozedur im Modulfenster befindet.
⇒ Wählen Sie den Menüpunkt AUSFÜHREN - SUB/USERFORM AUSFÜHREN.
Alternativen: F5 oder ▶

Definition einer aufgerufenen Prozedur anzeigen

Um eine Prozedur anzuzeigen, die in einer anderen Prozedur aufgerufen wird, gehen Sie wie folgt vor:

⇒ Setzen Sie den Cursor in den Namen der aufgerufenen Prozedur.
⇒ Wählen Sie den Menüpunkt ANSICHT - DEFINITION.
Alternativen: ⇧ UMSCHALTEN F2 oder Kontextmenüpunkt DEFINITION

3.5.6 Tipps zum Arbeiten mit dem Modulfenster

Ansicht des Moduls ändern

Sie können wählen, ob gleichzeitig alle Prozeduren eines Moduls im Modulfenster angezeigt werden sollen oder immer nur eine.

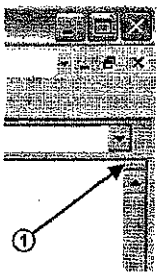
⇒ Klicken Sie bei geöffnetem Modulfenster auf die Symbole ☰ und ☰ am linken Rand der horizontalen Bildlaufleiste, um zwischen der Einzelprozeduransicht und der vollständigen Modulansicht zu wechseln.
⇒ Bei aktivierter Einzelprozeduransicht wechseln Sie mit STRG ↑ bzw. STRG ↓ zur vorherigen bzw. nachfolgenden Prozedur.

Sie können die Prozedur, die Sie bearbeiten möchten, auch direkt im Listenfeld PROZEDUR des Modulfensters auswählen.

Modulfenster in zwei Bereiche teilen

Möchten Sie Prozeduren gleichzeitig an unterschiedlichen Stellen ansehen, können Sie das Modulfenster teilen. Im oberen und unteren Fensterbereich können Sie dann zwei verschiedene Stellen einer Prozedur oder zwei verschiedene Prozeduren bearbeiten.

⇒ Positionieren Sie den Mauszeiger an die Stelle des Fensterteilers oberhalb der vertikalen Bildlaufleiste ①. Der Mauszeiger verwandelt sich in das Zeichen ÷.
⇒ Ziehen Sie den Fensterteiler bei gedrückter Maustaste nach unten.
VBA teilt das Fenster nun in zwei Bereiche.
Alternative: Doppelklick auf den Fensterteiler zum Teilen in zwei gleich große Bereiche



oder

⇒ Wählen Sie den Menüpunkt FENSTER - TEILEN.

Um zwischen den beiden Teilfenstern zu wechseln, gehen Sie wie folgt vor:

⇒ Klicken Sie in den jeweiligen Bildschirmbereich, oder betätigen Sie F6, um zwischen den beiden Fensterinhalten zu wechseln.

Ein Doppelklick auf den Fensterteiler beendet die Teilung.

Lesezeichen verwenden

Sie haben die Möglichkeit, an jeder beliebigen Stelle des Moduls ein Lesezeichen zu setzen. Dadurch wird das Auffinden einer Programmzeile wesentlich leichter, da Sie mit einem Befehl die auf diese Weise markierten Zeilen der Reihe nach anspringen können.

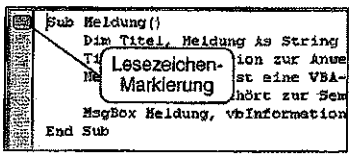
Lesezeichen können Sie auch mithilfe von Symbolen einfügen und löschen. Blenden Sie dazu die Symbolleiste BEARBEITEN ein.

Lesezeichen setzen

⇒ Setzen Sie den Cursor in die Zeile, auf die Sie ein Lesezeichen setzen möchten.

⇒ Wählen Sie den Menüpunkt BEARBEITEN - LESEZEICHEN - LESEZEICHEN SETZEN/ZURÜCKSETZEN.

Alternative: [F9]



Lesezeichen setzen

Die Entwicklungsumgebung markiert die betreffende Zeile durch ein hellblaues Rechteck am linken Rand.

Zwischen Lesezeichen wechseln

Haben Sie mehrere Lesezeichen gesetzt, können Sie zum nächsten bzw. vorherigen Lesezeichen wechseln.

⇒ Wählen Sie den Menüpunkt BEARBEITEN - LESEZEICHEN - NÄCHSTES LESEZEICHEN bzw. BEARBEITEN - LESEZEICHEN - VORHERIGES LESEZEICHEN.

Alternative: [Strg]+[F9] bzw. [Strg]+[Umschalt]+[F9]

Lesezeichen löschen

⇒ Setzen Sie den Cursor in die Zeile, in der Sie das Lesezeichen löschen wollen.

⇒ Wählen Sie den Menüpunkt BEARBEITEN - LESEZEICHEN - LESEZEICHEN SETZEN/ZURÜCKSETZEN.

Alternative: [F9]

oder

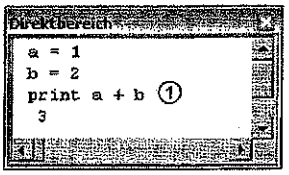
⇒ Wählen Sie den Menüpunkt BEARBEITEN - LESEZEICHEN - ALLE LESEZEICHEN LÖSCHEN.

Notizen

3.6 Das Direktfenster

Mit dem Direktfenster haben Sie die Möglichkeit, VBA-Anweisungen direkt einzugeben und auszuführen. So können Sie zu Testzwecken beispielsweise eine Anweisung ausführen oder während der Laufzeit Variableninhalte und Eigenschaften abfragen. Dies können Sie zur Fehlersuche einsetzen.

⇒ Wählen Sie den Menüpunkt ANSICHT - DIREKTFENSTER.
Alternative: STRG G
⇒ Geben Sie eine beliebige VBA-Anweisung in den Eingabebereich des Direktfensters ein ①.
⇒ Betätigen Sie am Ende der Zeile RETURN, um die Anweisung auszuführen.



Das Direktfenster

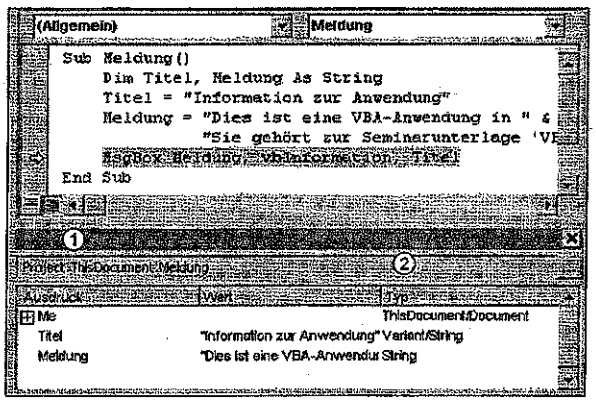
In der Abbildung wird den Variablen a und b jeweils ein Zahlenwert zugewiesen. Mit der Anweisung `print` wird nun die Summe der Variablen ausgegeben.

Im Programmcode können Sie die Anweisung `Debug.Print` verwenden, um beispielsweise Werte von Variablen zu Testzwecken im Direktfenster anzuzeigen. Das Programm wird dabei ohne Unterbrechung ausgeführt.

3.7 Das Lokalfenster

Das Lokalfenster liefert Ihnen zu allen lokalen und globalen Variablen des aktuellen Moduls den Datentyp und den aktuellen Wert.

⇒ Wählen Sie den Menüpunkt ANSICHT - LOKALFENSTER.
Das Lokalfenster ② wird eingeblendet, ist jedoch leer.
⇒ Rufen Sie den Menüpunkt DEBUGGEN - EINZELSCHRITT auf.
Alternative: F8
⇒ Führen Sie die Anweisungen im Einzelschrittmodus aus. Sobald der Wert einer Variablen verändert wird, erscheint sie im Lokalfenster.



Das Modulfenster mit dem Lokalfenster

Das Lokalfenster enthält ein aufklappbares Listenfeld ②. Darin sind die aktiven Prozeduren aufgeführt. Mehrere Prozeduren sind nur dann gleichzeitig aktiv, wenn eine Prozedur durch eine andere aufgerufen wurde.

Das Lokalfenster können Sie zur Fehlerbeseitigung und Überwachung in der Programmierphase benutzen.

3.8 Schnellübersicht

Was bedeutet ...	
VBA-Entwicklungsumgebung	Zu Office 2003 gehörendes Programm, in dem die VBA-Programmierung erfolgt
Projekt-Explorer	Enthält alle Komponenten der aktuellen Anwendung zusammengefasst und hierarchisch geordnet
Eigenschaftenfenster	Enthält eine Auflistung der Eigenschaften eines Objekts
Modulfenster	Editor-Fenster zur Eingabe des Programmcodes
Direktfenster	In das Direktfenster können einzelne Anweisungen eingegeben und direkt ausgeführt werden.
Lokalfenster	Zeigt gerade gültige Variablen mit ihrem Inhalt an

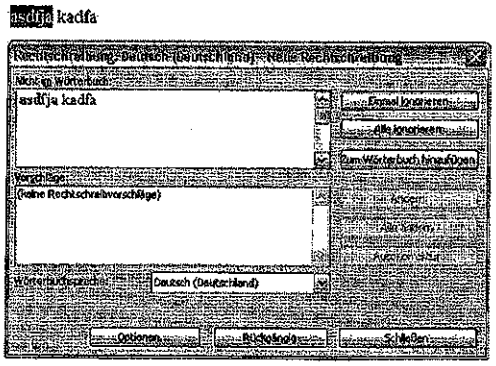
Sie möchten ...	
die QuickInfo anzeigen	BEARBEITEN - QUICKINFO oder STRG I
Eigenschaften/Methoden anzeigen	BEARBEITEN - EIGENSCHAFTEN/METHODEN ANZEIGEN oder STRG J
ein Wort vervollständigen	BEARBEITEN - WORT VERVOLLSTÄNDIGEN oder STRG LEERTASTE
ein Lesezeichen setzen/löschen	BEARBEITEN - LESEZEICHEN - LESEZEICHEN SETZEN/ZURÜCKSETZEN
eine Prozedur testen	AUSFÜHREN - SUB/USERFORM AUSFÜHREN oder F5
das Direktfenster einblenden	ANSICHT - DIREKTFENSTER oder STRG G
das Lokalfenster einblenden	ANSICHT - LOKAL-FENSTER
die Darstellung des VBA-Codes ändern	EXTRAS - OPTIONEN, Register EDITORFORMAT

3.9 Übung

Übungsdatei: *Übung1.doc* **Ergebnisdatei:** *Übung1-E.doc*

① Schreiben Sie in Word eine Prozedur `Rechtschreibpruefung`, die die Überprüfung der Rechtschreibung des aktuellen Dokuments durchführen soll. Verwenden Sie dazu die Methode `CheckSpelling` des Objekts `ActiveDocument`.

② Schreiben Sie in Word einen Text mit einigen fehlerhaften Wörtern, und testen Sie die Prozedur `Rechtschreibpruefung`.



4 Die Sprachelemente von VBA

In diesem Kapitel erfahren Sie

→ wie Sie modulorientiert programmieren
→ welche Sprachelemente in VBA zur Verfügung stehen
→ wie Sie Kontrollstrukturen einsetzen

Voraussetzungen

✓ Arbeiten mit der VBA-Entwicklungsumgebung
✓ Grundlegende Programmierkenntnisse

4.1 Module verwenden

Strukturierte Programmierung

Bereits kleinere Programmierprojekte weisen oft eine hohe Komplexität auf. Nur mit einer strukturierten Vorgehensweise lässt sich diese Komplexität in die Programmiersprache umsetzen. Das Prinzip der strukturierten Programmierung bedeutet, dass Sie die Gesamtaufgabe Ihres Programms in verschiedene Teilaufgaben zerlegen. Die Teilaufgaben können Sie wiederum in kleinere Teilaufgaben aufgliedern. Dies wird auch als schrittweise Verfeinerung bezeichnet. VBA unterstützt Sie dabei durch ein modulorientiertes Programmschema.

Diese Vorteile bringt die strukturierte Programmierung:

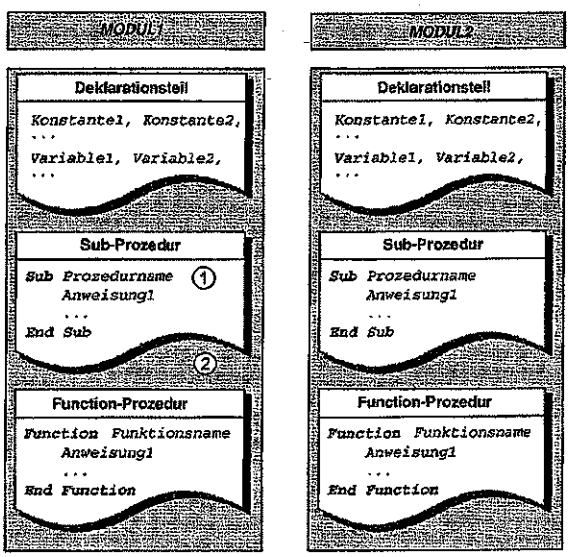
☑ Spätere Änderungen am Programm lassen sich leichter durchführen, da die Programmstruktur übersichtlicher und einfacher zu erfassen ist.
☑ Logische Fehler des Programmablaufs werden schneller erkannt, wenn das verursachende Modul identifiziert werden kann.

Der Aufwand des Programmierprojekts insgesamt wird verringert.

Programmierung mit Modulen

Durch das modulorientierte Programmschema gibt es keine lineare Anordnung der Programmanweisungen. Stattdessen bestehen VBA-Programme aus einer Zusammenstellung von Prozeduren ① in so genannten Modulen ②. Jedes Modul ist ein Behälter für Prozeduren, die alle gemeinsam eine abgeschlossene Aufgabe erfüllen. Die in Modulen enthaltenen Prozeduren werden meist durch bestimmte Ereignisse aufgerufen, wie z. B. das Anklicken einer Schaltfläche oder über andere Prozeduren.

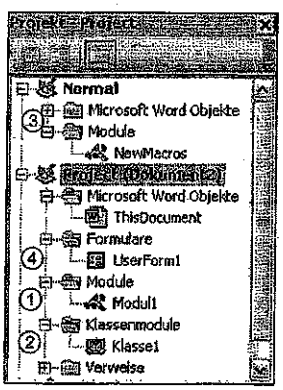
Bei integrierten Office-Projekten mit VBA wird meist nur in einer Office-Anwendung programmiert und von dort aus auf eine andere Anwendung zugegriffen (z. B. um Daten aus Access in Word einzufügen).



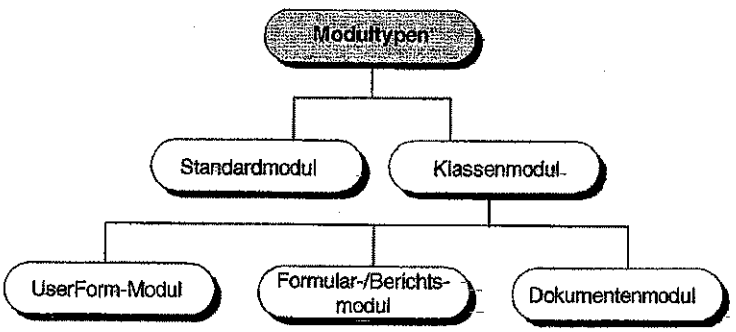
Programmieren mit Modulen

Modultypen

Standardmodule	☑ Sind für ein gesamtes Projekt gültig und werden auch als "globale Module" bezeichnet ☑ Enthalten Prozeduren für häufig und an verschiedenen Stellen vorkommende Aufgaben ☑ Erscheinen im Projekt-Explorer im Zweig MODULE ①
Klassenmodule	☑ Sind untrennbar mit einem Objekt verbunden (z. B. einem Dokument) oder definieren ein neues Objekt mit seinen Eigenschaften und Methoden ☑ Erscheinen im Projekt-Explorer im Zweig KLASSENMODULE ② bzw. im Zweig des übergeordneten Objekts ③ ☑ Spezialfälle sind UserForm-Dialoge, Dokumentenmodule und in Access Formular- und Berichtsmodule.
Dokumentenmodule	☑ Sind an ein bestimmtes Dokument gebunden ☑ Enthalten Prozeduren, die nur in diesem Dokument benötigt werden ☑ Existieren nicht in Access
UserForm-Dialoge	☑ Module, die zu einem benutzerdefinierten Formular gehören ☑ Ereignisprozeduren eines UserForms werden hier erstellt und gespeichert. ☑ Erscheinen im Projekt-Explorer im Zweig FORMULARE ④
Formular- bzw. Berichtsmodule	☑ Existieren nur in Access ☑ Besonderes Klassenmodul für ein Formular oder einen Bericht ☑ Alle Ereignisprozeduren eines Formulars bzw. Berichts werden hier erstellt und gespeichert.



Verschiedene Modultypen im Projekt-Explorer (hier in Word)



Modultypen im Überblick

Dokumente (wie z. B. Word-Dokumente oder Excel-Arbeitsmappen) stellen aus der Sicht von VBA Objekte dar. Dadurch stellen sie automatisch ein zugehöriges Klassenmodul zur Verfügung, das Sie verwenden können, um VBA-Anweisungen zu speichern, die nur in diesem Dokument zur Verfügung stehen sollen.

Arbeiten mit Modulen

Standardmodule erstellen

⇒ Rufen Sie in der VBA-Entwicklungsumgebung den Menüpunkt EINFÜGEN - MODUL auf.

VBA blendet ein leeres Modulfenster ein. Alle Prozeduren, die Sie darin erstellen, können an anderen Stellen im Projekt aufgerufen werden.

Modul bearbeiten

⇒ Wählen Sie im Projekt-Explorer das gewünschte Standard-, Dokumenten- oder UserForm-Modul.

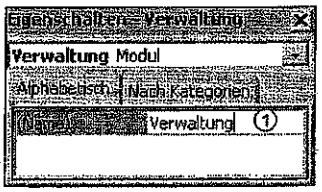
⇒ Rufen Sie den Menüpunkt ANSICHT - CODE auf.

Alternativen: F7 oder Doppelklick auf den Modulnamen

Der VBA-Programmcode des gewählten Moduls wird im Modulfenster angezeigt.

Standardmodul benennen

Neue Standardmodule werden von VBA automatisch MODUL1, MODUL2 .... benannt. Für eine bessere Übersicht sollten Sie jedem Modul einen Namen geben, der auf den Verwendungszweck hinweist.



Standardmodul benennen

⇒ Wenn das Eigenschaftenfenster nicht geöffnet ist, rufen Sie es mit dem Menüpunkt ANSICHT - EIGENSCHAFTENFENSTER auf.

Alternative: F4

⇒ Geben Sie im Eingabefeld für die Eigenschaft NAME einen aussagekräftigen Namen ein ①.

Modul löschen

⇒ Wählen Sie im Projekt-Explorer das gewünschte Modul.

⇒ Rufen Sie den Kontextmenüpunkt ENTFERNEN VON auf. VBA blendet ein Dialogfenster ein, in dem Sie wählen können, ob Sie das Modul vorher exportieren möchten.

⇒ Wenn Sie die im Modul enthaltenen VBA-Anweisungen in einer Datei sichern wollen, bestätigen Sie mit JA.

Module, die fest mit einem Objekt verbunden sind (z. B. Dokumentmodule), können nicht gelöscht werden.

Modul exportieren bzw. importieren

Sie können die Programmanweisungen eines Moduls in einer Datei speichern (exportieren) bzw. aus einer Datei auslesen (importieren).

⇒ Wählen Sie den Menüpunkt DATEI - DATEI EXPORTIEREN bzw. DATEI - DATEI IMPORTIEREN.

⇒ Markieren Sie im Dialogfenster eine Datei, bzw. geben Sie einen Dateinamen ein, und bestätigen Sie mit Ok.

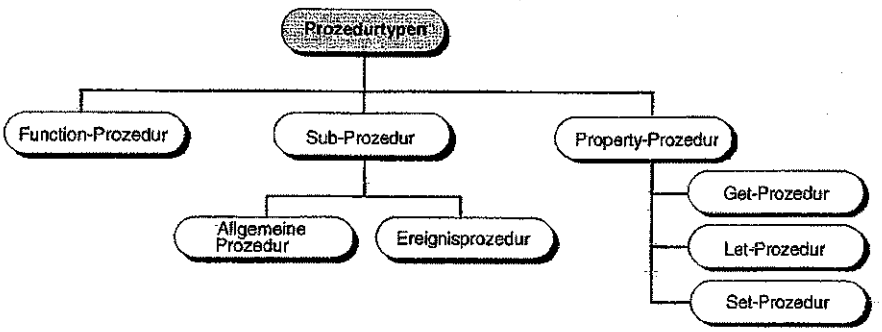
Die VBA-Entwicklungsumgebung speichert die Programmanweisungen beim Exportieren in einer Textdatei mit der Dateiendung **.bas*.

4.2 Mit Prozeduren programmieren

Prozedurtypen

VBA unterscheidet grundsätzlich zwischen `Function`-, `Sub`- und `Property`-Prozeduren. In vielen anderen Programmiersprachen werden diese Programmelemente Funktionen, Prozeduren und Properties bzw. Attribute genannt. Auch alle Ereignisprozeduren bei der ereignisorientierten Programmierung mit VBA sind Sub-Prozeduren.

Function-Prozeduren	☑	Liefern einen Wert zurück (z. B. das Ergebnis einer Berechnung) und können daher Variablen zugewiesen werden
	☑	Werden z. B. innerhalb einer anderen Prozedur aufgerufen
	☑	Können in allen Modultypen erstellt werden
Sub-Prozeduren	☑	Geben keinen Wert zurück
	☑	Werden in allgemeine Prozeduren und Ereignisprozeduren unterteilt
Allgemeine Prozeduren	☑	Sind in der gesamten Anwendung und allen Modulen gültig
	☑	Werden in Standardmodulen erstellt
	☑	Aufruf erfolgt durch eine andere Prozedur oder den Anwender direkt
Ereignisprozeduren	☑	Werden als Reaktion auf ein Ereignis ausgeführt (z. B. Klicken auf eine Schaltfläche)
	☑	Werden in Modulen gespeichert, die mit einem Objekt verbunden sind (z. B. UserForm)
Property-Prozeduren	☑	Regeln den Zugriff auf die Eigenschaften eines Objekts



Prozedurtypen in VBA

Aufbau von Sub-Prozeduren

```
① Sub BerechneSumme()
②   Summe = 2 + 6
③ End Sub
```

① Der Kopf einer Sub-Prozedur besteht aus dem Schlüsselwort Sub, auf das der Name der Sub-Prozedur folgt. Nach dem Prozedurnamen folgt ein rundes Klammernpaar, das für eine Parameterübergabe verwendet werden kann. Im Beispiel wurden keine Parameter übergeben.

② Anweisungen werden in den Prozeduren zwischen Sub und End Sub eingefügt.

③ Jede Sub-Prozedur endet mit den Schlüsselwörtern End Sub.

Notizen

Aufbau von Function-Prozeduren

Da Function-Prozeduren einen Wert zurückgeben können, besitzen Sie einen etwas anderen Aufbau als Sub-Prozeduren.

```
①  Function BerechneSumme() [As Datentyp]
     Dim Summe as Integer
     Summe = 2 + 6
②    BerechneSumme = Summe
③  End Function
```

① Eine Function-Prozedur wird mit dem Schlüsselwort Function eingeleitet, auf das der Name der Prozedur folgt. Nach dem Namen steht ein rundes Klammernpaar, das für eine Parameterübergabe verwendet werden kann. Im Beispiel wurden keine Parameter übergeben. Anschließend kann optional der Datentyp des Rückgabewertes angegeben werden.

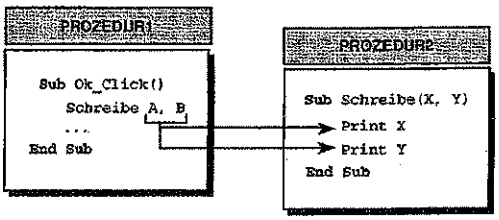
② Am Ende des Anweisungsteils wird dem Namen dieser Function-Prozedur ein Wert übergeben. Dies ist das Ergebnis, das die Prozedur zurückgeben soll.

③ Jede Function-Prozedur endet mit den Schlüsselwörtern End Function.

Prozeduren mit Argumenten

Einer Prozedur können Sie beim Aufruf Variablenwerte oder Konstanten als Argumente (auch Parameter genannt) übergeben. Dies ist in der modularen Programmierung oft notwendig, wenn mehrere Prozeduren mit diesen Werten arbeiten sollen.

Beim Einsatz von Prozeduren mit Argumenten können Sie Werte übergeben, ohne globale Variablen und Konstanten zu verwenden. Der Programmcode wird dadurch übersichtlicher und leichter nachvollziehbar.



Eine Prozedur wird mit Argumenten aufgerufen

Syntax der Prozedurdefinition

```
Sub Prozedurname(ByVal Argument1, ByRef Argument2, ...)
   ...
End Sub
```

oder

```
Function Prozedurname(Argument1, ...)
   ...
End Function
```

☑ Nach dem Namen der Sub- bzw. Function-Prozedur werden in runden Klammern alle Argumente aufgelistet, die beim Aufruf der Prozedur übergeben werden sollen.

☑ Mehrere Argumente werden, durch Kommas getrennt, nacheinander aufgelistet.

☑ Beim Aufruf der Prozedur werden die Argumente als Variablen eingerichtet. Diese Variablen nehmen die beim Aufruf übergebenen Werte auf und können innerhalb der Prozedur verwendet werden.

☑ Mittels der Schlüsselwörter ByVal und ByRef können die Argumente als Werte oder als Referenz übergeben werden. Standardmäßig (ohne Angabe) werden sie in VBA als Referenz übergeben.

Syntax der Property-Prozeduren

```
Property Get Name[(Arumentliste)] [As Datentyp]
  ...
  Name = Eigenschaft
End Property
...
Property Let Name(Wert As Datentyp)
```

```
  ...
  Eigenschaft = Wert
End Property
...
Property Set Name(Wert As Objecttyp)
  ...
  Eigenschaft = Wert
End Property
```

- ☑ Diese Prozeduren werden für den kontrollierten Zugriff auf die in der Regel privaten Eigenschaften eines Objekts verwendet. Sie müssen deshalb im Datentyp mit dieser Eigenschaft übereinstimmen. Property-Prozeduren sind grundsätzlich `public`, wenn Sie nicht ausdrücklich als `private` deklariert wurden.
- ☑ Die `Get`-Prozedur dient dem Lesezugriff und die `Let`- bzw. `Set`-Prozedur dem Schreibzugriff auf die entsprechende Eigenschaft, wobei die `Set`-Prozedur vor allem bei der Zuweisung von Objektvariablen verwendet wird.
- ☑ Innerhalb der Propertyprozeduren können Sie die zugewiesenen Werte behandeln bzw. überprüfen.

4.3 Variablen, Konstanten und deren Datentypen

4.3.1 Der Begriff Variable

Variablen können nacheinander verschiedene Werte aufnehmen und speichern. So müssen z. B. häufig Berechnungen mit unterschiedlichen Werten erfolgen. Diese Werte und die Ergebnisse der Berechnungen können Sie in Variablen speichern.

Für jede Variable wird ein Speicherplatz im Arbeitsspeicher Ihres Computers belegt. Der Variablenname ist der Schlüssel, damit Sie den dort gespeicherten Wert auslesen oder einen neuen Wert zuweisen können.

Variablennamen

- ☑ Der Variablenname muss mit einem Buchstaben beginnen.
- ☑ Es dürfen nur Buchstaben, Zahlen und der Unterstrich (_) verwendet werden. Alle anderen Zeichen wie Leer-, Satz- oder Sonderzeichen sind nicht erlaubt.
- ☑ Der Name muss eindeutig sein und darf nicht mit einem reservierten VBA-Schlüsselwort (z. B. `Dim`, `Function`, `Print`) übereinstimmen.
- ☑ Groß- und Kleinschreibung von Variablennamen wird nicht beachtet. Allerdings achtet VBA während der Eingabe darauf, dass an allen Stellen die gleiche Schreibweise verwendet wird.

Notizen

4.3.2 Deklaration von Variablen

Unter der Deklaration von Variablen wird das Vereinbaren von Variablen vor ihrem ersten Gebrauch verstanden. Mit der Deklaration machen Sie VBA Ihre Variablen bekannt und weisen jeder Variablen einen Datentyp zu.

Arten der Variablendeklaration

VBA erfordert im Gegensatz zu anderen Programmiersprachen nicht unbedingt eine Deklaration der Variablen am Beginn einer Prozedur. So ist diese Notwendigkeit standardmäßig auch ausgeschaltet. In diesem Fall wird eine Variable automatisch definiert, wenn sie das erste Mal verwendet wird. Solch ein Verfahren wird auch implizite Variablendeklaration genannt. Das Gegenstück dazu ist die explizite Variablendeklaration, bei der jede Variable vor ihrer Verwendung unbedingt deklariert werden muss.

Mit der Anweisung `Option Explicit` im Deklarationsteil eines Moduls können Sie die explizite Variablendeklaration für das betreffende Modul erzwingen.

Variablendeklaration	
Implizite Variablendeklaration	**Explizite Variablendeklaration**
☑ Variablen müssen vor ihrer ersten Verwendung nicht deklariert werden.	☑ Variablen müssen vor ihrer ersten Verwendung deklariert werden.
☑ Die Anweisung `Option Explicit` ist im Deklarationsteil des Moduls nicht vorhanden.	☑ Im Deklarationsteil des Moduls steht die Anweisung `Option Explicit`.

Sie können die explizite Variablendeklaration für alle neuen Module standardmäßig aktivieren. Wählen Sie dazu den Menüpunkt EXTRAS - OPTIONEN. Im Register EDITOR des Dialogfensters OPTIONEN aktivieren Sie das Kontrollfeld VARIABLENDEKLARATION ERFORDERLICH.

Die Deklaration von Variablen kann an zwei Stellen erfolgen. Im Deklarationsteil des Moduls können Sie Variablen deklarieren, die im gesamten Modul gültig sind. Die am Anfang einer Prozedur deklarierten Variablen sind dagegen nur in der jeweiligen Prozedur gültig.

Syntax der Variablendeklaration

Die Variablendeklaration kann auf zwei Arten erfolgen:

```
Dim Variable1 As Datentyp, Variable2 As Datentyp, ...
```

oder

```
Dim Variablenname1Typkennzeichen, Variablenname2Typkennzeichen, ...
```

- ☑ Die Variablendeklaration wird mit dem Schlüsselwort `Dim` eingeleitet, gefolgt vom Namen der Variablen.
- ☑ Nach dem Schlüsselwort `As` folgt der Datentyp, z. B. `String`. Wird kein Datentyp angegeben, deklariert VBA die Variable automatisch als `Variant`.
- ☑ Als Alternative lässt sich der Datentyp einer Variablen auch in einer Kurzform angeben. Dazu setzen Sie direkt an den Variablennamen (ohne Leerzeichen) das Typkennzeichen. Für den Datentyp `String` ist dies z. B. das Zeichen `$`.
- ☑ Mit einem Komma trennen Sie mehrere Variablendeklarationen in einer Zeile. Dabei muss das Schlüsselwort `Dim` nur einmal pro Zeile angegeben werden. Die Angabe des Datentyps ist jedoch für jede Variable einzeln notwendig. VBA verwendet sonst den Datentyp `Variant`.
- ☑ Numerische Variabelen werden mit 0, `Variant`-Variablen mit `Empty` und Zeichenketten mit "" (Leer) vorbelegt.

Mit der Anweisung `Dim` zur Deklaration wird eine lokale Variable erzeugt, was in den meisten Fällen auch erwünscht ist. VBA bietet darüber hinaus noch die Schlüsselwörter `Private`, `Public` und `Static`, die an Stelle der Anweisung `Dim` verwendet werden können, um die Gültigkeitsdauer und den Gültigkeitsbereich genauer festzulegen. Die grundlegende Syntax der Deklaration ändert sich dabei nicht.

Im Folgenden finden Sie einige Beispiele für die Deklaration von Variablen:

```
① Public WochenStd As Date
② Private Anzahl As Integer, NachName As String
③ Dim AnzStd
④ Dim Zähler%, Kontonummer&, Text$
```

① Hier wird eine globale Variable `WochenStd` deklariert, die in allen Modulen und darin enthaltenen Prozeduren bekannt ist.

② Diese zwei Variablen sind nur im aktuellen Modul verfügbar. Dort können sie jedoch von jeder Prozedur verwendet werden, um z. B. prozedurübergreifend Werte zu speichern.

③ Die Variable `AnzStd` wird als lokale Variable deklariert. Sie ist nur in der jeweiligen Prozedur gültig. Da kein Datentyp angegeben wurde, setzt VBA diesen automatisch auf `Variant`.

④ Hier erfolgt eine Deklaration in der Kurzform mit Typkennzeichen. Der Variablen `Zähler` wird der Datentyp `Integer` zugewiesen. Die Variable `Kontonummer` besitzt den Datentyp `Long` und die Variable `Text` den Typ `String`.

4.3.3 Konstanten deklarieren und verwenden

Wie für Variablen lassen sich auch für Konstanten Namen vereinbaren. Jedoch wird einer Konstanten bereits bei der Deklaration ein Wert zugewiesen, der später im Programmverlauf nicht mehr verändert werden kann.

```
Const Variable1 [As Datentyp] = Ausdruck
```

☑ Dem Namen folgt die Wertzuweisung über ein Gleichheitszeichen. Der Datentyp der Konstante wird durch den Datentyp des zugewiesenen Ausdrucks automatisch festgelegt. Mit der Anweisung `As Datentyp` hinter der Wertzuweisung können Sie den Datentyp aber auch ausdrücklich festlegen.

☑ Einer Konstanten können absolute Werte wie Zahlen, Zeichen oder Zeichenketten, aber auch ganze Ausdrücke (beispielsweise Ergebnisse aus Berechnungen) zugewiesen werden.

Vor der Ausführung eines Programms ersetzt VBA alle Vorkommen einer Konstanten durch den zugehörigen Wert. Eine erneute Wertzuweisung ist nicht möglich und erzeugt eine Fehlermeldung. Sie können Konstanten einsetzen, um einen Namen für einen Wert festzulegen, der an verschiedenen Stellen verwendet wird, z. B. für die Versionsnummer.

Integrierte Konstanten

VBA und die Office XP-Anwendungen stellen eine große Anzahl integrierter Konstanten zur Verfügung. Diese Konstanten erleichtern Ihnen die Arbeit mit den Funktionen und Objekten der Office-Anwendungen, da sie mit ihrem Namen den jeweiligen Wert und dessen Bedeutung erkennen lassen. Sie stehen meist für Zahlenwerte.

Notizen

An den ersten Zeichen des Konstantennamens können Sie erkennen, zu welcher Anwendung oder welchem Bereich die Konstante gehört.

vb	VBA-Konstante, z. B. `vbYesNo`
wd	Word-Konstanten, z. B. `wdAlignTabLeft`
xl	Excel-Konstanten, z. B. `xlFixedWidth`
ac	Access-Konstanten, z. B. `acCmdFindNext`
ad	Konstanten für Datenbankzugriff (ADO), z. B. `adLockOptimistic`
pp	PowerPoint-Konstanten, z. B. `ppEffectCoverRight`
mso	Office-Konstanten, z. B. `msoBarBottom`
fm	MSForms-Bibliothek für Formulare, z. B. `fmActionPaste`

Durch das Verwenden der integrierten Konstanten wird der VBA-Programmcode deutlich verständlicher als beim Einsatz der zugrunde liegenden Zahlen.

Damit die VBA-Entwicklungsumgebung alle Konstanten der Office-Anwendungen kennt, müssen die entsprechenden Objektbibliotheken eingebunden werden. Standardmäßig sind nur die Konstanten der geöffneten Office-Anwendung verfügbar.

4.3.4 Datentypen von VBA

Der Datentyp einer Variablen oder Konstanten legt den Wertebereich und somit die Größe des reservierten Speicherplatzes im Computer fest. Damit Sie in den verwendeten Ausdrücken die richtigen Ergebnisse bekommen, sollten Sie darauf achten, die richtigen Datentypen zuzuweisen. Nicht jeder Datentyp ist für das Speichern von bestimmten Werten (z. B. Fließkommazahlen) geeignet. Auch um nicht unnötig viel Speicherplatz zu reservieren, ist es sinnvoll, einen geeigneten Datentyp auszuwählen.

VBA kennt unter anderem die folgenden Datentypen:

Datentyp	Typkennzeichen	Verwendungszweck	Wertebereich/Bemerkung
`Integer`	%	Ganze Zahlen	-32.768 bis 32.767
`Long`	&	Ganze Zahlen	-2.147.483.648 bis 2.147.483.647
`Single`	!	Fließkommazahlen	Negativer Wertebereich: -3.4E+38 bis -1.4E-45 Positiver Wertebereich: 1.4E-45 bis 3.5E38 (7 Ziffern Genauigkeit)
`Double`	#	Fließkommazahlen	Negativer Wertebereich: -1.8E+308 bis -4.94E-324 Positiver Wertebereich: 4.94E-324 bis 1.8E308 (15 Ziffern Genauigkeit)
`Currency`	@	Fließkommazahlen	-9.22E+14 bis 9.22E+14 (Der Datentyp besitzt 15 Vor- und 4 Nachkommastellen und ist für genaue Berechnungen mit Währungen gedacht.)
`String`	$	Zeichenketten	Eine Zeichenkette kann 0 bis 65.535 Zeichen umfassen.
`Date`	Nicht vorhanden	Datum und Zeitangaben	Datum: 01. Januar 100 bis 31. Dezember 9999 Zeit: 00:00:00 bis 23:59:59
`Boolean`	Nicht vorhanden	Logische Werte	`True` oder `False` (`Wahr` oder `Falsch`)
`Variant`	Nicht vorhanden	Beliebige Daten	Numerischer Wert im Bereich des Datentyps `Double`, eine beliebige Zeichenkette oder eine Datums- bzw. Zeitangabe

Wenn Sie einer Variablen bei der Deklaration keinen Datentyp zuweisen oder die implizite Deklaration verwenden, wird standardmäßig der Datentyp `Variant` verwendet. Dieser Datentyp benötigt jedoch den meisten Speicherplatz.

4.3.5 Datenfelder (Arrays)

Datenfelder, auch "Arrays" genannt, ermöglichen es Ihnen, mehrere Werte desselben Datentyps unter einem einzigen Bezeichner (Variablennamen) zu speichern. Dies bietet sich an, um z. B. in einer Schleife über einen Index auf die einzelnen Feldelemente zuzugreifen. Um beispielsweise bei einer Berechnung zehn Ergebnisse zu speichern, benötigen Sie statt zehn einzelner Variablen nur ein Array, das alle Ergebnisse aufnehmen kann. Um auf die einzelnen Werte zugreifen zu können, werden der Array-Name und der Index benötigt.

Eindimensionale Arrays deklarieren

Bei der Deklaration eines Arrays legen Sie fest, welchen Datentyp die gespeicherten Werte haben und wie viele Werte gespeichert werden sollen.

```
Dim Arrayname(IndexUntergrenze To IndexObergrenze) As Datentyp
```

oder

```
Dim Arrayname(IndexObergrenze) As Datentyp
```

- ☑ Mit `IndexUntergrenze To IndexObergrenze` legen Sie die Größe des Arrays fest. Möglich ist z. B. eine Angabe `1 To 100`. In diesem Fall kann das Array 100 Werte aufnehmen. Der erste Wert wird mit dem Index 1 angesprochen.
- ☑ Wenn Sie nur die Obergrenze des Arrays angeben, verwendet VBA die Standarduntergrenze 0. Der erste Wert muss deshalb mit dem Index 0 angesprochen werden. Die Gesamtgröße des Arrays ist bei der Untergrenze 0 um 1 höher als die Obergrenze.

Mehrdimensionale Arrays deklarieren

Eindimensionale Arrays sind vergleichbar mit einer Liste oder einer Spalte aus einer Tabelle. Zweidimensionale Arrays beschreiben dagegen eine Matrix bzw. eine Tabelle mit Zeilen und Spalten. Jeder Wert der Matrix ist durch mehrere Indexwerte gekennzeichnet. Bei einer Tabelle sind dies beispielsweise die Zeile und die Spalte. Ein dreidimensionales Array wäre mit einer Arbeitsmappe aus mehreren Tabellen vergleichbar.

```
Dim Arrayname(Dimension1, Dimension2,...) As Datentyp
```

- ☑ Jede Dimension kann wie bei eindimensionalen Arrays nur mit der Obergrenze und mit der Unter- und Obergrenze des Index angegeben werden.
- ☑ Für ein zweidimensionales Array könnte die Deklaration beispielsweise lauten: `Dim TestArray(1 To 50, 1 To 50) As Integer`. Das Array `TestArray` bietet dann Platz für 50 mal 50 Werte (= 2.500 Werte) des Datentyps `Integer`.

Notizen

Arrays verwenden

Um auf einen einzelnen Wert eines Arrays zuzugreifen, geben Sie nach dem Array-Namen den Indexwert in runden Klammern an. Auf diese Weise können Sie Werte im Array speichern oder aus diesem auslesen.

Beispiel: *Sprachelemente.doc*, Modul `Beispiele`

Das folgende Beispiel demonstriert einige Verwendungsmöglichkeiten für ein- und mehrdimensionale Arrays.

```
Sub ArrayBsp()
  Const AnzNamen = 49
  Dim Wochentag(1 To 7) As String
  Dim NamenTabelle(5, AnzNamen) As String
  Dim Nachname As String, Vorname As String
  Nachname = "Meier"
  Wochentag(1) = "Montag"
  NamenTabelle(0, 0) = Nachname
  NamenTabelle(1, 0) = "Franz"
  Vorname = NamenTabelle(1, 0)
  Debug.Print Wochentag(1)
  Debug.Print Vorname + " " + Nachname
End Sub
```

① An dieser Stelle wird die Konstante `AnzNamen` deklariert.

② Hier wird ein eindimensionales Array `Wochentag` des Datentyps `String` deklariert. Es kann insgesamt sieben Werte aufnehmen, beginnend mit dem Indexwert 1.

③ `NamenTabelle` wird als ein zweidimensionales Array deklariert. Da für beide Dimensionen kein unterer Indexwert angegeben wurde, nimmt VBA automatisch 0 an. Die Obergrenze der zweiten Dimension wird von der Konstanten `AnzNamen` bestimmt. Das Resultat ist ein Array, das einer Tabelle mit sechs Spalten und 50 Zeilen entspricht.

④ Der Indexwert wird in Klammern hinter dem Array-Namen angegeben. Hier wird dem ersten Wert des Arrays `Wochentag` ein Text zugewiesen.

⑤ Bei mehrdimensionalen Arrays ist das Vorgehen gleich. Die einzelnen Indexwerte werden je durch ein Komma getrennt. Im Beispiel wird dem ersten Element in beiden Dimensionen ein Wert zugewiesen. Bei einer Tabelle wäre dieses das Element in der ersten Spalte der ersten Zeile.

⑥ Beim Auslesen eines Wertes aus einem Array wird analog der Zuweisung vorgegangen.

⑦ Im Direktfenster der Entwicklungsumgebung erfolgt eine Kontrollausgabe der Daten.

Dynamische Arrays deklarieren

Die Deklaration eines dynamischen Arrays erfolgt wie bei statischen Arrays. Es wird jedoch keine untere oder obere Indexgrenze angegeben. Die Größe des Arrays können Sie während der Programmausführung (also dynamisch) verändern.

```
Dim Arrayname() As Datentyp
```

Größe dynamischer Arrays verändern

Sowohl vor der ersten Benutzung als auch im späteren Programmverlauf können Sie mit der Anweisung `ReDim` die Unter- und Obergrenze des Arrays festlegen.

```
ReDim Arrayname(Dimension1, Dimension2,...)
```

☑ Im Gegensatz zur Array-Deklaration mit der Anweisung `Dim` können bei `ReDim` auch Variablen zur Festlegung der Indexgrenzen verwendet werden.

☑ Mit `ReDim` kann die Größe des Arrays auch mehrfach im Programmablauf geändert werden.

Bei der Verwendung von `ReDim` werden alle bereits im Datenfeld gespeicherten Werte gelöscht. Um dies zu verhindern und zumindest die Werte zu erhalten, die in das neu definierte Array passen, können Sie das Schlüsselwort `Preserve` verwenden:

```
ReDim Preserve Arrayname(Dimension1, Dimension2, ...)
```

Mit der Anweisung `Erase Arrayname` können Sie ein dynamisches Datenfeld löschen und den verwendeten Speicherplatz freigeben.

4.4 Programmablauf mit Kontrollstrukturen steuern

Im Programmablauf kann es oft vorkommen, dass das Programm auf eine bestimmte Situation oder Aktion mit einer Entscheidung reagieren muss und damit von mehreren Anweisungen eine oder gar keine Anweisung durchführt. So soll ein Programm beispielsweise anhand eines unteren und oberen Grenzwertes für einen Zahlenwert diesen in einer jeweils eigenen Prozedur verarbeiten. Ebenfalls von Nutzen ist es, wenn bestimmte Anweisungen nicht nur einfach, sondern mehrfach hintereinander durchlaufen werden. VBA bietet für diese Zwecke Verzweigungen und Schleifen, so genannte Kontrollstrukturen, die durch Ausdrücke gesteuert werden, die eine Bedingung enthalten.

4.4.1 Verzweigungen

Bei einer Verzweigung wird in Abhängigkeit vom Wert eines Ausdrucks eine bestimmte Anweisung bzw. ein bestimmter Anweisungsblock ausgeführt.

Übersicht der Verzweigungsarten in VBA

Verzweigunsart	Syntax	Beispiel

Einseitige Auswahl
If-Then-Anweisung
	`If Ausdruck Then`
	`  Anweisung1`
	`  Anweisung2`
	`  ...`
	`End If`
		`If Zähler = 10 Then`
		`  Zähler = 0`
		`  MsgBox "Berechnung beendet"`
		`End If`

Zweiseitige Auswahl
If-Then-Else-Anweisung
	`If Ausdruck Then`
	`  Anweisungsblock1`
	`Else`
	`  Anweisungsblock2`
	`End If`
		`If GesamtPreis >= 100 Then`
		`  Ausgabe = "Rabatt möglich!"`
		`Else`
		`  Ausgabe = "Kein Rabatt möglich!"`
		`End If`

Mehrstufige Auswahl
If-Then-ElseIf-Anweisung
	`If Ausdruck1 Then`
	`  Anweisungsblock1`
	`ElseIf Ausdruck2 Then`
	`  Anweisungsblock2`
	`ElseIf Ausdruck3 Then`
	`  Anweisungsblock3`
	`Else`
	`  Anweisungsblock`
	`End If`
		`If GesamtPreis < 100 Then`
		`  Ausgabe = "Kein Rabatt möglich!"`
		`ElseIf GesamtPreis < 200 Then`
		`  Ausgabe = "5% Rabatt möglich!"`
		`ElseIf GesamtPreis < 250 Then`
		`  Ausgabe = "10% Rabatt möglich!"`
		`Else`
		`  Ausgabe = "15% Rabatt möglich!"`
		`End If`

Fallauswahl
Select-Case-Anweisung
	`Select Case Selektor`
	`  Case Auswahlwert1`
	`  Anweisungsblock1`
	`  Case Auswahlwert2`
	`  Anweisungsblock2`
	`  ...`
	`  Case Else`
	`  Anweisungsblock`
	`End Select`
		`Select Case Note`
		`  Case 1`
		`  Text = "Note Sehr gut"`
		`  Case 2`
		`  Text = "Note Gut"`
		`  Case 3 To 4`
		`  Text = "Note Befriedigend"`
		`  Case 5 To 6`
		`  Text = "Note Ungenügend"`
		`  Case Else`
		`  Text = "Ungültige Eingabe"`
		`End Select`

Bei einer Fallauswahl können auch Operatoren wie `Is` zum Einsatz kommen, z. B. `Case Is < 20`.

Beispiel: *Sprachelemente.doc*, Modul `Beispiele`

Ein CD-Händler gewährt je nach Bestellmenge einen unterschiedlich hohen Rabatt. In Abhängigkeit von einem festen Preis und einer einzugebenden Bestellmenge sollen der Rechnungsbetrag und der dafür mögliche Rabattwert angezeigt werden.

```
Sub Rabatt()
  Dim Anzahl As String, Ausgabe As String
  Dim EinzelPreis As Double, GesamtPreis As Double
①  Anzahl = InputBox("Wie viele CDs wurden bestellt: ")
②  If Anzahl <> "" Then
    EinzelPreis = 15.90
③   GesamtPreis = Val(Anzahl) * EinzelPreis
    Ausgabe = "Rechnungsbetrag: " & GesamtPreis & " €" & vbCrLf
④   If GesamtPreis < 50 Then
      Ausgabe = Ausgabe & "Kein Rabatt möglich!"
⑤   ElseIf GesamtPreis < 100 Then
      Ausgabe = Ausgabe & "5% Rabatt möglich!"
⑥   ElseIf GesamtPreis < 150 Then
      Ausgabe = Ausgabe & "10% Rabatt möglich!"
⑦   Else
      Ausgabe = Ausgabe & "15% Rabatt möglich!"
    End If
    MsgBox Ausgabe, vbOKOnly + vbInformation
  End If
End Sub
```

① Mit einem Eingabedialog wird die Anzahl der bestellten CDs abgefragt und als Zeichenkette in der Variablen `Anzahl` gespeichert.

② Wird der Eingabedialog mit der Schaltfläche ABBRECHEN beendet oder wurde keine Anzahl eingegeben, enthält die Variable `Anzahl` eine leere Zeichenkette. Die `If`-Anweisung überprüft dies und führt den folgenden Anweisungsblock nur aus, wenn die Variable nicht leer ist.

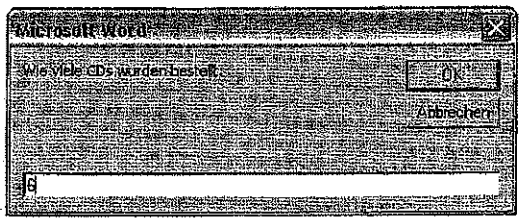
③ Mit der Funktion `Val` wird die Zeichenkette `Anzahl` in einen numerischen Wert konvertiert.

④ Ist der Gesamtpreis unter 50 €, erfolgt ein Hinweis, dass kein Rabatt möglich ist.

⑤ Das Schlüsselwort `ElseIf` leitet eine neue, untergeordnete Bedingung ein, die überprüft wird, wenn der Gesamtpreis nicht unter 50 € liegt. Falls der Preis nun unter 100 € liegt, erfolgt ein entsprechender Hinweis.

⑥ Hier wird überprüft, ob der Gesamtpreis unter 150 € liegt. Wenn ja, wird ein Hinweis zur Ausgabe hinzugefügt.

⑦ Falls alle bisherigen Bedingungen falsch waren, wird die Anweisung nach `Else` ausgeführt. Dies ist für alle Gesamtpreise der Fall, die über 150 € liegen.



Eingeben der Anzahl bestellter CDs

Ausgabe

Einfache Wertzuweisung mit einer Bedingung

Oft soll mit einer `If-Then-Else`-Anweisung kein großer Anweisungsblock ausgeführt werden, sondern nur eine Wertzuweisung an eine Variable in Abhängigkeit von einer Bedingung erfolgen. Für diesen Fall verfügt VBA über die vordefinierte Funktion `Iif`.

```
Variable = Iif(Ausdruck, Wert1, Wert2)
```

Ergibt die Auswertung des Ausdrucks den Wert `True`, wird der Variablen der `Wert1` zugewiesen, sonst der `Wert2`.

4.4.2 Schleifen

Mit Schleifen (auch Wiederholungen genannt) wird eine bestimmte Anweisung bzw. ein bestimmter Anweisungsblock wiederholt ausgeführt. VBA unterscheidet dabei zählergesteuerte und bedingte Schleifen.

Übersicht der Schleifen in VBA

Schleife	Syntax	Beispiel

Zählergesteuerte Wiederholung
For-Next-Schleife
	`For Zähler = Start To Ende`
	`  Anweisungsblock`
	`Next Zähler`
		`For Zähler = 1 To 20`
		`  Debug.Print "Nr. " & Zähler`
		`Next`

Kopfgesteuerte bedingte Wiederholung
Do-While/Until-Schleife
	`Do While/Until Ausdruck`
	`  Anweisungsblock`
	`Loop`
		`Do While Zahl2 <> 0`
		`  temp = Zahl2`
		`  Zahl2 = Zahl1 Mod Zahl2`
		`  Zahl1 = temp`
		`Loop`

Fußgesteuerte bedingte Wiederholung	`Do`
	`  Anweisungsblock`
	`Loop While/Until Bedingung`
		`Do`
		`  temp = Zahl2`
		`  Zahl2 = Zahl1 Mod Zahl2`
		`  Zahl1 = temp`
		`Loop Until Zahl2 = 0`

Bei der Prüfung des Ausdrucks nach dem Schlüsselwort `While` wird eine Schleife so lange ausgeführt, bis die Auswertung den Wert `False` ergibt. Mit dem Schlüsselwort `Until` wird die Schleife dagegen durchlaufen, bis die Bedingung wahr ist.

Notizen

Beispiel: *Sprachelemente.doc*, Modul `Beispiele`

Eine Prozedur soll die Anzahl der Tage von heute bis zu einem einzugebenden Datum berechnen. Der Eingabedialog soll so oft angezeigt werden, bis der Anwender die Schaltfläche ABBRECHEN wählt.

```
Sub AnzahlTage()
  Dim Heute As Date, Datum As String, Ausgabe As String
①  Heute = Date
②  Do
③    Datum = InputBox("Berechne die Anzahl der Tage bis zum:", _
                      "Anzahl Tage berechnen")
④    If Datum <> "" Then
⑤      Datum = DateValue(Datum)
       Ausgabe = "Heute ist der " & Heute & "." & vbCrLf & _
                 "Bis zum " & Datum & " sind es noch " & _
⑥                DateDiff("d", Heute, Datum) & " Tage."
       Auswahl = MsgBox(Ausgabe, vbOKOnly + vbExclamation, _
                        "Anzahl Tage berechnen")
     End If
⑦  Loop While Datum <> ""
End Sub
```

Die Ausgabe der Prozedur

① Die Funktion `Date` liefert das aktuelle Systemdatum.

② Mit dem Schlüsselwort `Do` beginnt die Schleifenstruktur. Die Schleife wird mindestens einmal durchlaufen, denn die Bedingung wird erst am Ende geprüft.

③ Die Funktion `InputBox` erzeugt ein Eingabefenster, das den Anwender zur Eingabe eines Datums auffordert.

④ Wenn der Anwender die Schaltfläche ABBRECHEN betätigt, enthält `Datum` eine leere Zeichenkette. In diesem Fall soll keine Berechnung erfolgen.

⑤ Mit der Funktion `DateValue` wird das eingegebene Datum in ein gültiges Format umgewandelt.

⑥ Die Funktion `DateDiff` berechnet die Differenz zwischen zwei Datumsangaben. Da das Ergebnis in Tagen ausgegeben werden soll, wird als Argument der Wert `"d"` übergeben.

⑦ Nach dem Schlüsselwort `Loop` beginnt die Prüfung des Ausdrucks, der die Abbruchbedingung enthält. In diesem Fall wird die Schleife so lange ausgeführt, bis der Anwender die Schaltfläche ABBRECHEN betätigt und die Variable `Datum` eine leere Zeichenkette enthält.

Schleifenabbruch mit `Exit`

Sie haben die Möglichkeit, Schleifen in Abhängigkeit von einer Bedingung vorzeitig zu verlassen, um so z. B. auf einen Abbruch durch den Anwender oder eine unzulässige Eingabe zu reagieren.

Mit der Anweisung `Exit` können sowohl zählergesteuerte als auch bedingte Wiederholungen abgebrochen werden:

☑ Eine zählergesteuerte Wiederholung wird mit `Exit For` abgebrochen.

☑ Eine bedingte Wiederholung kann mit `Exit Do` verlassen werden.

Mit der Anweisung `Exit Sub` können Sie die Ausführung einer Prozedur abbrechen.

4.5 Vordefinierte Funktionen

VBA bietet eine Reihe vordefinierter Funktionen, die bei der Programmentwicklung oft sehr nützlich sind, z. B. zum Bearbeiten von Zeichenketten oder Umwandeln von Datentypen. Erläuterungen und Beispiele zu allen Funktionen finden Sie im Online-Hilfesystem von VBA.

Dazu gehören unter anderem folgende Funktionen:

- ☑ Funktionen, die Teile einer Zeichenkette liefern bzw. ändern, z. B. `Left()`, `Len()`, `Mid()`
- ☑ Funktionen zum Rechnen mit Zeit und Datum, z. B. `Date()`, `Now()`, `Month()`
- ☑ Funktionen zum Umwandeln von Datentypen, z. B. `Val()`, `Int()`, `CDbl()`
- ☑ Mathematische und weitere Funktionen, z. B. `Exp()`, `Abs()`, `Rnd()`

4.6 Schnellübersicht

Was bedeutet ...	
Variable	Symbolischer Name für einen Speicherbereich im Computer, in dem während des Programmablaufs Werte gespeichert werden können; diese Werte lassen sich jederzeit auslesen und verändern.
Konstante	Symbolischer Name für einen Speicherbereich im Computer, dem einmalig ein Wert zugewiesen wird; der Wert kann nach der Definition nicht mehr geändert werden.
Datentyp	Legt fest, welche Daten eine Variable oder Konstante speichern kann (Wertebereich), und bestimmt die Speicherplatzgröße
Anweisung	Arbeitsschritt, den ein Programm ausführen soll und der nach bestimmten Regeln (Syntax) geschrieben wird.
Deklaration	Vereinbarung von Variablen oder Konstanten im Programm und Zuweisen eines Datentyps und/oder eines Anfangswerts
Array	Datenfeld, das mehrere Elemente eines Datentyps enthält, auf die über einen Index zugegriffen wird
Ausdruck	Ein oder mehre Werte, die über Vergleichsoperatoren und logische Operatoren verknüpft werden können. Ein Ausdruck kann entweder wahr (`True`) oder falsch (`False`) sein; z. B. ein einzelner Wert, ein Funktionsaufruf, der einen Wahrheitswert repräsentiert

Notizen

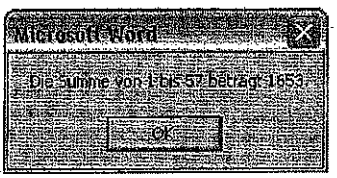
Sie möchten ...	
eine Sub-Prozedur deklarieren	`Sub ... End Sub`
eine Function-Prozedur deklarieren	`Function ... End Function`
Variablen deklarieren	`Dim Variablenname [As Datentyp]`
Konstanten deklarieren	`Const Konstantenname [As Datentyp] = Wert`
ein Array (Datenfeld) deklarieren	`Dim Arrayname(Dimension1, Dimension2,...) As Datentyp`
auf einen Array-Wert zugreifen	`Variable = Arrayname(Index1, Index2, ...)`
mehrere Zeichenketten verknüpfen	`String1 & String2 & ... oder String1 + String2 ...`
Vergleichsoperatoren einsetzen	`=,<,>,<>,<=,>=, Like`
logische Operatoren einsetzen	`Not, And, Or, Xor`
eine Auswahl erstellen	`If...Then...[ElseIf]...[Else]...End If`
eine Fallauswahl verwenden	`Select Case...Case...Case...Case Else...End Select`
eine zählergesteuerte Wiederholung erstellen	`For..To...[Step]...Next`
eine bedingte Wiederholung einsetzen	`Do While/Until...Loop` `Do ... Loop While/Until`

4.7 Übung

Übung 1

Übungsdatei: *Übung1.doc* **Ergebnisdatei:** *Übung1-E.doc*

① Öffnen Sie die Datei *Übung1.doc* und rufen Sie die VBA-Entwicklungsumgebung auf.
② Erstellen Sie im Modul ÜBUNG eine Sub-Prozedur AddZahlen, die alle Zahlen von 1 bis zu einem Endwert addiert.
③ Benutzen Sie eine Konstante, um den Endwert festzulegen.
④ Mit einer zählergesteuerten Wiederholung (For-Next-Schleife) addieren Sie alle Zahlen von 1 bis zum Endwert.
⑤ Das Ergebnis präsentieren Sie in einem Meldungsfenster. Geben Sie dabei auch den verwendeten Endwert an.



Ausgabe

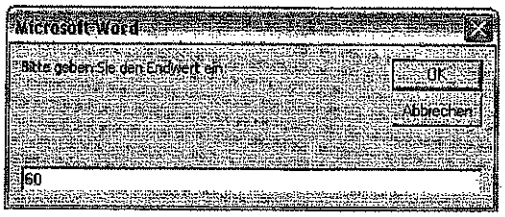
Übung 2

Übungsdatei: *Übung2.doc* **Ergebnisdatei:** *Übung2-E.doc*

① Erstellen Sie im Modul ÜBUNG eine neue Prozedur AddZahlenWiederholt, indem Sie die Prozedur aus Übung 1 kopieren.
② Der Endwert soll diesmal nicht mit einer Konstanten festgelegt, sondern durch einen Eingabedialog vom Anwender abgefragt werden. Verwenden Sie dazu die VBA-Funktion INPUTBOX, und speichern Sie die Rückgabe in einer Variablen.

③ Der Anwender soll auf Nachfrage die Berechnung wiederholen können. Erweitern Sie dazu das Meldungsfenster um eine Abfrage, ein Fragezeichen (Konstante `vbQuestion`) und die Schaltflächen JA und NEIN (Konstante `vbYesNo`).

④ Die Berechnung soll mindestens einmal durchgeführt werden und danach so oft, bis der Anwender die Schaltfläche NEIN betätigt. Benutzen Sie dazu eine `Do-Loop-Until`-Schleife. Prüfen Sie dabei den Rückgabewert von `MsgBox` auf den Wert `vbNo`.



Endwert abfragen

Ausgabe und Wiederholungsabfrage

Notizen

5 Objektorientierte Programmierung mit VBA

In diesem Kapitel erfahren Sie

- → wie VBA das Programmieren mit Objekten unterstützt
- → wie Sie auf Eigenschaften und Methoden zugreifen
- → wie Sie mit Auflistungen arbeiten
- → wie der Objektkatalog verwendet wird

Voraussetzungen

- ✓ Grundlagen der VBA-Programmierung
- ✓ Kenntnisse im Umgang mit der VBA-Entwicklungsumgebung

5.1 Was sind Objekte?

In der objektorientierten Programmierung stellt ein Objekt eine abgeschlossene Einheit dar, die bestimmte Daten sowie die Prozeduren (innerhalb von Objekten als "Methoden" bezeichnet) zum Verarbeiten dieser Daten enthält. Was sich im Inneren des Objekts befindet, bleibt dadurch weit gehend verborgen und geschützt. Auf diese Weise lässt sich eine bestimmte Funktionalität kapseln. Objekte kommunizieren mit anderen Objekten über festgelegte Schnittstellen, das sind vor allem die Methoden des Objekts.

Alle Elemente in Office, wie beispielsweise Dokumente, Tabellen, Arbeitsblätter, Formulare und Steuerelemente, werden als Objekte betrachtet. VBA erschließt Ihnen damit die Möglichkeit, auf alle Objekte der einzelnen Office-Anwendungen von einem VBA-Programm aus zuzugreifen, um beispielsweise Daten auszulesen oder zu verändern.

Die Office-Anwendungen Word, Excel usw. stellen selbst ebenfalls Objekte dar. Auf diese Weise wird Ihnen der Zugriff auf diese Anwendungen bei der Programmierung integrierter Lösungen vereinfacht.

Jedes Objekt verfügt über eine Anzahl individueller Eigenschaften und Methoden. Zu den Eigenschaften gehören z. B. Größe, Position, Farbe und Name eines Objekts. Mit den Methoden können Sie das Verhalten eines Objekts steuern. Ein Word-Dokument besitzt andere Eigenschaften als beispielsweise ein Access-Formular.

Die im Programmcode befindliche Definition für bestimmte Objekte wird als Klasse bezeichnet. Von einer Klasse können beliebig viele Objekte (auch Instanzen genannt) gebildet werden. Wenn Sie z. B. ein neues Dokument anlegen, erzeugt Word eine neue Instanz der Klasse `Document` und fügt es der Auflistung `Documents` der entsprechenden Instanz des `Application`-Objekts hinzu.

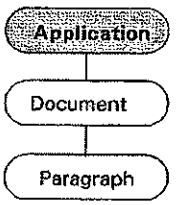
Wichtige Office-Objekte

Microsoft Office besteht aus einer sehr großen Anzahl von Objekten, auf die Sie in VBA Zugriff haben. Die folgende Tabelle gibt einen Überblick über einige Office-Objekte:

`Application`	Bezeichnet das Objekt der jeweiligen Anwendung (Word, Excel usw.)
`Document`	Klasse für Word-Dokumente
`Range`	Ein Textbereich in einem Word-Dokument bzw. ein Zellenbereich in einer Excel-Tabelle
`Workbook`	Eine Excel-Arbeitsmappe
`Form`	Klasse für Access-Formulare
`Table`	Klasse für Tabellen in Word, Excel oder Access
`Presentation`	Klasse für eine PowerPoint-Präsentation

5.2 Die Objekthierarchien (Objektmodelle)

Objekte existieren in der Regel nicht unabhängig voneinander. Sie sind in Hierarchiestufen angeordnet und in einem Objektmodell zusammengefasst. Für die Programmierung dieser Objekte ist es nicht nur wichtig, ihre Eigenschaften und Methoden zu kennen, sondern auch ihre jeweilige Hierarchiestufe im Objektmodell, da sie nur über diese Stufen angesprochen werden können.



Teil der Word-Objekthierarchie

Die höchste Ebene der Office-Objekte bildet das Objekt `Application`, das die jeweilige Anwendung darstellt. Alle anderen Objekte sind diesem untergeordnet und befinden sich dementsprechend in der Hierarchie unterhalb dieses Objektes. Das `Application`-Objekt wird automatisch beim Öffnen einer Anwendung erzeugt.

Beispiel: *Absatz.doc*, Modul `ThisDocument`

Im folgenden Beispiel sollen mithilfe von VBA zwei Absätze eines Word-Dokuments einmal fett und einmal kursiv formatiert werden.

```
Sub AbsatzFormatieren()
①   Application.Documents(1).Paragraphs(2).Range.Bold = True
②   Paragraphs(3).Range.Italic = True
End Sub
```

① Ausgehend vom `Application`-Objekt (der Anwendung Word) wird im ersten geöffneten Dokument der gesamte zweite Absatz fett formatiert.

② Hier wird der dritte Absatz kursiv formatiert. Da sich die Anweisungen in einem Dokumentenmodul befinden, erkennt VBA selbstständig das aktuelle Dokument als Bezugsobjekt.

VBA kann am Kontext oft erkennen, welches Objekt gemeint ist, z. B. wenn Sie in einem Dokumentenmodul programmieren. Sie müssen in diesen Fällen nicht die gesamte Hierarchie benennen. VBA stellt zusätzlich das Schlüsselwort `Me` zur Verfügung, welches sich immer auf das gerade aktuelle Objekt bezieht.

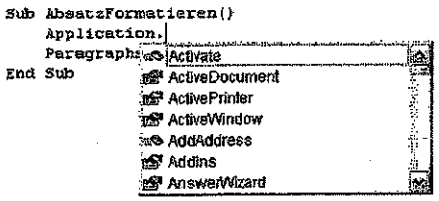
5.3 Eigenschaften und Methoden

Jedes Objekt verfügt über Eigenschaften und Methoden, über die Sie das Objekt beeinflussen und steuern können. Eigenschaften sind Variablen des Objekts, die Sie auslesen und verändern können. Als Methoden werden die Prozeduren eines Objekts bezeichnet.

Notizen

5.3.1 Eigenschaften von Objekten

Ein großer Teil der Programmierung besteht darin, Objekteigenschaften auszulesen und zu verändern. Die meisten Objekte verfügen über eine sehr große Anzahl Eigenschaften. Innerhalb von Objekten wird der Zugriff auf Eigenschaften meist über Property-Prozeduren (`Get`, `Let`, `Set`) geregelt bzw. kontrolliert, um die Kapselung der Daten sicherzustellen.



Eigenschaften und Methoden werden automatisch aufgelistet

Die Entwicklungsumgebung unterstützt Sie bei der Programmierung, indem sie die verfügbaren Eigenschaften (und Methoden) eines Objekts automatisch auflistet.

Auch der Objektkatalog (Taste F2) hilft Ihnen, den Überblick über die verfügbaren Eigenschaften eines Objekts zu behalten.

Wichtige Eigenschaften

Name	Wichtigste Eigenschaft vieler Objekte. Über den Namen wird das Objekt angesprochen und referenziert. Die Eigenschaft kann zur Laufzeit nur gelesen werden.
Caption	Beschriftung, z. B. von Schaltflächen, Formularen oder Fenstern
Selection	Bezeichnet die Markierung, beispielsweise einer Textstelle, Grafik usw.
Value	Ermöglicht den Zugriff auf den Inhalt eines Objekts, z. B. den Eintrag in einem Textfeld oder den Inhalt einer Tabellenzelle

Beispiel: *Eigenschaften.xls*, Modul `Tabelle1`

Im folgenden Beispiel wird einigen Zellen eines Excel-Arbeitsblattes mithilfe von VBA ein neuer Wert zugewiesen. Zusätzlich erhält eine Zelle eine neue Hintergrundfarbe, und der Wert einer Zelle wird ausgelesen.

```
Sub Eigenschaften()
①   Range("A1:C1").Value = 123
②   Range("C1").Interior.Color = RGB(255, 255, 0)
③   MsgBox Range("A1").Value
End Sub
```

① Da sich die Anweisungen in einem Tabellenmodul befinden, kann auf die Angabe der übergeordneten Objekthierarchie verzichtet werden. Hier wird mittels der Eigenschaft `Value` in die Zellen A1 bis C1 der Wert 123 eingetragen.

② Über die Eigenschaft `Color` wird der Zelle C1 eine gelbe Hintergrundfarbe zugewiesen. Das Objekt `Interior` bezeichnet dabei den Innenbereich der Zelle. Die VBA-Funktion `RGB` erzeugt den benötigten Farbcode.

③ Um den Wert der Zelle A1 zu erhalten, wird die Eigenschaft `Value` ausgelesen und das Ergebnis in einem Meldungsfenster angezeigt.

Syntax des Zugriffs auf Objekteigenschaften

```
Variablenname = Objektverweis.Eigenschaftsname
```

oder

```
Objektverweis.Eigenschaftsname = Ausdruck
```

☑ `Objektverweis` ist der Name eines Objektes oder ein Ausdruck, der das entsprechende Objekt bestimmt.

☑ Nach einem Punkt folgt der Name der Eigenschaft. Falls eine hierarchische Objektfolge vorliegt, werden die einzelnen Objekte ebenfalls durch Punkte getrennt.

☑ Eine Objekteigenschaft kann als Variable betrachtet werden, die Sie auslesen oder der Sie einen Wert zuweisen können.

Eigenschaften besitzen wie Variablen einen bestimmten Datentyp. So liefern viele Eigenschaften, z. B. `Caption`, eine Zeichenkette. Eigenschaften, die eine Auswahl zwischen `True` und `False` erlauben, sind vom Datentyp `Boolean`. Daneben gibt es einige Eigenschaften, bei denen eine Reihe von zulässigen Einstellungen möglich sind. Diese Eigenschaften besitzen den Datentyp `Integer`. Dabei ist jeder Einstellung ein Zahlenwert zugeordnet. Ein Beispiel für eine solche Eigenschaft ist die Eigenschaft STANDARDANSICHT (`DefaultView`) eines Access-Formulars. Möglich sind die Einstellungen EINZELNES FORMULAR, ENDLOSFORMULAR und DATENBLATTANSICHT. Bei der Programmierung mit VBA werden dafür intern die Zahlenwerte 0, 1 und 2 verwendet.

5.3.2 Methoden von Objekten

Methoden sind Prozeduren, die einem Objekt eine Anweisung erteilen, damit es eine Aufgabe erledigt. Sie werden wie normale `Sub`-Prozeduren oder `Function`-Prozeduren verwendet.

Beispiel: *Eigenschaften.xls*, Modul `Tabelle1`

Der von Daten verwendete Bereich eines Arbeitsblattes soll mit VBA-Befehlen markiert und anschließend in die Zwischenablage kopiert werden.

```
Sub Zwischenablage()
①   ActiveSheet.UsedRange.Select
②   Selection.Copy
End Sub
```

① Die Eigenschaft `UsedRange` des `ActiveSheet`-Objekts verweist auf den verwendeten Bereich des aktiven Tabellenblatts. Der Aufruf der Methode `Select` markiert diesen Bereich.

② Die Methode `Copy` des `Selection`-Objekts kopiert den markierten Zellbereich in die Zwischenablage.

Syntax des Zugriffs auf Objektmethoden

```
Objektverweis.Methodenname ①
Me.Methodenname ②
Methodenname ③
```

☑ `Objektverweis` ① ist der Name eines Objektes oder eine Zeichenfolge, die das Objekt bestimmt. Innerhalb des Objekts kann der Verweis auf das Objekt selbst auch entfallen ③ oder durch das Schlüsselwort `Me` ② hergestellt werden.

☑ Nach einem Punkt folgt der Name der Methode.

☑ Methoden sind Prozeduren des Objekts, die auch Werte zurückliefern oder Argumente erwarten können.

Notizen

5.3.3 Die With-Anweisung zum Zugriff auf Eigenschaften und Methoden

VBA ermöglicht eine vereinfachte Form, um nacheinander auf Eigenschaften und Methoden des gleichen Objekts zuzugreifen. Dazu dient die Anweisung `With`.

Beispiel: *Eigenschaften.xls*, Modul `Beispiele`

Mithilfe der `With`-Anweisung werden einige Eigenschaften des Arbeitsblattes ausgelesen und im Direktfenster angezeigt. Zusätzlich soll das betreffende Arbeitsblatt aktiviert werden.

```
Sub EigenschaftenAnzeige()
  With Workbooks(1)
    Debug.Print .Name
    Debug.Print .FullName
    Debug.Print .Path
    .Activate
  End With
End Sub
```

① Mit der Anweisung `With` wird der Block eingeleitet. Danach folgt die Angabe des Objekts, auf das sich die folgenden Eigenschaften und Methoden beziehen.

② Um auf die Eigenschaften des Objekts zuzugreifen, reicht es, wenn der Punkt, gefolgt vom Namen, angegeben wird. Hier werden einige Eigenschaften im Direktfenster angezeigt.

③ Der Zugriff auf Methoden erfolgt auf die gleiche Weise. An dieser Stelle wird das Tabellenblatt mit der Methode `Activate` aktiviert.

Syntax der `With`-Struktur

```
With Objektverweis
  ...
  [.Eigenschaftsname]
  [.Methodenname]
  ...
End With
```

☑ Nach der Angabe des Schlüsselwortes `With` folgt die Angabe des Objektnamens. Dieser muss für den gesamten Block nur einmal angegeben werden.

☑ Innerhalb der `With`-Struktur entfällt beim Zugriff auf Eigenschaften und Methoden die Angabe des Objekts. Nur der Punkt, gefolgt vom Namen der Eigenschaft oder Methode, muss angegeben werden.

☑ Soll innerhalb der `With`-Struktur vereinzelt auf ein anderes Objekt zugegriffen werden, darf der Objektverweis dort nicht entfallen.

`With`-Strukturen können beliebig verschachtelt werden. Beachten Sie jedoch, dass die Übersicht gewahrt bleibt. Eigenschaften und Methoden beziehen sich immer auf die zuletzt eingeleitete `With`-Struktur.

5.4 Objektvariablen

Um den Zugriff auf Objekte zu vereinfachen, können Sie Objektvariablen einsetzen. Dies sind spezielle Variablen, die Verweise auf Objekte speichern. Objektvariablen repräsentieren einen Verweis auf ein Objekt, z. B. auf den selektierten Bereich eines Dokuments.

Objektvariablen vereinfachen besonders den Zugriff auf Objekte, die weiter unten in der Objekthierarchie angeordnet sind. Der Aufbau langer Objektreferenzen wird dadurch vermieden, und der Programmcode wird übersichtlicher.

Beispiel: *Absatz.doc*, Modul `ThisDocument`

Mithilfe einer VBA-Prozedur sollen für den ersten Absatz eines Word-Dokuments bestimmt werden, wie viele Zeichen, Wörter und Sätze er enthält. Das Ergebnis wird in einem Meldungsfenster angezeigt.

```
Sub AbsatzEigenschaften()
①   Dim Absatz As Range
    Dim Meldung As String
②   Set Absatz = Application.Selection.Paragraphs(1).Range
③   Meldung = "Der Absatz enthält " & Absatz.Characters.Count & " Zeichen, "
    Meldung = Meldung & Absatz.Words.Count & " Wörter und " & _
              Absatz.Sentences.Count
    Meldung = Meldung & " Sätze."
④   MsgBox Meldung, vbOKOnly + vbInformation
End Sub
```

① An dieser Stelle wird die Objektvariable `Absatz` deklariert. Als Objektdatentyp wird `Range` angegeben; die Variable kann damit Verweise auf Textflächen im Dokument aufnehmen. Word verwendet das `Range`-Objekt, um auf Teile des Texts, z. B. einen Absatz, zugreifen zu können.

② Im Unterschied zu normalen Variablen wird der Objektvariablen an dieser Stelle ein Objektverweis mithilfe von `Set` zugewiesen. `Absatz` verweist danach auf die Textfläche des Absatzes, in dem sich gerade der Cursor befindet (`Selection`-Objekt).

③ Die Objektvariable wird wie eine Objektangabe verwendet. Hier werden verschiedene Eigenschaften ausgelesen und in der Zeichenkette `Meldung` formatiert.

④ Die Informationen werden in einem Meldungsfenster angezeigt.

Informationen über den aktuellen Absatz

Syntax der Deklaration von Objektvariablen

```
Dim Objektvariablenname As Objektdatentyp
```

☑ Die Deklaration wird wie bei normalen Variablen mit der Anweisung `Dim` eingeleitet. Danach folgt der Name der Objektvariablen.

☑ Mit der Anweisung `As` wird der Datentyp des Objekts bestimmt. Jeder verfügbare Objektname kommt dafür in Betracht. Der Datentyp legt fest, welche Verweise später gespeichert werden können.

☑ Es gelten alle Regeln, die auch für normale Variablen Gültigkeit haben, beispielsweise die Einschränkungen bei der Wahl des Variablennamens.

Syntax des Zuweisens eines Objektverweises

```
Set Objektvariablenname = Objektverweis
```

☑ Die Zuweisung wird mit der Anweisung `Set` eingeleitet. Danach folgt der Name der Objektvariablen.

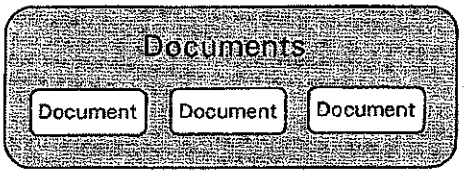
☑ Mit einem Gleichheitszeichen wird der Verweis auf das gewünschte Objekt erzeugt.

☑ Eine Zuweisung nur per Gleichheitszeichen wie bei normalen Variablen ist nicht möglich.

5.5 Auflistungen in der Objekthierarchie

5.5.1 Objektauflistungen

Einige der in der Objekthierarchie vorkommenden Objekte sind Auflistungen. Auflistungen bestehen aus einer Gruppe gleicher Objekte, d. h., es existieren mehrere Objekte dieses Typs. Sie stellen eine Art Container für mehrere Einzelobjekte dar.



Document-Objekte in der Documents-Auflistung in Word

Die Objektauflistung `Documents` in Word enthält beispielsweise alle geöffneten Dokumente. Jedes einzelne Dokument wird dabei innerhalb dieser Auflistung durch ein eigenes `Document`-Objekt mit seinen Eigenschaften und Methoden repräsentiert. Objektauflistungen werden in VBA durch den Namen des Einzelobjekts in der Plural-Form ("s" am Ende) gekennzeichnet, z. B. `Documents`, `Forms`, `Controls`, `Worksheets` usw.

Auf die Eigenschaften und Methoden eines einzelnen Objektes (z. B. eines Formulars oder Steuerelements) können Sie nur über das Einzelobjekt zugreifen. Jedoch besitzt auch die Auflistung selbst einige Eigenschaften, z. B. `Count` (ermittelt die Anzahl der in der Auflistung enthaltenen Objekte).

5.5.2 Auf einzelne Objekte einer Auflistung zugreifen

VBA behandelt Auflistungen ähnlich wie Arrays. Auf einzelne Objekte der Auflistung greifen Sie daher über einen Indexwert zu.

Im Gegensatz zu Arrays besitzen Objektauflistungen eine dynamische Länge, und die Zählung des Indexwerts beginnt bei 1.

Beispiel: *Eigenschaften.xls*, Modul `Beispiele`

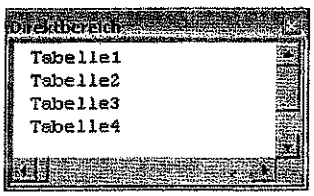
In einer Schleife werden die Namen aller Arbeitsblätter einer Excel-Arbeitsmappe ausgelesen und im Direktfenster angezeigt.

```
Sub Arbeitsblätter()
  Dim Anzahl As Integer, Index As Integer
①  Anzahl = Worksheets.Count
②  For Index = 1 To Anzahl
③    Debug.Print Worksheets(Index).Name
  Next
End Sub
```

① Die Eigenschaft `Count` der Auflistung `Worksheets` gibt die Anzahl der Arbeitsblätter an und bestimmt damit den größten Indexwert für den Zugriff auf die Auflistung. Der Wert wird in der Variablen `Anzahl` gespeichert.

② In einer `For-Next`-Schleife werden die einzelnen Elemente der Auflistung `Worksheets` vom Indexwert 1 bis `Anzahl` durchlaufen.

③ Mittels der Variable `Index` wird ein bestimmtes `Worksheet`-Objekt der Auflistung angesprochen und die Eigenschaft `Name` im Direktfenster angezeigt.



Die Ausgabe im Direktfenster

Das Direktfenster können Sie benutzen, um einfache Ausgaben zu testen. Falls es noch nicht angezeigt wird, öffnen Sie es über den Menüpunkt ANSICHT - DIREKTFENSTER bzw. den Tastaturbefehl STRG G.

Syntax des Zugriffs auf einzelne Objekte einer Auflistung

```
Auflistung(Index).Eigenschaftsname
```

oder

```
Auflistung(Index).Methodenname
```

- ☑ Der Zugriff auf einzelne Objekte einer Auflistung erfolgt ähnlich wie bei einem Array.
- ☑ Nach dem Namen der Auflistung folgt in runden Klammern ein Indexwert. Der Index beginnt bei dem Wert 1.
- ☑ Durch einen Punkt getrennt werden die Eigenschaften und Methoden des Einzelobjekts angesprochen, beispielsweise des `Document`-Objekts bei einer `Documents`-Auflistung.

5.5.3 Namen beim Zugriff auf Auflistungen verwenden

Außer dem Zugriff über Indexwerte kann bei vielen Auflistungen auch der Name des gewünschten Objekts verwendet werden, beispielsweise der Name eines Word-Dokuments oder eines Excel-Arbeitsblatts.

Beispiel für den Zugriff auf Objekte einer Auflistung über Namen:

```
Documents("Markierung.doc")
Workbooks("Mappe1.xls").Worksheets("Tabelle2").Range("A1:B5")
Forms("Mitarbeiter")
```

Der Name ist in den genannten Beispielen die Bezeichnung, die bei dem gewünschten Objekt in der Titelleiste der Anwendung angezeigt wird. Falls es sich um einen Dateinamen handelt, muss die Dateierweiterung, z. B. **.doc*, angegeben werden.

Wenn Sie die Namen der in einer Auflistung enthaltenen Objekte nicht kennen, können Sie sich diese in einer `For-Each`-Schleife ausgeben lassen (vgl. Beispiel im folgenden Abschnitt).

5.5.4 Auf alle Objekte einer Auflistung nacheinander zugreifen

Um nacheinander auf alle Objekte einer Auflistung zugreifen zu können, bietet VBA die `For-Each`-Schleife. Mit dieser Schleife wird der Zugriff auf die Einzelobjekte einfacher als bei einer `For`-Schleife mit Index-Werten.

Notizen

Beispiel: *Smile.ppt*, Modul `Beispiele`

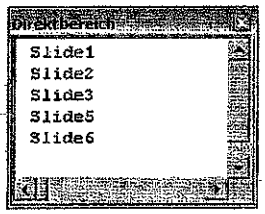
In einer Sub-Prozedur werden die Foliennamen der aktuellen PowerPoint-Präsentation ermittelt und im Direktfenster ausgegeben.

```
Sub FolienNamen()
  Dim Folie As Slide
  For Each Folie In ActivePresentation.Slides
    Debug.Print Folie.Name
  Next
End Sub
```

① Hier wird die Objektvariable `Folie` definiert, die in der Lage ist, verschiedene Objekte des Typs `Slide` (PowerPoint-Folie) aufzunehmen.

② Nacheinander wird bei jedem Durchlauf der `For-Each`-Schleife der Variablen `Folie` ein neues Objekt der Auflistung `Slides` zugewiesen. Das Objekt `ActivePresentation` verweist auf die aktuelle PowerPoint-Präsentation.

③ Der Name der Folie wird mit der Eigenschaft `Name` ermittelt und im Direktfenster angezeigt.



Die Ausgabe im Direktfenster

Syntax des Zugriffs auf alle Objekte einer Auflistung (nacheinander)

```
For Each Objektvariable In Auflistung
   ...
Next
```

☑ Mit den Schlüsselwörtern `For Each` wird die Schleife eingeleitet. Danach folgt eine Objektvariable, der nacheinander die einzelnen Objekte der Auflistung zugewiesen werden.

☑ Nach dem Schlüsselwort `In` folgt der Name der Auflistung, die durchlaufen werden soll.

☑ Innerhalb der Schleife kann über die Objektvariable auf die Methoden und Eigenschaften des Einzelobjekts zugegriffen werden.

Der Objektvariablen muss bei der Deklaration der Objekttyp der Einzelobjekte der Auflistung oder der allgemeine Objekttyp `Object` zugewiesen werden.

5.6 Den Objektkatalog verwenden

VBA verfügt über eine Vielzahl Objekte, die wiederum viele Eigenschaften und Methoden zur Verfügung stellen. Der Objektkatalog der VBA-Entwicklungsumgebung hilft Ihnen, den Überblick zu behalten.

⇒ Wählen Sie den Menüpunkt ANSICHT - OBJEKTKATALOG, um den Objektkatalog anzuzeigen.
Alternativen: F2 oder
⇒ Mit F7 wechseln Sie wieder zur Anzeige des vorher aktiven Modulfensters.

Sie können, im Objektkatalog der Hierarchie folgend, ein bestimmtes Objekt nachschlagen. Um schnell die Eigenschaften und Methoden eines Objekts anzuzeigen, verwenden Sie die Suchfunktion.

⇒ Tragen Sie im Listenfeld ① den gewünschten Suchbegriff ein, z. B. *save*.

⇒ Betätigen Sie nun die Taste RETURN, um den eingegebenen Begriff zu suchen.

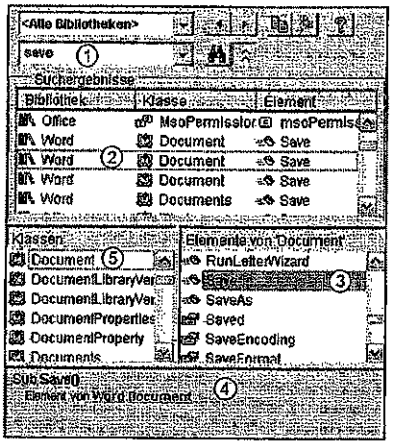
Alternative:

Die Auflistung ② zeigt alle gefundenen Begriffe.

⇒ Wählen Sie einen davon aus.

Im Listenfeld ③ sind nun alle Eigenschaften und Methoden des gewählten Objekts aufgeführt. Wählen Sie einen Eintrag aus, werden im unteren Textbereich ④ eine kurze Erklärung und die Syntax des Aufrufs angezeigt.

⇒ Im Listenfeld KLASSEN ⑤ können Sie auch ohne vorherige Suche eine Klasse nachschlagen.



Der Objektkatalog nach einer Suche

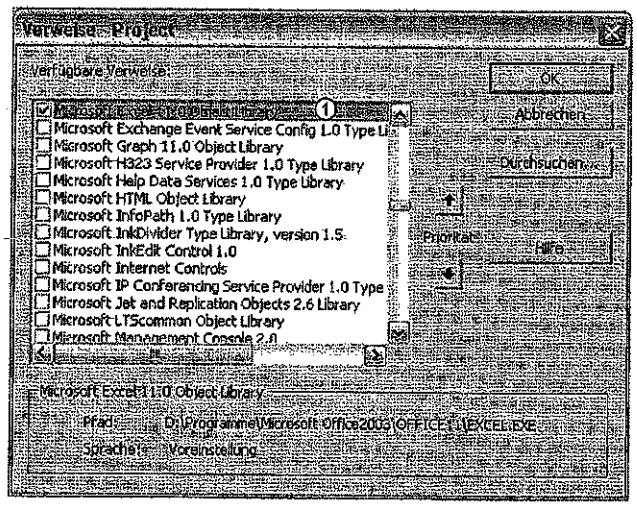
Zusätzliche Objektbibliotheken in der Entwicklungsumgebung aktivieren

Die VBA-Entwicklungsumgebung kennt standardmäßig nur die Objekte von VBA und der Office-Anwendung, in der gerade programmiert wird. Damit die Programmierhilfen, wie z. B. der Objektkatalog und das automatische Auflisten, auch mit den Objekten der anderen Anwendungen funktionieren, müssen Sie die entsprechenden Bibliotheken einbinden.

⇒ Wählen Sie in der VBA-Entwicklungsumgebung den Menüpunkt EXTRAS VERWEISE.

⇒ Aktivieren Sie im Dialogfenster VERWEISE im Listenfeld ① den Eintrag für die benötigte Bibliothek.

⇒ Bestätigen Sie mit Ok.



Einbinden zusätzlicher Bibliotheken

Folgende Bibliotheken enthalten die wichtigsten Konstanten und zugehörigen Objekte:

- ☑ Visual Basic for Applications
- ☑ Microsoft Word 11.0 Object Library
- ☑ Microsoft Excel 11.0 Object Library
- ☑ Microsoft Access 11.0 Object Library
- ☑ Microsoft PowerPoint 11.0 Object Library
- ☑ Microsoft Outlook 11.0 Object Library
- ☑ Microsoft Publisher 11.0 Object Library
- ☑ Microsoft Office 11.0 Object Library
- ☑ Microsoft Forms 2.0 Object Library

Notizen

5.7 Schnellübersicht

Was bedeutet ...	
Objekt	Abgeschlossene Programmeinheit, die Eigenschaften und Methoden zur Verfügung stellt
Objekthierarchie	Beginnend mit dem `Application`-Objekt sind alle Objekte in einer bestimmten Reihenfolge angeordnet.
Eigenschaft	Variable eines Objekts, die Sie auslesen oder der Sie Werte zuweisen können
Methode	Prozedur eines Objekts, über die dem Objekt Aufgaben übertragen werden
Auflistung	Zusammenstellung mehrerer gleichartiger Objekte, z. B. mehrerer Dokumente
Objektvariable	Variable, um Verweise auf Objekte zu speichern

Sie möchten ...	
eine Objekteigenschaft auslesen	`Variablenname = Objektverweis.Eigenschaftsname`
eine Objekteigenschaft verändern	`Objektverweis.Eigenschaftsname = Ausdruck`
eine Objektmethode aufrufen	`Objektverweis.Methodenname [Argument1, Argument2, ...]`
auf mehrere Eigenschaften und Methoden eines Objekts zugreifen	`With ... End With`
die Anzahl der Objekte einer Auflistung bestimmen	`Auflistung.Count`
auf ein einzelnes Objekt einer Auflistung zugreifen	`Auflistung(Index)` `Auflistung("Name")`
nacheinander auf alle Objekte einer Auflistung zugreifen	`For Each ... In ... Next`
einer Objektvariablen ein Objekt zuweisen	`Set Objektvariable = Objektverweis`

5.8 Übung

Übungsdatei: *Übung1.doc* **Ergebnisdatei:** *Übung1-E.doc*

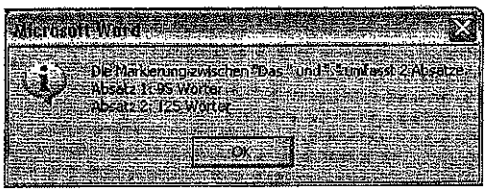
① Erstellen Sie im Dokumentenmodul der Datei *Übung1.doc* eine Prozedur `Markierunginformationen`. Die Prozedur soll anzeigen, wie viele Absätze die Markierung umfasst. Für jeden Absatz soll zusätzlich die Wörterzahl angezeigt werden.

② Erstellen Sie eine Objektvariable, die einen Verweis auf ein `Selection`-Objekt aufnehmen kann, und weisen Sie ihr die aktuelle Markierung zu.

③ Ermitteln Sie die Anzahl der Absätze in der Markierung.

④ Programmieren Sie eine `For-Next`-Schleife, die nacheinander alle Absätze der Markierung durchläuft und die Wörterzahl anzeigt.

⑤ Sammeln Sie alle Ausgaben in einer Zeichenkette, und zeigen Sie diese in einem Meldungsfenster an (vgl. Abb.).



Meldungsfenster mit den Absatzinformationen

Notizen

6 Gemeinsam genutzte VBA-Elemente

In diesem Kapitel erfahren Sie

- → wie Sie Anwendungs- und Dokumentenfenster steuern
- → wie Sie nach Dateien suchen
- → wie Sie Eingabedialoge und Meldungsfenster verwenden

Voraussetzungen

- ✓ Grundlagen der VBA-Programmierung
- ✓ Kenntnisse im Umgang mit den Office-Anwendungen

6.1 VBA-Elemente für alle Office-Anwendungen

VBA verfügt über einige Objekte, Methoden und Eigenschaften, die in allen Office-Anwendungen zur Verfügung stehen. Ebenfalls in allen Anwendungen verfügbar sind integrierte VBA-Funktionen. Besonders wichtig sind dabei die Funktionen `MsgBox` und `InputBox` zur einfachen Dateneingabe und -ausgabe.

Die folgende Tabelle gibt einen Überblick über einige wichtige VBA-Elemente, die in allen Anwendungen verwendet werden können:

`Application`	Objekt, das die jeweilige Anwendung selbst sowie deren Anwendungsfenster repräsentiert
`Window`	Objekt zum Steuern der Dokumentenfenster
`FileDialog`	Objekt zum Anzeigen der Standarddialoge für Datei- und Ordnerauswahl
`FileSystem`	Objekt zum Ausführen von Dateioperationen
`FileSearch`	Objekt zum Suchen nach Dateien im Dateisystem
`Assistant`	Objekt des Office-Assistenten
`MsgBox`	VBA-Funktion zum Anzeigen eines Meldungsfensters
`InputBox`	VBA-Funktion zum Anzeigen eines Eingabedialogs

Diese Elemente werden im Folgenden näher erläutert.

`UserForm`-Dialoge können ebenfalls in fast allen Office-Anwendungen eingesetzt werden. Da sie jedoch beim Erstellen von benutzerfreundlichen integrierten Lösungen eine besondere Rolle spielen, werden sie in einem eigenen Kapitel behandelt.

6.2 Programmieren des Anwendungsfensters

Das `Application`-Objekt entspricht im VBA-Objektmodell der Office-Anwendung selbst und repräsentiert daher das Anwendungsfenster, in dem bei Word, Excel und PowerPoint verschiedene Dokumentenfenster geöffnet werden können.

In Access kann, resultierend aus der völlig anderen Programmstruktur, das Anwendungsfenster nicht beeinflusst werden. Das `Application`-Objekt erlaubt daher nur den Zugriff auf bestimmte Eigenschaften und Methoden der Anwendung.

In Outlook existieren keine Dokumentenfenster. Die Daten, z. B. E-Mails oder Kontakte, sind hier in Ordnern (Folder) organisiert.

Ändern von Größe und Position

Das `Application`-Objekt verfügt über verschiedene Eigenschaften und Methoden zur Steuerung von Größe und Position des Anwendungsfensters:

`Height`	Höhe des Anwendungsfensters
`Width`	Breite des Anwendungsfensters
`Top`	Abstand des Anwendungsfensters vom oberen Rand
`Left`	Abstand des Fensters vom linken Rand
`WindowState`	Ermittelt bzw. bestimmt den Anzeigemodus des Fensters (normal, minimiert, maximiert) über drei Konstanten

Alle Größenangaben erfolgen in Bildschirmpunkten (Pixel).

Beispiel: *Fenster.xls*, Modul `DieseArbeitsmappe`

Mit einer VBA-Prozedur soll das Excel-Anwendungsfenster auf eine Größe von 600 x 400 Pixeln eingestellt und gleichzeitig an eine neue Position auf dem Bildschirm verschoben werden.

```
Sub AnwendungsFensterGröße()
①   Application.WindowState = xlNormal
②   Application.Left = 40
    Application.Top = 40
③   Application.Height = 400
    Application.Width = 600
End Sub
```

① Eine Größenänderung des Anwendungsfensters kann nur erfolgen, wenn es weder minimiert noch maximiert ist. Über die Eigenschaft `WindowState` wird daher mithilfe der vordefinierten Konstanten `xlNormal` der Normalmodus des Fensters eingestellt.

② Über die Eigenschaften `Left` und `Top` wird der Abstand vom linken und oberen Rand auf 40 Bildschirmpunkte bestimmt.

③ Die Größe des Anwendungsfensters wird auf eine Breite von 600 Bildschirmpunkten und eine Höhe von 400 festgelegt.

Die Konstanten für die Eigenschaft `WindowState` des `Application`-Objekts tragen in jeder Office-Anwendung andere Bezeichnungen. Sie sind in der jeweiligen Anwendungsbibliothek definiert und durch ein anwendungsspezifisches Kürzel gekennzeichnet, beispielsweise `xlNormal` (Excel), `wdWindowStateNormal` (Word) oder `ppWindowNormal` (PowerPoint).

Notizen

Anwendungsnamen ermitteln

Mithilfe der Eigenschaft `Name` ermitteln Sie von VBA aus den Namen der entsprechenden Office-Anwendung. Dies ist beispielsweise notwendig, wenn Teile des Programmcodes nur in einer bestimmten Office-Anwendung ausgeführt werden sollen.

```
① MsgBox Application.Name
② If Application.Name <> "Microsoft Access" Then
    ...
  End If
```

① In einem Meldungsfenster wird der aktuelle Anwendungsname ausgegeben.

② Die `If`-Anweisung prüft, ob das Programm in Access ausgeführt wird. Wenn nicht, soll ein bestimmter Anwendungsblock ausgeführt werden, beispielsweise zum Verändern der Fenstergröße.

Weitere Möglichkeiten für das Anwendungsfenster

Das `Application`-Objekt bietet in den meisten Office-Anwendungen noch weitere Eigenschaften und Methoden.

Eigenschaft	Erklärung	verfügbar in
`Caption`	Liest die Anzeige in der Titelleiste des Anwendungsfensters aus oder verändert sie.	allen Office-Anwendungen
`DisplayStatusBar`	Ermittelt oder bestimmt, ob die Statuszeile angezeigt wird	Word, Excel
`DisplayScrollBars`	Ermittelt oder bestimmt, ob die Bildlaufleisten angezeigt werden	Word, Excel
`DisplayFullScreen`	Liefert zurück oder bestimmt, ob sich das Anwendungsfenster im Vollbildmodus befindet	Excel

Wollen Sie Eigenschaften des `Application`-Objekts auslesen oder verändern, sollten Sie stets die Objektbezeichnung `Application` angeben. Da einige Eigenschaften auch in anderen Objekten verfügbar sind (z. B. `Caption`), können sonst durch die Kontexterkennung von VBA falsche Ergebnisse auftreten.

6.3 Die Dokumentenfenster programmieren

Mit dem `Window`-Objekt steuern Sie die Dokumentenfenster der Office-Anwendung. Dokumentenfenster dienen der Darstellung der geöffneten Dokumente und sind dem Anwendungsfenster untergeordnet.

In Access und Outlook steht das `Window`-Objekt aufgrund einer anderen Programmierstruktur nicht zur Verfügung.

Das `Window`-Objekt stellte eine Vielzahl von Eigenschaften und Methoden zum Steuern eines Dokumentenfensters zur Verfügung. Einige davon sind nachfolgend dargestellt:

`Activate`	Aktiviert das entsprechende Dokumentenfenster
`Caption`	Ermöglicht das Auslesen bzw. Verändern der Anzeige in der Titelleiste
`Close`	Schließt das entsprechende Fenster
`Height/Width`	Ermöglichen das Auslesen bzw. Verändern der Höhe und Breite des Fensters
`Left/Top`	Ermöglichen das Auslesen bzw. Verändern des Abstands zum Bildschirmrand
`NewWindow`	Erstellt eine Kopie des entsprechenden Fensters

Panes	Auflistung mehrerer Pane-Objekte, die Fensterausschnitte darstellen
Visible	Ermittelt oder bestimmt, ob ein Fenster angezeigt wird
WindowState	Ermöglicht das Auslesen oder Bestimmen des Fensterzustands

In Word, Excel und PowerPoint können Sie mit dem Menübefehl EXTRAS - OPTIONEN im Dialogfenster OPTIONEN, Register ANSICHT wählen, ob die Dokumentenfenster innerhalb des Anwendungsfensters angezeigt, oder ob für jedes Dokument eine eigenständige Task erzeugt werden soll. Der Zugriff über das Window-Objekt ist jedoch in allen Fällen identisch.

Aktuelles Fenster festlegen und bearbeiten

Alle geöffneten Dokumente bzw. Präsentationen oder Arbeitsmappen werden in einer für jede Office-Anwendung speziellen Auflistung verwaltet.

Word	Documents-Auflistung
Excel	Workbooks-Auflistung
PowerPoint	Presentations-Auflistung
Alle Anwendungen	Windows-Auflistung

Access kennt aufgrund seiner anderen Struktur keine Dokumente. Hier werden z. B. Formulare, Berichte und Tabellen verwaltet, für die jeweils ein eigenes Objekt existiert. In Outlook werden Ordner verwendet, für die ebenfalls jeweils eigene Objekte zur Verfügung stehen.

Sie können über einen Indexwert oder über den Dokumentennamen auf ein bestimmtes Dokument zugreifen. Mit der Methode Activate wird ein bestimmtes Dokument und damit das entsprechende Dokumentenfenster aktiviert.

```
Documents(Index).Activate
Workbooks(Index).Activate
Presentations(Index).Windows(Index).Activate
Windows(Index).Activate
```

- ☑ Jede Office-Anwendung verwaltet die Dokumente in einer unterschiedlichen Weise.
- ☑ Der Zugriff auf ein bestimmtes Fenster kann über einen Index oder einen Namen erfolgen.
- ☑ Über die Methode Activate wird das betreffende Dokument aktiviert und das Fenster in den Vordergrund geholt.
- ☑ Die Windows-Auflistung ist in allen Office-Anwendungen verfügbar.

VBA stellt mit dem ActiveWindow-Objekt eine einfache Möglichkeit zur Verfügung, auf das aktuelle Dokumentenfenster zu verweisen. Dieses Objekt existiert in allen Office-Anwendungen und besitzt alle Eigenschaften und Methoden des Window-Objekts.

Notizen

Ändern von Größe und Position

Wie beim Anwendungsfenster wird das Dokumentenfenster mit den Eigenschaften `Height`, `Width`, `Left` und `Top` gesteuert. Eine Größen- und Positionsänderung ist nur im Normalzustand des Fensters möglich.

Beispiel: *Fenster.xls*, Modul `DieseArbeitsmappe`

Mit einer VBA-Prozedur werden im folgenden Beispiel die Größe und die Position des Fensters der ersten Excel-Arbeitsmappe verändert.

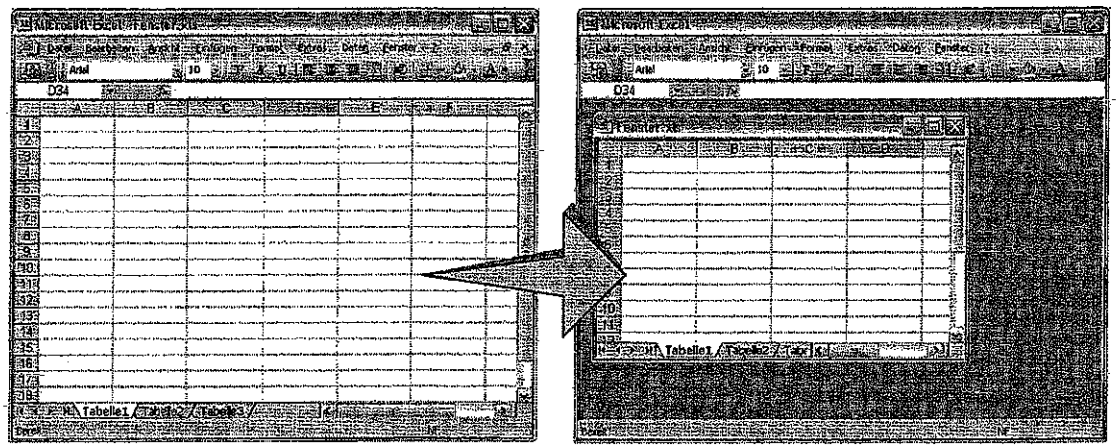
```
Sub DokumentenfensterGröße()
  Application.Windows(1).Activate
  With ActiveWindow
    .WindowState = xlNormal
    .Top = 10
    .Left = 10
    .Height = 200
    .Width = 300
  End With
End Sub
```

① An dieser Stelle wird das Fenster der ersten Arbeitsmappe als aktives Fenster definiert.

② Das Objekt `ActiveWindow` erlaubt den Zugriff auf das aktuelle Dokument. Mithilfe der `With`-Anweisung wird der Zugriff vereinfacht.

③ Da eine Größenänderung nur im Normalmodus erfolgen kann, wird das Fenster über die `WindowState`-Eigenschaft darauf eingestellt.

④ Mit den üblichen Eigenschaften werden im Folgenden Größe und Position des Fensters bestimmt.



Neues Fenster öffnen

Die Methode `NewWindow` öffnet ein neues Fenster für das bestehende Dokument. Sie ist für alle Objekte vom Typ `Window` verfügbar. Mit dem Schlüsselwort `Set` können Sie gleichzeitig einen Objektverweis auf das neue Fenster speichern.

```
ActiveWindow.NewWindow
Set Objektvariable = ActiveWindow.NewWindow
```

Verschiedene andere Objekte verfügen über die Methode `Open`, mit der sich gezielt Dokumente öffnen lassen. Diese Objekte sind jedoch anwendungsspezifisch, beispielsweise `Document.Open` in Word und `Workbook.Open` in Excel. Dabei müssen der genaue Pfad und der Dateiname eines Dokuments angegeben werden.

Fenster schließen

Die Methode `Close` schließt das betreffende Fenster. Falls noch weitere Fenster eines Dokuments existieren, wird das Dokument selbst nicht geschlossen. Es erfolgt daher keine Sicherheitsabfrage.

```
ActiveWindow.Close
```

6.4 Dateien suchen

In integrierten Office-Lösungen kann es oft notwendig sein, auf externe Dokumente zuzugreifen. VBA stellt Ihnen mit dem `FileSearch` ein Objekt zur Verfügung, das eine oder mehrere Dateien sucht. Das `FileSearch`-Objekt kann in allen Office-Anwendungen verwendet werden.

Folgende Eigenschaften und Methoden sind verfügbar:

`NewSearch`	Setzt alle Suchkriterien zurück und leitet eine neue Suche ein
`FileName`	Ermittelt oder bestimmt das Suchkriterium (Dateiname oder Suchmuster ohne Pfadangabe)
`FileTyp`	Legt einen bestimmten Dokumententyp fest, z. B. Word-Dokumente. Eine Dateiendung in der Eigenschaft `FileName` hat dabei Vorrang.
`SearchSubFolders`	Legt fest, ob Unterverzeichnisse in die Suche eingeschlossen werden
`LookIn`	Ermittelt oder bestimmt das Startverzeichnis für die Suche. Fehlt die Angabe, wird das aktuelle Verzeichnis angenommen.
`Execute`	Führt die Dateisuche aus und liefert die Anzahl der gefundenen Dateien zurück bzw. null, wenn die Suche nicht erfolgreich verlief
`FoundFiles`	Liefert eine Liste der gefundenen Dateien

Beispiel: *Dateisuche.doc*, Modul `ThisDocument`

Mithilfe einer VBA-Prozedur soll im Ordner *C:\Dokumente und Einstellungen* und in allen Unterverzeichnissen nach Word-Dateien gesucht werden. Die Anzeige der Anzahl der gefunden Dateien soll mit einem Meldungsfenster durchgeführt werden. Außerdem sollen alle Dateinamen in das geöffnete Word-Dokument geschrieben werden.

```
Sub DateiSuche()
  Dim Datei As Variant, Meldung As String
①  With Application.FileSearch
②    .NewSearch
③    .LookIn = "C:\Dokumente und Einstellungen"
④    .SearchSubFolders = True
⑤    .FileName = "*.doc"
⑥    If .Execute > 0 Then
⑦      Meldung = "Es wurden " & .FoundFiles.Count & _
                 " Dateien gefunden." & vbCrLf
⑧      For Each Datei In .FoundFiles
         Selection.TypeText (Datei & vbCrLf)
       Next
       MsgBox Meldung, vbInformation, "Dateisuche"
     End If
  End With
End Sub
```

① Mit einer `With`-Anweisung wird der Zugriff auf das `FileSearch`-Objekt weniger schreibaufwändig.

② `NewSearch` leitet eine neue Suche ein. Dadurch werden alle eventuell vorher gesetzten Suchkriterien zurückgesetzt. Es könnte sonst zu einer fehlerhaften Suche kommen.

③ Über die Eigenschaft `LookIn` wird der Startpfad definiert.

④ Die Suche in Unterverzeichnissen ist aktiviert.

⑤ Es werden alle Dateien mit der Endung **.doc* gesucht (Word-Dateien). Eine Suche mit dem Kriterium `.FileTyp=msoFileTypeWordDocuments` würde auch HTML-Dateien finden.

Das Suchergebnis im Meldungsfenster

⑥ Mit der Methode `Execute` wird die Suche gestartet und gleichzeitig abgefragt, ob die Suche erfolgreich war. Im Erfolgsfall liefert die Methode die Anzahl der gefundenen Dateien, sonst den Wert 0.

⑦ Die Eigenschaft `Count` der Auflistung `FoundFiles` liefert die Anzahl der gefundenen Dateien.

⑧ Nacheinander werden die gefundenen Dateien in der Auflistung `FileSearch` durchlaufen und ihre Dateinamen in das Word-Dokument geschrieben.

Die gefundenen Dateien enthalten den Namen der Datei inklusive Pfadangabe. Sie können daher auch als Argumente für die `Open`-Anweisung zum Öffnen der Datei oder die `Open`-Methode für `Document`- bzw. `Workbook`-Objekte dienen.

6.5 Dialogfenster zur Datei- und Ordnerauswahl

Office XP verfügt neben dem `FileSearch`-Objekt zusätzlich über das `FileDialog`-Objekt, das Bestandteil jedes `Application`-Objekts ist. Mit diesem Objekt können Sie die Standarddialogfenster von Windows zum Öffnen und Speichern von Dateien sowie zur Ordnerauswahl auch in eigenen VBA-Programmen verwenden.

Um einen Dialog zu verwenden, müssen Sie eine Objektvariable vom Typ `FileDialog` erstellen. Über `Application.FileDialog` erstellen Sie ein neues Dialogobjekt und entscheiden dabei mithilfe einer Konstanten, um welchen Dialog es sich handeln soll.

Konstante	Dialogfenster
`msoFileDialogOpen`	Dialogfenster DATEI ÖFFNEN
`msoFileDialogSaveAs`	Dialogfenster SPEICHERN UNTER
`msoFileDialogFilePicker`	Dialogfenster zur Dateiauswahl
`msoFileDialogFolderPicker`	Dialogfenster zur Ordnerauswahl

Das Dialogfenster wird mit der Methode `Show` angezeigt. Anschließend können Sie über die Auflistung `SelectedItems` auf die Namen der ausgewählten Dateien oder Ordner zugreifen.

Beispiel: *DateiAuswahl.xls*, Modul `DieseArbeitsmappe`

Eine VBA-Prozedur soll ein Dialogfenster zur Auswahl mehrerer Word-Dokumente anzeigen. Die vom Anwender ausgewählten Dateinamen werden in das aktuelle Dokument geschrieben.

```
Sub DateiAuswahl()
  Dim fd As FileDialog, Zaehler As Integer
①  Set fd = Application.FileDialog(msoFileDialogFilePicker)
②  fd.Filters.Add "Word-Dokumente", "*.doc; *.rtf", 1
③  fd.AllowMultiSelect = True
④  fd.Show
⑤  For Zaehler = 1 To fd.SelectedItems.Count
⑥    Application.Selection.TypeText (fd.SelectedItems(Zaehler))
  Next
End Sub
```

① Hier wird das neue `FileDialog`-Objekt erstellt. Mit der Konstante `msoFileDialogFilePicker` wird die Anzeige eines Dialogfensters zur Dateiauswahl erreicht.

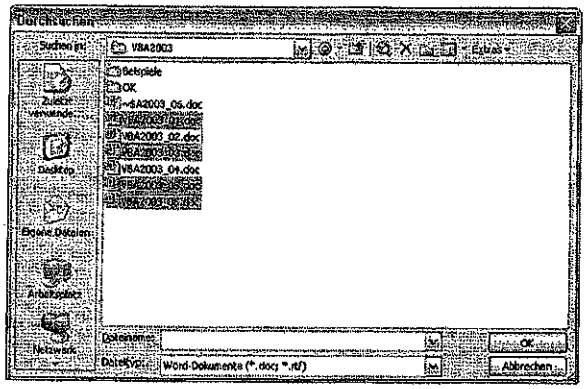
② Um die Dateiauswahl im Dialogfenster einzuschränken, wird an dieser Stelle ein Filter definiert. Unter dem Namen WORD-DOKUMENTE sollen nur Dateien mit den Endungen **.doc* oder **.rtf* angezeigt werden. Der Parameter 1 aktiviert diesen Filter als Standardeinstellung.

③ Damit der Anwender mehrere Dateien auswählen kann, wird die Eigenschaft `AllowMultiSelect` aktiviert.

④ Mithilfe der Methode `Show` wird das Dialogfenster angezeigt.

⑤ Sobald das Fenster geschlossen ist, wird die `For`-Schleife ausgeführt. Die Schleife durchläuft alle markierten Dateinamen.

⑥ Die Dateinamen werden in das aktuelle Word-Dokument geschrieben.



Das Dialogfenster zur Dateiauswahl von Word-Dokumenten

6.6 Mit dem Dateisystem programmieren

Neben dem `FileSearch`-Objekt existiert in VBA das Objekt `FileSystem`, das Methoden und Eigenschaften für Datei- und Verzeichnisoperationen zur Verfügung stellt. Beispielsweise können Sie mit der Methode `ChDir` das Verzeichnis wechseln und mit `FileLen` die Größe einer Datei bestimmen. Das `FileSystem`-Objekt kann in allen Office-Anwendungen eingesetzt werden.

Folgende Eigenschaften und Methoden sind beispielsweise verfügbar:

`ChDir`	Wechselt in ein Verzeichnis
`CurDir`	Ermittelt das aktuelle Verzeichnis
`FileCopy`	Kopiert eine Datei
`FileLen`	Bestimmt die Dateigröße
`Kill`	Löscht eine Datei
`FileDateTime`	Ermittelt Dateidatum und -zeit
`MkDir`	Erstellt ein neues Verzeichnis

Notizen

Beispiel: *Dateisuche.doc*, Modul `ThisDocument`

In einer VBA-Prozedur soll das aktuelle Verzeichnis gewechselt werden. Anschließend sollen von einer Datei in diesem Verzeichnis die Dateigröße und das Dateidatum ermittelt werden.

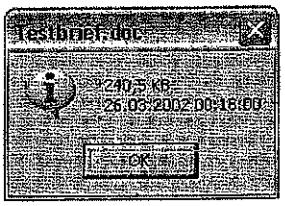
```
Sub DateiInfo()
  Dim Datei As String, Info As String
①  Datei = "Testbrief.doc"
②  FileSystem.ChDir ("C:\")
  If FileSystem.Dir(Datei) <> Datei Then
    MsgBox ("Die Datei " & Datei & " existiert nicht!")
    Exit Sub
  End If
③④ Info = Round(FileSystem.FileLen(Datei), 2) / 1024 & " KB " & _
            vbCrLf & FileSystem.FileDateTime(Datei)
  MsgBox Info, vbOKOnly + vbInformation, Datei
End Sub
```

① Beachten Sie, dass die Datei vorhanden sein muss, und legen Sie sie gegebenenfalls an.

② Die Methode `ChDir` wechselt das aktuelle Arbeitsverzeichnis. Als Argument wird das gewünschte Verzeichnis übergeben. Wenn die Datei nicht vorhanden ist, wird eine Fehlermeldung ausgegeben.

③ Mit der Methode `FileLen` wird die Dateigröße in Bytes ermittelt. Das Teilen durch 1.024 führt zur Dateigröße in KByte, die auf zwei Kommastellen gerundet wird.

④ Dateidatum und -zeit liefert ein Aufruf der Methode `FileDateTime`.



Anzeige von Dateigröße und Datum

6.7 Eingabedialoge und Meldungsfenster

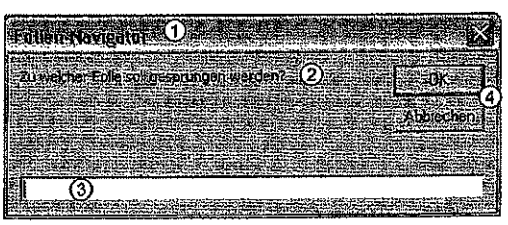
VBA verfügt über zwei vordefinierte Dialogfenster, mit denen Sie Meldungen ausgeben, Abfragen definieren und Eingaben anfordern können. Die Dialogfenster sind in allen Office-Anwendungen verfügbar und bieten eine einfache Möglichkeit, ohne aufwändige Formulare (Access) und UserForms Benutzerabfragen und -informationen zu realisieren.

Funktion	Verwendung
`InputBox`	Anzeige eines Eingabedialogs mit einem beliebigen Text und einer Eingabezeile; der vom Anwender eingegebene Wert wird an das Programm übergeben
`MsgBox`	Anzeige eines Meldungsfensters mit beliebigem Text, einem wählbaren Standardsymbol und verschiedenen Schaltflächen; der Wert der vom Anwender betätigten Schaltfläche kann zurückgeliefert und ausgewertet werden

6.7.1 Eingabedialoge verwenden

Einen Eingabedialog erzeugen Sie mit der Funktion `InputBox`. Das Eingabedialogfenster enthält folgende Elemente:

- ☑ Die Beschriftung des Fensters ① können Sie selbst gestalten. Auch ein beschreibender Text ② als Eingabeaufforderung ist von Ihnen frei wählbar.
- ☑ Das Eingabefeld ③ dient dem Anwender zur Eingabe der Daten. Es kann beliebige Zeichen aufnehmen.
- ☑ Die Schaltflächen OK und ABBRECHEN ④ werden standardmäßig eingeblendet.



Eingabedialog für einen Folien-Navigator

Beispiel: *Smile.ppt*, Modul `Beispiele`

Es soll eine VBA-Prozedur erstellt werden, die in einem Eingabedialog die Nummer einer Folie abfragt. Falls eine gültige Eingabe vorliegt, soll zu dieser Folie gesprungen werden.

```
Sub FolieAnzeigen()
  Dim Nummer As String
①  Nummer = InputBox("Zu welcher Folie soll gesprungen werden?", _
                   "Folien-Navigator")
②  If Nummer <> "" Then
③    ActivePresentation.Slides(Val(Nummer)).Select
  End If
End Sub
```

① Der Eingabedialog wird mit einer Eingabeaufforderung und einem Dialogtitel definiert. Der Variablen `Nummer` wird der vom Anwender in das Eingabefeld eingegebene Wert zugewiesen.

② Falls der Dialog mit ABBRECHEN geschlossen wird oder das Eingabefeld leer bleibt, wird eine leere Zeichenkette zurückgeliefert. In diesem Fall soll nicht zu einer bestimmten Folie gesprungen werden.

③ Das `Slide`-Objekt der gewünschten Folie wird mit der Variablen `Nummer` indiziert. Da das Dialogfeld eine Zeichenkette zurückliefert, wird sie mit `Val` in eine Zahl gewandelt. Mithilfe der Methode `Select` wird die Folie aktiviert.

Syntax der Funktion `InputBox`

```
Rückgabewert = InputBox("Eingabeaufforderung", "Titel", Vorgabe)
```

☑ Die Funktion `InputBox` zeigt einen vordefinierten Eingabedialog an und erwartet vom Benutzer eine Eingabe.

☑ Das Argument `Eingabeaufforderung` legt fest, welche Eingabeaufforderung angezeigt wird. Das Argument `Titel` bestimmt den Titel des Dialogfensters. Diese Argumente sind Zeichenketten und werden daher in Anführungszeichen ("") eingeschlossen.

☑ Das Argument `Vorgabe` legt fest, welcher Wert im Eingabefeld als Vorgabe angezeigt werden soll, bevor der Anwender eine Eingabe vornimmt. Dieses Argument kann auch entfallen; dann bleibt das Feld leer.

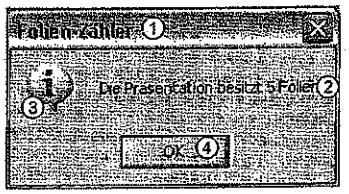
☑ Die `InputBox`-Funktion liefert als Rückgabewert die Eingabe des Anwenders als Zeichenkette. Betätigt der Anwender die Schaltfläche ABBRECHEN, ist der Rückgabewert eine leere Zeichenkette ("").

Die Position des Eingabedialogs wird von VBA vorgegeben und kann von Ihnen nicht beeinflusst werden. Die Größe des Dialoges können Sie über die optionalen Parameter xpos und ypos festlegen.

Notizen

6.7.2 Meldungsfenster verwenden

VBA bietet die Funktion MsgBox, um Meldungsfenster anzuzeigen. Ein Meldungsfenster besitzt die folgenden Elemente:



Ein Meldungsfenster in Aktion

- ☑ Die Beschriftung des Dialogfensters ① können Sie frei wählen. Das Gleiche gilt für den Meldungstext ②.
- ☑ Um die Meldung visuell zu unterstützen, können Sie ein beschreibendes Symbol anzeigen ③ (z. B. Ausrufezeichen, Fragezeichen).
- ☑ Standardmäßig ist die Schaltfläche OK ④ enthalten. Sie können auch eine oder mehrere andere Schaltflächen anzeigen.

Beispiel: *Smile.ppt*, Modul Beispiele

Als Erweiterung der Prozedur aus dem vorherigen Abschnitt soll zusätzlich mit einem Meldungsfenster die Anzahl der Folien in der aktuellen Präsentation angezeigt werden.

```
Sub FolieAnzeigen2()
  Dim Nummer As String
①  MsgBox "Die Präsentation besitzt " & ActivePresentation.Slides.Count & _
        " Folien.", vbOKOnly + vbInformation, "Folien-Zähler"
  Nummer = InputBox("Zu welcher Folie soll gesprungen werden?", _
                    "Folien-Navigator")
  If Nummer <> "" Then
    ActivePresentation.Slides(Val(Nummer)).Select
  End If
End Sub
```

① An dieser Stelle erfolgt die Anzeige des Meldungsfensters. Das Dialogfenster enthält einen Meldungstext, ein Symbol und einen Titel.

Syntax der Funktion MsgBox

Die Funktion MsgBox liefert standardmäßig den Wert der betätigten Schaltfläche zurück. Wenn Sie dies nicht benötigen, weil Sie z. B. nur eine OK-Schaltfläche anzeigen, können Sie MsgBox auch als Prozedur verwenden.

```
Rückgabewert = MsgBox("Meldung", Schaltflächen, "Titel")
```

oder

```
MsgBox "Meldung", Schaltflächen, "Titel"
```

- ☑ Die MsgBox-Funktion zeigt ein vordefiniertes Meldungsfenster an. Möchten Sie den Wert der betätigten Schaltfläche auswerten, können Sie eine Variable Rückgabewert zur Aufnahme dieses Wertes angeben. Die Argumente werden dann in Klammern eingeschlossen.
- ☑ Das Argument Meldung enthält den Text, der angezeigt werden soll. Mit dem Argument Titel bestimmen Sie den Titel des Dialogfensters. Beide Argumente sind Zeichenketten und werden daher in Anführungszeichen ("") eingeschlossen.
- ☑ Mit dem Argument Schaltflächen bestimmen Sie, welche Schaltflächen, aber auch welches Symbol, angezeigt werden.

Für das Anzeigen von Schaltflächen, Symbolen und anderen Einstellungen existieren in VBA vordefinierte Konstanten (bzw. deren Zahlenwerte), die Sie für die MsgBox-Funktion verwenden können:

Konstante	Wert	Element im Meldungsfenster
vbOkOnly	0	Schaltfläche OK
vbOkCancel	1	Schaltflächen OK und ABBRECHEN

vbAbortRetryIgnore	2	Schaltflächen ABBRECHEN, WIEDERHOLEN und IGNORIEREN
vbYesNoCancel	3	Schaltflächen JA, NEIN und ABBRECHEN
vbYesNo	4	Schaltflächen JA und NEIN
vbRetryCancel	5	Schaltflächen WIEDERHOLEN und ABBRECHEN
vbCritical	16	Symbol
vbQuestion	32	Symbol
vbExclamation	48	Symbol
vbInformation	64	Symbol
vbDefaultButton1	0	Die erste Schaltfläche ist standardmäßig aktiv
vbDefaultButton2	256	Die zweite Schaltfläche ist standardmäßig aktiv
vbDefaultButton3	512	Die dritte Schaltfläche ist standardmäßig aktiv

Um ein Symbol mit Schaltflächen zu kombinieren, müssen Sie den numerischen Wert des gewünschten Symbols (16, 32, 48, 64) mit dem numerischen Wert der Schaltflächenelemente (0 bis 5) addieren. Für ein Meldungsfenster mit einem Fragesymbol und den Schaltflächen JA und NEIN geben Sie die Werte 32+4 bzw. die Summe 36 an. Um das Programm besser lesbar zu gestalten, können Sie auch die numerischen Konstanten addieren, z. B. `vbYesNo + vbQuestion`.

Die folgende Tabelle zeigt die Rückgabewerte der Schaltflächen bei Anwendung der `MsgBox`-Funktion:

Konstante	Wert	Gewählte Schaltfläche
vbOk	1	OK
vbCancel	2	ABBRECHEN
vbAbort	3	ABBRECHEN
vbRetry	4	WIEDERHOLEN
vbIgnore	5	IGNORIEREN
vbYes	6	JA
vbNo	7	NEIN

Sie können im VBA-Programm auf das Betätigen einer bestimmten Schaltfläche reagieren. Vergleichen Sie dazu in einer Auswahl den Rückgabewert der `MsgBox`-Funktion mit einer der numerischen Konstanten, z. B. `vbYes`.

Wie der Eingabedialog kann auch das Meldungsfenster in der Größe und der Position nicht beeinflusst werden, da es von VBA vorgegeben wird.

Notizen

6.8 Schnellübersicht

Sie möchten ...	
den Status des Anwendungsfensters abfragen/setzen	`Application.WindowState`
Größe und Position des Anwendungsfensters einstellen	`Application.Height/.Width/.Left/.Top`
den Anwendungsnamen ermitteln	`Application.Name`
die Beschriftung des Anwendungsfensters auslesen/setzen	`Application.Caption`
auf das aktuelle Dokumentenfenster zugreifen	`ActiveWindow`-Objekt
ein bestimmtes Dokumentenfenster aktivieren	`Windows(Index).Activate` `Windows("Name").Activate`
Größe und Position eines Dokumentenfensters einstellen	`ActiveWindow.Height/.Width/.Left/.Top` `Windows(Index).Height/.Width/.Left/.Top`
das aktive Dokumentenfenster schließen	`ActiveWindow.Close`
nach Dateien suchen	`Application.FileSearch`-Objekt
ein Meldungsfenster anzeigen	`MsgBox "Text", Schaltflächen, "Titel"` `Rückgabewert=MsgBox("Meldung",Schaltflächen,"Titel")`
einen Eingabedialog einblenden	`Rückgabewert = InputBox("Eingabeaufforderung", "Titel", Vorgabe)`

6.9 Übung

Übung 1

Übungsdatei: *Übung1.doc* **Ergebnisdatei:** *Übung1_E.doc*

① Öffnen Sie die Datei *Übung1.doc* in Word, und starten Sie die VBA-Entwicklungsumgebung.

② Fügen Sie ein neues Modul hinzu. Ändern Sie den standardmäßig vergebenen Namen des Moduls in *Übung*.

③ Erstellen Sie eine neue Prozedur mit dem Namen `NeuesFenster`. Die Prozedur soll eine Kopie des aktuellen Dokumentenfensters öffnen und dessen Größe und Position ändern.

④ Deklarieren Sie eine Objektvariable mit einem geeigneten Objekttyp, und weisen Sie ihr einen Verweis auf das neue Fensterobjekt zu.

⑤ Ändern Sie die Position (20 Bildschirmpunkte Abstand vom oberen und linken Rand) und die Größe (400 Bildschirmpunkte Breite und Höhe) des neuen Fensters mithilfe der Objektvariablen. Verwenden Sie dazu eine `With`-Struktur.

Übung 2

① Erstellen Sie eine neue Prozedur `NeuesFensterAbfrage`, indem Sie die Prozedur aus Übung 1 kopieren.

② Weisen Sie der Prozedur (bzw. dem Makro) über den Menüpunkt EXTRAS - ANPASSEN eine Tastenkombination zu, und starten Sie die Prozedur auf diese Weise.

③ Erstellen Sie mit einem Meldungsfenster eine Abfrage, um dem Anwender die Wahl zu geben, das neue Fenster wieder zu schließen. Das Meldungsfenster soll ein Fragezeichensymbol und die Schaltflächen JA und NEIN enthalten (vgl. Abbildung).

④ Prüfen Sie die Auswahl des Anwenders, und programmieren Sie eine Anweisung zum Schließen des neuen Fensters.

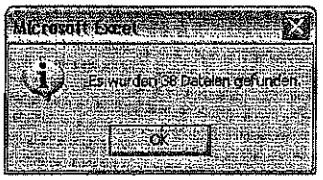
Übung 3

Übungsdatei: *Übung3.xls* **Ergebnisdatei:** *Übung3E.xls*

① Öffnen Sie die Datei *Dateisuche.xls* in Excel, und starten Sie die VBA-Entwicklungsumgebung.

② Erstellen Sie im Dokumentenmodul der Tabelle1 eine Prozedur `DateiSuche`. Die Prozedur soll Dateien nach Vorgabe suchen und in die Tabelle eintragen.

③ Erstellen Sie den Rumpf für eine Dateisuche mit dem `FileSearch`-Objekt. Als Startpfad soll das Verzeichnis C:\ verwendet werden. Unterverzeichnisse sollen durchsucht werden.



④ Programmieren Sie für die Abfrage des Dateinamens (des Suchkriteriums) einen Eingabedialog (vgl. Abbildung). Beim Betätigen der Schaltfläche ABBRECHEN soll die Suche nicht ausgeführt werden.

⑤ War die Suche erfolgreich, soll ein Meldungsfenster mit der Anzeige der gefundenen Dateien eingeblendet werden.

⑥ Verwenden Sie die `Cells`-Auflistung, um in einer `For`-`Next`-Schleife alle Dateinamen untereinander in die erste Spalte der Tabelle einzutragen. Eine einzelne Zelle adressieren Sie in der `Cells`-Auflistung nach dem Muster `Cells(Zeilenindex, Spaltenindex)`.

Notizen

7 Programmieren der Office-Anwendungen

In diesem Kapitel erfahren Sie

- → welche Objektmodelle die Office-Anwendungen besitzen
- → wie Sie auf Komponenten und Daten der Anwendungen zugreifen

Voraussetzungen

- ✓ Sprachelemente von VBA
- ✓ Programmieren mit Objekten
- ✓ Grundlegende Kenntnisse im Umgang mit den Office-Anwendungen

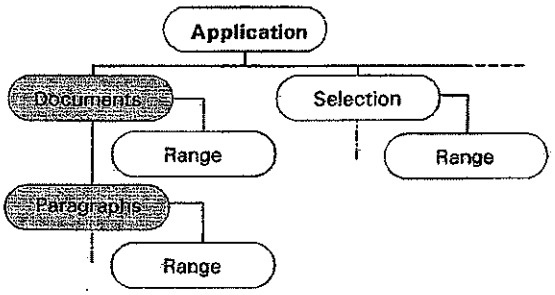
7.1 Mit Word programmieren

7.1.1 Das Word-Objektmodell

Word besitzt ein sehr komplexes Objektmodell. Alle Objekte sind dem `Application`-Objekt untergeordnet und befinden sich in der Hierarchie unterhalb dieses Objekts.

Jedes geöffnete Dokument wird durch ein `Document`-Objekt, alle geöffneten Dokumente werden durch die Auflistung `Documents` repräsentiert. Ein Dokument besteht aus Absätzen, die über die Auflistung `Paragraphs` angesprochen werden können.

Das `Selection`-Objekt bezieht sich stets auf die aktuelle Markierung. Dabei ist egal, ob nur ein Zeichen, ein Absatz oder mehrere Seiten markiert sind.



Ausschnitt aus dem Word-Objektmodell

Eine Besonderheit stellt das `Range`-Objekt dar, zu dem an vielen Stellen im Objektmodell verzweigt wird. Es stellt eine fortlaufende Fläche in einem Dokument dar, die durch eine Anfangs- und Endposition begrenzt wird. Sie kann dabei mehrere Absätze oder Seiten umfassen. Über das `Range`-Objekt können Sie auf den eigentlichen Inhalt eines Dokuments zugreifen und ihn verändern.

Das Word-Objektmodell enthält noch viele weitere Objekte, die jedoch nur in besonderen Fällen benötigt werden. Mehr Informationen finden Sie in der Online-Hilfe zu Word unter dem Stichwort *Microsoft-Word-Objekte* oder im Objektkatalog der Entwicklungsumgebung.

Word stellt zusätzlich einige Objekte zur Verfügung, die einen schnellen Zugriff auf ein gerade aktuelles Objekt erlauben. So können Sie beispielsweise mit dem Objekt `ActiveDocument` schnell auf das aktuelle Dokument verweisen, ohne seinen Namen oder seine Indexnummer in der `Documents`-Auflistung zu kennen. Weitere Beispiele sind die Objekte `ActiveWindow` und `ActivePrinter`.

7.1.2 Mit Dokumenten arbeiten

Dokumente öffnen, neu erstellen und schließen

Über verschiedene Methoden der `Documents`-Auflistung oder auch über das `ActiveDocument`-Objekt können Sie mithilfe von VBA neue Dokumente erstellen oder vorhandene öffnen bzw. schließen.

Beispiel: *WordObjekte.doc*, Modul `Beispiele`

Im folgenden Beispiel wird das Word-Dokument *Brief.doc* aus dem Stammverzeichnis *C:* geöffnet. Danach soll ein neues Dokument erstellt werden, das auf der Vorlage *MusterBrief.dot* beruht.

```
Sub DokumentÖffnenSchließen()
①   Documents.Open "C:\Testbrief.doc"
②   Documents.Add Template:="C:\MS Office\Vorlagen\MusterBrief.dot", _
                  NewTemplate:=False
End Sub
```

① Die Methode `Open` öffnet ein vorhandenes Dokument und fügt es der `Documents`-Auflistung hinzu. In diesem Fall wird die Datei *Testbrief.doc* im Verzeichnis *C:* geöffnet.

② Mit der Methode `Add` wird ein neues Dokument erzeugt, welches optional auf einer angegebenen Vorlage basiert, in diesem Fall auf der Vorlage *MusterBrief.dot*. Das Argument `NewTemplate` legt fest, ob das neue Dokument eine Vorlage werden soll (`True`) oder nicht (`False`). Da nur zwei bestimmten Argumenten der Methode Werte übergeben werden sollen, erfolgt die Zuweisung in der Form `Argumentname:=Wert`.

Dokument versteckt öffnen

Sie können ein Dokument auch unsichtbar für den Anwender öffnen. Dies ist besonders nützlich, wenn Sie mit VBA Teile des Dokuments auslesen oder verändern möchten, ohne dass der Anwender es sieht.

Versteckt geöffnete Dokumente belegen Speicherplatz, auch wenn sie für den Anwender nicht zu sehen sind. Nach der Bearbeitung sollten Sie daher die Dokumente mit der Methode `Close` schließen und den Speicherplatz durch Zuweisung von `Nothing` freigeben.

Weitere Möglichkeiten bei Dokumenten

Objekte vom Typ `Document` verfügen über eine Anzahl Methoden und Eigenschaften, die es erlauben, auf die zugrunde liegenden Dokumente zuzugreifen und sie zu steuern.

`Activate`	Aktiviert das Dokument und bringt das zugehörige Fenster in den Vordergrund
`Close`	Schließt und speichert das Dokument
`FullName`	Liefert Namen und Pfad des Dokuments
`IsMasterDocument` `IsSubDocument`	Ermittelt, ob das Dokument ein Zentral- oder Filialdokument ist
`Path`	Liefert den Pfad zum Speicherort des Dokuments
`PrintOut`	Startet den Ausdruck des Dokuments
`ReadOnly`	Ermittelt, ob das Dokument schreibgeschützt ist
`Protect`	Schützt das Dokument mit einem Passwort
`Save`	Speichert das Dokument
`Saved`	Ermittelt, ob das Dokument seit der letzten Änderung gespeichert wurde
`SaveAs`	Speichert das Dokument unter einem neuen Namen

Notizen

Beispiel: *WordObjekte.doc*, Modul `Beispiele`

Das Word-Dokument *C:\Testbrief.doc* wird für den Anwender unsichtbar geöffnet und danach wieder geschlossen.

```
Sub VersteckteÖffnen()
  Dim DocVersteckt As Document
①  Set DocVersteckt = Documents.Open(FileName:="C:\Testbrief.doc", _
                                     Visible:=False)
②  MsgBox (Documents(1).Name)
  DocVersteckt.Close
  Set DocVersteckt = Nothing
End Sub
```

① Das Argument `Visible` der `Open`-Methode legt fest, ob das Dokument versteckt (`False`) oder sichtbar geöffnet werden soll (`True`). Als Rückgabewert liefert die Methode einen Verweis auf das neue Dokument, der mit der Objektvariablen `DocVersteckt` gespeichert wird. Wird das Argument `Visible` nicht angegeben, öffnet VBA das Dokument sichtbar.

② Der Name des Dokuments wird angezeigt. Danach wird es geschlossen.

7.1.3 Auf Dokumenteninhalte zugreifen

Mit VBA können Sie nicht nur die Dokumente als Ganzes steuern, sondern Sie erhalten auch Zugriff auf den Inhalt der Dokumente. Dazu stellt VBA verschiedene Möglichkeiten zur Verfügung. Mit dem `Selection`-Objekt greifen Sie z. B. direkt auf einen markierten Bereich im Dokument zu. Über das `Range`-Objekt können Sie dagegen auf beliebige Teile eines Dokuments zugreifen, ohne diese zu markieren.

Auf markierten Bereich zugreifen

Das `Selection`-Objekt verfügt über eine Anzahl Unterobjekte, Methoden und Eigenschaften für den Zugriff auf einen markierten Bereich des aktuellen Dokuments. Die folgende Tabelle zeigt eine Auswahl davon:

Schlüsselwort	Typ	Erklärung
`Characters`	Auflistung	Erlaubt den Zugriff auf einzelne Zeichen der Markierung
`Sentences`	Auflistung	Erlaubt den Zugriff auf einzelne Sätze der Markierung
`Paragraphs`	Auflistung	Erlaubt den Zugriff auf einzelne Absätze der Markierung
`Words`	Auflistung	Erlaubt den Zugriff auf einzelne Wörter der Markierung
`Start/End`	Eigenschaft	Liefert die Start-/Endposition der Markierung
`Type`	Eigenschaft	Ermittelt den Markierungstyp (Blockmarkierung oder Einfügemarke)
`Information`	Eigenschaft	Liefert Informationen über die Markierung
`Text`	Eigenschaft	Liefert den Text der Markierung
`Move`	Methode	Verschiebt die Markierung
`WholeStory`	Methode	Erweitert die Markierung auf das gesamte Dokument
`TypeText`	Methode	Fügt den angegebenen Text an der Cursorposition ein
`Collapse`	Methode	Hebt die Markierung auf

Beispiel: *DokumentInhalt.doc*, Modul `ThisDocument`

Das folgende Beispiel überprüft die aktuelle Markierung und zeigt deren Typ an. Falls es sich um markierten Text handelt, soll angezeigt werden, wie viele Wörter die Markierung enthält, und der Text soll gleichzeitig fett formatiert werden. Hat der Anwender nichts markiert, soll eine entsprechende Meldung angezeigt werden.

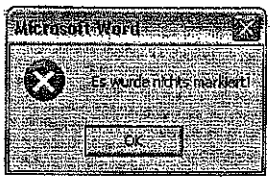
```
Sub MarkierungTesten()
  Select Case Selection.Type
    Case wdSelectionNormal
①        MsgBox "Es sind  " & Selection.Words.Count & " Wörter markiert."
         Selection.Font.Bold = True
②    Case wdSelectionShape
③        MsgBox "Sie haben eine Grafik markiert."
    Case wdSelectionIP
④        MsgBox "Es wurde nichts markiert!", vbCritical
    Case Else
         MsgBox "Der Typ der Markierung wurde nicht erkannt."
  End Select
End Sub
```

① Die Eigenschaft `Type` liefert Informationen über die Art der Markierung. Mit vordefinierten Konstanten kann der Markierungstyp ermittelt werden.

② Falls die Markierung normalen Text umfasst, wird über die Auflistung `Words` die Anzahl der markierten Wörter ermittelt, und der markierte Text wird fett formatiert.

③ Ist nur ein Grafikobjekt markiert, wird eine entsprechende Meldung ausgegeben.

④ Mit der Konstante `wdSelectionIP` wird an dieser Stelle geprüft, ob der Textcursor aktiv ist und keine Markierung besteht.



Eine mögliche Ausgabe der Prozedur

Auf beliebige Textbereiche zugreifen

Mit dem `Range`-Objekt kann auf beliebige Teile des Dokuments zugegriffen werden, ohne dass diese markiert sind. Das Objekt benötigt dazu definierte Start- und Endpositionen.

Viele Unterobjekte des `Document`-Objekts, beispielsweise `Words`, `Sentences` oder `Paragraphs`, liefern direkt oder indirekt einen Verweis auf ein `Range`-Objekt, das ein bestimmtes Wort, einen Satz oder einen Absatz umfasst. Die Angabe einer Start- und einer Endposition ist dabei nicht erforderlich.

Beispiel: *DokumentInhalt.doc*, Modul `ThisDocument`

Die VBA-Prozedur im folgenden Beispiel greift auf den zweiten Absatz des aktuellen Dokuments zu und ermittelt dessen erstes Wort. Zusätzlich wird der erste Buchstabe dieses Worts rot markiert.

```
Sub AbsatzInformation()
    Dim Absatz As Range
①   Set Absatz = ActiveDocument.Paragraphs(2).Range
②   MsgBox "Der 2. Absatz beginnt mit dem Wort """ & Absatz.Words.First & """."
③   Absatz.Characters.First.Font.Color = wdColorRed
End Sub
```

① Über die `Range`-Eigenschaft wird ein Verweis auf den Textbereich des zweiten Absatzes des aktuellen Dokuments ermittelt und der Objektvariablen `Absatz` zugewiesen.

② Die `Words`-Auflistung enthält alle Wörter des Textbereichs. Hier wird das erste Wort ausgegeben.

③ Ähnlich der `Words`-Auflistung enthält `Characters` alle Zeichen des Bereichs. Jedes Zeichen besitzt das Unterobjekt `Font`, das ein Verändern der Zeichenformatierung erlaubt. An dieser Stelle wird das erste Zeichen des Absatzes rot formatiert.

Weitere Möglichkeiten in Word

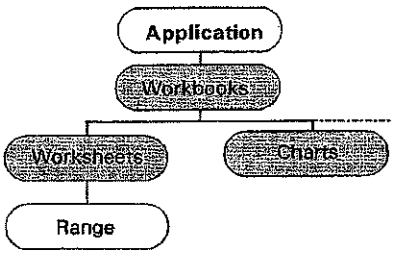
Word stellt noch viele weitere Objekte zur Verfügung, auf die Sie mit VBA zugreifen können. Eine kleine Auswahl zeigt die folgende Tabelle.

Name	Unterobjekt von	Erklärung
Table	Document, Selection	Zugriff auf Tabellen im Dokument
Font	Word, Sentence, Range, Paragraph, Character, Selection	Ändern und Auslesen von Zeichenformaten
Fields	Document, Selection, Range	Zugriff auf Felder im entsprechenden Bereich
Find	Range, Selection	Durchführen einer Suche (bzw. Suchen und Ersetzen)
SmartTag	Document, Range	Auslesen, Hinzufügen und Ändern von SmartTags
Style	Document	Zugriff auf Formatvorlagen des Dokuments
TaskPane	-	Repräsentiert einen einzelnen Aufgabenbereich

7.2 Mit Excel programmieren

7.2.1 Das Excel-Objektmodell

VBA ermöglicht den Zugriff auf jede Excel-Komponente. Als Objekte stehen Ihnen beispielsweise Arbeitsmappen, Tabellenblätter, Diagramme und Zellen zur Verfügung. Auch im Objektmodell von Excel sind alle Objekte dem Application-Objekt untergeordnet.



Ausschnitt aus dem Excel-Objektmodell

Jede geöffnete Arbeitsmappe wird durch ein Workbook-Objekt innerhalb der Workbooks-Auflistung repräsentiert. Jede Arbeitsmappe kann ihrerseits mehrere Tabellenblätter enthalten. Diese werden über die Worksheets-Auflistung angesprochen. Enthält eine Arbeitsmappe Diagramme, so sind sie über die Auflistung Charts zu erreichen. Wie Word verfügt auch Excel über ein Range-Objekt. Mit ihm können Sie auf einzelne Zellen, Zeilen oder Spalten in einem Tabellenblatt zugreifen und diese verändern.

Das Excel-Objektmodell enthält noch viele weitere Objekte. Mehr Informationen finden Sie in der Online-Hilfe zu Excel unter dem Stichwort *Microsoft Excel-Objekte* oder im Objektkatalog.

Excel stellt zusätzlich einige Objekte zur Verfügung, die einen schnellen Zugriff auf ein gerade aktuelles Objekt erlauben. So können Sie beispielsweise mit dem Objekt ActiveCell schnell auf die aktuelle Zelle verweisen. Weitere Beispiele sind die Objekte ActiveChart, ActiveSheet und ActivePrinter.

7.2.2 Mit Arbeitsmappen und Tabellenblättern arbeiten

Arbeitsmappen öffnen, neu erstellen und schließen

Für die Programmierung mit Arbeitsmappen stehen Ihnen die Auflistung Workbooks und das Workbook-Objekt zur Verfügung. Über die Methoden der Workbooks-Auflistung können Sie Arbeitsmappen erstellen, vorhandene öffnen oder schließen.

Add	Erstellt eine leere Arbeitsmappe. Rückgabewert ist ein Verweis auf die neue Arbeitsmappe als Workbook-Objekt.
Open [Pfad]	Öffnet eine bereits vorhandene Arbeitsmappe

Die folgenden Methoden und Eigenschaften stellen alle `Workbook`-Objekte zur Verfügung:

`Save`	Speichert die Arbeitsmappe
`SaveAs`	Speichert die Arbeitsmappe unter einem neuen Namen
`Saved`	Ermittelt, ob die Arbeitsmappe gespeichert wurde
`Close`	Schließt die Arbeitsmappe mit einer Nachfrage, falls Änderungen zu speichern sind

Auf die gerade angezeigte Arbeitsmappe verweisen Sie mit dem `ActiveWorkBook`-Objekt. Die VBA-Anweisungen können sich dabei auch in einer anderen Arbeitsmappe als der angezeigten befinden. Wollen Sie die Arbeitsmappe ansprechen, in der sich der Programmcode befindet, können Sie `ThisWorkBook` verwenden.

Arbeiten mit Tabellenblättern

Alle Tabellenblätter einer Arbeitsmappe sind in der Auflistung `WorkSheets` enthalten. Mit den Methoden der Auflistung können Sie Tabellenblätter erstellen oder löschen.

`Add`	Erzeugt ein leeres Tabellenblatt
`Move`	Verschiebt ein Tabellenblatt

Die Auflistung `Sheets` enthält im Gegensatz zu `WorkSheets` nicht nur alle Tabellenblätter, sondern auch alle Diagramme.

Einzelne Tabellenblätter besitzen den Objekttyp `Worksheet` mit den folgenden Eigenschaften und Methoden:

`Activate`	Aktiviert das Tabellenblatt
`Delete`	Löscht das Tabellenblatt
`Name`	Ermittelt den Namen des Tabellenblatts oder legt diesen fest
`Range`	Verweist auf ein `Range`-Objekt, das den Zugriff auf Zellen und Zellbereiche erlaubt

Beispiel: *TabellenZugriff.xls*, Modul `DieseArbeitsmappe`

Wenn eine Excel-Arbeitsmappe sehr viele Arbeitsblätter enthält, kann der Aufruf des gewünschten Arbeitsblatts langwierig sein. Die folgende VBA-Prozedur erlaubt das gezielte Suchen eines Arbeitsblatts nach dessen Namen.

Notizen

```
Sub SucheTabellenblatt()
  Dim TabBlatt As Object, BlattName As String
  BlattName = InputBox("Bitte geben Sie einen Suchbegriff ein!", _
                       "Tabellenblatt suchen")
  For Each TabBlatt In Sheets
    If InStr(LCase(TabBlatt.Name), LCase(BlattName)) > 0 Then
      TabBlatt.Activate
      Exit Sub
    End If
  Next
  MsgBox "Leider existiert kein Tabellenblatt mit diesem Namen." , _
         vbInformation
End Sub
```

① Die Objektvariable `TabBlatt` soll Verweise auf einzelne Tabellenblätter speichern.

② Mit einem Eingabedialog wird ein Suchbegriff abgefragt und in der Variablen `BlattName` gespeichert.

③ Die Schleife durchläuft alle in der aktuellen Arbeitsmappe enthaltenen Tabellenblätter. Die Variable `TabBlatt` speichert dabei jeweils einen Objektverweis.

④ An dieser Stelle erfolgt die eigentliche Suche. Mit der VBA-Funktion `InStr` wird geprüft, ob `BlattName` im Namen des jeweiligen Tabellenblatts (`Name`-Eigenschaft) vorkommt. Die VBA-Funktion `Lcase` wandelt dabei beide Begriffe in Kleinbuchstaben um. War die Suche nicht erfolgreich, liefert `InStr` den Wert Null.

⑤ Das erste gefundene Tabellenblatt wird über die Methode `Activate` aktiviert.

⑥ Die Angabe `Exit Sub` bricht die Prozedur nach erfolgreicher Suche ab.

⑦ Wird der eingegebene Suchbegriff in keinem Namen gefunden, zeigt ein Meldungsfenster eine entsprechende Information an.

7.2.3 Zugriff auf Zellen und Zellbereiche

Bei der Entwicklung von integrierten Lösungen mit VBA ist der Zugriff auf einzelne Zellen oder Bereiche einer Tabelle die häufigste Zugriffsart. Über das `Range`-Objekt eines Tabellenblatts können Sie gezielt auf einzelne Zellen verweisen. Dabei können Sie deren Inhalt auslesen oder verändern. Sie besitzen außerdem Zugriff auf alle Formatierungen.

Folgende Eigenschaften und Methoden stehen Ihnen bei einem `Range`-Objekt zur Verfügung (Auswahl):

`Column`	Liefert die erste Spalte eines angegebenen Bereichs als Indexwert
`Row`	Liefert die erste Zeile eines angegebenen Bereichs Zeile als Indexwert
`Columns`	Liefert eine Auflistung aller Spalten in einem Bereich
`Rows`	Liefert eine Auflistung aller Zeilen in einem Bereich
`EntireColumn`	Verweist auf ein `Range`-Objekt, das eine oder mehrere Spalten umfasst
`EntireRow`	Verweist auf ein `Range`-Objekt, das eine oder mehrere Zeilen umfasst
`Count`	Liefert die Anzahl der Zellen
`Select`	Markiert einen Zellbereich
`Interior`	Zugriff auf die Formatierung des Innenbereichs einer Zelle
`Formula`	Liefert die Formel einer Zelle
`CurrentRegion`	Verweist auf einen von leeren Zellen umschlossenen Bereich

Das Objekt `ActiveCell` ermöglicht einen schnellen Zugriff auf die aktuelle Zelle. Mit den Eigenschaften `Address`, `Column` und `Row` können Sie bestimmen, um welche Zelle es sich handelt.

Zellen adressieren

Beim Zugriff über das Range-Objekt erfolgt die Adressierung wie in Excel üblich über Buchstaben (Spalten) und Zahlen (Zeilen). Dabei sind alle in Excel möglichen Schreibweisen, beispielsweise für die Adressierung ganzer Zellbereiche, zugelassen.

Beispiel zur Zelladressierung mit der Range-Eigenschaft eines Tabellenblatts:

```
Worksheets(1).Range("A4").Value
Worksheets("Konten").Range("B2:B8").Count
Range("A1:A5","B1:B5").Count
```

Wenn Sie kein Tabellenblatt angeben, gebraucht VBA beim Verwenden des Range-Objekts automatisch das aktuelle Tabellenblatt.

VBA bietet weiterhin die Möglichkeit, auf einzelne Zellen über einen numerischen Index zu verweisen. Dazu können Sie die Cells-Eigenschaft eines Tabellenblatts verwenden, die ebenfalls ein Range-Objekt zurückgibt. Beispiele zur Zelladressierung mit der Cells-Eigenschaft eines Tabellenblatts:

```
Worksheets(1).Cells(4,1).Value        ' Vierte Zelle in der ersten Spalte (A4)
Cells(516).Value                      ' Dritte Zelle in der vierten Spalte (D3)
Range(Cells(3,3),Cells(4,6)).Select   ' Gleichbedeutend mit Range("C3:F4").Select
```

Der Indexwert kann auch fortlaufend verwendet werden. Die erste Zeile umfasst dabei die Zellen von 1 bis 256, die zweite Zeile beginnt danach mit dem Zellindex 257.

Mit der Cells-Zugriffeigenschaft lassen sich nur einzelne Zellen adressieren. Erst durch eine Kombination von Range und Cells können Zellbereiche auch mit Indexwerten berechnet werden.

Beispiel: *Artikel.xls*, Modul DieseArbeitsmappe

In einem Excel-Arbeitsblatt wird eine Artikelliste gepflegt. Mithilfe einer VBA-Prozedur soll eine Möglichkeit geschaffen werden, gezielt nach einer Artikelnummer zu suchen und den zugehörigen Namen und Preis mit einem Meldungsfenster anzuzeigen.

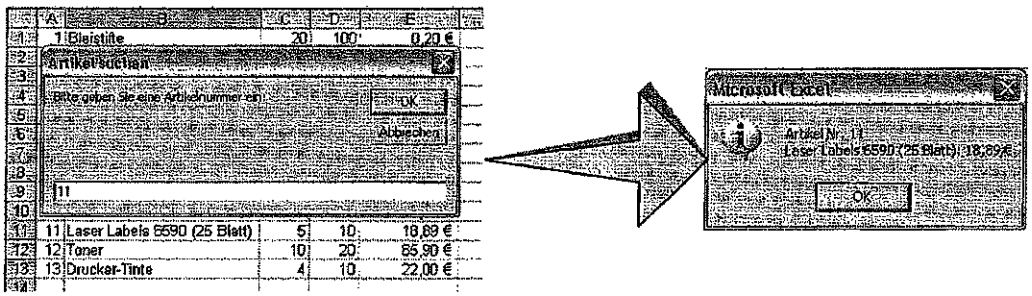
```
Sub ArtikelSuchen()
  Dim SuchNr As String, Zaehler As Integer
  SuchNr = InputBox("Bitte geben Sie eine Artikelnummer ein:", _
                    "Artikel suchen")
①  ThisWorkbook.Sheets("Artikel").Activate
②  For Zaehler = 1 To Range("A1").CurrentRegion.Rows.Count
③    If Cells(Zaehler, 1).Value = SuchNr Then
       MsgBox "Artikel Nr. " & SuchNr & vbCrLf & _
④      Cells(Zaehler, 2).Value & " - " & _
       Cells(Zaehler, 5).Value & " €", vbInformation
       Exit Sub
     End If
   Next
   MsgBox "Der gewünschte Artikel wurde nicht gefunden.", vbCritical
End Sub
```

① Hier wird das Tabellenblatt mit dem Namen *Artikel* aktiviert. Über das Objekt ThisWorkBook wird dabei auf die Arbeitsmappe verwiesen, in der sich die Prozedur befindet.

② In einer For-Schleife werden alle gefüllten Zeilen, beginnend mit der Zelle A1, durchlaufen. Das Objekt CurrentRegion bezeichnet einen Bereich gefüllter Zellen, die von leeren Zellen umgeben sind.

③ An dieser Stelle wird geprüft, ob der Wert in der Spalte 1 und der aktuellen Zeile mit der gesuchten Artikelnummer übereinstimmt.

④ War die Suche erfolgreich, werden die Daten des gefundenen Artikels mit der Hilfe der Cells-Zugriffseigenschaft ausgelesen und in einem Meldungsfenster angezeigt.



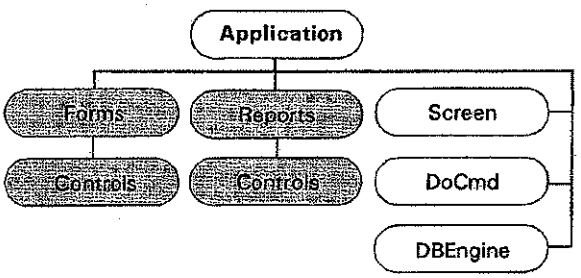
Nach Eingabe einer Artikelnummer werden die zugehörigen Artikeldaten gesucht

7.3 Mit Access programmieren

7.3.1 Das Access-Objektmodell

Die große Anzahl Objekte im Access-Objektmodell erlaubt den Zugriff auf alle Komponenten der Anwendung. Sie können damit z. B. Daten aus Formularen lesen oder Daten hinzufügen.

Jedes geöffnete Formular einer Access-Datenbank wird durch ein `Form`-Objekt in der `Forms`-Auflistung repräsentiert. Den Zugriff auf die einzelnen Kontrollelemente erlauben die untergeordneten `Control`-Objekte, die sich in `Controls`-Auflistungen befinden. Die `Reports`-Auflistung verweist auf die zum Ausdruck vorgesehenen Berichte der Datenbank.



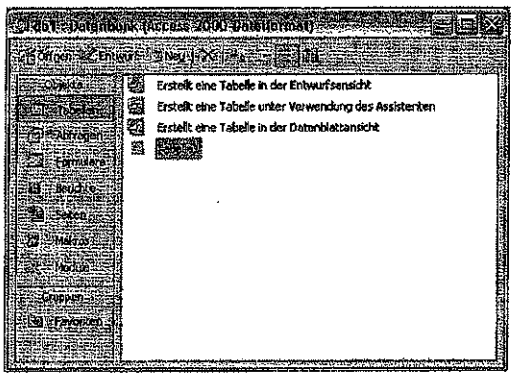
Ausschnitt aus dem Access-Objektmodell

Das Access-Objektmodell enthält viele weitere Objekte, die jedoch nicht alle in Standardanwendungen benötigt werden. Mehr Informationen finden Sie in der Online-Hilfe zu Access unter dem Stichwort *Microsoft Access-Objekte* oder im Objektkatalog.

Die verschiedenen Access-Objekte erlauben zunächst nur das Ermitteln und Verändern von Daten in Formularen oder Berichten. Es lassen sich zwar auch einzelne Werte aus Tabellen und Abfragen ermitteln, der Zugriff auf den in Tabellen gespeicherten Datenbestand bleibt jedoch verwehrt. Wenn Sie mit diesen Daten arbeiten möchten, um z. B. Datenabfragen aus einer Tabelle zu erstellen, müssen Sie auf eine so genannte Datenbank-Engine zugreifen. VBA stellt Ihnen dafür zwei verschiedene Objektmodelle zur Verfügung: die schon in älteren Access-Versionen verwendeten Datenzugriffsobjekte (DAO - **D**ata **A**ccess **O**bjects und seit Access 2000 ADO - **A**ctiveX **D**ata **O**bjects). Das ADO-Objektmodell hat den Vorteil, dass es sehr kompakt ist und den Zugriff auf Datenquellen verschiedener Herkunft vereinfacht.

7.3.2 Mit Access-Datenbanken arbeiten

Access speichert alle Komponenten einer Anwendung, wie Tabellen, Formulare, Berichte, Module, in einer Datenbank. Das `Application`-Objekt erlaubt das Öffnen, Erstellen und Schließen solcher Datenbanken. Im Gegensatz zu anderen Office-Anwendungen kann in Access nur jeweils eine Datenbank geöffnet sein.



Die Elemente einer Access-Datenbank werden im Datenbankfenster verwaltet

Folgende Methoden des Application-Objekts stehen zur Verfügung:

OpenCurrentDatabase "Pfad"	Öffnet eine vorhandene Datenbank
CloseCurrentDatabase	Schließt die aktuelle Datenbank
NewCurrentDatabase "Pfad"	Erstellt eine neue Datenbank mit dem angegebenen Dateinamen

Das CurrentProject-Objekt erlaubt den schnellen Zugriff auf alle Komponenten der geöffneten Datenbank.

Das dem Application-Objekt untergeordnete Objekt DoCmd ermöglicht das einfache Ausführen von Makrobefehlen, beispielsweise, um Formulare zu öffnen oder Daten zu suchen. Alle Makrobefehle sind als Methoden des Objekts realisiert. Die folgende Tabelle zeigt einige wichtige Methoden des DoCmd-Objekts.

OpenForm "Name"	Öffnet das angegebenen Formular der Datenbank
OpenReport "Name"	Öffnet den angegebenen Bericht
Quit	Beendet Access
Close	Schließt das aktuelle Fenster oder die Datenbank
GoToRecord	Springt zu einem bestimmten Datensatz in einem Formular

7.3.3 Zugriff auf Steuerelemente in Formularen und Berichten

Formulare und Berichte ansprechen

Auf geöffnete Formulare greifen Sie über die Forms-Auflistung zu. Diese Auflistung enthält aber nicht die geschlossenen Formulare der Datenbank. Berichte erreichen Sie auf die gleiche Weise über die Reports-Auflistung.

Forms(Indexnummer)	Reports(Indexnummer)
Forms("Formularname")	Reports("Berichtname")
Forms!Formularname	Reports!Berichtname
Forms![Formularname]	Reports![Berichtname]

- ☑ In Access wird die Schreibweise mit einem Ausrufezeichen (!) bevorzugt. Enthält der Formular- bzw. Berichtname Sonderzeichen oder Leerzeichen, wird er in eckige Klammern eingeschlossen.
- ☑ Auf Eigenschaften und Methoden des Formulars bzw. Berichts wird über einen nachgestellten Punkt zugegriffen.

Befindet sich der Programmcode in einem Formular- oder Berichtsmodul, ermöglicht das Schlüsselwort Me einen schnellen Verweis auf das aktuelle Formular bzw. den aktuellen Bericht.

Notizen

Steuerelemente ansprechen

Steuerelemente dienen in Formularen und Berichten zum Darstellen und Eingeben von Daten. Sie werden in der Auflistung `Controls` verwaltet, die jedem Formular bzw. Bericht untergeordnet ist.

```
Forms!Formularname!Steuerelementname
Forms![Formularname]![Steuerelementname]
Forms("Formularname")("Steuerelementname")
```

- ☑ Um ein einzelnes Steuerelement zu adressieren, verwenden Sie wieder das Ausrufezeichen (!). Steuerelemente in Berichten werden auf die gleiche Weise angesprochen wie Steuerelemente in Formularen. Sie sind jedoch schreibgeschützt.
- ☑ Auf Eigenschaften und Methoden der Steuerelemente greifen Sie über einen nachgestellten Punkt zu.

Jedes Steuerelement besitzt den Objekttyp `Control` und stellt unter anderem folgende Eigenschaften und Methoden zur Verfügung:

Eigenschaft/Methode	Steuerelemente	Beschreibung
`Value`	alle	Auslesen oder Zuweisen des Steuerelementinhalts
`ItemsSelected`	Listenfelder	Auflistung mit den Indexwerten aller markierten Elemente eines Listenfelds
`ItemData(Indexwert)`	Listenfelder	Liefert den Wert eines bestimmten Elements
`Visible`	Alle	Steuerelement ein- bzw. ausblenden
`Name`	Alle	Ermittelt den Namen des Steuerelements

Beispiel: *Gehalt.mdb*, Modul `FormularZugriff`

Die folgende VBA-Prozedur öffnet ein Access-Formular und liest, solange der Anwender es wünscht, nacheinander die Namen von Mitarbeitern aus. Am Ende wird das Formular wieder geschlossen.

```
Sub FormularLesen()
  Dim Meldung As String, Auswahl As Integer
①  On Error GoTo Ende
②  DoCmd.OpenForm "Mitarbeiter"
③  DoCmd.GoToRecord acDataForm, "Mitarbeiter", acFirst
  Do
④   Meldung = "Name: " & Forms!Mitarbeiter!Vorname & " " & _
          Forms!Mitarbeiter!Name & vbCr & vbCr & _
          "Weiteren Datensatz anzeigen?"
    Auswahl = MsgBox(Meldung, vbYesNo)
⑤   DoCmd.GoToRecord acDataForm, "Mitarbeiter", acNext
⑥  Loop Until Auswahl = vbNo
Ende:
⑦   DoCmd.Close
End Sub
```

① Falls ein Fehler auftritt, z. B. durch den Versuch, nach dem letzten Datensatz weiterzublättern, soll die Prozedur beendet werden.

② Mithilfe des `DoCmd`-Objektes wird das Formular MITARBEITER geöffnet. Falls es bereits offen ist, wird es aktiviert.

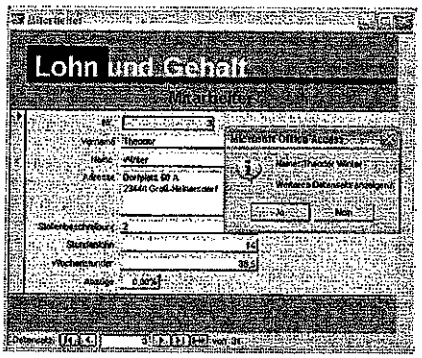
③ Die Methode `GoToRecord` ermöglicht das Navigieren in einem Formular. An dieser Stelle wird zum ersten Datensatz gesprungen.

④ Der Vorname und der Nachname des Mitarbeiters werden aus den entsprechenden Formularfeldern (Steuerelementen) gelesen und in einem Meldungsfenster ausgegeben. Der Anwender kann dabei gleichzeitig wählen, ob weitere Datensätze angezeigt werden sollen.

⑤ Hier wird mit `GoToRecord` zum nächsten Datensatz gewechselt.

⑥ Es werden so lange Datensätze angezeigt, bis das Meldungsfenster mit der Schaltfläche NEIN geschlossen wird. In diesem Fall enthält die Variable `Auswahl` den konstanten Wert `vbNo`.

⑦ Mit der `DoCmd`-Methode `Close` wird das aktuelle Fenster, in diesem Fall das Formular MITARBEITER, geschlossen.

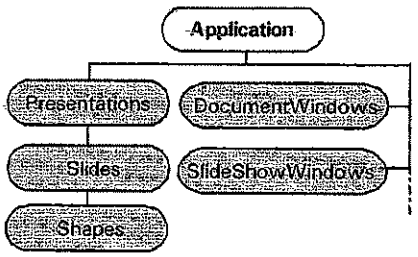


Der erste Datensatz wird angezeigt

7.4 Mit PowerPoint programmieren

7.4.1 Das PowerPoint-Objektmodell

Auch PowerPoint ermöglicht in VBA den Zugriff auf alle Komponenten der Anwendung. Als Objekte stehen Ihnen beispielsweise Präsentationen und Folien zur Verfügung. Beginnend mit dem `Application`-Objekt können Sie auf die einzelnen Objekte zugreifen und PowerPoint automatisieren.



Ausschnitt aus dem PowerPoint-Objektmodell

Jede geöffnete Präsentation wird durch ein `Presentation`-Objekt innerhalb der `Presentations`-Auflistung repräsentiert. Jede Präsentation kann wiederum mehrere Folien (`Slide`-Objekte) enthalten. Alle Formen auf einer Folie sind über die `Shapes`-Auflistung verfügbar.

Alle PowerPoint-Objekte finden Sie in der Online-Hilfe zu PowerPoint unter dem Stichwort *Microsoft PowerPoint-Objekte* oder im Objektkatalog.

PowerPoint stellt, wie die meisten anderen Office-Anwendungen auch, zusätzliche Objekte zur Verfügung, die einen schnellen Zugriff auf ein gerade aktuelles Objekt erlauben. So können Sie beispielsweise mit dem Objekt `ActivePresentation` schnell auf die aktuelle Präsentation verweisen.

7.4.2 Mit Präsentationen und Folien arbeiten

Präsentationen öffnen, neu erstellen und schließen

Für die Programmierung mit PowerPoint-Präsentationen stehen Ihnen die Auflistung `Presentations` und das `Presentation`-Objekt zur Verfügung. Alle in PowerPoint geöffneten Präsentationen verwaltet VBA in der `Presentations`-Auflistung.

Notizen

Folgende Methoden stellt Ihnen die `Presentations`-Auflistung zur Verfügung:

`Add`	Erstellt eine leere Präsentation. Der Rückgabewert ist ein Verweis auf die neue Präsentation als `Presentation`-Objekt.
`Open "Pfad"`	Öffnet eine bereits vorhandene Präsentation
`Close`	Schließt eine Präsentation
`Save`	Speichert eine Präsentation

Auf Folien zugreifen

Die Auflistung `Slides` verwaltet alle Folien einer Präsentation. Auch diese Auflistung und das `Slide`-Objekt selbst stellen unter anderem folgende Eigenschaften und Methoden zur Verfügung:

`Add`	Fügt eine neue Folie ein
`Count`	Ermittelt die Anzahl der vorhandenen Folien
`Delete`	Löscht eine Folie
`Background`	Unterobjekt für den Zugriff auf den Folienhintergrund
`Select`	Aktiviert eine Folie (im Entwurfsmodus)
`Layout`	Ermittelt oder bestimmt die Layoutvorlage einer Folie
`SlideIndex`	Ermittelt die Position einer Folie in der Präsentation

7.4.3 Zugriff auf den Folieninhalt

Der Inhalt einer Folie setzt sich vor allem aus so genannten Formen zusammen, die Sie auf einer Folie platzieren können. Diese Formen können auch Grafiken oder Text enthalten. Mithilfe der Auflistung `Shapes` und dem zugehörigen `Shape`-Objekt ist der Zugriff auf alle Formen einer Folie möglich.

Formen hinzufügen

Neue Formen fügen Sie über spezielle Methoden der `Shapes`-Auflistung ein:

Methode	Formtyp	Beispiel
`AddLine`	Linienobjekt	`ActivePresentation.Slides(1).Shapes.AddLine 10, 10, 20, 10`
`AddPicture`	Grafikobjekt aus einer Datei	`ActivePresentation.Slides(1).Shapes.AddPicture "c:\bild.pcx"`
`AddShape`	Grafikobjekt, z. B. AutoForm	`ActivePresentation.Slides(1).Shapes.AddShape msoShapeCube`
`AddTable`	Tabelle	`ActivePresentation.Slides(1).Shapes.AddTable 4, 2`
`AddTextBox`	Textrahmen	`ActivePresentation.Slides(1).Shapes.AddTextbox msoTextOrientationHorizontal, 20, 20, 150, 10`
`AddTextEffect`	WordArt-Objekt	`ActivePresentation.Slides(1).Shapes.AddTextEffect msoTextEffect2, "Text"`

Formatieren von Formen

Über verschiedene Unterobjekte und Eigenschaften des `Shape`-Objekts können Sie eine Form gezielt bearbeiten und formatieren.

Fill	Erlaubt den Zugriff auf das Füllmuster
Line	Verweist auf die Außenlinie einer Form
Left, Top	Ermitteln oder bestimmen die Position der Form
IncrementRotation	Dreht die Form um einen bestimmten Betrag
TextFrame	Erlaubt den Zugriff auf einen in der Form enthaltenen Textrahmen

Mit dem Unterobjekt TextRange des TextFrame-Objekts erhalten Sie Zugriff auf den Text eines Textrahmens. Sie können Text zuweisen, verändern oder formatieren.

Beispiel: *FolienZugriff.ppt*, Modul FolienZugriff

Mit einer VBA-Prozedur soll eine neue Folie mit farbigem Hintergrund erstellt werden. Danach wird ein Textrahmen mit einer Überschrift erzeugt und formatiert.

```
Sub FolienBearbeiten()
  Dim NeueFolie As Slide
①  Set NeueFolie = ActivePresentation.Slides.Add(1, ppLayoutBlank)
②  NeueFolie.FollowMasterBackground = msoFalse
③  NeueFolie.Background.Fill.ForeColor.RGB = RGB(210, 150, 40)
④  With NeueFolie.Shapes.AddTextbox(msoTextOrientationHorizontal, _
                                   50, 50, 350, 50)
⑤    .TextFrame.WordWrap = msoFalse
⑥    .TextFrame.TextRange.Text = "Überschrift der Folie"
     .TextFrame.TextRange.Font.Name = "Arial"
     .TextFrame.TextRange.Font.Size = 32
  End With
End Sub
```

① An dieser Stelle wird mithilfe der Methode Add vor der ersten Folie eine neue leere Folie (ppLayoutBlank) eingefügt.

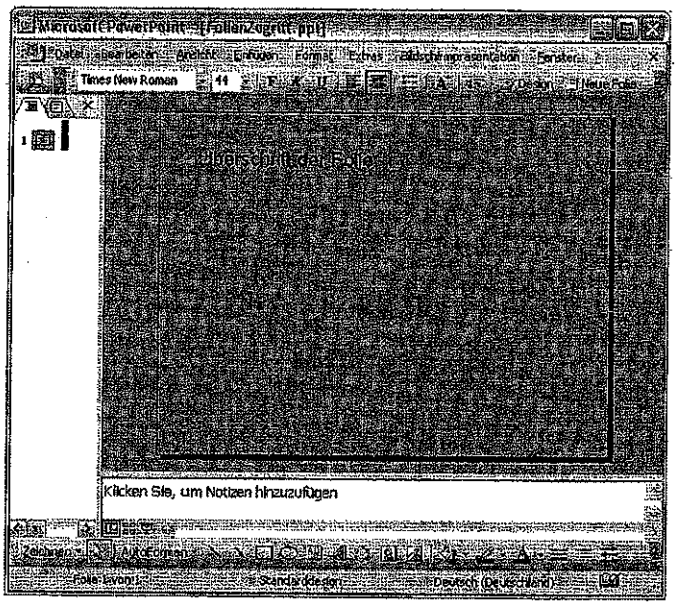
② Damit der Folienhintergrund bearbeitet werden kann, wird zuerst der Hintergrund der Masterfolie abgeschaltet.

③ Über das Background-Objekt und dessen Füllung (Fill-Objekt) wird hier eine Vordergrundfarbe für den Hintergrund zugewiesen und im nächsten Schritt damit ein Farbverlauf erzeugt.

④ Mit der Methode AddTextBox wird ein neuer Textrahmen erzeugt. Der zurückgegebene Objektverweis auf die neue Form wird für einen einfachen Objektzugriff mit der With-Anweisung verwendet.

⑤ Über das TextFrame-Objekt kann auf den Textrahmen zugegriffen werden. Hier wird der Zeilenumbruch ausgeschaltet.

⑥ Das TextRange-Objekt ermöglicht eine Texteingabe (Eigenschaft Text) und im Anschluss das Formatieren des Textes.



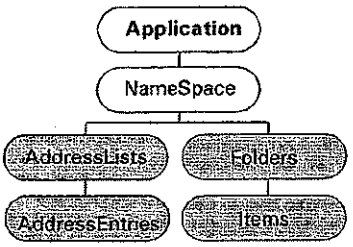
Die neue Folie mit eingefügtem Text

7.5 Mit Outlook programmieren

7.5.1 Das Outlook-Objektmodell

Outlook besitzt, verglichen mit den anderen Office-Anwendungen, eingeschränkte Programmiermöglichkeiten. Dennoch ermöglicht es den Zugriff auf Adressbücher, Ordner und die darin enthaltenen Elemente.

An der Spitze des Outlook-Objektmodells steht das `Application`-Objekt. Über das untergeordnete `NameSpace`-Objekt erhalten Sie Zugriff auf den Nachrichtenspeicher der Outlook-Daten. Dazu gehören z. B. die Ordnerstruktur und Menübefehle, die Sie, beispielweise aus einer anderen Anwendung heraus, automatisieren können.



Ausschnitt aus dem Outlook-Objektmodell

VBA-Anweisungen in Outlook werden für jeden Benutzer zentral in der Datei *VBAProject.otm* gespeichert, auch wenn sie in Modulen organisiert sind. Das Speichern von Anweisungen in einzelnen Outlook-Elementen ist nicht möglich.

Den Programmcode eines Outlook-Moduls können Sie nicht mit einem Dokument weitergeben bzw. speichern. Sie können das entsprechende Modul jedoch als **.bas*-Datei exportieren und später importieren.

Alle Outlook-Objekte finden Sie in der Online-Hilfe zu Outlook unter dem Stichwort MICROSOFT OUTLOOK-OBJEKTE oder im Objektkatalog.

7.5.2 Auf Ordner zugreifen

Das `NameSpace`-Objekt ist der Schlüssel zu allen vorhandenen Outlook-Ordnern. Jeder Ordner besitzt den Objekttyp `MAPIFolder`.

Bevor Sie über das `NameSpace`-Objekt auf Outlook-Elemente zugreifen können, müssen Sie stets mithilfe der `Application`-Methode `GetNameSpace` einen Verweis auf das `NameSpace`-Objekt zurückgeben.

```
Set Objektverweis = Application.GetNameSpace("MAPI")
```

- ☑ Die Methode liefert einen Verweis auf das `NameSpace`-Objekt einer Outlook-Datenquelle. Die einzige durch VBA unterstützte Datenquelle ist dabei `MAPI`.
- ☑ Über den zurückgelieferten Objektverweis können Sie auf alle untergeordneten Objekte des `NameSpace`-Objekts zugreifen, z. B. Adressbücher und Ordner.

MAPI (Messaging Application Program Interface) ist eine Schnittstelle unter Windows, die Programmen beispielsweise die Nutzung von E-Mail-Diensten ermöglicht.

Folgende Unterobjekte stehen für ein `NameSpace`-Objekt zur Verfügung:

`AddressEntries`	Ermöglicht den Zugriff auf Adressbücher (Kontakte)
`Folders`	Ermöglicht den Zugriff auf die Outlook-Ordner vom Typ `MAPIFolder`

Das `NameSpace`-Objekt verfügt über die Methode `GetDefaultFolder`, mit der Sie einen Verweis auf einen Outlook-Standardordner, z. B. POSTEINGANG, erstellen können.

```
Set OrdnerVerweis = NameSpaceVerweis.GetDefaultFolder(TypKonstante)
```

7.5.3 Auf Elemente zugreifen

Neue Elemente erstellen

Elemente sind die Outlook-Entsprechung zu Dokumenten. Mögliche Elemente sind beispielsweise E-Mails, Aufgaben oder Notizen. Auf vorhandene Elemente greifen Sie über das `NameSpace`-Objekt zu. Für das Erstellen neuer Elemente ist jedoch die Methode `CreateItem` des `Application`-Objekts zuständig.

```
Set Objektverweis = Application.CreateItem(TypKonstante)
```

- ☑ Die Methode liefert einen Verweis auf das neue Element vom Objekttyp `Item`.
- ☑ Über die Typkonstante wird bestimmt, welches Element erzeugt werden soll.

Folgende Konstanten sind beispielsweise möglich:

`olMailItem`	E-Mail-Nachricht
`olNoteItem`	Notiz
`olContactItem`	Kontakt
`olTaskItem`	Aufgabe

Auf vorhandene Elemente zugreifen

Auf vorhandene Outlook-Elemente greifen Sie in der Regel über die `Items`-Auflistung eines `MAPIFolder`-Objekts zu. Die `Items`-Auflistung enthält unterschiedliche Objekte, je nach Art der im Ordner enthaltenen Elemente. Eine Auswahl der möglichen Objekte zeigt die folgende Auflistung:

`AppointmentItem`	Verweist auf einen Termin im Ordner Kalender
`ContactItem`	Verweist auf einen Kontakt im Kontakteordner
`MailItem`	Stellt eine E-Mail-Nachricht in einem Posteingangsordner dar
`NoteItem`	Verweist auf eine Notiz im Ordner NOTIZEN

Jedes Element verfügt über Unterobjekte, Methoden und Eigenschaften, die eine für das jeweilige Element spezifische Bearbeitung erlauben. So verfügt das Objekt `MailItem` beispielsweise über eine Auflistung `Recipients`, die alle Empfänger der E-Mail-Nachricht enthält, und die Eigenschaft `Body`, die den Text der E-Mail enthält.

Notizen

Beispiel: *LetzteNachricht.bas*

Den Programmcode des Beispiels finden Sie in der Datei *LetzteNachricht.bas*. Wählen Sie im Menü DATEI den Befehl DATEI IMPORTIEREN, um den in der Datei gespeicherten Programmcode zu importieren. Outlook fügt danach ein neues Modul MODUL1 ein, das die Beispiel-Prozedur enthält.

Die VBA-Prozedur soll das Datum der zuletzt empfangenen Nachricht ermitteln und zusammen mit der Anzahl der Nachrichten im Posteingang anzeigen.

```
Sub LetzteNachricht()
  Dim MeinNS As NameSpace, Posteingang As MAPIFolder
  Dim Email As MailItem, Anzahl As Integer
①  Set MeinNS = Application.GetNamespace("MAPI")
②  Set Posteingang = MeinNS.GetDefaultFolder(olFolderInbox)
③  Anzahl = Posteingang.Items.Count
④  Set Email = Posteingang.Items(Anzahl)
   MsgBox "Es befinden Sich " & Anzahl & " Nachrichten im Posteingang" & _
          vbCrLf & vbCrLf & "Die letzte Nachricht wurde empfangen: " & _
⑤         Email.CreationTime, vbInformation
End Sub
```

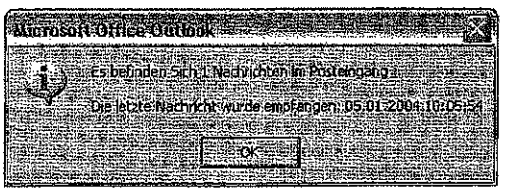
① Über die Methode `GetNameSpace` wird ein Verweis auf das `NameSpace`-Objekt erzeugt.

② Die `NameSpace`-Methode `GetDefaultFolder` liefert an dieser Stelle einen Verweis auf den Standardordner POSTEINGANG.

③ Über die `Items`-Auflistung wird die E-Mail-Anzahl im Ordner ermittelt.

④ Die Objektvariable `Email` erhält hier einen Verweis auf die letzte E-Mail im Ordner.

⑤ Die Eigenschaft `CreationTime` liefert Datum und Zeit des Empfangs der E-Mail.



Das Ergebnis der Prozedur

7.6 Schnellübersicht

Sie möchten ...	
Word-Dokumente öffnen bzw. schließen	`Documents.Open` bzw. `Documents(Index).Close`
auf das aktuelle Word-Dokument verweisen	`ActiveDocument`
auf eine Markierung in Word zugreifen	`Selection`
auf Textabschnitte eines Word-Dokuments zugreifen	`Range`
Excel-Arbeitsmappen öffnen bzw. schließen	`Workbooks.Open` `Workbooks(Index).Close`
auf Excel-Tabellenblätter zugreifen	`Sheets`-Auflistung, `WorkSheet`-Objekt
auf einzelne Zellen in Excel zugreifen	`Range("Adresse")` oder `Cells(Index, Index)`
eine Access-Datenbank öffnen bzw. schließen	`Appication.OpenCurrentDatabase` `Application.CloseCurrentDatabase`
Access-Makrobefehle ausführen	`DoCmd`-Objekt
auf Steuerelemente in Access-Formularen oder -Berichten zugreifen	`Forms!Formularname!Steuerlement` `Reports!Berichtname!Steuerelement`

PowerPoint-Präsentationen öffnen bzw. schließen	`Presentations.Open` `Presentation(Index).Close`
auf Folien in PowerPoint zugreifen	`Slides-Auflistung`
auf Folieninhalte einer PowerPoint-Folie zugreifen	`Shapes-Auflistung`
Zugriff auf Daten in Outlook-Ordnern	`Application.GetNameSpace("MAPI")`
auf Outlook-Elemente zugreifen	`Item`-Objekte (z. B. `MailItem`, `NoteItem` usw.)
neue Outlook-Elemente erstellen	`Application.CreateItem(Typ)`

7.7 Übung

Übung 1

① Öffnen Sie nacheinander alle im Kapitel aufgeführten Beispiele für Word, Excel, Access und PowerPoint.

② Testen Sie die VBA-Anweisungen, und versuchen Sie, die Aufgaben der einzelnen Objekte zu verstehen.

③ Nehmen Sie kleine Änderungen an den Beispielen vor, z. B. um eine andere Zelle in Excel oder ein anderes Steuerelement in Access auszulesen.

Übung 2

① Starten Sie Outlook, und öffnen Sie die VBA-Entwicklungsumgebung.

② Fügen Sie ein neues Modul ein.

③ Übernehmen Sie die Prozedur `LetzteNachricht` aus diesem Kapitel.

④ Testen Sie die Prozedur, und verändern Sie diese so, dass sie das Datum einer anderen Nachricht anzeigt.

Notizen

8 Kommunikation zwischen Office-Anwendungen

In diesem Kapitel erfahren Sie

- → welche Möglichkeiten zum Datenaustausch zwischen Office-Anwendungen existieren
- → wie Sie die ActiveX-Automation für integrierte Lösungen nutzen
- → wie Sie auf Automationsobjekte mit VBA zugreifen

Voraussetzungen

- ✓ Objektmodelle der Office-Anwendungen
- ✓ Programmierung mit VBA

8.1 Integrierte Office-Automatisierung

Typen integrierter Projekte

Es gibt zwei wesentliche Möglichkeiten zur Integration von Office-Anwendungen: Die Integration von Funktionalitäten der einzelnen Office-Anwendungen und die Integration von Daten.

Integration von Funktionalitäten	Eine Office-Anwendung kann über verschiedene im Office-Paket vorhandene Funktionalitäten verfügen und diese für eigene Zwecke einsetzen. So können Sie z. B. Datenreihen in Access verwalten und speichern und zusätzlich in einem VBA-Programm von Access aus die Funktionalität von Excel verwenden, um die Daten zu analysieren.
Integration der Daten	Eine Office-Anwendung kann Daten verwenden, die außerhalb dieser Anwendung existieren. Sie können so z. B. Adressbestände aus Excel oder Access zum Erstellen von Serienbriefen in Word verwenden.

In den meisten Fällen sind beide Integrationsmöglichkeiten eng miteinander verbunden. Um die Funktionalität einer Office-Anwendung nutzen zu können, ist häufig auch ein Datenaustausch notwendig.

Zugriff auf Daten anderer Office-Anwendungen

Es existieren viele verschiedene Möglichkeiten, Daten zwischen Office-Anwendungen auszutauschen. Nicht alle davon eignen sich auch für den Einsatz unter VBA.

Zwischenablage	Über die Zwischenablage, die Windows zur Verfügung stellt, können Daten zwischen vielen Anwendungen in verschiedenen Formaten ausgetauscht werden. Die Zwischenablage ist vor allem für einen manuellen Datenaustausch von Bedeutung.
COM-Automation	COM-Automation (Component Object Model) ermöglicht den Datenaustausch als Verbindung von Objekten. Alle Office-Anwendungen unterstützen dieses Verfahren.
ODBC	ODBC (Open Database Connectivity) ermöglicht den Austausch von Daten mit beliebigen Anwendungen (vor allem Datenbanken), die einen entsprechenden ODBC-Treiber zur Verfügung stellen.

ADO/DAO	ADO (ActiceX Data Objects) und DAO (Data Access Objekts) sind spezielle Objektmodelle von Microsoft, die den Zugriff auf und die Manipulation von Daten mit VBA vereinfachen.
DDE	DDE (Dynamic Data Exchange) ist ein älteres Verfahren zum Datenaustausch, das in früheren Office-Versionen zum Einsatz kam. Es ist nicht objektorientiert und nicht einfach anzuwenden.

Steht eine funktionale Integration im Vordergrund, d. h., soll auf bestimmte Funktionen einer fremden Office-Anwendung zugegriffen werden, wird heute vor allem die COM-Automation eingesetzt. Sie ist mit VBA sehr einfach umzusetzen.

Sollen bei einer integrierten Anwendung große Mengen Daten ausgetauscht und manipuliert werden, können Sie ActiveX-Datenobjekte (ADO) verwenden. ADO verfügt über ein einfaches Objektmodell, mit dem Sie auf verschiedene Datenquellen zugreifen können.

Für den Datenaustausch über die Zwischenablage stellen einige Objekte in Word, Excel und PowerPoint die Methoden `Copy` und `Paste` bereit.

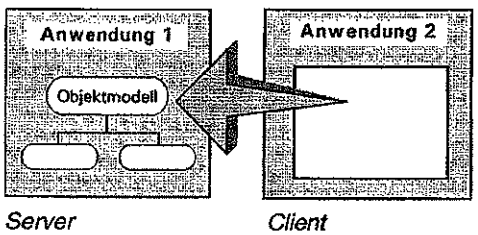
8.2 Technische Grundlagen

Begriffsklärung

Das Component Object Model (COM) ist die Basis der Ansteuerung von Microsoft Office und sämtlicher Anwendungen, die diese Spezifikation implementieren. Es stellt eine standardisierte Schnittstelle zur Verfügung, über die eine Anwendung unabhängig von der verwendeten Programmiersprache gesteuert werden kann. Auf COM bauen weitere Techniken und Protokolle auf, die in der folgenden Tabelle kurz vorgestellt werden.

COM	Dies ist das Basisprotokoll zur Ansteuerung anderer Anwendungen. Die betreffenden Anwendungen besitzen nach außen eine Schnittstelle, die von anderen Programmen zur Ansteuerung genutzt werden kann. Es können nur Anwendungen auf dem gleichen Rechner gesteuert werden.
OLE-Automatierung	Ein COM-Objekt erhält hier einen speziellen Rahmen, um es in eine andere Anwendung einzubetten oder darin zu verlinken. Beispielsweise können Sie eine Excel-Tabelle in ein Word-Dokument einbetten und direkt in Word bearbeiten. OLE ist die Abkürzung für Object Linking and Embedding (Objekt-Verknüpfung und -einbettung).
OCX / ActiveX	Ein OCX ist eine Komponente, die sich in andere Anwendungen integrieren und damit wiederverwenden lässt. Nachteilig ist hierbei, dass die OCX-Komponente separat weitergegeben und auf dem Zielsystem registriert werden muss, da sie nicht in eine **.exe*-Datei eingebunden werden kann. Für die Verwendung von OCX-Komponenten im Internet wurden kleinere (in Bezug auf die Programmgröße) Komponenten geschaffen, die auf einer einfacheren OCX-Spezifikation aufbauen.
DCOM	Durch Distributed COM (verteiltes COM) können Anwendungen rechnerübergreifend gesteuert werden.
COM+	COM+ stellt eine Erweiterung vom COM um administrative Verbesserungen und neue Komponententypen ((nicht) konfigurierbar) dar.

Eine Anwendung, die Dienste einer anderen Anwendung benutzt, wird als Client bezeichnet. Die Anwendung, die die angeforderten Dienste ausführt, heißt Server bzw. Automation-Server. Das Verwenden von COM/ActiveX/OLE in der Programmierung wird daher auch als Automation bezeichnet. OLE bezeichnet dagegen das manuelle Verknüpfen von Anwendungen über eingebettete Dokumente.



Wenn Sie über Automation Zugriff auf die Objekte einer fremden Anwendung erhalten, sind dabei auch alle Methoden und Eigenschaften eingeschlossen. Sie bekommen dadurch die Möglichkeit, die durch Objekte erschlossene Funktionalität einer Anwendung (z. B. Excel) und deren Daten in einer anderen Anwendung (z. B. Word) einzusetzen.

Die COM-/ActiveX-/OLE-Technologie ist nicht automatisch in allen Windows-Anwendungen vorhanden. Eine entsprechende Anwendung muss ihre Objekte explizit zur Verfügung stellen. Dies ist bei allen Office-Anwendungen der Fall, aber auch z. B. bei Visio und anderen Anwendungen.

Automation mit VBA verwenden

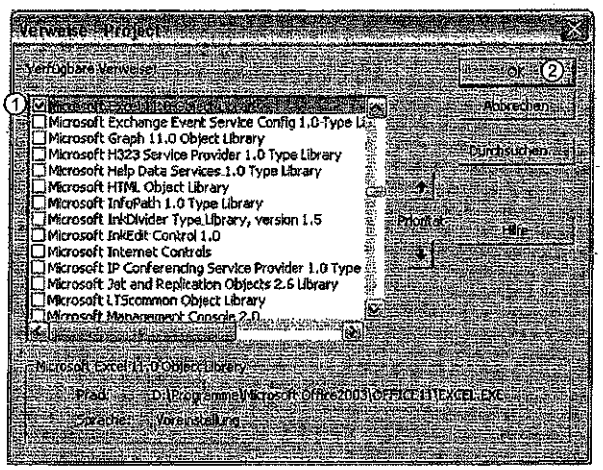
Das Programmieren integrierter Office-Lösungen über COM-Automation-Server unterscheidet sich nur wenig von der Programmierung einer einzelnen Office-Anwendung. Es sind jedoch einige zusätzliche Schritte notwendig:

⇒ Erstellen Sie einen Verweis auf die Objektbibliothek der Office-Anwendung, auf die Sie zugreifen möchten.
⇒ Erzeugen Sie eine Instanz des fremden Objekts, und weisen Sie dieses einer Objektvariablen zu.
⇒ Programmieren Sie die benötigten Anweisungen. Über das Automationsobjekt erhalten Sie Zugriff auf alle Unterobjekte, Methoden und Eigenschaften der fremden Anwendung.
⇒ Geben Sie die Automationsobjekte frei, wenn kein weiterer Zugriff notwendig ist.

8.3 Verweis auf eine Objektbibliothek erstellen

Standardmäßig ist in der VBA-Entwicklungsumgebung nur die Objektbibliothek der aktuellen Office-Anwendung bekannt. Bei der Automation greifen Sie jedoch auf fremde Objekte zu. Damit die VBA-Entwicklungsumgebung alle Funktionen der Programmierunterstützung bereitstellen kann, erstellen Sie einen Verweis auf die fremde Objektbibliothek.

⇒ Öffnen Sie die VBA-Entwicklungsumgebung in der Office-Anwendung, in der Sie programmieren möchten.
⇒ Rufen Sie den Menüpunkt EXTRAS - VERWEISE auf, um das Dialogfenster VERWEISE zu öffnen.
⇒ Aktivieren Sie in der Auflistung VERFÜGBARE VERWEISE ① das Kontrollfeld für die benötigte Objektbibliothek.
⇒ Schließen Sie den Dialog mit OK ②.



Dialogfenster VERWEISE

Wenn Sie einen Verweis erstellt haben, erkennt die VBA-Entwicklungsumgebung automatisch die neuen Objekte und stellt die üblichen Funktionen zur Programmierunterstützung zur Verfügung, z. B. den Objektkatalog oder das automatische Auflisten der Methoden und Eigenschaften.

8.4 Deklarieren von Objektvariablen für die Automation

Es existieren zwei Möglichkeiten, um Objektvariablen für die Automation zu deklarieren: *early binding* (frühes Binden) und *late binding* (spätes Binden). Beide Formen unterscheiden sich im zugewiesenen Objekttyp.

Das Binden einer Variablen bezeichnet in diesem Zusammenhang das Festlegen eines Objekttyps für eine Objektvariable.

Deklaration mit spätem Binden

```
Dim appXL As Object
```

- ☑ Die Deklaration erfolgt mit dem allgemeinen Typ `Object`.
- ☑ Der Zugriff auf das Objekt erfolgt langsamer, da VBA den genauen Objekttyp zur Laufzeit ermitteln muss.

Deklaration mit frühem Binden

```
Dim appXL As Objektklasse
```

- ☑ Bei der Deklaration wird angegeben, auf welche Objektklasse sich die Objektvariable beziehen soll.
- ☑ Der Zugriff auf das Objekt erfolgt schneller, da VBA die genaue Objektklasse bereits kennt.

Durch das Deklarieren mit der Angabe des genauen Objekts wird die Objektvariable frühzeitig an das Automationsobjekt gebunden. Dadurch erreichen Sie einen Geschwindigkeitszuwachs. Die Objektvariable kann im Programmverlauf jedoch auf keine anderen Objekte verweisen, beispielsweise zuerst auf ein Word- und später auf ein Excel-Objekt.

Objektklasse angeben

Der Objektklasse setzt sich aus der Bezeichnung der Anwendung, z. B. Word, Excel, und der Angabe eines Objekts, eines Typs oder einer Objektklasse zusammen.

```
Anwendungsname.Objekttyp        Beispiel:   Access.Application
                                            Excel.Sheet
                                            Word.Application
```

Die folgende Übersicht zeigt einige der möglichen Objektklassen für die Office-XP-Anwendungen:

Anwendung	Objektklasse	Wiedergegebenes Objekt
Access	`Access.Application`	Access-Anwendung
Access	`Access.Form`	Access-Formular
Excel	`Excel.Application`	Excel-Anwendung
Excel	`Excel.Sheet`	Excel-Arbeitsmappe
Word	`Word.Application`	Word-Anwendung
PowerPoint	`PowerPoint.Application`	PowerPoint-Anwendung

Optional kann, mit einem Punkt getrennt, die gewünschte Version der Anwendung angeschlossen werden, z. B. `Access.Application.11`. Dies ist besonders wichtig, wenn mehr als eine Version der Anwendung installiert ist.

Wenn Sie frühes Binden verwenden möchten, müssen Sie einen Verweis auf die zugrunde liegende Objektbibliothek einrichten.

Notizen

8.5 Erzeugen einer Objektinstanz

8.5.1 Grundlagen

Wenn Sie ein VBA-Programm in einer Office-Anwendung erstellen, können Sie sofort auf alle Objekte der betreffenden Anwendung zugreifen. Für den Zugriff auf fremde Objekte benötigen Sie eine Instanz des Objekts, die Sie gegebenenfalls erzeugen und einer Objektvariablen zuweisen müssen.

Beispiel: *AccessZugriff.doc*, Modul `AccessZugriff`

Mithilfe einer VBA-Prozedur soll durch Automation eine Access-Datenbank im gleichen Verzeichnis geöffnet werden. Aus dem Formular MITARBEITER wird danach der Name eines Mitarbeiters ausgelesen und in das aktuelle Word-Dokument eingefügt.

```
Sub AccessFormularÖffnen()
  Dim appAC As Access.Application
  MDBPfad = ActiveDocument.Path & " \gehalt.mdb"
  Set appAC = CreateObject("Access.Application")
  appAC.OpenCurrentDatabase MDBPfad
  appAC.Visible = True
  appAC.DoCmd.OpenForm "Mitarbeiter"
  ActiveDocument.Range.InsertAfter "Mitarbeitername: " & _
    appAC.Forms("Mitarbeiter")!Name
  Set appAC = Nothing
End Sub
```

① Die Objektvariable `appAC` wird deklariert und an dieser Stelle an das `Access.Application`-Objekt gebunden.

② Da sich die Datenbank im gleichen Verzeichnis wie das Word-Dokument befinden soll, wird zuerst dessen Dateipfad ermittelt und mit dem Namen der Datenbank verbunden.

③ Mithilfe der VBA-Funktion `CreateObject` wird eine Instanz des Objekts erzeugt und der Objektvariablen `appAC` zugewiesen.

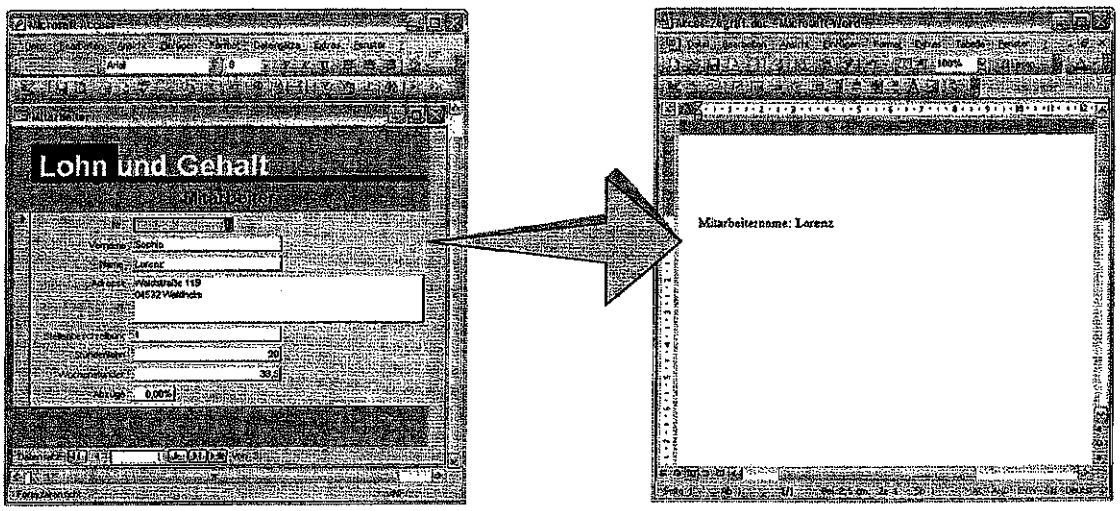
④ Die Objektvariable `appAC` kann nun wie ein `Application`-Objekt in Access verwendet werden. Mit der Methode `OpenCurrentDatabase` wird die angegebene Datenbank geöffnet.

⑤ Standardmäßig wird die Datenbank unsichtbar im Hintergrund geöffnet. In diesem Fall ist dies nicht notwendig, daher wird die Eigenschaft `Visible` mit dem Wert `True` belegt.

⑥ Über das `DoCmd`-Objekt wird das Formular MITARBEITER geöffnet.

⑦ Der Mitarbeitername wird aus dem Datensatz des Formulars ausgelesen und an der Cursorposition im aktuellen Word-Dokument eingefügt.

⑧ Durch das Zuweisen von `Nothing` wird das Automationsobjekt aus dem Speicher entfernt.



Der Name des Mitarbeiters wird im Access-Formular ausgelesen und im Word-Dokument eingefügt

Das Beispiel ist nur funktionsfähig, wenn Sie für das Modul zuerst einen Verweis zur Access-Objektbibliothek erstellen.

Möglichkeiten zum Erzeugen einer Objektinstanz

VBA ermöglicht drei verschiedene Möglichkeiten, um die Objektinstanz für eine Automation zu erzeugen bzw. zu ermitteln.

`CreateObject`-Funktion	Startet die zugrunde liegende Anwendung und liefert einen Objektverweis zurück. Die Objektvariable muss vorher deklariert werden. Über einen Fehlercode kann ermittelt werden, ob die Anwendung installiert ist.
`GetObject`-Funktion	Liefert einen Objektverweis auf eine bereits gestartete Anwendung zurück. Über einen Fehlercode kann überprüft werden, ob die Anwendung bereits gestartet ist.
Schlüsselwort `New`	Erzeugt eine Objektinstanz. Die zugrunde liegende Anwendung wird dabei gestartet. Im Fehlerfall wird kein Fehlercode zurückgegeben.

8.5.2 Objektinstanz mit `CreateObject` erzeugen

Mit der Funktion `CreateObject` erzeugen Sie eine neue Objektinstanz, wobei die zugrunde liegende Anwendung gestartet wird.

```
Set Objektvariable = CreateObject("Objektklasse")
```

- ☑ Die Funktion `CreateObject` erzeugt eine Instanz der als Zeichenkette angegebenen Objektklasse, z. B. `Word.Application` oder `Access.Application.11`.
- ☑ Die Funktion liefert einen Verweis auf das neue Objekt zurück. Der Objektverweis muss einer vorher deklarierten Objektvariablen zugewiesen werden.
- ☑ Die zugrunde liegende Office-Anwendung wird stets neu gestartet, auch wenn sie bereits aktiv ist.

Prüfen, ob eine Anwendung installiert ist

Die Funktion `CreateObject` liefert den Fehlercode 429, falls die benötigte Anwendung nicht installiert ist.

```
On Error Resume Next
Set Objektvariable = CreateObject("Objektklasse")
If Err = 429 Then
  MsgBox "Anwendung nicht installiert"
End If
```

Der Fehler mit dem Fehlercode 429 tritt immer dann auf, wenn eine ActiveX-Komponente ein Objekt nicht erzeugen kann oder wenn es nicht möglich ist, eine Referenz auf ein bestimmtes Objekt zurückzugeben.

Notizen

8.5.3 Objektinstanz mit `GetObject` erzeugen

Mit der Funktion `GetObject` können Sie die Objektinstanz einer bereits gestarteten Anwendung ermitteln und zurückgeben.

```
Set Objektvariable = GetObject("Pfad", "Objektklasse")
```

- ☑ Das Argument `Pfad` ist optional. Es beschreibt den vollständigen Pfad und den Namen der Anwendung bzw. des Dokuments einer Anwendung.
- ☑ Wird kein Pfad angegeben, greift die Funktion auf eine bereits gestartete Anwendung zu. In diesem Fall muss eine Objektklasse angegeben werden.

Das Argument `Pfad` steuert das Verhalten der Funktion `GetObject`. Sie besitzen folgende Möglichkeiten:

Pfad-Angabe	Erklärung	Beispiel
keine Angabe	Die Funktion greift auf eine bereits gestartete Anwendung zu.	`appXL=GetObject(, "Excel.Application")`
""	Die Funktion erzeugt eine neue Instanz und startet dabei die Anwendung (entspricht `CreateObject`).	`appXL=GetObject("", "Excel.Application")`
"Pfad"	Die Funktion öffnet das angegebene Dokument.	`appXL=GetObject("C:\Daten\Statistik.xls")`

Die Funktion `GetObject` bietet eine bessere Geschwindigkeit im Programmablauf als `CreateObject`, wenn mehrfach auf eine bestimmte Anwendung zugegriffen werden soll, da das aufwändige Starten der Anwendung entfällt. Zusätzlich schonen Sie dabei die Speicherressourcen des Anwenders.

Prüfen, ob eine Anwendung gestartet ist

Die Funktion `GetObject` liefert den Fehlercode 429, falls die benötigte Anwendung nicht gestartet ist.

```
On Error Resume Next
Set Objektvariable = GetObject(, "Objektklasse")
If Err = 429 Then
  MsgBox "Anwendung ist nicht aktiv, bitte starten!"
End If
```

8.5.4 Objektinstanz mit `New` erstellen

Mit dem Schlüsselwort `New` erhalten Sie eine weitere Möglichkeit, eine Objektinstanz zu erstellen.

```
Dim Objektvariable As New Objektklasse
```

- ☑ Die Deklaration erfolgt in der üblichen Form. Vor der Angabe der Objektklasse wird das Schlüsselwort `New` angegeben. Die Objektklasse bezeichnet den Typ der Objektvariablen und das Objekt für das Erstellen einer neuen Instanz.
- ☑ `New` kann nur verwendet werden, wenn ein Verweis auf die Objektbibliothek eingerichtet wurde.
- ☑ Die dem Objekt zugrunde liegende Anwendung wird gestartet.

8.6 Objekt schließen und Arbeitsspeicher freigeben

Jede für eine Automation deklarierte Objektvariable belegt Arbeitsspeicher, den Sie freigeben sollten, wenn das Objekt nicht mehr benötigt wird. Im Gegensatz zu normalen lokalen Variablen erledigt dies VBA in der Regel nicht automatisch.

Objekte mit eigener Methode schließen

Die meisten größeren Objekte, z. B. das `Application`-Objekt, besitzen die Methoden `Close` oder `Quit` zum Schließen. Nach dem Aufruf wird das entsprechende Objekt geschlossen und steht für erneute Zugriffe nicht mehr zur Verfügung.

```
Objektvariable.Close
Objektvariable.Quit
```

Arbeitsspeicher freigeben

Durch den Aufruf der Methode `Close` wird ein Objekt ordnungsgemäß geschlossen. Oft bleibt die zugehörige Anwendung jedoch geladen und verwendet weiterhin Arbeitsspeicher.

Mithilfe des Task-Managers von Windows können Sie überprüfen, ob eine Anwendung im Hintergrund geladen ist. Auch wenn kein Anwendungsfenster zu sehen ist, erscheint ein Eintrag in der Liste der geladenen Anwendungen.

Sie können mit dem Schlüsselwort `Nothing` eine Objektvariable freigeben. Dabei wird das Objekt geschlossen, und der belegte Arbeitsspeicher wird freigegeben.

```
Set Objektvariable = Nothing
```

Das Schlüsselwort `Nothing` bezeichnet ein undefiniertes Objekt, das keinen Speicherplatz benötigt. Das vorher zugewiesene Objekt wird ungültig, und der zugehörige Arbeitsspeicher wird freigegeben.

Erzeugen Sie in einer Prozedur mehrfach neue Objektinstanzen und tauschen Sie per Automation Daten aus, kann dadurch ein beträchtlicher Arbeitsspeicherbedarf entstehen. Da die Objekte auch nach dem Beenden der Prozedur für den Anwender unsichtbar im Arbeitsspeicher gehalten werden, kann es zu Programmabstürzen durch Speichermangel kommen, ohne dass ein Zusammenhang zur VBA-Prozedur erkennbar ist. Sie sollten daher am Ende einer Prozedur alle für die Automation benötigten Objektvariablen freigeben.

Bei der Verwendung des `Application`-Objekts sollten Sie zuerst das Objekt mit der `Close`-Methode schließen und anschließend die Objektvariable freigeben. Für alle anderen Objekte reicht das Freigeben der Objektvariablen mit `Nothing`.

Notizen

Beispiel: *SummeAusExcel.doc*, Modul `ThisDocument`

Für das Erstellen einer Rechnung soll aus einer Excel-Arbeitsmappe die Rechnungssumme ausgelesen werden. Um Arbeitsspeicher zu sparen und eine schnellere Ausführung der Prozedur zu erreichen, soll zuerst geprüft werden, ob Excel bereits gestartet ist.

```
Sub SummeAusExcel()
  On Error GoTo FehlerSub                                          ①
  Dim appXL As Excel.Application
  Set appXL = GetObject(, "Excel.Application")                     ②
  appXL.Workbooks.Open (ActiveDocument.Path & "\Werte.xls")        ③
  ActiveDocument.Bookmarks("SUMME").Select                         ④
  Selection.InsertAfter appXL.ActiveSheet.Range("C1")              ⑤
  appXL.Quit                                                       ⑥
  Set appXL = Nothing                                              ⑦
  Exit Sub
FehlerSub:
  If Err = 429 Then                                                ⑧
    Set appXL = GetObject("", "Excel.Application")
  Else
    Err.Raise Err.Number
  End If
  Resume Next                                                      ⑨
End Sub
```

① An dieser Stelle wird die Fehlerüberwachung eingeschaltet. Im Fehlerfall wird zur Sprungmarke `FehlerSub` verzweigt.

② Mit der Funktion `GetObject` wird versucht, eine neue Objektinstanz von der bereits laufenden Excel-Anwendung zu erzeugen. Ist dies nicht möglich, wird der Fehler 429 erzeugt.

③ Über die Excel-`Workbooks`-Auflistung wird die Arbeitsmappe *Werte.xls* geöffnet. Da sich die Excel-Datei im gleichen Verzeichnis wie das aktuelle Word-Dokument befindet, wird dessen Dateipfad ermittelt.

④ Im Word-Dokument wurde die Textmarke *SUMME* an der Stelle eingefügt, an der die Summe erscheinen soll. Mithilfe der `Bookmarks`-Auflistung wird diese Textmarke markiert.

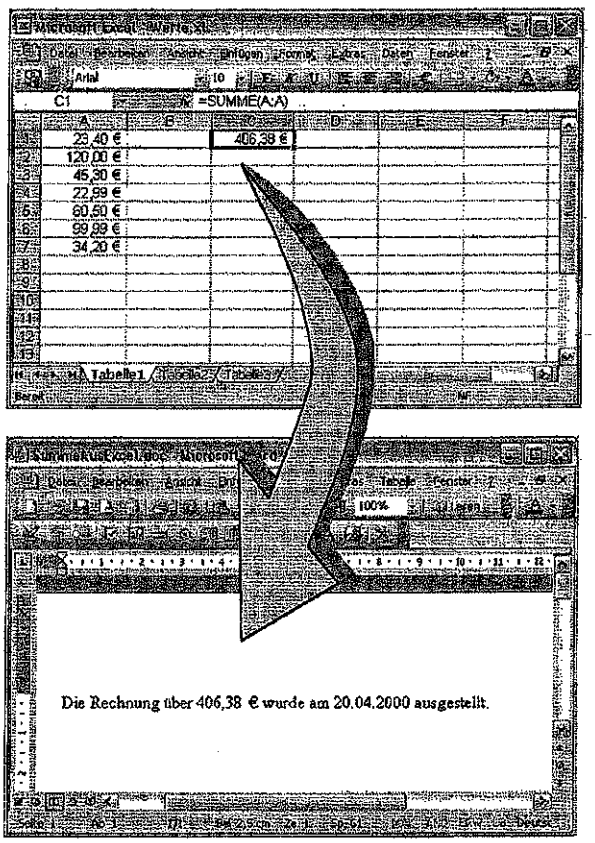
⑤ Aus der Excel-Tabelle wird der Inhalt der Zelle C1 ausgelesen und hinter der Markierung (bzw. nach der Textmarke) eingefügt.

⑥ Die Methode `Quit` schließt die Excel-Anwendung.

⑦ Durch das Zuweisen von `Nothing` wird auch die Objektvariable und der zugehörige Arbeitsspeicher freigegeben.

⑧ Falls Excel beim Aufruf der Prozedur noch nicht gestartet war, wird der Fehler 429 erzeugt. In diesem Fall wird durch das Argument `""` der Funktion `GetObject` Excel geladen, und es wird eine Objektinstanz erzeugt.

⑨ Die Anweisung `Resume Next` verzweigt zu der Anweisung, die der fehlerverursachenden Anweisung folgt. Wird der Fehler bei ② erzeugt, ist dies die Anweisung ③.



Auslesen eines Werts aus Excel und Einfügen in Word

8.7 Meldungen des Automationsobjekts unterdrücken

Die Office-Anwendungen blenden häufig Meldungsfenster ein, die Fehler oder Warnungen anzeigen und den Anwender zur Eingabe einer Information oder zum Bestätigen auffordern. Beispielsweise fragen alle Office-Anwendungen beim Beenden der Anwendung, ob ungesicherte Dokumente gespeichert werden sollen.

Die Ausführung eines VBA-Programms wird so lange unterbrochen, bis das Meldungsfenster durch eine Aktion des Anwenders bestätigt wird. Bei der Programmierung mit Automationsobjekten stören diese Meldungen jedoch. Unter Umständen können sie auch zu falschen Ergebnissen beim Datenaustausch mit der Anwendung führen.

Falls das Automationsobjekt unsichtbar geöffnet wird, werden Meldungsfenster nicht am Bildschirm angezeigt und können daher auch nicht bestätigt werden. Solche Fälle führen zu einem Aufhängen des Programms.

Access, Excel und Word verfügen jedoch über Möglichkeiten, Meldungsfenster des Automationsobjekts über VBA auszuschalten.

Meldungen in Word unterdrücken

Die Eigenschaft `DisplayAlerts` bestimmt, ob bzw. welche Meldungen angezeigt werden:

```
Application.DisplayAlerts = Konstante
```

Konstante	Beschreibung
`wdAlertsNone`	Es werden keine Meldungsfenster und Fehler angezeigt. Alle Meldungsfenster werden intern mit der Standardeinstellung bestätigt, und die Programmausführung wird fortgesetzt.
`wdAlertsMessageBox`	Es werden nur Meldungsfenster angezeigt. Alle Fehlermeldungen werden intern abgefangen und der Prozedur zurückgegeben.
`wdAlertsAll`	Alle Meldungsfenster und Fehlermeldungen werden eingeblendet. Fehler werden der Prozedur zurückgegeben.

Meldungen in Excel unterdrücken

Auch in Excel steuern Sie die Anzeige von Meldungen über die Eigenschaft `DisplayAlerts`, es kommen hier jedoch andere Konstanten zum Einsatz.

```
Application.DisplayAlerts = Konstante
```

Konstante	Beschreibung
`False`	Es werden keine Meldungsfenster und Fehler angezeigt. Alle Meldungsfenster werden intern mit der Standardeinstellung bestätigt, und die Programmausführung wird fortgesetzt.
`True`	Alle Meldungsfenster und Fehlermeldungen werden eingeblendet. Fehler werden der Prozedur zurückgegeben.

Auch in Excel müssen Sie die Anzeige von Meldungen am Ende einer Prozedur wieder einstellen, da dies nicht automatisch geschieht.

Meldungen in Access unterdrücken

In Access können Sie die Methode `SetWarnings` des `DoCmd`-Objekts verwenden, um Meldungen auszuschalten.

```
DoCmd.SetWarnings Konstante
```

Konstante	Beschreibung
False	Es werden keine gebundenen Warnungen angezeigt, d. h. keine Meldungsfenster, die eine Bestätigung erwarten.
True	Alle gebundenen Warnungen werden angezeigt.

Bildschirmaktualisierung unterdrücken

Neben dem Ausblenden von Meldungen bieten Word und Excel auch eine Möglichkeit, die Bildschirmaktualisierung zu unterdrücken.

```
Application.ScreenUpdating = Konstante
```

- ☑ Die Methode `ScreenUpdating` erwartet als Argument eine der Konstanten `True` bzw. `False`. Mit `False` deaktivieren Sie die Bildschirmaktualisierung. Mit `True` stellen Sie diese wieder ein.
- ☑ Das Abschalten der Bildschirmaktivierung wird eingesetzt, wenn Formatierungen unsichtbar durchgeführt werden sollen, oder wenn die Ausführung rechenintensiver Makros beschleunigt werden soll.

Ähnlich der Methode `ScreenUpdating` in Word bzw. Excel verfügt auch Access über eine Möglichkeit, die Bildschirmaktualisierung zu unterdrücken.

```
Application.Echo Konstante
```

Die Methode `Echo` erwartet als Argument eine der Konstanten `True` bzw. `False`. Mit `False` deaktivieren Sie die Bildschirmaktualisierung. Mit `True` stellen Sie diese wieder ein.

Die Anzeige von Meldungen und die Bildschirmaktualisierung werden beim Beenden einer Prozedur nicht automatisch wieder aktiviert. Sie müssen dies durch Zuweisen der entsprechenden Konstanten selbst erledigen. Anderenfalls kann es zu Fehlern bei der Ausführung anderer Prozeduren kommen.

8.8 Schnellübersicht

Was bedeutet ...	
OLE	Object Linking and Embedding ermöglicht den Datenaustausch über in Dokumente eingebettete Objekte.
Automation	Automation erlaubt den Datenaustausch mithilfe von VBA über Objekte, die durch Anwendungen zur Verfügung gestellt werden.
COM	Component Object Model beschreibt ein einheitliches Objektmodell für das wechselseitige Bereitstellen von Methoden und Eigenschaften über eine binäre Schnittstelle für verschiedene Anwendungen.
Automationsserver	Anwendung, deren Objekt durch eine andere Anwendung über Automation verwendet werden
Automationsclient	Anwendung, die Objekte einer fremden Anwendung über Automation verwendet

Sie möchten ...	
einen Verweis auf eine Objektbibliothek erstellen	Menüpunkt EXTRAS - VERWEISE
eine Objektvariable für Automation mit frühem Binden deklarieren	`Dim Objektvariable As Objektklasse`
eine Objektvariable für Automation mit spätem Binden deklarieren	`Dim Objektvariable As Object`

eine Objektinstanz für Automation erzeugen (Anwendung neu starten)	`Set Objektvariable = CreateObject("Objektklasse")` `Dim Objektvariable As New Objektklasse`
eine Objektinstanz einer bereits laufenden Anwendung nutzen	`Set Objektvariable = GetObject(, "Objektklasse")`
ein Automationsobjekt schließen	`Objektvariable.Quit/Close`
ein Automationsobjekt freigeben	`Set Objektvariable = Nothing`

8.9 Übung

Übungsdatei: *Übung1.doc; Gehalt.mdb* **Ergebnisdatei:** *Übung1-E.doc*

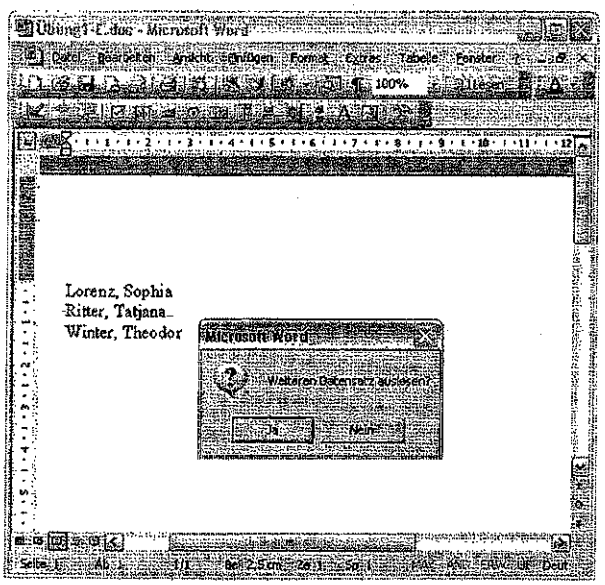
① Öffnen Sie in Word die Datei *Übung1.doc*. Rufen Sie die VBA-Entwicklungsumgebung auf.

② Binden Sie die Access-Objektbibliothek ein, da Sie diese für das Beispiel benötigen.

③ Erstellen Sie im Modul `AccessZugriff` eine neue Prozedur `NamenAuslesen`. Die Prozedur soll Namen aus einem Access-Formular auslesen, solange der Anwender es wünscht. Verwenden Sie als Grundlage die Prozedur `AccessFormularÖffnen` aus diesem Kapitel.

④ Erzeugen Sie eine neue Objektinstanz der `Access.Application`-Klasse, indem Sie das Schlüsselwort `New` verwenden.

⑤ Ermitteln Sie den Speicherort der Datenbank *gehalt.mdb*, indem Sie den Pfad des aktuellen Word-Dokuments auslesen. Weisen Sie den Datenbankpfad einer Variablen zu.



Ausgabe der Namen aus Access und Abfrage des Anwenders

⑥ Öffnen Sie von Word aus die Beispieldatenbank *gehalt.mdb* und das Formular MITARBEITER.

⑦ Lesen Sie aus dem Formular Name und Vorname des Mitarbeiters aus, und geben Sie diese im Word-Dokument in der Form "Name, Vorname" aus.

⑧ Verwenden Sie die Methode `GoToRecord` des `DoCmd`-Objekts, um zum nächsten Datensatz des Formulars zu wechseln.

⑨ In einem Meldungsfenster fragen Sie ab, ob ein weiterer Datensatz ausgelesen werden soll. Das Meldungsfenster soll dazu die Schaltflächen JA und NEIN sowie ein Fragezeichensymbol enthalten.

⑩ Die Namen sollen so lange ausgelesen werden, bis der Anwender die Schaltfläche NEIN betätigt. Verwenden Sie zu diesem Zweck eine `Do-Loop-Until`-Schleife.

⑪ Damit der Anwender die bereits im Dokument ausgegebenen Namen lesen kann, rufen Sie vor dem Einblenden des Meldungsfensters die Methode `ScreenRefresh` des `Application`-Objekts von Word auf.

⑫ Beenden Sie Access am Ende der Prozedur mit der Methode `Quit`, und geben Sie die Objektvariable frei.

9 Benutzerdefinierte Dialoge verwenden

In diesem Kapitel erfahren Sie

- → welche Möglichkeiten zur Anwenderkommunikation existieren
- → wie Sie benutzerdefinierte Dialoge erstellen
- → wie Sie Steuerelemente und UserForm-Dialoge programmieren

Voraussetzungen

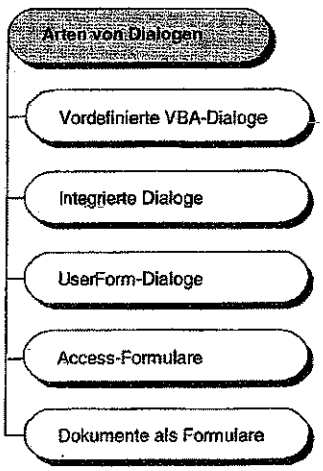
- ✓ Kenntnisse in der ereignisorientierten Programmierung

9.1 Kommunikation mit dem Anwender

Die Kommunikation zwischen einem Anwender und einer Anwendung erfolgt über Dialoge und Formulare. Der Anwender kann in ein Dialogfenster beispielsweise Daten eintragen oder Angaben darin auswählen, die die weiteren Aktionen der Anwendung bestimmen.

Arten von Dialogen und Formularen

- ☑ **Vordefinierte VBA-Dialoge:** VBA besitzt vordefinierte Ein- und Ausgabedialoge (z. B. `InputBox` bzw. `MsgBox`), die Sie für einfache Benutzereingaben und -ausgaben direkt verwenden können.
- ☑ **Integrierte Dialoge:** Sie haben die Möglichkeit, alle in den Office-Anwendungen zur Verfügung stehenden Dialoge, beispielsweise DRUCKEN oder SPEICHERN, in Ihrem VBA-Code zu verwenden.
- ☑ **UserForm-Dialoge (Benutzerdefinierte Dialoge):** In einem speziellen Modus der VBA-Entwicklungsumgebung haben Sie die Möglichkeit, eigene Formulare zu gestalten, so genannte UserForm-Dialoge.
- ☑ **Formulare in Access:** Formulare sind in Access Teile der Datenbank. Sie werden direkt in Access gestaltet und verwaltet.
- ☑ **Dokumente als Formulare:** Dokumente, z. B. in Word, können Steuerelemente enthalten, die Sie programmieren können.

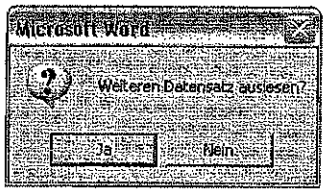


Sie kennen bereits die durch VBA vordefinierten Meldungsfenster und Eingabedialoge, die für eine einfache Kommunikation mit dem Anwender ausreichen. Für umfangreiche integrierte Lösungen benötigen Sie jedoch häufig Formulare, die Sie nach Ihren Anforderungen selbst gestalten können. Für diesen Zweck verwenden Sie UserForm-Dialoge, die außer in Access in allen Office-Anwendungen zur Verfügung stehen.

Mit UserForm-Dialogen können Sie Dialogfenster selbst gestalten, mit Daten füllen und programmieren.

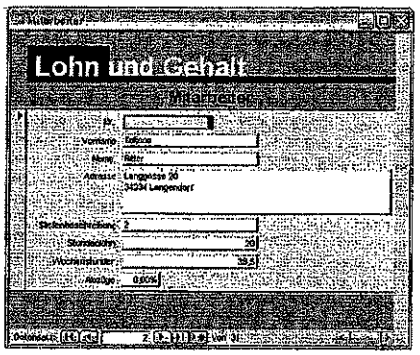
Access verfügt über eigene Formulare, die einen wichtigen Teil jeder Datenbankanwendung darstellen. Access-Formulare werden daher grundlegend anders erstellt, verwendet und programmiert als UserForm-Dialoge.

Beispiele für Dialoge und Formulare



Ein vordefinierter VBA-Dialog (MsgBox)

Ein benutzerdefiniertes Formular



Ein Access-Formular

9.2 UserForm-Dialoge erstellen

Benutzerdefinierte Formulare (engl. "User Forms") gestalten Sie visuell innerhalb der VBA-Entwicklungsumgebung. Dabei stehen Ihnen verschiedene Steuerelemente zur Verfügung, beispielsweise Listenfelder, Schaltflächen oder Eingabefelder.

Mit verschiedenen Ereignisprozeduren können Sie auf Benutzereingaben und Benutzerverhalten reagieren, beispielsweise auf das Betätigen einer Schaltfläche oder das Anzeigen des Formulars. Sie können während der Bearbeitung (per Hand) und während der Anzeige (per VBA-Programmcode) des Formulars Werte zu Steuerelementen hinzufügen und auslesen.

Sie können fertige Formulare probeweise direkt aus der VBA-Entwicklungsumgebung heraus starten. Im Allgemeinen werden Formulare durch eine Prozedur im Programmcode angezeigt.

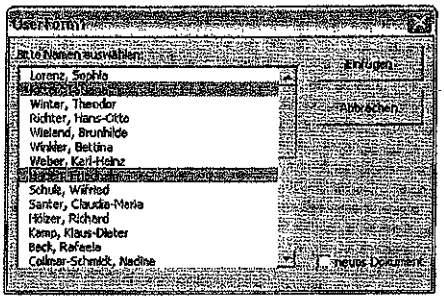
Beispiel: *NamenAusAccessFormular.doc*, Formular `UserForm1`, Modul `AccessZugriff`

In einem Word-Dokument sollen Mitarbeiternamen aus einer Access-Datenbank in einem benutzerdefinierten Formular angezeigt werden. Die VBA-Prozedur soll die Daten aus einem Access-Formular auslesen.

```
Sub NamenAuslesen()
  Dim appAC As New Access.Application
  Dim Auswahl As Long
  MDBPfad = ActiveDocument.Path & "\Gehalt.mdb"
  appAC.OpenCurrentDatabase MDBPfad
  appAC.Visible = False
  appAC.DoCmd.OpenForm "Mitarbeiter"
  Do While appAC.Forms("Mitarbeiter")!Name <> ""
    UserForm1.lst_Namen.AddItem appAC.Forms("Mitarbeiter")!Name & ", " _
                                & appAC.Forms("Mitarbeiter")!Vorname
    appAC.DoCmd.GoToRecord , , acNext
  Loop
  appAC.Quit
  Set appAC = Nothing
End Sub
```

① Die Schleife liest so lange Namen aus dem Access-Formular *Mitarbeiter*, bis nach dem letzten Datensatz der Name leer bleibt.

② Das benutzerdefinierte Formular `UserForm1` enthält das Listenfeld `lst_Namen`, in das an dieser Stelle der Name und der Vorname jedes Mitarbeiters aus dem Access-Formular eingetragen werden.



Das Formular zum Anzeigen der Namen

Die Prozedur `NamenAuslesen` soll aufgerufen werden, wenn das benutzerdefinierte Formular angezeigt wird. Dazu erfolgt ein Aufruf in einer geeigneten Ereignisprozedur des Formulars:

```
① Private Sub UserForm_Activate()
    NamenAuslesen
  End Sub
```

① Der Prozedurname `UserForm_Activate` ist durch VBA vorgegeben. Wenn sich eine Prozedur mit diesem Namen im Modul des Formulars befindet, wird sie beim Anzeigen des Formulars automatisch ausgeführt. Im Beispiel erfolgt dabei der Aufruf der Prozedur `NamenAuslesen`.

Angezeigt wird das Formular mithilfe einer weiteren Prozedur:

```
① Sub FormularZeigen()
    UserForm1.Show
  End Sub
```

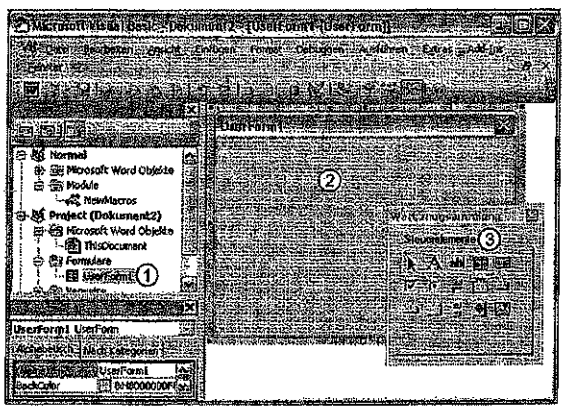
① Zum Anzeigen des Formulars dient die Methode `Show`.

Einen neuen UserForm-Dialog erstellen

UserForm-Dialoge erstellen Sie innerhalb der VBA-Entwicklungsumgebung im so genannten UserForm-Fenster.

→ Rufen Sie den Menüpunkt EINFÜGEN - USERFORM auf.
Alternative: [F7]

Die VBA-Entwicklungsumgebung fügt das Formular UserForm mit einer fortlaufenden Nummer in den Ordner Formulare des aktuellen Projekts ein ①. Gleichzeitig wird ein Fenster mit einem leeren Dialog mit Rasterpunkten ② und der Werkzeugsammlung ③ geöffnet. Die Werkzeugsammlung enthält Symbole für verschiedene Steuerelemente, die Sie in Dialoge einfügen können.



VBA-Entwicklungsumgebung mit leerem UserForm-Dialog

UserForm-Dialoge gehören stets zu einem bestimmten Projekt, beispielsweise zu einem Dokument.

Verfügbare Steuerelemente

Folgende Steuerelemente können Sie in UserForm-Dialogen verwenden:

Steuerelemente	Symbol	Beschreibung
Bezeichnungsfeld	A	Beschreibungstext ohne Eingabemöglichkeit
Textfeld	ab	Feld zur Eingabe und Anzeige von Text, Zahlen etc.
Kombinationsfeld		Einzeiliges Feld für eine Liste gleichartiger Daten mit einem Pfeilsymbol, das bei Klick die Liste öffnet
Listenfeld		Ein- oder mehrzeiliges Feld für die Auswahl eines oder mehrerer Einträge aus einer Liste
Kontrollfeld		Feld zur Aktivierung bzw. Deaktivierung einer Option; es können gleichzeitig mehrere Optionen aktiviert sein

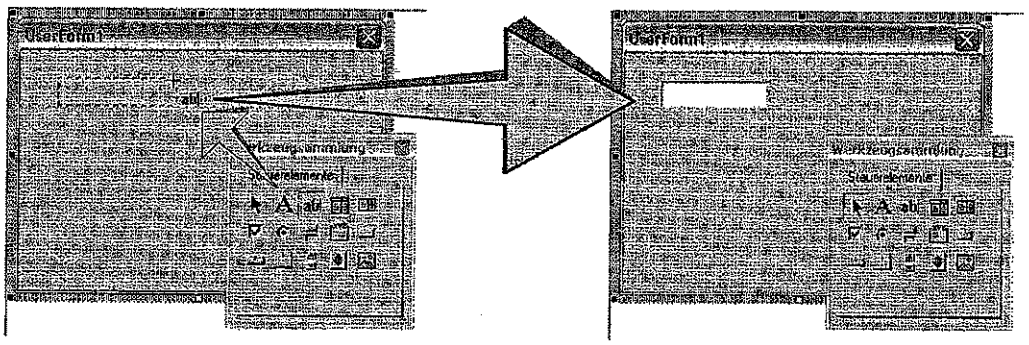
Optionsfeld		Feld zur Aktivierung bzw. Deaktivierung einer Option; meist zur Aktivierung einer einzigen Option in Verbindung mit einem Rahmen
Umschaltfeld		Schaltflächen, mit denen Sie einen Zustand ein- oder ausschalten können
Rahmen		Rahmen zur Gruppierung von Steuerelementen, insbesondere von Optionsfeldern, die durch einen Rahmen logisch miteinander verbunden werden, wodurch jeweils nur ein Optionsfeld aktiviert werden kann
Befehlsschaltfläche		Schaltfläche zum Auslösen einer Aktion
Register		Registergruppe (TabStrip) mit standardmäßig zwei Registern (Tabs), die inhaltlich zusammenhängende Steuerelemente beinhalten
Multiseiten		Wie Registersteuerelement, stellt aber mehrere Seiten für Informationen aus unterschiedlichen Kategorien zusammen
Bildlaufleiste		Horizontale oder vertikale Bildlaufleiste
Drehfeld		Pfeilsymbole zum Erhöhen oder Verringern von Werten
Anzeige		Anzeige eines Bildes oder einer Grafik

Der Pfeil dient zum Markieren und Bearbeiten der Steuerelemente.

Steuerelemente hinzufügen

Der UserForm-Dialog ist in ein Raster eingeteilt, um die Steuerelemente genau anordnen zu können. Das Raster ist nur in der Entwurfsansicht sichtbar.

- Klicken Sie auf das Steuerelemente in der Werkzeugsammlung, das Sie einfügen möchten. Der Cursor verändert seine Gestalt in ein Kreuz mit dem Symbol des Steuerelements.
- Ziehen Sie im Bereich des UserForm-Dialogs bei gedrückter linker Maustaste das Steuerelement auf die gewünschte Größe.
- Lassen Sie die Maustaste wieder los, um das Steuerelement einzufügen.

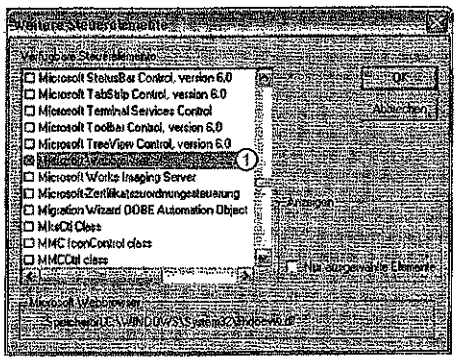


Einfügen eines Textfelds in einen leeren UserForm-Dialog

Notizen

Sie möchten...	
Ein Steuerelement in seiner Standardgröße einfügen	⇒ Klicken Sie mit der linken Maustaste auf das Formular, um das zuvor ausgewählte Steuerelement einzufügen.
ein Steuerelement verschieben	⇒ Zeigen Sie mit der Maus auf das Steuerelement, und ziehen Sie es bei gedrückter linker Maustaste an die gewünschte Stelle.
die Größe eines Steuerelements ändern	⇒ Markieren Sie das Steuerelement, und ziehen Sie es durch Anklicken eines Eckpunkts auf die gewünschte Größe.
ein Steuerelement löschen	⇒ Markieren Sie das Steuerelement, und betätigen Sie die ENTF-Taste.

Es gibt viele zusätzliche ActiveX-Steuerelemente, die sie zur Werkzeugsammlung hinzufügen und in Ihren Projekten verwenden können. Über den Kontextmenüpunkt ZUSÄTZLICHE STEUERELEMENTE wird ein Dialog geöffnet, in dem Sie Steuerelemente, die sich bereits auf Ihrem Rechner befinden, auswählen können, z. B. das ADOBE ACROBAT CONTROL FOR ACTIVEX oder der MICROSOFT WEB BROWSER ①. Weitere Steuerelemente sind im Internet in Form von Code-Bibliotheken mit den Dateierweiterungen **.ocx* bzw. **.dll* verfügbar.



Diese Steuerelemente können, wie bereits beschrieben, auf das Formular gezogen und über ihren Namen angesprochen werden. Das Web-Browser-Steuerelement besitzt z. B. die Methode `Navigate`, der Sie die anzuzeigende URL übergeben können ②.

Eigenschaften festlegen

Alle Objekte eines UserForm-Dialogs besitzen Eigenschaften, die das Aussehen und das Verhalten steuern, beispielsweise die Größe, die Farbe oder den Namen.

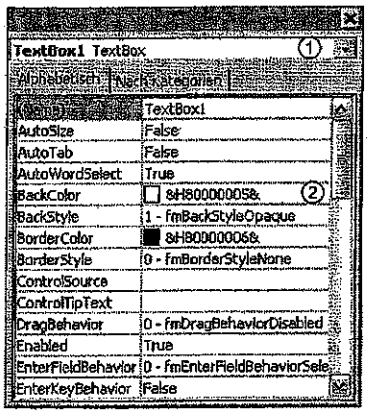
Beim Hinzufügen eines Steuerelements werden ihm Standardeigenschaften zugewiesen. Diese Werte können Sie nachträglich im Eigenschaftenfenster ändern und an Ihre Bedürfnisse anpassen.

⇒ Markieren Sie das Steuerelement bzw. den UserForm-Dialog.

oder Wählen Sie das Objekt aus dem Listenfeld ① im Eigenschaftenfenster.

⇒ Klicken Sie in der Liste der Eigenschaften auf den zu ändernden Wert ②.

⇒ Je nach dem Typ der Eigenschaft können Sie den vorgegebenen Wert überschreiben, einen neuen Wert aus einem Listenfeld auswählen oder ein Dialogfenster zur Werteauswahl anzeigen.



Eigenschaftenfenster bei einem Textfeld

Falls das Eigenschaftenfenster nicht angezeigt wird, können Sie es über den Menüpunkt ANSICHT - EIGENSCHAFTENFENSTER (Alternative: F4) einblenden.

UserForm-Dialog testen

Haben Sie einen UserForm-Dialog fertig gestellt, können Sie ihn auch unabhängig vom Aufruf in einer Prozedur testen.

⇒ Klicken Sie auf das Fenster des UserForm-Dialogs, um es zu aktivieren.
⇒ Rufen Sie den Menüpunkt AUSFÜHREN - SUB/USERFORM AUSFÜHREN auf.
Alternativen: ▶ oder F5
⇒ Schließen Sie den UserForm-Dialog durch Betätigen des Schließsymbols ☒.

Zwar lässt sich jeder UserForm-Dialog durch das Schließsymbol beenden, für den späteren Einsatz sollten Sie jedoch eine ABBRECHEN-Schaltfläche einfügen.

9.3 UserForm-Dialoge verwenden und programmieren

UserForm-Dialoge und Ereignisprozeduren

Wird ein UserForm-Dialog am Bildschirm angezeigt, erzeugt jede Aktion des Anwenders, z. B. das Betätigen einer Schaltfläche oder die Auswahl eines Wertes, ein Ereignis. Mit speziellen Ereignisprozeduren können Sie im VBA-Programm auf diese Ereignisse reagieren.

Mit Ereignisprozeduren können Sie beispielsweise

- ☑ eingegebene Daten auf Gültigkeit prüfen
- ☑ aktivierte Optionsfelder abfragen
- ☑ andere Steuerelemente aktivieren oder deaktivieren
- ☑ weitere UserForm-Dialoge einblenden oder einen Dialog beenden

Beispiel: *Briefanrede.doc*, UserForm-Dialog `UserForm1`, Modul `ThisDocument`

Über ein Dialogformular soll der Anwender einen Namen aus einer Liste auswählen. Für den ausgewählten Namen soll nach dem Betätigen einer Schaltfläche eine passende Anrede in das Word-Dokument eingefügt werden. Die Namen, Vornamen und Anredekürzel werden in einer dreispaltigen Excel-Tabelle verwaltet.

```
①  Dim appXL As Excel.Application

②  Private Sub UserForm_Activate()
     Dim Zähler As Long
     Set appXL = CreateObject("Excel.Application")
     appXL.Workbooks.Open (ActiveDocument.Path & "\Namen.xls")
     appXL.Sheets("Mitarbeiter").Activate
     appXL.Range("A2").Activate
     appXL.Range("A2").CurrentRegion.Sort appXL.Range("A1"), xlAscending
     For Zähler = 1 To appXL.Range("A2").CurrentRegion.Rows.Count
③      Me.lst_Namen.AddItem appXL.Cells(Zähler, 2) & " " & _
                            appXL.Cells(Zähler, 1)
     Next
④    Me.lbl_Anzahl.Caption = Me.lst_Namen.ListCount & " Namen geladen"
   End Sub

⑤  Private Sub btn_Abbrechen_Click()
     appXL.DisplayAlerts = False
     appXL.Quit
     appXL.DisplayAlerts = True
     Set appXL = Nothing
⑥    Unload UserForm1
   End Sub

⑦  Private Sub lst_Namen_Change()
```

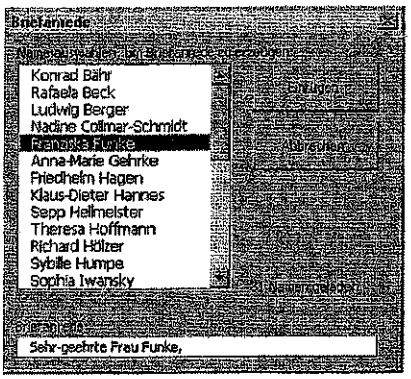
```
    Dim Zeile As Long, Anrede As String
    Zeile = Me.lst_Namen.ListIndex + 1
    If appXL.Cells(Zeile, 3) = 1 Then
        Anrede = "Sehr geehrte Frau " & appXL.Cells(Zeile, 1) & ","
    Else
        Anrede = "Sehr geehrter Herr " & appXL.Cells(Zeile, 1) & ","
    End If
    Me.txt_Anrede = Anrede
End Sub

⑧ Private Sub btn_Einfügen_Click()
    Selection.TypeText Me.txt_Anrede
    btn_Abbrechen_Click
End Sub
```

① Die Objektvariable für das Automationsobjekt wird im Deklarationsteil des Formularmoduls deklariert, da sie in mehreren Prozeduren zur Verfügung stehen soll.

② Die Prozedur für das `Activate`-Ereignis wird aufgerufen, sobald der UserForm-Dialog eingeblendet wird. In dieser Prozedur wird das Automation-Objekt initialisiert und die benötigte Excel-Arbeitsmappe geöffnet.

③ An dieser Stelle wird das Listenfeld `lst_Namen` mit Einträgen gefüllt. Die Methode `AddItem` fügt die angegebene Zeichenfolge am Ende des Listenfelds hinzu.

④ Ein Bezeichnungsfeld wie `lbl_Anzahl` kann keinen Wert enthalten. Damit es dennoch die Anzahl der geladenen Namen anzeigt, wird die Beschriftung des Feldes über die Eigenschaft `Caption` geändert. Die `ListCount`-Eigenschaft des Listenfelds liefert dabei die Anzahl der Einträge.



Der UserForm-Dialog mit den gelisteten Namen

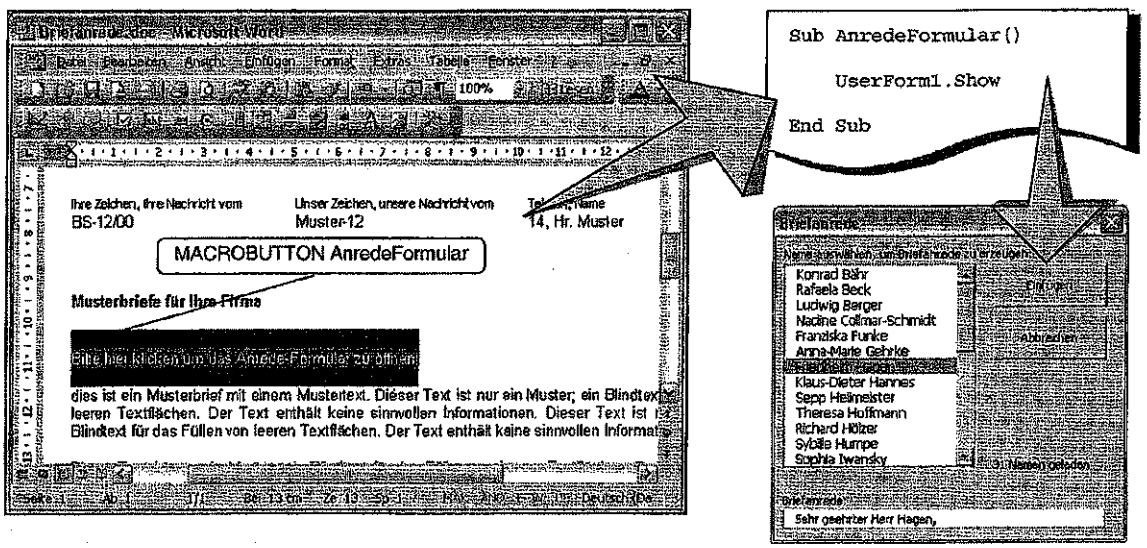
⑤ Diese Ereignisprozedur wird aufgerufen, wenn der Anwender die Schaltfläche ABBRECHEN betätigt. Hier wird das Automation-Objekt geschlossen und der UserForm-Dialog ausgeblendet.

⑥ Mithilfe der `Unload`-Anweisung wird der UserForm-Dialog entladen und der zugewiesene Speicher freigegeben.

⑦ Nach der Auswahl eines Namens im Listenfeld wird das `Change`-Ereignis ausgelöst und die zugehörige Prozedur ausgeführt.

⑧ Nach dem Betätigen der Schaltfläche EINFÜGEN wird mithilfe dieser Prozedur die Briefanrede aus dem UserForm-Dialog in das Dokument eingefügt.

Der UserForm-Dialog USERFORM1 wird durch die Feldfunktion `MACROBUTTON` im Dokument aufgerufen:



Ereignisprozeduren erstellen

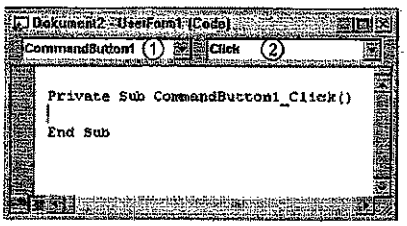
→ Klicken Sie doppelt auf das Element, zu dem Sie eine Ereignisprozedur schreiben möchten. Wenn Sie für das UserForm-Objekt eine Ereignisprozedur erstellen möchten, klicken Sie innerhalb des Fensters auf eine leere Fläche.
Alternativen: ANSICHT - CODE oder F7

Das Modulfenster wird geöffnet. Es enthält standardmäßig das Gerüst der Ereignisprozedur, das Sie mit eigenem VBA-Code ergänzen können.

Der Name der Ereignisprozedur wird nach folgendem Muster automatisch vergeben:

```
Steuerelementname_Ereignis
```

- ☑ Der Steuerelementname ist der Name, der durch die Eigenschaft Name festgelegt wird. VBA vergibt standardmäßig einen Namen, den Sie allerdings im Eigenschaftenfenster ändern können.
- ☑ Nach einem Unterstrich folgt der Name des Ereignisses. Diese englische Bezeichnung ist von VBA vorgegeben und kann nicht geändert werden.



Modulfenster mit dem Rumpf einer Ereignisprozedur für eine Schaltfläche

Das Modulfenster enthält am oberen Rand zwei Listenfelder, über die Sie den Namen eines Steuerelements ① und ein dazugehöriges Ereignis ② auswählen können. Die VBA-Entwicklungsumgebung fügt danach das Gerüst für die entsprechende Ereignisprozedur ein.

UserForm-Dialog anzeigen

Jeder UserForm-Dialog kann durch eine Prozedur eingeblendet werden. Dazu rufen Sie die Methode Show des betreffenden UserForm-Objekts auf. Der Name des UserForm-Dialogs kann im Eigenschaftenfenster geändert werden.

```
UserFormName.Show
```

Wird ein UserForm-Dialog eingeblendet, löst VBA das Activate-Ereignis aus. Dabei spielt es keine Rolle, ob der Dialog manuell oder durch eine Prozedur geöffnet wurde. In einer Ereignisprozedur für dieses Ereignis können Sie verschiedene Voreinstellungen vornehmen oder Steuerelemente initialisieren (mit Werten füllen).

```
Sub UserFormName_Activate()
    ...
End Sub
```

Notizen

UserForm-Dialog ausblenden und schließen

Mit der Methode `Hide` können Sie einen UserForm-Dialog ausblenden. Der Dialog wird dabei jedoch nicht vollständig geschlossen, sondern nur verborgen. Alle Werte in den verschiedenen Steuerelementen und der zugewiesene Speicherplatz bleiben erhalten.

```
UserFormName.Hide
```

Mit der `Unload`-Anweisung wird der UserForm-Dialog entladen und der zugewiesene Speicherplatz freigegeben. Alle Steuerelemente verlieren gleichzeitig ihren Inhalt.

```
Unload UserFormName
```

9.4 Steuerelemente programmieren

Die Steuerelemente eines UserForm-Dialogs dienen dem Anzeigen und Abfragen von Werten. Sie können Steuerelementen Werte zuweisen, beispielsweise über die `Activate`-Ereignisprozedur, Sie können jedoch auch die durch einen Anwender eingegebenen Werte auslesen.

Alle Steuerelemente lösen verschiedene Ereignisse aus, auf die Sie mit einer geeigneten Ereignisprozedur reagieren können. So ist es beispielsweise möglich, nach dem Betätigen einer Schaltfläche oder nach dem Ändern eines Werts in einem Steuerelement eine Aktion auszulösen.

Die folgenden Tabellen erläutern Standardaufgaben bei einigen wichtigen Steuerelementen.

Eingabefelder

Werte zuweisen bzw. auslesen	`UserFormName.EingabefeldName.Value` Die Angabe der Eigenschaft `Value` kann weggelassen werden, da sie die Standardeigenschaft ist.
Wichtiges Ereignis	`EingabefeldName_Change` Das Ereignis wird beim Ändern des Steuerelementwerts ausgelöst.

Schaltflächen

Schaltflächenbeschriftung einstellen	`UserFormName.SchaltflächenName.Caption`
Schaltfläche aktivieren bzw. deaktivieren	`UserFormName.SchaltflächenName.Enabled = True/False` Durch Zuweisen von `True` wird die Schaltfläche aktiviert, durch `False` deaktiviert.
Wichtiges Ereignis	`SchaltflächenName_Click` Das `Click`-Ereignis wird beim Betätigen der Schaltfläche ausgelöst.

Listen- und Kombinationsfelder

Listenelemente hinzufügen	`UserFormName.ListenfeldName.AddItem "Wert"` Die `AddItem`-Methode fügt einen Wert am Ende der Liste ein.
Listeneinträge entfernen	`UserFormName.ListenfeldName.RemoveItem Indexwert` Die Methode löscht den mit Indexwert referenzierten Wert der Liste. Die Indexzählung beginnt bei 0.

Anzahl der Listeneinträge bestimmen	`UserFormName.ListenfeldName.ListCount`
Index des ausgewählten Elements bestimmen	`UserFormName.ListenfeldName.ListIndex` Die Eigenschaft liefert den Indexwert des ausgewählten Listenelements.
Text des ausgewählten Elements bestimmen	`UserFormName.ListenfeldName.Text` Die Eigenschaft ermittelt den Text des ausgewählten Elements.
Ausgewählte Elemente bei Mehrfachauswahl bestimmen	`UserFormName.ListenfeldName.Selected(IndexWert)` Die Methode liefert `True` oder `False`, je nachdem, ob das angegebene Element ausgewählt ist.
Wichtiges Ereignis	`ListenfeldName_Change` Das Ereignis wird ausgelöst, sobald ein anderes Element ausgewählt wird.

Kontrollfelder

Auswahl eines Kontrollfelds prüfen	`UserFormName.KontrollfeldName.Value` Wenn das Kontrollfeld aktiviert ist, wird `True` zurückgegeben, anderenfalls `False`. Die Angabe der Eigenschaft `Value` kann entfallen.
Wichtiges Ereignis	`KontrollfeldName_Click` Das Ereignis wird ausgelöst, wenn auf das Kontrollfeld geklickt wird.

9.5 Schnellübersicht

Sie möchten ...	
einen neuen UserForm-Dialog einfügen	EINFÜGEN - USERFORM oder
einen UserForm-Dialog testen	AUSFÜHREN - SUB/USERFORM AUSFÜHREN oder F5 oder
einen UserForm-Dialog anzeigen	`UserFormName.Show`
einen UserForm-Dialog ausblenden	`UserFormName.Hide`
einen UserForm-Dialog entladen	`Unload UserFormName`
eine Ereignisprozedur erstellen	`Sub UserFormName_Ereignisname` `Sub SteuerelementName_Ereignisname`
Aktionen beim Anzeigen des UserForm-Dialogs ausführen	`UserFormName_Activate`

Notizen

9.6 Übung

Übungsdatei: *Übung1.doc, Gehalt.mdb* **Ergebnisdatei:** *Übung1-E.doc*

① Öffnen Sie die VBA-Entwicklungsumgebung in Word, und fügen Sie einen neuen leeren UserForm-Dialog ein.

② Erstellen Sie die in der nebenstehenden Abbildung angegebenen Steuerelemente, und benennen Sie diese. Der Dialog soll in der Titelleiste die Bezeichnung NAMEN EINFÜGEN enthalten.

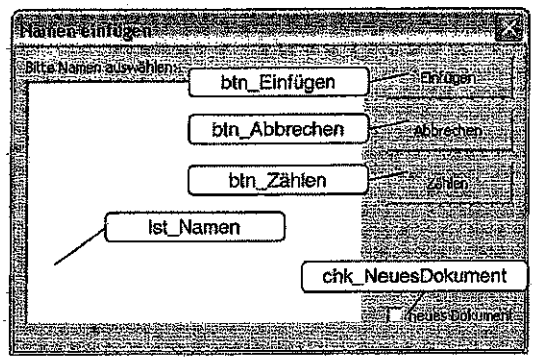
③ Das Listenfeld soll eine Mehrfachauswahl edauben. Stellen Sie dazu für die Eigenschaft `MultiSelect` den Wert `fmMultiSelectMulti` ein.

④ Fügen Sie ein neues Modul hinzu, und benennen Sie es `AccessZugriff`.

⑤ Erstellen Sie im Modul `AccessZugriff` eine Prozedur `NamenAuslesen`, die alle Namen aus dem Formular `Mitarbeiter` der Datenbank *Gehalt.mdb* auslesen soll. Die Datenbank befindet sich im gleichen Verzeichnis wie das aktuelle Dokument.

⑥ Der Prozedur soll als Argument der Name des Listenfelds übergeben werden. Die ausgelesenen Namen sollen nacheinander dem Listenfeld hinzugefügt werden. Schließen und entladen Sie danach das Automationsobjekt.

⑦ Erstellen Sie eine Ereignisprozedur für das `Activate`-Ereignis des UserForm-Dialogs, das die Prozedur `NamenAuslesen` aufruft, falls das Listenelement keine Einträge enthält.



Der Dialog in der Entwurfsansicht

Zählen der Listeneinträge

⑧ Erstellen Sie eine Ereignisprozedur für die Schaltfläche ABBRECHEN, die den UserForm-Dialog ausblendet.

⑨ Nach dem Betätigen der Schaltfläche ZÄHLEN soll ein Meldungsfenster mit der Anzahl der geladenen Namen angezeigt werden. Erstellen Sie dafür eine geeignete Ereignisprozedur.

⑩ Für die Schaltfläche EINFÜGEN sollen Sie eine Ereignisprozedur erstellen, die alle im Listenfeld ausgewählten Namen in einer neuen Zeile einfügt. Verwenden Sie dazu eine `For Next`-Schleife, die für jedes Listenelement prüft, ob es markiert ist (`Selected`-Methode).

⑪ Prüfen Sie zu Beginn der Ereignisprozedur, ob das Kontrollfeld NEUES DOKUMENT aktiviert ist. Falls dies der Fall ist, erzeugen Sie ein neues Dokument, und fügen Sie die Namen dort ein. Verwenden Sie dabei eine Objektvariable des Typs `Document`, der Sie je nach Auswahl den Verweis auf ein neues oder auf das aktuelle Dokument zuweisen.

⑫ Fügen Sie am Ende der Ereignisprozedur eine Anweisung zum Aktualisieren des Bildschirms ein.

Notizen

10 Datenbankzugriff in Office-Anwendungen

In diesem Kapitel erfahren Sie

- → wie Sie auf Tabellen einer Datenbank zugreifen
- → wie Sie mit VBA Datensätze auslesen und bearbeiten
- → wie Sie auf Datenfelder zugreifen

Voraussetzungen

- ✓ VBA-Programmierung
- ✓ Struktur und Aufbau von Datenbanken

10.1 Objekte zum Datenbankzugriff

10.1.1 Grundlagen

Bisher haben Sie beim Zugriff auf Access-Datenbanken nur Daten aus Formularen ermittelt. Dazu wurde das `Form`-Objekt einer geöffneten Access-Anwendung verwendet. Das Auslesen der Daten aus Formularen ist für das Erstellen integrierter Office-Lösungen jedoch umständlich und wenig vorteilhaft. Oft ist es günstiger, direkt auf die zugrunde liegenden Tabellen zuzugreifen und gezielt die benötigten Daten auszulesen.

Für diesen Datenzugriff stellt VBA verschiedene Objektmodelle zur Verfügung. Zum einen sind dies die Datenzugriffsobjekte, auch Data Access Objects (DAO) genannt, zum anderen die ActiveX-Datenobjekte, die ActiveX Data Objects (ADO). Das DAO-Objektmodell wurde vor allem in früheren Office-Versionen verwendet. Beim Programmieren mit Office 2003 ist das aktuellere ADO-Objektmodell die beste Wahl. Es vereinfacht und vereinheitlicht den Zugriff auf Datenquellen unterschiedlicher Art und Herkunft und wird daher für die Programmierung empfohlen. Im Gegensatz zum Datenzugriff mit DAO-Objekten erlaubt ADO den universellen Zugriff auf alle OLE-DB-Provider und kann dadurch Verbindung zu vielen unterschiedlichen Datenbanken aufnehmen.

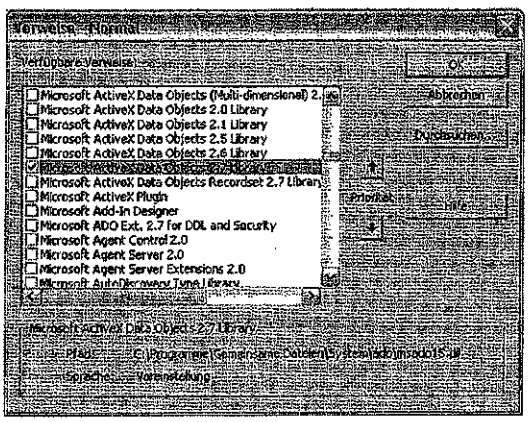
In den folgenden Abschnitten wird ausschließlich auf die Programmierung mit den ADO-Objekten eingegangen. Im Zuge der Einführung der .NET-Technologie gibt es eine weiterentwickelte Form des Zugriffs auf Datenbanken, die als ADO .NET bezeichnet wird. Um ADO .NET für die Office-Integration zu verwenden, ist die Verwendung von Visual Studio .NET sowie zusätzlich der Visual Studio Tools für Office 2003 empfehlenswert. In diesem Fall benötigen Sie außerdem Kenntnisse in einer .NET-Programmiersprache (z. B. Visual Basic .NET oder Visual C# .NET).

Das ADO-Objektmodell in der VBA-Entwicklungsumgebung aktivieren

Damit die Programmierhilfen der Entwicklungsumgebung funktionieren, müssen Sie die Unterstützung für das ADO-Objektmodell einschalten.

⇒ Öffnen Sie die VBA-Entwicklungsumgebung.

⇒ Wählen Sie den Menüpunkt EXTRAS - VERWEISE.



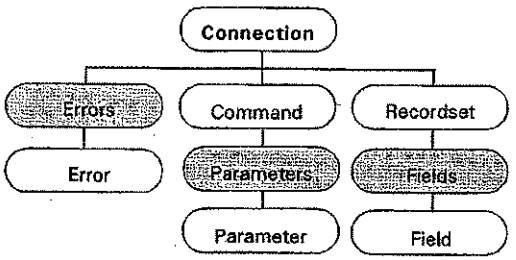
ADO im Dialogfenster VERWEISE aktivieren

⇒ Im Dialogfenster VERWEISE aktivieren Sie das Kontrollfeld MICROSOFT ACTIVEX DATA OBJECTS 2.7 LIBRARY. Wenn Sie eine neuere ADO-Version installiert haben, ist möglicherweise eine höhere Programmversion als 2.7 angegeben.

⇒ Bestätigen Sie das Dialogfenster mit OK.

10.1.2 Hierarchie des ADO-Objektmodells

Die Objekte und Auflistungen des ADO-Objektmodells sind wie alle Objekte der Office-Anwendungen in Hierarchiestufen angeordnet. Allerdings umfasst es (im Gegensatz zum früher verwendeten DAO-Objektmodell) nur wenige Objekte in einer flachen Hierarchie. Dies vereinfacht den Umgang mit den Objekten bei der Programmierung.



Die Hierarchie des ADO-Objektmodells

Die Objekte `Errors`, `Parameters` und `Fields` sind Auflistungen von dazugehörigen Einzelobjekten.

Objekt	Beschreibung
`Connection`	Mit diesem Objekt wird eine Datenverbindung zu einer Datenquelle hergestellt. Das Objekt repräsentiert die Verbindung.
`Command`	Das `Command`-Objekt definiert einen Befehl, der auf eine bestimmte Datenquelle ausgeführt wird.
`Recordset`	Dieses Objekt ermöglicht den Zugriff auf die einzelnen Datensätze einer geöffneten Datenquelle.
`Error`	Das Objekt liefert Einzelheiten zu den aufgetretenen Fehlern beim Datenzugriff.
`Parameter`	Das `Parameter`-Objekt definiert Parameter und Argumente für ein `Command`-Objekt.
`Field`	Dieses Einzelobjekt der `Fields`-Auflistung repräsentiert die Datenfelder einer Datenquelle, z. B. einer Tabelle. Ein Datenfeld entspricht dabei einer ganzen Spalte mit Daten des gleichen Typs.

10.1.3 Ablauf des Datenzugriffs mit ADO

Der Datenzugriff mit den ADO-Objekten erfolgt stets nach einem vorgegebenen Schema:

- ☑ Verbindung zu einer Datenquelle über das `Connection`-Objekt (oder vereinfacht über das `Recordset`-Objekt) herstellen
- ☑ Mit dem `Command`-Objekt einen Befehl angeben, um auf die Datenquelle zuzugreifen
- ☑ Den Befehl über die `Connection`-, `Command`- oder `Recordset`-Objekte ausführen
- ☑ Die zurückgegebenen Datensätze mithilfe des `Recordset`-Objekts in einem Zwischenspeicher speichern
- ☑ Die gespeicherten Datensätze bearbeiten, indem Daten mittels des `Recordset`-Objekts hinzugefügt, geändert oder gelöscht werden

Notizen

- ☑ Datenquelle mit den Änderungen der Datensätze im Zwischenspeicher über das `Recordset`-Objekt aktualisieren
- ☑ Eventuell Fehler mit dem `Error`-Objekt auswerten
- ☑ Verbindung mithilfe des `Connection`-Objekts schließen

Nicht alle der hier angegebenen Schritte müssen bei jedem Datenzugriff durchlaufen werden. In den meisten Fällen soll nur auf eine durch Access geöffnete Datenbank in einem Einzelbenutzersystem zugegriffen werden, sodass weniger Aufwand nötig ist.

10.2 Verbindung zu einer Datenquelle herstellen

Verbindung erstellen

Im Regelfall wird das Öffnen der Datenverbindung durch das `Connection`-Objekt realisiert. Dies ist die flexibelste Methode, denn sie ermöglicht das Verbinden zu beliebigen Datenquellen, z. B. Access-Datenbanken, Excel-Tabellen, Text-Dateien, aber auch Datenquellen außerhalb der Office-Umgebung.

In den meisten Fällen ist jedoch lediglich ein Zugriff auf die Werte einer bestimmten Datentabelle innerhalb der Office-Umgebung gefordert. Da für das Speichern von Daten im Office-Umfeld in der Regel Access eingesetzt wird, soll im Folgenden nur der Zugriff auf eine Access-Datenbank beschrieben werden.

Beispiel: *ADODatenbankZugriff.xls*, Modul `ADOProzeduren`

In einer Excel-Arbeitsmappe sollen Daten aus einer Access-Datenbank verwendet werden. Da das Auslesen eines Access-Formulars nicht leistungsfähig genug ist, wird eine Datenverbindung über ADO-Objekte verwendet.

```
Sub VerbindungÖffnen()
  Dim ADOCnn As ADODB.Connection                                    ①
  Set ADOCnn = New ADODB.Connection                                ②
  ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"                      ③
  ADOCnn.Open (ActiveWorkbook.Path & "\gehalt.mdb")                ④
  ...                                                              ⑤
  ADOCnn.Close                                                     ⑥
  Set ADOCnn = Nothing
End Sub
```

① Die Objektvariable `ADOCnn` wird mit dem Typ `ADODB.Connection` deklariert. Sie soll den Bezug zu einer geöffneten Datenquelle, in diesem Fall einer Access-Datenbank, herstellen.

② Mit dem Schlüsselwort `New` wird eine neue Instanz des `Connection`-Objekts erzeugt.

③ Die Eigenschaft `Provider` bestimmt, welche Datenquellen verwendet werden können. In diesem Fall wird der Microsoft Jet 4.0 OLE DB-Provider verwendet, der den direkten (nativen) Zugriff auf Access-Datenbanken erlaubt.

④ Über die `Open`-Methode des `Connection`-Objekts wird die angegebene Datenbank geöffnet. Dabei wird davon ausgegangen, dass sich die Datenbankdatei im gleichen Verzeichnis wie die Excel-Datei befindet.

⑤ Hier können Daten aus der Datenbank gelesen oder geändert werden (vgl. folgende Beispiele).

⑥ Mit der Methode `Close` wird die Verbindung zur Datenquelle wieder geschlossen. Anschließend kann das `Connection`-Objekt durch Zuweisen des Schlüsselworts `Nothing` freigegeben werden.

Syntax des Öffnens einer Verbindung zur Datenbank

```
Dim DBVerbindung As ADODB.Connection
Set DBVerbindung = New ADODB.Connection
DBVerbindung.Provider = "Provider-Bezeichnung"
DBVerbindung.Open "Datenquelle"
```

- ☑ Zuerst wird eine Objektvariable mit dem Datentyp `ADODB.Connection` deklariert.
- ☑ Durch das Schlüsselwort `New` wird eine neue Objektinstanz vom Typ `ADODB.Connection` erzeugt.
- ☑ Damit die Kommunikation mit der Datenquelle funktioniert, wird ein geeigneter so genannter OLE DB-Provider benötigt. Der Provider fungiert als Treiber für den Zugriff auf unterschiedliche Datenquellen.
- ☑ Mit der Methode `Open` wird die Verbindung geöffnet, wobei der Pfad zur Datenbankdatei anzugeben ist.

Der Zugriff auf die Access-Datenbank erfolgt über die Jet-Datenbank-Engine von Microsoft, die zusammen mit Access installiert wird. Der geeignete Provider für diese Datenquelle ist der Microsoft Jet 4.0 OLE DB-Provider. Der Bezeichnungs-String für diesen Provider lautet `Microsoft.Jet.OLEDB.4.0`.

Die Jet-Datenbank-Engine kann auch auf Windows-Systemen ohne Access verwendet werden. In den älteren Versionen (bis 2.5) von MDAC (Microsoft Data Access Components) war sie enthalten. Bei allen neueren Versionen muss Microsoft Jet 4.0 gesondert heruntergeladen und installiert werden. Danach ist es möglich, auch ohne Access per Programmcode Access-Datenbanken zu erzeugen und auf sie zuzugreifen.

Die folgende Tabelle zeigt einige OLE DB-Provider, die Sie mit ADO verwenden können, in der Übersicht:

OLE DB-Provider	Bezeichnungs-String	Erklärung
OLE DB Provider for Microsoft Jet	`Microsoft.Jet.OLEDB.4.0`	Zugriff auf Access-Datenbanken ab Access 2000
Microsoft OLE DB Provider for SQL Server	`SQLOLEDB`	Zugriff auf den Microsoft SQL-Server
Microsoft OLE DB Provider for ODBC	`MSDASQL`	Zugriff auf Datenbanken über ODBC
Microsoft OLE DB Provider for Oracle	`MSDAORA`	Zugriff auf einen Oracle-Datenbankserver
OLE DB Provider for Microsoft Active Directory Service	`ADSDSOObject`	Zugriff auf den Active Directory Service in einem Windows 2000-Netzwerk

Für eine in Access geöffnete Datenbank richtet Access automatisch ein ADO-`Connection`-Objekt ein, auf das Sie mit `CurrentProject.Connection` zugreifen können.

Das Erzeugen der neuen Objektinstanz kann, ähnlich wie bei den Automationsobjekten, auch bei der Deklaration erfolgen. Dadurch entfällt der zusätzliche Schritt mit der `Set`-Anweisung.

```
Dim DBVerbindung As New ADODB.Connection
```

Notizen

Syntax des Schließens einer Verbindung zur Datenbank

```
DBVerbindung.Close
Set DBVerbindung = Nothing
```

- ☑ Mit der `Close`-Methode wird die Verbindung zur Datenquelle geschlossen.
- ☑ Durch Zuweisen des Schlüsselworts `Nothing` werden das Objekt und der zugehörige Speicherplatz freigegeben.

10.3 Datensatzgruppe auswählen und öffnen

Die in einer Datenquelle gespeicherten Werte sind in Tabellen angeordnet. Nachdem Sie eine Verbindung zu einer Datenquelle geöffnet haben, können Sie eine bestimmte Tabelle öffnen und erhalten damit Zugriff auf die gespeicherten Datensätze.

Das ADO-Objektmodell verfügt über das Objekt `Recordset`. Das `Recordset`-Objekt kann alle Datensätze einer Tabelle umfassen oder nur einige ausgewählte. Das Objekt repräsentiert daher eine Datensatzgruppe.

Beispiel: *ADODatenbankZugriff.xls*, Modul `ADOProzeduren`

In der Excel-Arbeitsmappe wurde eine ADO-Verbindung geöffnet. Darüber soll nun eine Tabelle zum Datenzugriff geöffnet werden.

```
Sub TabelleÖffnen()
①   Dim ADOCnn As ADODB.Connection
    Dim ADOTab As ADODB.Recordset
    Set ADOCnn = New ADODB.Connection
    ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    ADOCnn.Open (ActiveWorkbook.Path & "\gehalt.mdb")
②   Set ADOTab = New ADODB.Recordset
③   ADOTab.Open "Mitarbeiter", ADOCnn, adOpenDynamic, adLockOptimistic
    ADOCnn.Close
    Set ADOCnn = Nothing
End Sub
```

① Die Verbindung zur Datenquelle wird wie beschrieben geöffnet. Zusätzlich wird eine Objektvariable `ADOTab` mit dem Typ `ADODB.Recordset` deklariert. Sie soll den Bezug zur Datensatzgruppe herstellen.

② Mit dem Schlüsselwort `New` wird eine neue Instanz des `Recordset`-Objekts erzeugt.

③ An dieser Stelle wird die Verbindung zur Tabelle *Mitarbeiter* geöffnet. Dazu wird die Methode `Open` des `Recordset`-Objekts verwendet. Als Argument wird dabei die geöffnete Datenverbindung `ADOCnn` übergeben. Über die zwei folgenden Konstanten werden einige Eigenschaften der Verbindung bestimmt.

Syntax des Öffnens einer Verbindung mit dem Recordset-Objekt

```
Dim Datensatzgruppe As ADODB.Recordset
Set Datensatzgruppe = New ADODB.Recordset
Datensatzgruppe.Open "Tabelle", ADO-Verbindung, Cursortyp, Sperrtyp
```

- ☑ Zuerst wird eine Objektvariable mit dem Datentyp `ADODB.Recordset` deklariert. Diese Variable wird später für den Datenzugriff verwendet.
- ☑ Durch das Schlüsselwort `New` wird eine neue Objektinstanz vom Typ `ADODB.Recordset` erzeugt. Der Objektvariablen vom gleichen Typ wird ein Verweis auf diese Objektinstanz zugewiesen. Dies wird mit der `Set`-Anweisung erreicht. Die Variable ist nun dafür vorbereitet, eine Datensatzgruppe zu speichern.
- ☑ Das Erzeugen der Objektinstanz kann auch in einem Schritt mit der Deklaration erfolgen.
- ☑ Mit der Methode `Open` wird die Verbindung geöffnet. Die Methode benötigt einige Argumente, die die Verbindung näher spezifizieren.

Statt des Namens einer Tabelle können Sie der Open-Methode auch eine gültige SQL-Anweisung in der Form `SELECT Felder FROM Tabelle WHERE Kriterium` übergeben. In diesem Fall erhalten Sie nur eine Gruppe von passenden Datensätzen.

Argumente der Open-Methode

Argument	Beschreibung	Typisches Beispiel
`Datenquelle`	Das Argument bezeichnet die Datenquelle für alle weiteren Operationen. Dies kann eine Tabelle, eine Abfrage oder eine SQL-Anweisung sein.	`"Mitarbeiter"` `"SELECT Name, Vorname FROM Mitarbeiter"`

`Connection-Verweis`	Diese Angabe bestimmt die Verbindung, unter der die Datenquelle gefunden werden kann. Für die aktuelle Datenbank gilt die Angabe im nebenstehenden Beispiel.	`ADOCnn`
		in Access:
		`CurrentProject.Connection`

`Cursortyp`	Diese Konstante bestimmt, wie der Schreib- und Lesezugriff zur Datenquelle erfolgt. Für Datenbanken in Einzelbenutzerumgebungen kann zum Beispiel ein statischer Cursor verwendet werden, der Änderungen anderer Nutzer nicht anzeigt.	`adOpenDynamic` `adOpenStatic`
`Sperrtyp`	Die Konstante bestimmt die Art der Sperrung, die während des Bearbeitens der Datenquelle verwendet wird. Dies ist nur für Datenbanken mit mehreren gleichzeitigen Benutzern von Bedeutung.	`adLockOptimistic`

10.4 Datensätze und Datenfelder auslesen

Nachdem Sie eine Datenverbindung geöffnet haben, können Sie auf die Datensätze und deren Datenfelder zugreifen, diese auslesen, ändern, neue Datensätze hinzufügen oder nicht mehr benötigte Datensätze löschen. Für den Zugriff auf Datensätze verwenden Sie das ADO-Objekt `Recordset`. Dafür stellt es verschiedene Eigenschaften und Methoden zur Verfügung.

10.4.1 Datensätze auslesen

Beim Zugriff auf die Datensatzgruppe wird intern ein Datensatzzeiger (Cursor) verwendet, der den aktuellen Datensatz bestimmt. Nur in diesem Datensatz ist es möglich, Werte auszulesen oder zu verändern. Wenn Sie den Datensatzzeiger verschieben, können Sie nacheinander die einzelnen Datensätze einer Datensatzgruppe durchlaufen.

Notizen

Die folgende Tabelle zeigt Ihnen, welche Methoden des Recordset-Objekts zum Verschieben des Datensatzzeigers zur Verfügung stehen:

Methode	Syntax	Beschreibung
MoveFirst	Datensatzgruppe.MoveFirst	Bewegt den Datensatzzeiger zum ersten Datensatz
MoveNext	Datensatzgruppe.MoveNext	Positioniert den Datensatzzeiger auf den nächsten Datensatz
MovePrevious	Datensatzgruppe.MovePrevious	Positioniert den Datensatzzeiger auf den vorherigen Datensatz
MoveLast	Datensatzgruppe.MoveLast	Bewegt den Datensatzzeiger zum letzten Datensatz
Move	Datensatzgruppe.Move Anzahl	Bewegt den Datensatzzeiger um Anzahl Datensätze

Nach dem Öffnen einer Verbindung ist der Datensatzzeiger automatisch auf dem ersten Datensatz positioniert. Befindet sich der Datensatzzeiger auf dem letzten Datensatz, wenn Sie die MoveNext-Methode anwenden, wird der Datensatzzeiger ungültig. Eine Fehlermeldung tritt jedoch erst beim Zugriff auf den vermeintlichen Datensatz auf, nicht beim Verschieben! Um den Zustand des Datensatzzeigers vorher zu prüfen, stellt das Recordset-Objekt die Eigenschaften EOF und BOF zur Verfügung.

Eigenschaft	Syntax	Beschreibung
EOF	Datensatzgruppe.EOF	Die Eigenschaft EOF (End Of File) liefert False, solange der Datensatzzeiger gültig ist. Wurde der letzte Datensatz überschritten, liefert sie True.
BOF	Datensatzgruppe.BOF	Die Eigenschaft BOF (Begin Of File) gibt den Wert True zurück, wenn sich der Datensatzzeiger vor dem ersten Datensatz befindet. Anderenfalls liefert sie den Wert False.

Beispiel: *ADODatenbankZugriff.xls*, Modul ADOProzeduren

In der Excel-Arbeitsmappe wurde eine ADO-Verbindung geöffnet und ein Recordset-Objekt für eine Tabelle der Datenbank erstellt. Nacheinander sollen alle Mitarbeiternamen ausgelesen und in das Tabellenblatt eingetragen werden.

```
Sub DatensätzeLesen()
  Dim ADOCnn As ADODB.Connection
  Dim ADOTab As ADODB.Recordset
  Dim Index As Long
  Set ADOCnn = New ADODB.Connection
  ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
  ADOCnn.Open (ActiveWorkbook.Path & "\gehalt.mdb")
  Set ADOTab = New ADODB.Recordset
  ADOTab.Open "Mitarbeiter", ADOCnn, adOpenDynamic, adLockOptimistic
①  Do While ADOTab.EOF = False
    Index = Index + 1
②   Cells(Index, 1) = ADOTab!Vorname & " " & ADOTab!Name
③   ADOTab.MoveNext
  Loop
  ADOCnn.Close
  Set ADOCnn = Nothing
End Sub
```

① Nachdem die Verbindung zur Datenquelle geöffnet und die Datensatzgruppe aus der Tabelle MITARBEITER erzeugt wurde, erfolgt das Auslesen der Mitarbeiternamen. Die Do-While-Schleife wird dabei so lange durchlaufen, bis der Datensatzzeiger ungültig wird.

② Aus dem aktuellen Datensatz werden die Felder *Vorname* und *Nachname* ausgelesen und die Werte in eine Zelle der aktuellen Excel-Tabelle eingefügt.

③ Mit der MoveNext-Methode wird der Datensatzzeiger auf den folgenden Datensatz verschoben.

10.4.2 Zugriff auf Datenfelder

Eine Tabelle besteht aus Spalten und Zeilen. Jede Zeile stellt einen Datensatz dar. Die Spalten bezeichnen dagegen ein Feld für das Speichern eines Wertes pro Datensatz und tragen einen Namen. Das durch eine Spaltenbezeichnung und eine Zeile definierte Feld wird Datenfeld genannt.

Der Zugriff auf die Datenfelder einer Tabelle erfolgt entweder über deren Namen oder über einen numerischen Indexwert. Die Datenfelder werden beim Erstellen der Datenbank benannt. Falls Sie die verwendeten Namen nicht kennen, können Sie die `Fields`-Auflistung des `Recordset`-Objekts verwenden, um die Datenfelder über einen Indexwert anzusprechen.

Alle Datenfelder sind `ADODB.Field`-Objekte und verfügen beispielsweise über die Methoden `Name` und `Value`, um den Namen und den aktuellen Wert des Datenfelds zu ermitteln.

Beispiel: *ADODatenbankZugriff.xls*, Modul `ADOProzeduren`

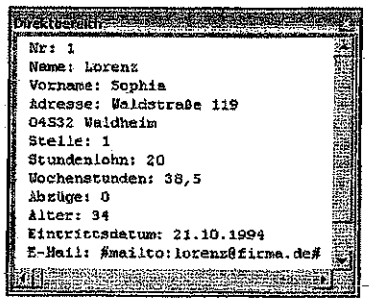
In einer Excel-Arbeitsmappe wird eine ADO-Verbindung zur Access-Datenbank *gehalt.mdb* geöffnet. Für den ersten Datensatz der Tabelle *Mitarbeiter* sollen alle Datenfelder mit ihrem zugehörigen Wert im Direktfenster ausgegeben werden.

```
Sub DatenfelderZugriff()
  Dim ADOCnn As New ADODB.Connection
  Dim ADOTab As New ADODB.Recordset
①  Dim ADOField As ADODB.Field
  ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
  ADOCnn.Open (ActiveWorkbook.Path & "\gehalt.mdb")
  ADOTab.Open "Mitarbeiter", ADOCnn, adOpenDynamic, adLockOptimistic
②  For Each ADOField In ADOTab.Fields
③    Debug.Print ADOField.Name & ": " & ADOField.Value
  Next
  ADOCnn.Close
  Set ADOCnn = Nothing
End Sub
```

① Die Objektvariable `ADOField` wird mit dem Typ `ADODB.Field` deklariert. Sie kann Verweise auf Datenfelder aufnehmen.

② In einer `For-Each`-Schleife werden alle Datenfelder aus der `Fields`-Auflistung durchlaufen.

③ Für jedes Datenfeld werden im Direktfenster der Name und der aktuelle Wert ausgegeben.



Ausgabe im Direktfenster

Notizen

Syntax zum Zugriff auf Datenfelder mithilfe von Namen

```
Datensatzgruppe!Datenfeld
```

- ☑ Die Namen der Datenfelder sind identisch mit denen der zugrunde liegenden Datenquelle, beispielsweise einer Tabelle. Die Namen werden beim Erstellen der Tabelle zugewiesen.
- ☑ Ein Ausrufezeichen (!) trennt den Feldnamen vom Objektnamen der Datensatzgruppe.

Syntax zum Zugriff auf Datenfelder mithilfe von Indexwerten

```
Datensatzgruppe.Fields(Indexwert)
```

- ☑ Der Indexwert bestimmt die Position des Feldes innerhalb der Auflistung. Die Zählung beginnt bei Null.

Mit einer `For-Each`-Schleife können Sie auf einfache Weise alle Datenfelder einer Tabelle auslesen.

10.4.3 Datensätze suchen

Mit der `Find`-Methode des `Recordset`-Objekts können Sie gezielt nach einem Datensatz in einer Datensatzgruppe suchen.

Beispiel: *ADODatenbankZugriff.xls*, Modul `ADOProzeduren`, Tabellenblatt `Tabelle1`

Von einem Excel-Tabellenblatt aus soll ein Mitarbeitername in der Access-Datenbank *gehalt.mdb* gesucht werden. Ein Teil des Namens kann dazu in eine Zelle des Tabellenblatts eingegeben werden. Über ADO-Datenzugriff auf die Tabelle *Mitarbeiter* wird der korrekte Name gesucht und danach in die Zelle eingetragen. Die Prozedur `DatensatzSuchen` wird durch eine Schaltfläche im Tabellenblatt aufgerufen.

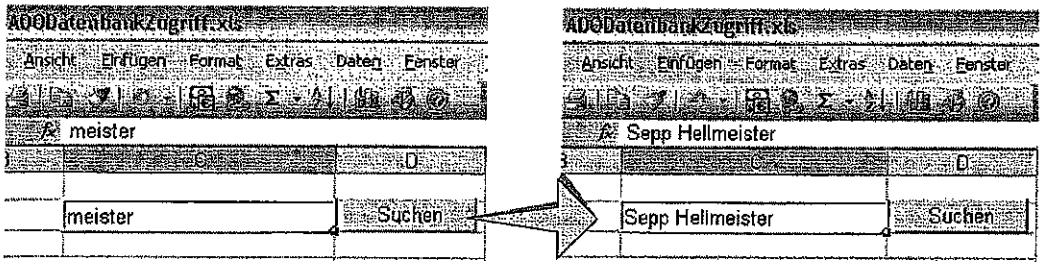
```
Sub DatensatzSuchen()
  Dim ADOCnn As New ADODB.Connection
  Dim ADOTab As New ADODB.Recordset
  ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
  ADOCnn.Open (ActiveWorkbook.Path & "\gehalt.mdb")
  ADOTab.Open "Mitarbeiter", ADOCnn, adOpenDynamic, adLockOptimistic
  With ADOTab
    .Find "Name Like '*" & Range("C2") & "*'"
    If .EOF Then
      MsgBox "Es wurde kein passender Datensatz gefunden!", vbInformation
    Else
      Range("C2") = !Vorname & " " & !Name
    End If
  End With
  ADOCnn.Close
  Set ADOCnn = Nothing
End Sub
```

① Das Erzeugen der Objektinstanzen für `Connection` und `Recordset` erfolgt in einem Schritt mit der Deklaration.

② Mithilfe der `Find`-Methode wird zum ersten auf das Suchkriterium passenden Datensatz gesprungen. In diesem Fall wird nach einem Datensatz gesucht, der den im Tabellenblatt angegebenen Suchbegriff im Feld *Name* enthält.

③ Wird kein Datensatz gefunden, ist der Datensatzzeiger ungültig, und es kann mit einem Meldungsfenster auf die misslungene Suche aufmerksam gemacht werden.

④ Bei erfolgreicher Suche wird der vollständige Name in die Zelle C2 eingetragen.



Suchen des eingegebenen Begriffs in der Access-Datenbank

Syntax zum Suchen von Datensätzen

```
Datensatzgruppe.Find Suchkriterium, AnzahlÜberspringen, Suchrichtung, Start
```

- ☑ Die Methode `Find` benötigt einige Argumente. Außer dem `Suchkriterium` können diese allerdings entfallen. Ist dies der Fall, erfolgt eine schrittweise Suche durch alle Datensätze, beginnend beim ersten Datensatz.
- ☑ War die Suche erfolgreich, wird der Datensatzzeiger auf den gefundenen Datensatz positioniert. Anderenfalls befindet sich der Datensatzzeiger hinter dem letzten Datensatz und ist damit ungültig. Dies kann mit der Eigenschaft `EOF` überprüft werden.

10.5 Datensätze hinzufügen und bearbeiten

10.5.1 Datensätze hinzufügen

Die folgende Tabelle zeigt eine Auswahl der Methoden des `Recordset`-Objekts, um Datensätze einer Datensatzgruppe hinzuzufügen.

Methode	Syntax	Beschreibung
`AddNew`	`Datensatzgruppe.AddNew`	Fügt der Datensatzgruppe einen neuen leeren Datensatz hinzu
`Update`	`Datensatzgruppe.Update`	Speichert den Datensatz in der Datensatzgruppe und aktualisiert die Datenquelle
`Close`	`Datensatzgruppe.Close`	Schließt die Verbindung mit der Datenquelle

Notizen

Beispiel: *ADODatenbankZugriff.xls*, Modul `ADOProzeduren`, Tabellenblatt `Tabelle1`

Mithilfe von ADO-Objekten sollen Vor- und Nachname eines Mitarbeiters aus einem Excel-Tabellenblatt ausgelesen und als neuer Datensatz in die Access-Datenbank *gehalt.mdb* eingefügt werden. Die Prozedur `DatensatzHinzuFügen` wird durch eine Schaltfläche im Tabellenblatt aufgerufen.

```
Sub DatensatzHinzufügen()
  Dim ADOCnn As New ADODB.Connection, ADOTab As New ADODB.Recordset
  ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
  ADOCnn.Open (ActiveWorkbook.Path & "\gehalt.mdb")
  ADOTab.Open "Mitarbeiter", ADOCnn, adOpenDynamic, adLockOptimistic
  If (Range("C6") = "" Or Range("C7") = "") Then
    MsgBox "Bitte geben Sie einen Namen ein!", vbInformation
  Else
    ADOTab.AddNew                          ①
    ADOTab!Name = Range("C7")              ②
    ADOTab!Vorname = Range("C6")
    ADOTab.Update                          ③
  End If
  ADOCnn.Close
  Set ADOCnn = Nothing
End Sub
```

① Die Methode `AddNew` erzeugt einen leeren Datensatz am Ende der Datensatzgruppe und verschiebt den Datensatzzeiger an diese Stelle.

② Die Werte werden aus dem Excel-Tabellenblatt ausgelesen und dem neuen Datensatz zugewiesen.

③ Mithilfe der Methode `Update` wird die Datenquelle aktualisiert und der Datensatz gültig.

Syntax des Hinzufügens von Datensätzen

```
Datensatzgruppe.AddNew
Datensatzgruppe!Datenfeld = Wert
...
Datensatzgruppe.Update
```

☑ Die Methode `AddNew` erstellt einen neuen leeren Datensatz.

☑ Mit der Methode `Update` erfolgt die Speicherung des neuen Datensatzes in der Datenquelle.

10.5.2 Datensätze ändern und löschen

Datensätze ändern

Um einen Datensatz zu ändern, ist es notwendig, den Datensatzzeiger auf den gewünschten Datensatz zu positionieren. Dazu können Sie beispielsweise die `Find`-Methode benutzen. Weisen Sie danach allen Datenfeldern, deren Inhalt sich ändert, den neuen Wert zu. Nach einem Aufruf der `Update`-Methode werden die Änderungen gespeichert.

```
Datensatzgruppe.Find Suchkriterium
Datensatzgruppe!Datenfeld = Wert
...
Datensatzgruppe.Update
```

Soll der geänderte Datensatz nicht in der Datenquelle gespeichert werden, verwenden Sie statt `Update` die Methode `CancelUpdate`. Sie verwirft die Änderungen an einem Datensatz.

Datensätze löschen

Auch zum Löschen eines Datensatzes positionieren Sie zuerst den Datensatzzeiger auf den gewünschten Datensatz. Anschließend können Sie mithilfe der Methode `Delete` das Löschen durchführen.

```
Datensatzgruppe.Find Suchkriterium
Datensatzgruppe.Delete
```

Nach dem Ausführen der `Delete`-Methode wird der Datensatz sofort in der Datenquelle gelöscht. Sie müssen daher nicht zusätzlich die Methode `Update` aufrufen.

Die Methode `Delete` des `Recordset`-Objekts verschiebt den Datensatzzeiger nicht. Er wird dadurch ungültig. Ein erneuter Zugriff erzeugt daher eine Fehlermeldung. Um dies zu vermeiden, können Sie den Datensatzzeiger vor dem Zugriff mit der Eigenschaft `EOF` auf Gültigkeit prüfen.

10.6 Schnellübersicht

Was bedeutet ...	
Datenquelle	Speicherort für Daten, z. B. Tabellen, Datenbanken, Dateien
Datenfeld	Gruppe von Feldern einer Datenquelle mit gleichem Datentyp; Spalte einer Datenquelle
Datensatz	Zeile einer Datenquelle; zusammengehörige Datenfelder
Datensatzgruppe	Auswahl von Datensätzen einer Datenquelle
Kriterium	Bedingung zur Auswahl eines Datensatzes
Datensatzzeiger	Verweist auf den aktuell ausgewählten Datensatz
ADO	ActiveX-Daten-Objekte (ActiveX Data Objects): Objektmodell zum Zugriff auf Datenquellen in VBA

Sie möchten ...	
eine Verbindung zu einer Datenquelle erstellen	`Dim Verbindung As New ADODB.Connection` `Verbindung.Open ("PfadZuDatenquelle")`
einen Provider für den Zugriff auf Access-Datenbanken verwenden	`Verbindung.Provider="Microsoft.Jet.OLEDB.4.0"`
eine Datensatzgruppe auswählen und öffnen	`Dim Datensatzgruppe As New ADODB.Recordset` `Datensatzgruppe.Open`
einen Datensatz hinzufügen	`Datensatzgruppe.AddNew`

den Datensatzzeiger verschieben	`Datensatzgruppe.MoveFirst`
	`Datensatzgruppe.MoveLast`
	`Datensatzgruppe.MovePrevious`
	`Datensatzgruppe.MoveNext`
	`Datensatzgruppe.Move`

einen Datensatz löschen	`Datensatzgruppe.Delete`
Änderungen an einem Datensatz speichern	`Datensatzgruppe.Update`
Änderungen an einem Datensatz nicht speichern	`Datensatzgruppe.CancelUpdate`
einen Datensatz suchen	`Datensatzgruppe.Find`
den Datensatzzeiger prüfen	`Datensatzgruppe.EOF` `Datensatzgruppe.BOF`
eine Datenquelle schließen	`Datensatzgruppe.Close`

10.7 Übung

Übungsdatei: *Übung1.xls; Gehalt.mdb* **Ergebnisdatei:** *Übung1-E.xls*

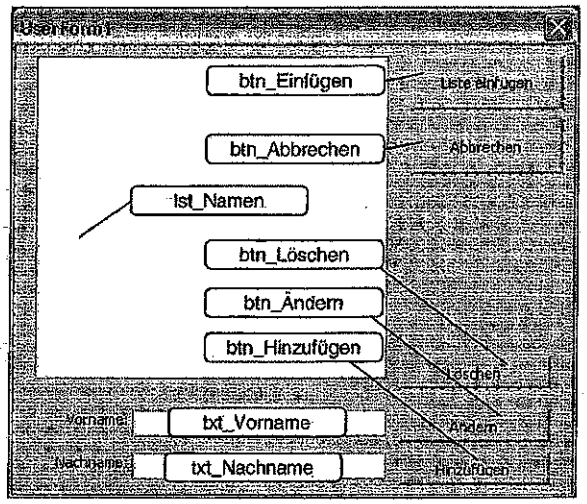
① Öffnen Sie die Datei *Übung1.xls*, und starten Sie die VBA-Entwicklungsumgebung.

② Erstellen Sie den in der Abbildung dargestellten UserForm-Dialog, und benennen Sie die Steuerelemente wie angegeben.

③ Das Listenfeld soll drei Spalten für Mitarbeiternummer, Name und Vorname enthalten. Verändern Sie zu diesem Zweck die Eigenschaften *ColumnCount* und *ColumnWidth*.

④ Deklarieren Sie die benötigten Objektvariablen für eine ADO-Verbindung und eine Datensatzgruppe global am Anfang des Moduls. Gleichzeitig soll eine neue Objektinstanz erzeugt werden.

⑤ Erstellen Sie eine Ereignisprozedur, die beim Start des UserForm-Dialogs eine Verbindung zur Datenbank *Gehalt.mdb* im gleichen Verzeichnis erstellt und die Tabelle *Mitarbeiter* öffnet. Damit die Anzahl der Datensätze bestimmt werden kann, verwenden Sie dabei den Cursortyp `adOpenStatic`.



Der UserForm-Dialog in der Entwurfsansicht

⑥ Fügen Sie eine neue Prozedur `ListeFüllen` hinzu, die das Listenfeld mit Einträgen aus der Datenbank füllt. Verwenden Sie dazu ein dynamisches zweidimensionales Array, dem Sie die Datenfelder *Nr*, *Name* und *Vorname* für alle Einträge aus der Datenbank zuweisen. Vor der ersten Verwendung des Arrays bestimmen Sie mit der Eigenschaft `RecordCount` die Anzahl der Datensätze der Tabelle. Legen Sie danach mit der Anweisung `ReDim` die Dimensionen des Arrays fest. Setzen Sie den Datensatzzeiger vor dem Auslesen der Werte an den Anfang der Tabelle, da die Prozedur mehrfach aufgerufen wird.

⑦ Das gefüllte Array weisen Sie dem Listenfeld mit der Methode `List` in der Form `Datensatzgruppe.Fields(0).Array zu`.

⑧ Erstellen Sie eine Ereignisprozedur für die Schaltfläche ABBRECHEN. Schließen Sie die Datenbankverbindung, und entladen Sie den UserForm-Dialog.

⑨ Nach der Auswahl eines Eintrags im Listenfeld sollen Vor- und Nachname in den entsprechenden Textfeldern angezeigt werden. Erstellen Sie dazu eine geeignete Ereignisprozedur. Die Namen ermitteln Sie mit der Eigenschaft `Column` des Listenfelds.

⑩ Die Namen in den Textfeldern können geändert werden. Erzeugen Sie für die Schaltfläche ÄNDERN eine Ereignisprozedur, die den aktuellen Datensatz mit den neuen Werten ändert. Um den richtigen Datensatz zu finden, verwenden Sie die `Find`-Methode und suchen Sie nach der eindeutigen Mitarbeiternummer des ausgewählten Listenfeldeintrags. Nach dem Ändern rufen Sie die Prozedur `ListeFüllen` auf, um die Änderungen sichtbar zu machen.

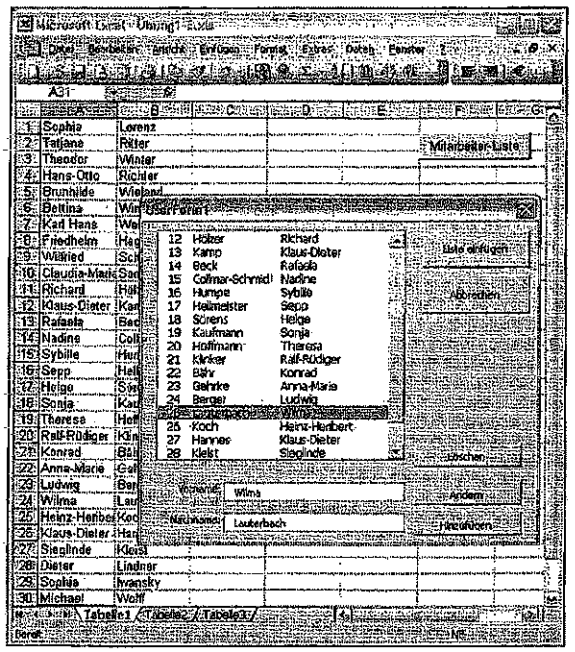
⑪ Erstellen Sie für die Schaltfläche HINZUFÜGEN eine Ereignisprozedur, die die Werte in den Textfeldern als neuen Eintrag in die Datenbank einfügt. Aktualisieren Sie danach den Inhalt des Listenfelds.

⑫ Nach dem Betätigen der Schaltfläche LÖSCHEN soll der im Listenfeld ausgewählte Eintrag aus der Datenbank gelöscht werden. Erstellen Sie dazu eine Ereignisprozedur, und aktualisieren Sie den Inhalt des Listenfelds.

⑬ Erzeugen Sie eine Ereignisprozedur für die Schaltfläche LISTE EINFÜGEN, die alle Werte der Datenbank in die aktuelle Excel-Tabelle einfügt. Fügen Sie dazu mithilfe der `Cells`-Methode in eine Spalte die Vornamen und in eine weitere Spalte die Nachnamen ein.

⑭ Ändern Sie die Ereignisprozeduren zum Ändern, Löschen und Hinzufügen so, dass keine Fehler auftreten, wenn im Listenfeld kein Eintrag gewählt wurde oder die Textfelder leer sind.

⑮ Erzeugen Sie im Tabellenblatt eine Formularschaltfläche MITARBEITER-LISTE, die eine geeignete Prozedur zum Öffnen des UserForm-Dialogs aufruft.



Der UserForm-Dialog mit der Mitarbeiterliste in Aktion

Notizen

11 Integrierte Lösungen mit Word

In diesem Kapitel erfahren Sie

- → welche Möglichkeiten es zur Automation mit Word gibt
- → wie Sie Funktionen fremder Anwendungen in Word nutzen
- → wie Sie Daten aus einer Datenbank in Word einfügen

Voraussetzungen

- ✓ Einsatz und Möglichkeiten von Automation-Objekten
- ✓ ADO-Objekte zum Datenaustausch

11.1 Automation mit Word

Word 2003 besitzt als Textverarbeitung umfangreiche und komplexe Möglichkeiten zur Arbeit mit Texten. Neben der manuellen Eingabe von Textabschnitten kann es notwendig sein, Daten aus Tabellen oder Datenbanken einzufügen. Häufig handelt es sich dabei um Adressen oder Zahlenwerte aus komplexen Tabellenkalkulationen. In diesen Fällen können Sie VBA-Programme erstellen, die Automation-Objekte verwenden, um diese Daten zu ermitteln.

Eine weitere Notwendigkeit für das Erstellen integrierter Lösungen mit Word ist das Nachrüsten fehlender Funktionen, bei denen Sie auf die Funktionalität anderer Office-Anwendungen zurückgreifen können. Beispielsweise ist es möglich, über ein Automation-Objekt die Diagrammfunktionen von Excel zu nutzen und für eigene Zwecke einzusetzen.

In den geschilderten Fällen arbeitet Word als Automation-Client und nutzt die Funktionen anderer Anwendungen, die ihrerseits als Server arbeiten.

Folgende beispielhafte Möglichkeiten ergeben sich bei integrierten Lösungen mit Word als Client:

Word als Client mit Excel als Server	☑	Ermitteln von Werten aus Tabellen
	☑	Berechnen von komplexen Formeln
	☑	Verwenden der Diagrammfunktionen von Excel
Word als Client mit Access als Server	☑	Ermitteln von einzelnen Werten aus umfangreichen Datenbeständen
	☑	Auslesen und Eingeben von Werten in eine Datenbank
Word als Client mit Outlook als Server	☑	Nutzung der Outlook-Adressbücher
	☑	Verwenden der Terminfunktion von Outlook

Die folgenden Abschnitte zeigen jeweils anhand eines Beispiels zwei verschiedene Möglichkeiten der Automation mit Word.

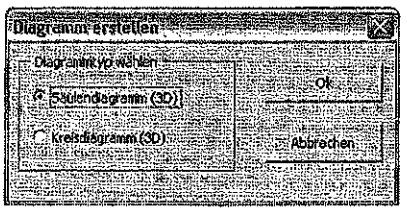
11.2 Diagrammfunktion von Excel nutzen

Word verfügt über umfangreiche Tabellenfunktionen, die sogar einfache Berechnungen erlauben. Eine leistungsfähige Diagrammfunktion gibt es jedoch nicht. Wollen Sie aus den Werten einer Tabelle in Word ein aussagekräftiges Diagramm erstellen, können Sie die Funktionen von Excel über Automation-Objekte verwenden.

Beispiel: *DiagrammErstellen.doc*, Modul `Diagramm`, UserForm-Dialog `frm_Diagramm`

Das Beispiel liest die Werte aus den Spalten einer Word-Tabelle und erstellt mit diesen Daten in einer versteckten Excel-Instanz ein dreidimensionales Diagramm, das in die Zwischenablage kopiert wird. Es kann danach an einer beliebigen Stelle im Dokument eingefügt werden.

Über einen UserForm-Dialog kann der Anwender den gewünschten Diagrammtyp wählen. In der Prozedur wird dabei nur eine Beispielformatierung verwendet.



Auswahl eines Diagrammtyps

Sie starten das Beispiel durch Ausführen der Prozedur `FormularAnzeigen` im Modul `Diagramm`. Diese Prozedur können Sie auch mit einer Schaltfläche in der Menüleiste verknüpfen.

Das Beispiel benötigt einen Verweis auf die Excel-Objektbibliothek.

Ereignisprozeduren

Es werden zwei Ereignisprozeduren benötigt, um auf das Betätigen der Schaltflächen im UserForm-Dialog zu reagieren.

```
Private Sub btn_Abbrechen_Click()
  Unload frm_Diagramm
End Sub

Private Sub btn_Ok_Click()
  DiagrammErstellen
  Unload frm_Diagramm
End Sub
```

① Nach dem Betätigen der Schaltfläche ABBRECHEN wird der UserForm-Dialog geschlossen und entladen.

② Die Ereignisprozedur für die OK-Schaltfläche startet die Prozedur `DiagrammErstellen`, um die Excel-Automation zu starten. Anschließend wird der UserForm-Dialog geschlossen.

Die Automationsprozedur

In einer Sub-Prozedur erfolgt die Kommunikation mit Excel. Dabei wird die Tabelle aus dem Word-Dokument über die Zwischenablage in Excel eingefügt, um daraus das Diagram zu erzeugen.

```
Sub DiagrammErstellen()
  Dim appXl As New Excel.Application, DiaTyp As Long, DiaPlot As Long
  If Selection.Information(wdWithInTable) = True Then
    Selection.Tables(1).Select
    Selection.Copy
  Else
    MsgBox "Bitte den Cursor in eine Tabelle setzen!", vbInformation
    frm_Diagramm.Hide
    Exit Sub
  End If
  If frm_Diagramm.opt_DiaTyp1 = True Then
    DiaTyp = xl3DColumn
    DiaPlot = xlRows
  ElseIf frm_Diagramm.opt_DiaTyp2 = True Then
    DiaTyp = xl3DPie
    DiaPlot = xlColumn
  End If
  frm_Diagramm.Hide
  With appXl
    .Workbooks.Add
    .ActiveSheet.Paste
```

```
⑨      .Charts.Add
       With ActiveChart
         .ChartType = DiaTyp
⑩        .SetSourceData appXl.Sheets("Tabelle1").UsedRange, DiaPlot
         .Location xlLocationAsNewSheet
⑪        .ChartArea.Copy
       End With
       .ActiveWorkbook.Close False
⑫      .Quit
     End With
     Set appXl = Nothing
     MsgBox "Das Diagramm wurde in die Zwischenablage kopiert.", vbInformation
End Sub
```

① Für die Kommunikation mit Excel wird die Objektvariable `appXL` deklariert. Gleichzeitig wird mit dem Schlüsselwort `New` eine neue Instanz der Excel-Objektklasse `Excel.Application` erstellt.

② An dieser Stelle wird mithilfe des `Selection`-Objekts von Word geprüft, ob sich der Cursor innerhalb einer Tabelle befindet. Ist dies der Fall, wird die gesamte Tabelle markiert (`Select`-Methode) und in die Zwischenablage kopiert (`Copy`-Methode). Anderenfalls wird eine Fehlermeldung ausgegeben und die Prozedur beendet.

③ Hier erfolgt eine Überprüfung, welcher Diagrammtyp im UserForm-Dialog gewählt wurde. Je nach Auswahl werden die Variablen `DiaPlot` und `DiaTyp` entsprechend eingestellt.

④ Der geöffnete UserForm-Dialog lässt keine Tastatureingaben in eine andere Anwendung zu. Das Einfügen der kopierten Tabelle in Excel würde daher nicht funktionieren. Aus diesem Grund kann der Dialog an dieser Stelle geschlossen werden.

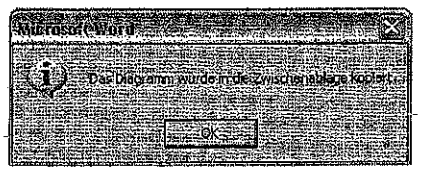
⑤ Für die einzufügende Tabelle erstellt die `Add`-Methode eine neue leere Arbeitsmappe.

⑥ Die in die Zwischenablage kopierte Tabelle wird an dieser Stelle in das aktive Tabellenblatt kopiert.

⑦ Über die `Add`-Methode der `Charts`-Auflistung wird ein leeres Diagramm erstellt.

⑧ Das neue Diagramm wird automatisch aktiv. Der Zugriff kann daher über `ActiveChart` erfolgen.

⑨ Der Eigenschaft `ChartType` des aktiven Diagramms wird der vorher eingestellte Diagrammtyp zugewiesen. Über die Eigenschaft `SetSourceData` wird festgelegt, dass die Werte für das Diagramm aus dem belegten Bereich des ersten Tabellenblatts gelesen werden sollen (`appXl.Sheets("Tabelle1").UsedRange`). Mittels der `Location`-Methode wird ein neues Tabellenblatt für das Diagramm erstellt.

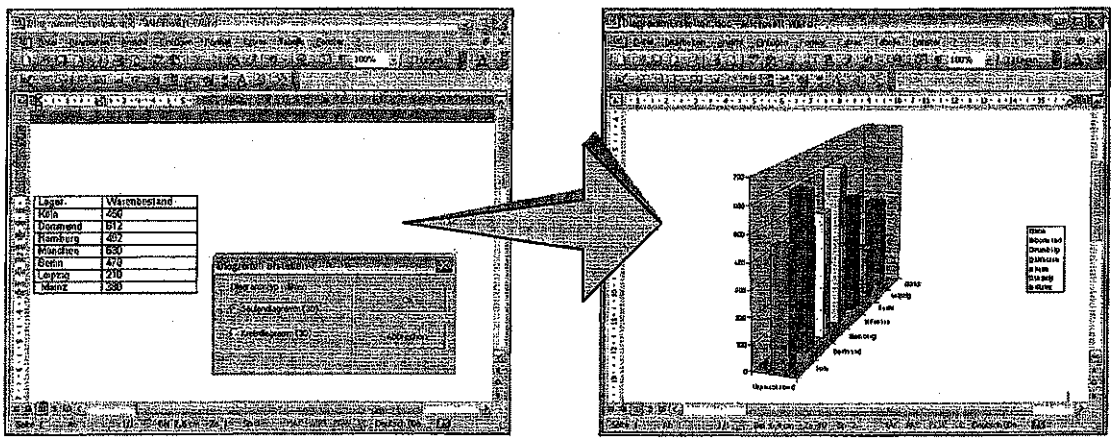


Mit einem Meldungsfenster wird der Anwender über den Erfolg informiert

⑩ Der Diagrammbereich wird an dieser Stelle in die Zwischenablage kopiert.

⑪ Um die Aufforderung zum Speichern der neuen Arbeitsmappe zu umgehen, wird die Arbeitsmappe mit der Methode `Close` geschlossen. Das Argument `False` bedeutet, dass auf das Speichern verzichtet wird. Erst danach wird Excel selbst geschlossen.

⑫ Am Ende der Prozedur wird der vom Automation-Objekt belegte Speicher freigegeben und ein Meldungsfenster eingeblendet.



Die Beispielprozedur hat ein 3D-Säulendiagramm aus den Werten einer Word-Tabelle erstellt

Indem Sie in der Prozedur `DiagrammErstellen` den Variablen `DiaTyp` und `DiaPlot` neue Werte zuweisen, können Sie das Beispiel um weitere Diagrammtypen ergänzen.

Datenaustausch über die Zwischenablage

Für einen einfachen Datenaustausch wurde im Beispiel die Zwischenablage verwendet. In vielen Fällen ist dies eine geeignete Methode. Sie hat jedoch den Nachteil, dass der Anwender keinen Einfluss auf die einzelnen übergebenen Werte hat.

Wenn Sie Tabellen aus Word über die Zwischenablage in Excel einfügen, werden die Daten automatisch in Spalten und Zeilen aufgeteilt. Sie sparen sich dadurch das manuelle Vorgehen mit `Range`- oder `Cells`-Objekten. Beachten Sie dabei, dass Zahlenwerte bei der Verwendung eines Dezimalpunkts oder Leerzeichens als Tausendertrennzeichen eventuell verfälscht als Datum oder Text eingefügt werden.

Syntax der Methoden für die Arbeit mit der Zwischenablage

```
Objektbezeichnung.Copy
Objektbezeichnung.Paste
```

- ☑ Die Methoden `Copy` und `Paste` werden nur von einigen Objekten zur Verfügung gestellt. Die `Copy`-Methode kopiert je nach Objekt unterschiedliche Daten.
- ☑ Die `Paste`-Methode fügt die in der Zwischenablage befindlichen Daten an der durch das Objekt vorgegebenen Stelle ein.

Die Methoden `Copy` und `Paste` stehen nur bei einigen Objekten zur Verfügung. Die folgende Tabelle nennt wichtige Objekte, die eine Verwendung der Zwischenablage erlauben:

Objekt	Anwendung
`Selection`	Word, PowerPoint
`Range`	Word, Excel
`Shape`	Excel, PowerPoint
`Slide`	PowerPoint

11.3 Adressen aus Access übernehmen

Word verfügt über eine nützliche Serienbrieffunktion, bei der die Adressdaten auch aus einer Access-Datenbank übernommen werden können. Es gibt jedoch keine vorbereitete Funktion, um eine einzelne Adresse aus einer Datenbank auszulesen, beispielsweise um einen einzelnen Brief zu schreiben.

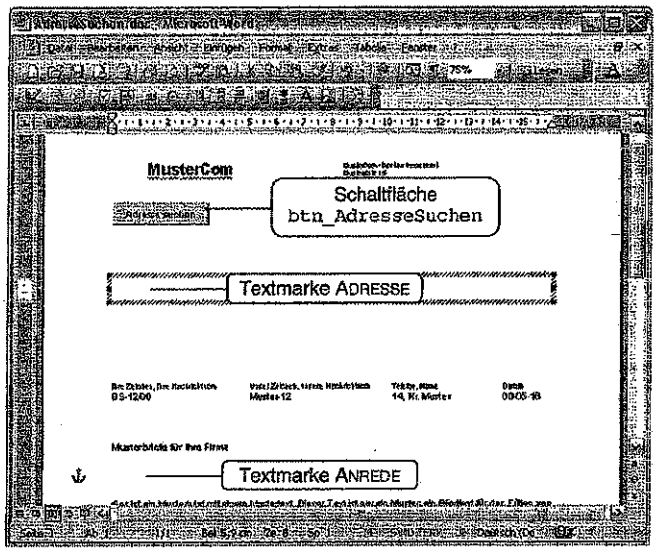
Über den Datenbankzugriff mit ADO können Sie diese fehlende Funktion jedoch leicht nachrüsten.

Notizen

Beispiel: *AdresseSuchen.doc*, Modul `AdresseSuchen`, Datei *Adressen.mdb*

Das Beispiel realisiert eine integrierte Lösung mit Word und einer Access-Datenbank. Die Datenbank ADRESSEN.MDB enthält eine Liste mit Adressen und Ansprechpartnern. Zu jedem Datensatz ist zusätzlich die gewünschte Anrede gespeichert.

Über einen einfachen Eingabedialog kann in Word nach einem bestimmten Namen in der Datenbank gesucht werden. Wurde ein passender Datensatz gefunden, fügt die Prozedur die Adresse und die dazugehörige Briefanrede ein. Zu diesem Zweck existieren in dem Word-Dokument zwei Textmarken mit den Namen ADRESSE und ANREDE. Die Prozedur wird über die Schaltfläche `btn_AdresseSuchen` im Dokument gestartet.



Markierung der Einfügepositionen mit Textmarken

Gesuchten Nachnamen eingeben

```
Sub AdresseSuchen()
  Dim ADOCnn As ADODB.Connection, ADOTab As ADODB.Recordset
  Dim Name As String
  Set ADOCnn = New ADODB.Connection
  Name = InputBox("Nach welchem Nachnamen soll gesucht werden?", _
                  "Adresse suchen", "Wenger")
  If Name > "" Then
    ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
    ADOCnn.Open (ActiveDocument.Path & "\adressen.mdb")
    Set ADOTab = New ADODB.Recordset
    With ADOTab
      .Open "Adressen", ADOCnn, adOpenDynamic, adLockOptimistic
      .Find "Nachname='" & Name & "'"
      If Not .EOF Then
        ActiveDocument.GoTo(What:=wdGoToBookmark, _
                            Name:="Adresse").Select
        Selection.InsertAfter .Fields("Vorname") & " " & _
                              .Fields("Nachname") & vbCr
        Selection.InsertAfter .Fields("Adresse") & vbCrLf
        Selection.InsertAfter .Fields("Land") & "-" & .Fields("PLZ") _
                              & " " & .Fields("Ort")
        ActiveDocument.GoTo(What:=wdGoToBookmark, Name:="Anrede").Select
        Selection.InsertAfter .Fields("Briefanrede") & ","
      Else
        MsgBox "Es wurde kein passender Datensatz gefunden!", vbCritical
      End If
      .Close
    End With
```

```
⑪     ADOCnn.Close
      Set ADOCnn = Nothing
    End If
End Sub
```

① Die für einen Datenaustausch über ADO-Objekte notwendigen Objektvariablen werden an dieser Stelle deklariert. Zur korrekten Funktion muss vorher ein Verweis auf die ADO-Objektbibliothek erstellt werden.

② Über das `ADODB.Connection`-Objekt wird die Umgebung für den Zugriff auf die Access-Datenbank geschaffen.

③ Der passende Provider für eine Access-Datenbank wird festgelegt und danach die Datenbank geöffnet.

④ `ADOTab` wird an dieser Stelle die Instanz einer ADO-Datensatzgruppe zugewiesen.

⑤ Mithilfe der `Open`-Methode wird die Tabelle *Adressen* über die bestehende Datenverbindung geöffnet.

⑥ Die `Find`-Methode erlaubt das Suchen nach einem bestimmten Kriterium. Hier wird im Datenfeld *Nachname* nach dem eingegebenen Namen gesucht.

⑦ Nach erfolgreicher Suche wird zuerst zur Textmarke *Adresse* des aktuellen Dokuments verzweigt, und die betreffende Stelle wird markiert.

⑧ Der Cursor befindet sich an der gewünschten Stelle. Über die `InsertAfter`-Methode wird nun die Adresse eingefügt, die aus den Datenfeldern der Datenbank gelesen wurde.

⑨ Nach dem Einfügen der Adresse wird die Textmarke *Anrede* angesprungen, um an dieser Stelle die Briefanrede aus der Datenbank einzufügen.

⑩ War die Suche nicht erfolgreich, wird ein entsprechendes Meldungsfenster ausgegeben.

⑪ Am Ende der Prozedur wird die Datenverbindung geschlossen und der belegte Speicher freigegeben.

Textmarken als Einfügemarkierungen

Das Beispiel verwendet zwei Textmarken, die festlegen, an welche Stellen im Dokument Daten eingefügt werden sollen. Diese Vorgehensweise ist besonders bei größeren, strukturierten Dokumenten, wie z. B. Briefen oder Anschreiben, vorteilhaft.

Textmarken können Sie mit der `Goto`-Methode gezielt anspringen und anschließend markieren. Durch das Markieren wird der Cursor an die gewünschte Stelle gesetzt und das Einfügen von Text ermöglicht.

Mit Textmarken erreichen Sie eine hohe Flexibilität in der Struktur der zu verarbeitenden Dokumenten, da bei der Programmierung nur die Namen der Textmarken verbindlich sind, nicht jedoch deren Position.

Syntax für das Verzweigen zu Textmarken

```
Documentobjekt.GoTo(What:=wdGoToBookmark, Name:="Adresse").Select
```

☑ Die `GoTo`-Methode liefert ein `Range`-Objekt zurück, das sich auf die angegebene Textmarke bezieht. Durch den Aufruf der Methode `Select` für dieses `Range`-Objekt wird der Cursor zu der Textmarke bewegt.

☑ Ein typisches `Document`-Objekt ist beispielsweise `ActiveDocument`, das einen Verweis auf das aktuelle Dokument darstellt.

Notizen

11.4 Schnellübersicht

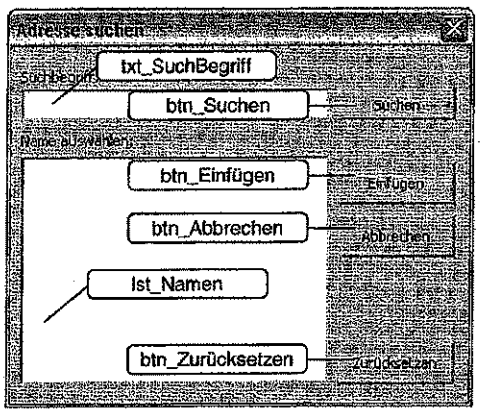
Sie möchten ...	
Daten in die Zwischenablage kopieren	`Objektbezeichnung.Copy`
Daten aus der Zwischenablage einfügen	`Objektbezeichnung.Paste`
eine Textmarke anspringen	`ActiveDocument.Goto (What:=wdGoToBookmark, Name:="Adresse")`
eine Textmarke anspringen und markieren	`ActiveDocument.Goto(What:=wdGoToBookmark, Name:="Adresse").Select`
Text an der Cursorposition eingeben	`Selection.InsertAfter Text`

11.5 Übung

Übungsdatei: *Übung1.doc, Adressen.mdb* **Ergebnisdatei:** *Übung1-E.doc*

① Öffnen Sie die Datei *AdresseSuchenFormular.doc*, und starten Sie die VBA-Entwicklungsumgebung. Aufbauend auf diesem Beispiel soll eine Lösung mit besserer Benutzeroberfläche erstellt werden.

② Fügen Sie einen neuen UserForm-Dialog hinzu, und erstellen Sie ein Textfeld, ein Listenfeld und vier Schaltflächen. Benennen Sie die Steuerelemente wie angegeben.



Der UserForm-Dialog im Entwurfsmodus

③ Fügen Sie dem Deklarationsteil des UserForm-Moduls die Deklarationen für die benötigten ADO-Objekte hinzu.

④ Erstellen Sie eine Ereignisprozedur, die beim Aktivieren des Dialogs eine ADO-Datenverbindung zur Datenbank *Adressen.mdb* im aktuellen Verzeichnis öffnet. Diese Verbindung sollen auch die anderen Ereignisprozeduren nutzen können.

⑤ Ergänzen Sie die Ereignisprozedur so, dass sie das Listenfeld lst_Namen mit allen Nachnamen aus der Datenbank füllt.

⑥ Für die Schaltfläche ABBRECHEN erstellen Sie eine Ereignisprozedur, die die ADO-Verbindung schließt, den Speicher freigibt und das Formular entlädt.

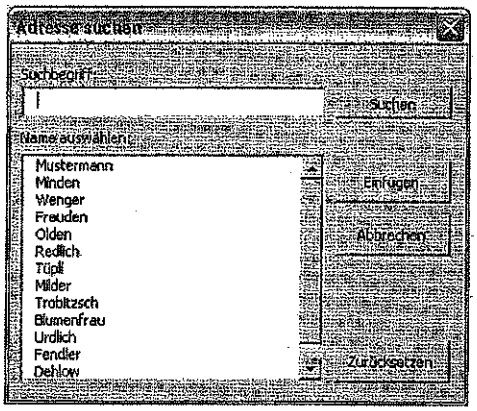
⑦ Nach dem Betätigen der Schaltfläche SUCHEN soll in allen Nachnamen in der Datenbank nach dem im Textfeld eingegebenen Suchbegriff gesucht werden. Erstellen Sie dafür eine Ereignisroutine. Damit Sie alle passenden Datensätze im Listenfeld anzeigen können, verwenden Sie innerhalb einer `While`-Schleife die `Find`-Methode mit dem `Like`-Operator im Suchkriterium und mit dem Offset 1.

```
Datensatzgruppe.Find Kriterium, Offset
```

⑧ Erstellen Sie eine Ereignisprozedur für die Schaltfläche ZURÜCKSETZEN. Sie soll das Listenfeld mit möglichen Suchergebnissen leeren und erneut mit allen Nachnamen aus der Datenbank füllen.

⑨ Beim Betätigen der Schaltfläche EINFÜGEN soll der im Listenfeld ausgewählte Name verwendet werden, um die zugehörige Adresse in das Word-Dokument einzufügen. Verzweigen Sie dazu nacheinander zu den Textmarken *Adresse* und *Anrede*, und fügen Sie die Daten aus der Datenbank an der Cursorposition ein.

⑩ Erstellen Sie für die im Dokument befindliche Schaltfläche eine Ereignisroutine, die den UserForm-Dialog öffnet.



Der UserForm-Dialog in Aktion

Notizen

12 Integrierte Lösungen mit Excel

In diesem Kapitel erfahren Sie

→ welche Möglichkeiten es zur Automation mit Excel gibt
→ wie Sie Funktionen anderer Office-Anwendungen in Excel nutzen
→ wie Sie Daten aus Outlook und Access in Excel verwenden

Voraussetzungen

✓ Programmieren von Automation-Objekten
✓ Datenaustausch mit ADO-Objekten

12.1 Automation mit Excel

Excel eignet sich besonders für das Analysieren und Auswerten von Daten in Tabellenform. Für diesen Zweck besitzt es sehr spezialisierte Funktionen, die über Automation auch in anderen Anwendungen genutzt werden können. Excel wird daher sehr häufig als Automation-Server eingesetzt, der Funktionen zur Datenbearbeitung anbietet oder Daten liefert.

Sollen jedoch sehr große Mengen Daten verwaltet werden, ist eine Datenbank, wie z. B. Access, effektiver als Excel. In solchen Fällen können Sie integrierte Lösungen erstellen, die mithilfe von Automation-Objekten gezielt Werte aus einer Datenbank auslesen, um sie in Excel zu verwenden.

Eine weitere Notwendigkeit für das Erstellen integrierter Lösungen mit Excel ist das Nachrüsten fehlender Funktionen. Beispielsweise ist es möglich, über ein Automation-Objekt die flexiblere Druckfunktion von Word zu nutzen.

Excel wird häufig als Automation-Server eingesetzt, um Clients den Zugriff auf die Excel-Funktionen und die in Tabellen gespeicherten Werte zu erlauben. Möglich ist jedoch auch der Zugriff auf andere Automation-Server, um deren Funktionen und Daten einzubinden.

Folgende Möglichkeiten ergeben sich für integrierte Lösungen mit Excel:

Excel als Server mit Word als Client	☑	Bereitstellen von Tabellenwerten
	☑	Bereitstellen der speziellen Excel-Funktionen
Excel als Client mit Word als Server	☑	Nutzen der Textverarbeitungsfunktionen
	☑	Verwenden spezieller Word-Funktionen
Excel als Server mit Access als Client	☑	Analysieren von Daten mit Excel-Funktionen
Excel als Client mit Outlook als Server	☑	Verwenden der E-Mail-Funktion
	☑	Auslesen der Outlook-Adressbücher

Die folgenden Abschnitte zeigen jeweils anhand eines Beispiels zwei verschiedene Möglichkeiten der Automation mit Excel.

12.2 Druckfunktion von Word nutzen

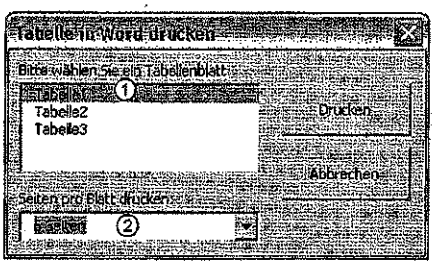
Das folgende Beispiel demonstriert, wie Sie Funktionen eines fremden Automation-Servers, z. B. Word, für eigene Zwecke nutzen können. Im Gegensatz zu Excel verfügt Word über eine Druckfunktion, die es erlaubt, mehrere Seiten (zwei bis 16 Seiten) verkleinert auf ein einziges Blatt zu drucken. Gerade für umfangreiche Tabellen ist dies eine nützliche und Papier sparende Funktion, über die Excel nicht verfügt.

Das Beispiel kopiert den mit Werten belegten Teil eines Tabellenblatts über die Zwischenablage in ein neues Word-Dokument. Anschließend wird das Dokument mit den Parametern für die gewählte Seitenzahl pro Blatt ausgedruckt.

Beispiel *WordDruck.xls*, Modul `WordDruckFunktion`, UserForm-Dialog `UserForm1`

Die integrierte Anwendung wird mit dem Aktivieren des UserForm-Dialogs `UserForm1` gestartet.

Über den UserForm-Dialog kann der Anwender im Listenfeld ① auswählen, welches Tabellenblatt der aktuellen Arbeitsmappe verwendet wird. Im Kombinationsfeld ② kann ausgewählt werden, wie viele Seiten Word auf ein Blatt Papier drucken soll. Nach dem Betätigen der Schaltfläche DRUCKEN erfolgt der automatische Ausdruck über Word.



Auswahl eines Diagrammtyps

Das Beispiel benötigt einen Verweis auf die Word-Objektbibliothek.

Ereignisprozeduren

Die Ereignisprozeduren werden beim Öffnen des Formulars bzw. beim Betätigen der Schaltflächen ausgeführt.

```
①  Private Sub btn_Abbrechen_Click()
     Unload UserForm1
   End Sub

②  Private Sub btn_Drucken_Click()
     Me.Hide
     WordDruck
   End Sub

③  Private Sub UserForm_Activate()
     Dim TabBlatt As Worksheet
④    If Me.cmb_Seiten.ListCount = 0 Then
⑤      With Me.cmb_Seiten
         .AddItem "2 Seiten"
         .AddItem "4 Seiten"
         .AddItem "6 Seiten"
         .AddItem "8 Seiten"
         .AddItem "16 Seiten"
       End With
       For Each TabBlatt In ActiveWorkbook.Worksheets
⑥        Me.lst_Tabellen.AddItem TabBlatt.Name
       Next
     End If
   End Sub
```

① Nach dem Betätigen der Schaltfläche ABBRECHEN wird der UserForm-Dialog geschlossen und entladen.

② Die Ereignisprozedur für die Schaltfläche DRUCKEN blendet den UserForm-Dialog aus und startet die Prozedur `WordDruck`, um die Word-Automation zu starten.

③ Beim Start des UserForm-Dialogs initialisiert diese Ereignisprozedur die verschiedenen Steuerelemente.

④ An dieser Stelle wird geprüft, ob das Kombinationsfeld bereits Werte enthält. Wenn dies der Fall ist, wurde der UserForm-Dialog bereits initialisiert und mit der Hide-Methode geschlossen. Ein erneutes Zuweisen der Steuerelementwerte ist daher nicht notwendig.

⑤ Über die AddItem-Methode werden dem Kombinationsfeld cmb_Seiten Werte für eine Auswahl der Seiten pro Blatt übergeben.

⑥ Das Listenfeld lst_Tabellen wird mit den Namen der Tabellenblätter der aktuellen Arbeitsmappe gefüllt.

Die Automationsprozedur

Die Sub-Prozedur WordDruck übernimmt die Automation von Word. Dazu wird Word gestartet und ein neues Dokument erzeugt. Mithilfe der Zwischenablage erfolgt der Datenaustausch zwischen Excel und Word. Das Word-Dokument wird danach mit den gewählten Einstellungen gedruckt.

```
Sub WordDruck()
①  Dim AppWD As New Word.Application
   Dim TabXL As Worksheet
   Dim Zeilen As Long, Spalten As Long, Seiten As Long
②  If UserForm1.lst_Tabellen > 0 And UserForm1.cmb_Seiten > 0 Then
③    Seiten = Val(UserForm1.cmb_Seiten.Value)
     Select Case Seiten
       Case 2
         Zeilen = 1
         Spalten = 2
       Case 4, 6, 8
         Zeilen = 2
         Spalten = Seiten / 2
       Case 16
         Zeilen = 4
         Spalten = 4
     End Select
④    Set TabXL = ActiveWorkbook.Worksheets(UserForm1.lst_Tabellen.ListIndex + 1)
⑤    TabXL.UsedRange.Select
⑥    Selection.Copy
     With AppWD
⑦      .Visible = True
       .Activate
       .Documents.Add
       .Selection.Paste
⑧      .PrintOut Background:=False, PrintZoomRow:=Zeilen, _
                                          PrintZoomColumn:=Spalten
⑨      .ActiveDocument.Close SaveChanges:=False
       .Quit SaveChanges:=False
     End With
⑩    Set AppWD = Nothing
     Unload UserForm1
   Else
⑪    MsgBox "Bitte wählen Sie ein Tabellenblatt und die _
           Seiten pro Blatt aus!", vbInformation
     UserForm1.Show
   End If
End Sub
```

① Für die Kommunikation mit Excel wird die Objektvariable `AppWD` deklariert. Gleichzeitig wird mit dem Schlüsselwort `New` eine neue Instanz der Word-Objektklasse `Word.Application` erstellt.

② An dieser Stelle wird überprüft, ob im UserForm-Dialog im Listen- und Kombinationsfeld ein Wert ausgewählt wurde. Nur dann kann die Automation erfolgen.

③ Die `Val`-Funktion wandelt eine Zeichenkette in einen Zahlenwert um. Alle ungültigen Zeichen, wie z. B. Buchstaben, werden dabei verworfen. Die Variable `Seiten` enthält danach die im Kombinationsfeld ausgewählte Anzahl Seiten pro Blatt. Beispielsweise wird `Seite` der Zahlenwert 6 zugewiesen, wenn im Kombinationsfeld der Eintrag 6 SEITEN ausgewählt wurde. In der folgenden `Select`-Anweisung werden, auf der Seitenzahl basierend, Variablen für die Druckfunktion von Word initialisiert.

④ Die Variable `TabXL` erhält einen Verweis auf das im Listenfeld ausgewählte Tabellenblatt. Der Index des Listenfelds wird dabei um den Wert 1 erhöht, sodass er im Gegensatz zur Zählung der Tabellenblätter mit 0 beginnt.

⑤ An dieser Stelle wird der mit Werten belegte Bereich der Tabelle (`UsedRange`) mit der `Select`-Methode ausgewählt.

⑥ Die Auswahl wird hier in die Zwischenablage kopiert.

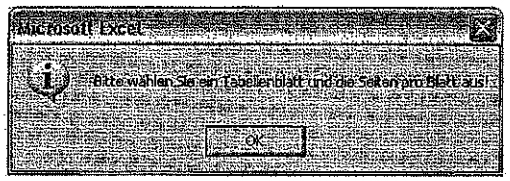
⑦ Über das Word-Automation-Objekt wird die Word-Anwendung sichtbar gesetzt, aktiviert, und ein neues leeres Dokument wird hinzugefügt. In dieses Dokument wird danach der Inhalt der Zwischenablage eingefügt (`Selection.Paste`). Word kann durch Zuweisen des Wertes `False` an die `Visible`-Eigenschaft auch versteckt im Hintergrund gestartet werden.

⑧ Die `PrintOut`-Methode startet den Ausdruck des Dokuments. Als Argument dienen dabei die Variablen `Zeilen` und `Spalten`, die festlegen, wie viele Spalten und Zeilen für den Ausdruck der Seiten verwendet werden sollen.

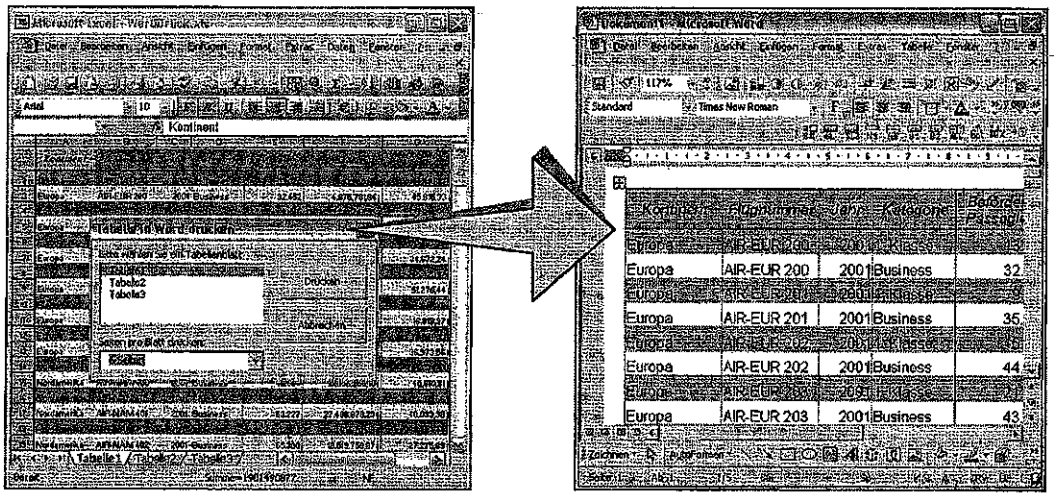
⑨ Nach dem Ausdruck wird das aktive Dokument geschlossen und dabei die Abfrage zum Speichern von Änderungen unterdrückt. Auch die Word-Anwendung wird mithilfe der `Quit`-Methode geschlossen und das Speichern von Änderungen unterbunden.

⑩ Die Variable `AppWD` wird nicht mehr benötigt und kann durch Zuweisen von `Nothing` freigegeben werden.

⑪ Die Abfrage in ② verzweigt an diese Stelle, falls im UserForm-Dialog kein Wert ausgewählt wurde. In diesem Fall wird ein Meldungsfenster ausgegeben und der Dialog erneut angezeigt.



Im UserForm-Dialog wurden nicht alle benötigten Werte ausgewählt



Die Beispielprozedur kopiert eine Tabelle in ein Word-Dokument und druckt sie aus

Je nach Umfang und Format der Excel-Tabelle kann das Einfügen in Word einige Zeit dauern.

Damit Word nicht beendet wird, bevor das Dokument gedruckt ist, wird in diesem Fall das Drucken im Hintergrund unterdrückt. Dies wird mit der Übergabe des Parameter `Background:=False` für die `PrintOut`-Methode erreicht.

12.3 Daten aus Outlook und Access importieren

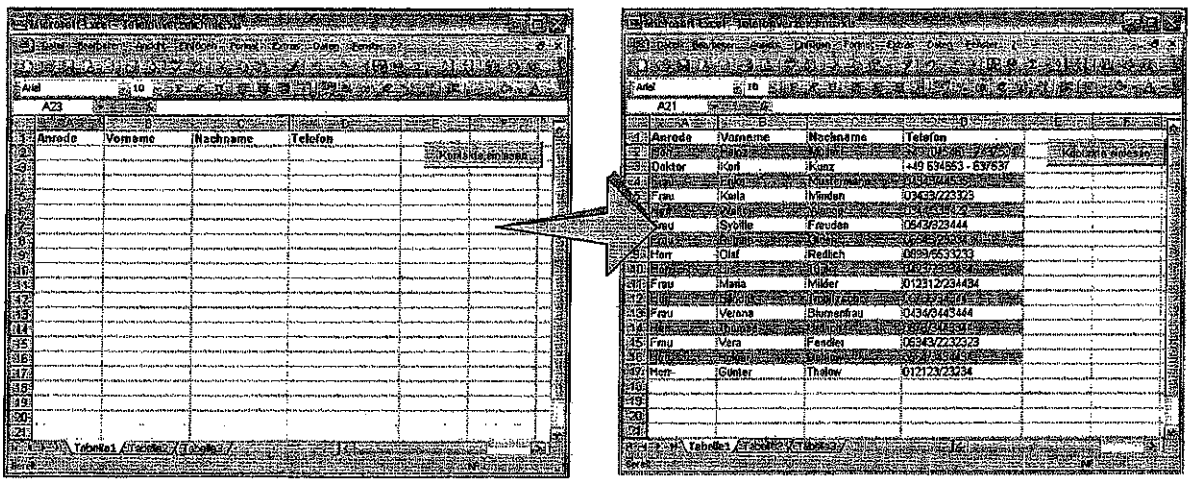
Excel wird häufig eingesetzt, um strukturierte Daten auf einfache Weise zu verwalten. Dabei kann es vorkommen, dass Sie Daten aus verschiedenen Quellen auslesen und importieren müssen. Mithilfe der Automation-Objekte können Sie dies komfortabel realisieren.

Im folgenden Beispiel soll ein Telefonverzeichnis als Excel-Tabelle erstellt werden. Die Namen und Telefonnummern dafür befinden sich zum einen als Kontakte in einem Outlook-Ordner, zum anderen in einer Access-Datenbank. Beide Datenbestände sollen durch eine integrierte Anwendung ausgelesen und in Excel zusammengeführt werden.

Beispiel: *Telefonverzeichnis.xls*, Modul `KontakteImport`, Datei *Adressen.mdb*

Das Beispiel besteht aus jeweils einer Prozedur zum Einlesen von Kontaktinformationen aus Outlook und einer Access-Datenbank. Der Importvorgang wird durch eine in das Dokument integrierte Schaltfläche gestartet. Am Ende erfolgt eine automatische Formatierung der Telefonliste.

Die Anwendung benötigt Verweise auf die Outlook- und die ADO-Objektbibliothek.



Nach dem Betätigen der Schaltfläche werden Kontakte aus Outlook und einer Datenbank eingelesen und formatiert

Kontakte aus Outlook importieren

Die Sub-Prozedur liest alle Namen und Telefonnummern aus dem Standard-Outlook-Kontakteordner.

```
Sub OutlookAdressen()
  Dim AppOL As New Outlook.Application
  Dim MeinNS As NameSpace, KontakteOrdner As MAPIFolder
  Dim MeinOrdner As MAPIFolder, Kontakt As ContactItem
  Dim Zeile As Long
  Set MeinNS = AppOL.GetNamespace("MAPI")
  Set KontakteOrdner = MeinNS.GetDefaultFolder(olFolderContacts)
  Zeile = ActiveSheet.UsedRange.Rows.Count + 1
  For Each Kontakt In KontakteOrdner.Items
    Cells(Zeile, 1) = Kontakt.Title
    Cells(Zeile, 2) = Kontakt.FirstName
    Cells(Zeile, 3) = Kontakt.LastName
    Cells(Zeile, 4) = Kontakt.BusinessTelephoneNumber
    Zeile = Zeile + 1
  Next
  AppOL.Quit
  Set KontakteOrdner = Nothing
```

```
    Set MeinNS = Nothing
    Set AppOL = Nothing
End Sub
```

① Für den Zugriff auf Outlook werden verschiedene Objektvariablen benötigt, die an dieser Stelle deklariert werden. Für die Variable `AppOL` wird eine neue Instanz der Outlook-Anwendung erzeugt. `MAPI Folder` ist die Objektklasse für Ordner in Outlook, `ContactItem` repräsentiert einzelne Einträge in einem Kontaktordner.

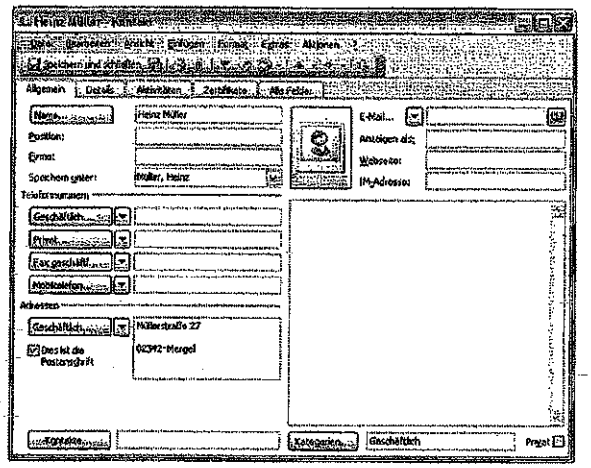
② Auf alle Outlook-Elemente und Nachrichtenspeicher kann nur über einen Verweis auf ein `Namespace`-Objekt zugegriffen werden. VBA unterstützt dabei nur die MAPI-Schnittstelle, die an dieser Stelle zugewiesen wird.

③ Der Variablen `KontakteOrdner` wird mithilfe der `GetDefaultFolder`-Methode ein Verweis auf den Standardordner für Kontakte übergeben.

④ Mithilfe der `UsedRange`-Methode wird an dieser Stelle die Anzahl der belegten Zeilen im aktiven Tabellenblatt ermittelt. Die Kontakte aus Outlook sollen ab der nächsten freien Zeile eingefügt werden. Die Outlook-Kontakte enthalten feste Felder, die die entsprechenden Daten beinhalten.

⑤ Die `Items`-Auflistung enthält alle Kontakteinträge des Ordners. In einer `For-Each`-Schleife werden diese nacheinander ausgelesen und in die Excel-Tabelle eingefügt.

⑥ Nach dem erfolgreichen Import wird Outlook geschlossen und die Objektvariablen freigegeben.



Ein Kontakt bei der Eingabe in Outlook

Notizen

Kontakte aus einer Datenbank importieren

Die Namen und Telefonnummern aus der Access-Datenbank *adressen.mdb* werden mithilfe eines ADO-Datenzugriffs ausgelesen.

```
Sub AccessAdressen()
  Dim ADOCnn As New ADODB.Connection, ADOTab As New ADODB.Recordset
  Dim ADOField As ADODB.Field, Zeile As Long
  ADOCnn.Provider = "Microsoft.Jet.OLEDB.4.0"
  ADOCnn.Open (ActiveWorkbook.Path & "\adressen.mdb")
  ADOTab.Open "Adressen", ADOCnn, adOpenDynamic, adLockOptimistic
  Zeile = ActiveSheet.UsedRange.Rows.Count + 1
  While Not ADOTab.EOF
    Cells(Zeile, 1) = ADOTab!Anrede
    Cells(Zeile, 2) = ADOTab!Vorname
    Cells(Zeile, 3) = ADOTab!Nachname
    Cells(Zeile, 4) = ADOTab!Telefon
    Zeile = Zeile + 1
    ADOTab.MoveNext
  Wend
  ADOCnn.Close
  Set ADOCnn = Nothing
End Sub
```

① Die Deklaration definiert die für einen Datenaustausch mit ADO benötigten Objektvariablen.

② Für den Zugriff auf eine Access-Datenbank wird der Provider für Microsoft Jet OLEDB benötigt.

③ Eine neue Instanz für die ADO-Verbindung wurde bereits bei der Deklaration erstellt. An dieser Stelle kann daher sofort die benötigte Datenbank geöffnet werden.

④ Über die geöffnete Verbindung zur Datenbank wird hier die Tabelle *Adressen* geöffnet.

⑤ Mithilfe der `UsedRange`-Methode wird die Anzahl der belegten Zeilen im aktiven Tabellenblatt ermittelt.

⑥ Nacheinander werden die Datensätze in einer `While`-Schleife durchlaufen und die entsprechenden Daten in die Excel-Tabelle eingefügt. Mithilfe der Methode `MoveNext` wird dabei zum nächsten Datensatz gewechselt.

⑦ Nach dem Datentransfer wird die Verbindung geschlossen und die Objektvariable freigegeben.

Ereignisprozedur der Schaltfläche

Die Ereignisprozedur für das Betätigen der Schaltfläche KONTAKTE EINLESEN startet den Import der Kontakte und formatiert jede zweite Zeile mit einer grauen Hintergrundfarbe.

```
Private Sub btn_Einlesen_Click()
  Dim Zeile As Long, Spalte As Long
  OutlookAdressen
  AccessAdressen
  For Zeile = 2 To ActiveSheet.UsedRange.Rows.Count Step 2
    For Spalte = 1 To ActiveSheet.UsedRange.Columns.Count
      Cells(Zeile, Spalte).Interior.Color = RGB(220, 220, 220)
    Next
  Next
End Sub
```

① Nacheinander werden die beiden Prozeduren zum Import der Kontakte aus Outlook und Access gestartet.

② Die erste `For`-Schleife durchläuft die Zeilen des mit Werten belegten Teils des Tabellenblatts. Durch die Anweisung `Step 2` wird erreicht, dass jede zweite Zeile übersprungen wird.

③ Die innere `For`-Schleife liefert über die Variable `Spalte` die Spaltennummern im verwendeten Tabellenbereich.

④ Der Zellenhintergrund wird an dieser Stelle mit einem hellen Grau formatiert.

Anrede	Vorname	Nachname	Telefon
Herr	Heinz	Müller	+49 (345/6) 7437547
Doktor	Karl	Kunz	+49 634653 - 637637
Frau	Erika	Mustermann	04434/44536
Frau	Karla	Minden	03433/223323
Herr	Walter	Wenger	0344/31 12
Frau	Sybille	Freuden	0543/323444
Frau	Regine	Olden	05 65/2 32 34
Herr	Olaf	Redlich	0899/6533233
Herr	Uli	Tauf	04 20/2 2 6 4
Frau	Maria	Milder	012312/234434
Herr	Hendrik	Trabisch	0676/74944
Frau	Verena	Blumenfrau	0434/3443444
Herr	Thomas	Ulrich	06 5 7/4 4 3 4 1
Frau	Vera	Fendler	06343/2232323
Herr	Holger	Dellow	0564/4 4 4 4
Herr	Gunter	Thalow	012123/23234

Die fertig formatierte Telefonliste

12.4 Schnellübersicht

Sie möchten ...	
in Word mehrere Seiten auf ein einziges Blatt Papier drucken	`WordObjekt.PrintOut PrintZoomRow:=Zeilen, PrintZoomColumn:=Spalten`
einen eigenen Outlook-Ordner öffnen	`Set StandardOrdner = NamespaceObjekt.GetDefaultFolder(Ordnerkonstante)` `Set MeinOrdner = StandardOrdner.Folders("Ordnername")`
den belegten Teil eines Tabellenblatts ermitteln	`WorkSheetObjekt.UsedRange`
die Anzahl belegter Zeilen bzw. Spalten in einem Tabellenblatt ermitteln	`WorkSheetObjekt.UsedRange.Rows.Count` `WorkSheetObjekt.UsedRange.Columns.Count`

Notizen

12.5 Übung

Übungsdatei: *Übung1.ppt* **Ergebnisdatei:** *Übung1-E.doc*

① Öffnen Sie die Datei *Übung1.ppt*, und starten Sie die VBA-Entwicklungsumgebung. In dieser Übung soll eine integrierte Lösung erstellt werden, die ein bestimmtes Tabellenblatt (oder Diagramm) einer Excel-Datei in PowerPoint importiert.

② Fügen Sie einen neuen UserForm-Dialog ein. Erstellen Sie darin ein Listenfeld und zwei Schaltflächen. Benennen Sie die Steuerelemente wie angegeben.

③ Fügen Sie dem Deklarationsteil des UserForm-Moduls die Deklarationen für ein `Excel.Application`-Objekt und einen Dateinamen (Datentyp `Variant`) ein. Für das Excel-Objekt soll gleichzeitig eine neue Instanz erstellt werden.

④ Erstellen Sie eine Ereignisprozedur für das Aktivieren des UserForm-Dialogs. Öffnen Sie über die `GetOpenFilename`-Methode von Excel einen Dateiauswahldialog für Excel-Dateien.

⑤ Falls eine Datei ausgewählt wurde, soll der Dateiname in der Titelzeile des UserForm-Dialogs angezeigt werden. Füllen Sie danach das Listenfeld mit den Namen der einzelnen Tabellenblätter. Falls keine Datei ausgewählt wurde, schließen Sie die Excel-Anwendung und den UserForm-Dialog.

⑥ Erstellen Sie eine Ereignisprozedur für die Schaltfläche ABBRECHEN, die den UserForm-Dialog entlädt.

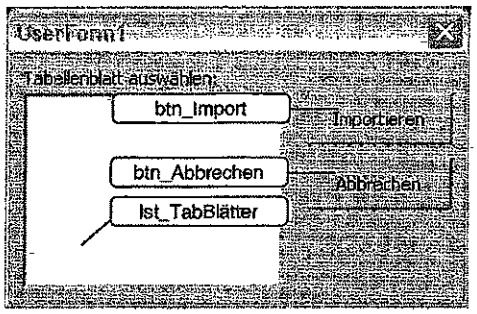
⑦ Nach dem Betätigen der Schaltfläche IMPORTIEREN soll der Import des ausgewählten Tabellenblatts in PowerPoint erfolgen. Prüfen Sie zuerst, ob im Listenfeld ein Wert ausgewählt wurde.

⑧ Aktivieren Sie mithilfe der `Select`-Methode das im Listenfeld ausgewählte Tabellenblatt. Speichern Sie anschließend die Arbeitsmappe, und schließen Sie Arbeitsmappe und Excel-Anwendung.

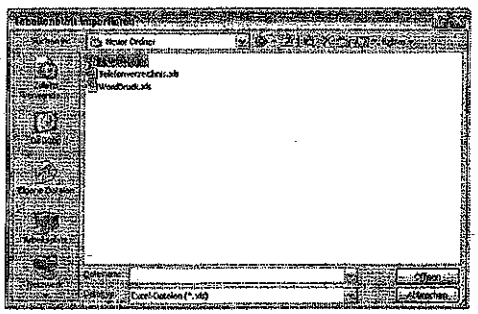
⑨ Fügen Sie mittels der `Slides`-Auflistung hinter der letzten Folie der aktiven Präsentation eine neue leere Folie ein. Über die `Shapes`-Auflistung der neuen Folie erstellen Sie ein OLE-Objekt aus der zu Beginn ausgewählten Datei (`AddOLEObject`-Methode). Das OLE-Objekt soll dabei nicht mit der Datei verknüpft werden.

⑩ PowerPoint soll sofort zur neuen Folie verzweigen. Zeigen Sie die Folie dazu mit der `GoToSlide`-Methode und der Indexnummer (`SlideIndex`-Methode) im aktuellen Fenster an.

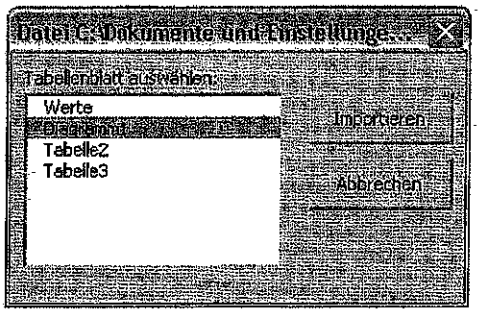
⑪ Erstellen Sie eine Ereignisprozedur, damit beim Doppelklicken auf das Listenfeld die gleiche Aktion ausgeführt wird, wie beim Betätigen der Schaltfläche IMPORTIEREN.



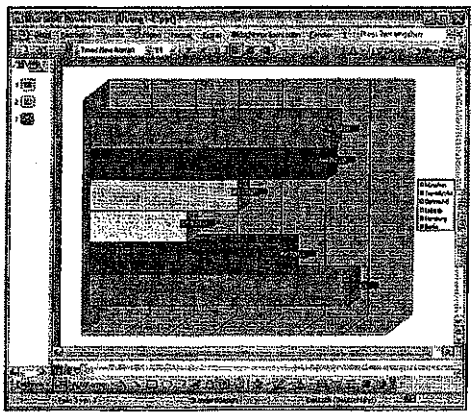
Der UserForm-Dialog im Entwurfsmodus



Der Dateiauswahldialog



Die Namen der Tabellenblätter werden angezeigt



Mithilfe der Anwendung können auch einzelne Diagramme eingefügt werden

Notizen

13 Integrierte Lösungen mit Access

In diesem Kapitel erfahren Sie

→ welche Möglichkeiten es zur Automation mit Access gibt

→ wie Sie eine Dokumentenverwaltung realisieren

Voraussetzungen

✓ Programmieren von Automation-Objekten

✓ Datenaustausch mit ADO-Objekten

✓ Erstellen von Formularen und Tabellen in Access

13.1 Automation mit Access

Mit Access können Sie relationale Datenbanken erstellen und pflegen. Im Gegensatz zu Excel eignet sich Access besonders zum Speichern und Verwalten umfangreicher Datenmengen. Aufgrund des relationalen Datenbankmodells in Access können die Daten in verschiedenen Tabellen gespeichert werden, die über Index-Felder miteinander verbunden sind. Durch die besondere Struktur einer Datenbank können Sie auch mit Tabellen, die viele 100.000 Datensätze umfassen, effektiv arbeiten.

Im Gegensatz zu allen anderen Office-Anwendungen wird Access nur selten als Automation-Server verwendet. Dies liegt daran, dass für den Zugriff auf Access-Datenbanken nicht das Access-Objektmodell, sondern in den meisten Fällen das ADO-Objektmodell verwendet wird. Mit dem ADO-Objektmodell erhalten Sie Zugriff auf die Tabellen und Abfragen einer Datenbank. Über das Access-Objektmodell können Sie dagegen nur auf Formulare und Berichte zugreifen.

Access eignet sich aufgrund seiner umfangreichen Programmiermöglichkeiten sehr gut für das Erstellen komplexer und vollwertiger Datenbankanwendungen, beispielsweise für die Lagerhaltung von Waren. Mithilfe von Automation-Prozeduren können solche Anwendungen noch komfortabler ausgestattet werden. So ist es beispielsweise möglich, nach dem Betätigen einer Schaltfläche einen vorgefertigten Bestellschein in Word zu öffnen oder verschiedene Werte aus der Datenbank in Excel mit einem Diagramm zu analysieren.

Access wird häufig als Automation-Client eingesetzt. In Datenbankanwendungen können dadurch die Vorteile anderer Office-Anwendungen genutzt werden. Der Zugriff auf die in einer Datenbank gespeicherten Werte erfolgt nicht über Access als Automation-Server, sondern über die spezialisierten ADO-Objekte.

Folgende Möglichkeiten ergeben sich für integrierte Lösungen mit Access:

Access als Client mit Excel als Server	☑	Analysieren von Daten mit Excel-Funktionen
	☑	Auslesen von Tabellenwerten und Import in die Datenbank
Access als Client mit Word als Server	☑	Nutzen der Textverarbeitungsfunktionen von Word
	☑	Erstellen von Serienbriefen aus der Datenbank heraus
Access als Server mit einer Office-Anwendung als Client	☑	Auslesen von Daten aus einem Formular
	☑	Öffnen und Drucken von Berichten
Office-Anwendungen mit Datenbankzugriff über ADO-Objekte	☑	Auslesen, Suchen, Verändern von Datensätzen
	☑	Einfügen und Löschen von Datensätzen

In den folgenden Abschnitten werden am Beispiel einer Dokumentenverwaltung die Erstellung von integrierten Lösungen mit Access als Client und der Datenbankzugriff über ADO-Objekte demonstriert.

13.2 Datenbank zur Dokumentenverwaltung

Aufgabe der Datenbankanwendung

Es soll ein einfaches Dokumentenmanagementsystem als Datenbankanwendung entwickelt werden, das es ermöglicht, Word- und Excel-Dokumente zu verwalten und mit Bemerkungen zu versehen. Auf diese Weise bleibt auch bei einem großen Bestand an Dokumenten auf der Festplatte die Übersicht erhalten. Über eine Stichwortsuche innerhalb des Datenbestandes ist eine deutlich schnellere Suche möglich als bei einer Dateisuche auf der Festplatte.

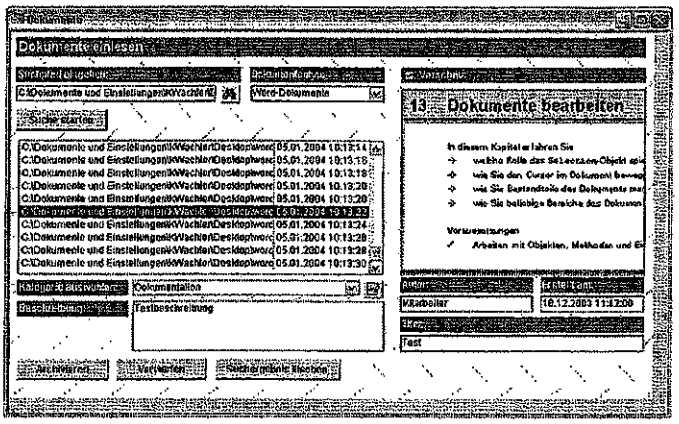
Mithilfe von Automation-Objekten soll beim Einlesen der Dokumente in die Datenbank eine Vorschaumöglichkeit realisiert werden. Die Dokumente sollen im Verwaltungsformular von der Access-Oberfläche aus geöffnet werden können. Der Zugriff auf die Tabellen der Datenbank erfolgt über ADO-Objekte.

Die realisierte Datenbankanwendung finden Sie unter dem Namen *Dokumente.mdb*.

Für die Arbeit mit der Dokumentenverwaltung ist der im Folgenden dargestellte Ablaufplan vorgesehen.

Dokumente einlesen

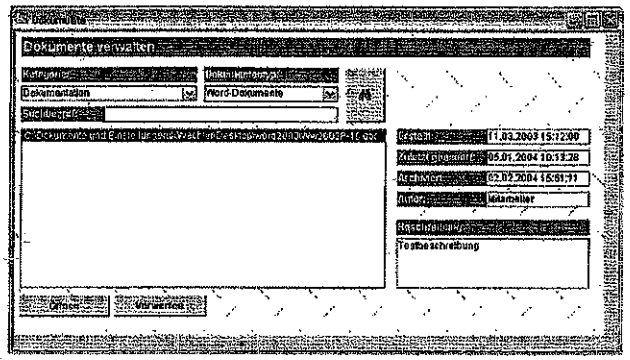
- ☑ Durchsuchen eines benutzerdefinierten Pfads nach Word- oder Excel-Dokumenten und Anzeige der gefundenen Dokumente in einem Listenfeld
- ☑ Begutachtung der gefundenen Dokumente in einer Vorschau
- ☑ Eingabe einer Bemerkung und Zuweisen eines Themengebiets, bei Bedarf Neueingabe eines Themas
- ☑ Auswahl, ob das Dokument archiviert oder verworfen werden soll
- ☑ Speichern der Daten in einer Tabelle



Formular zum Einlesen der Dokumente mit Vorschaufunktion

Dokumente suchen und verwalten

- ☑ Suche von Dokumenten nach Themengebiet und/oder Dokumententyp
- ☑ Suche nach bestimmten Stichwörtern in den Bemerkungen oder Dateinamen
- ☑ Anzeige der in der Datenbank gespeicherten Informationen zu einem gewählten Dokument
- ☑ Öffnen des Dokuments zum Bearbeiten in der zugehörigen Office-Anwendung



Formular zum Verwalten der archivierten Dokumente

Notizen

Bestandteile der Datenbank

Die Datenbank besteht aus mehreren Tabellen, Formularen und einer Abfrage. Alle VBA-Prozeduren sind als Ereignisprozeduren ausgelegt und in den Formularmodulen gespeichert.

Tabellen

Dokumente	In dieser Tabelle werden Dokumentbeschreibungen archiviert. Zu jedem Dokument werden der Pfadname der Datei, Zusatzinformationen (Bemerkung, Autor, Datum) und ein Verweis auf das Themengebiet gespeichert.
GefundeneDokumente	Diese Tabelle wird mit den durch die Suchfunktion gefundenen Dokumentbeschreibungen gefüllt. Diese Dokumente können für eine Archivierung ausgewählt oder verworfen werden. Nach der Archivierung wird der Eintrag in dieser Tabelle gelöscht.
Kategorien	Die Tabelle verwaltet die möglichen Themengebiete, die einem Dokument zugewiesen werden können.

Formulare

DokumenteEinlesen	Dieses Formular ermöglicht die Suche nach Word- oder Excel-Dokumenten auf der Festplatte. Über die Vorschaufunktion können die Dokumente begutachtet und auf Wunsch archiviert werden.
DokumenteVerwalten	Die in der Datenbank archivierten Dokumente sind über dieses Formular zugänglich. Es erlaubt die Suche nach einem bestimmten Dokument und zeigt die dazu in der Datenbank gespeicherten Informationen an.
KategorienBearbeiten	Mit diesem Formular können die vorhandenen Kategorien bearbeitet oder neue hinzugefügt werden.

Abfragen

AusgewähltesDokument	Die Abfrage liefert für das ausgewählte Dokument im Formular DOKUMENTEVERWALTEN den passenden Datensatz in der Tabelle DOKUMENTE.

13.3 Dokumente in die Datenbank einlesen

13.3.1 Das Formular DokumenteEinlesen

Arbeiten mit dem Formular

Für das Einlesen von Dokumenten in die Datenbank ist folgender Arbeitsablauf notwendig:

⇒ Betätigen Sie die Schaltfläche ①, um das Dialogfenster zur Auswahl des Suchpfads zu öffnen. Dieses Verzeichnis und alle untergeordneten Verzeichnisse werden später nach Dokumenten durchsucht. Sie können den Suchpfad auch manuell eingeben.

⇒ Wählen Sie im Listenfeld DOKUMENTENTYP ② einen Dokumententyp aus (Word- oder Excel-Dokumente).

⇒ Starten Sie die Suche durch Betätigen der Schaltfläche SUCHE STARTEN ③. Die gefundenen Dokumente werden im Listenfeld ④ mit ihrem Dateipfad und -datum angezeigt.

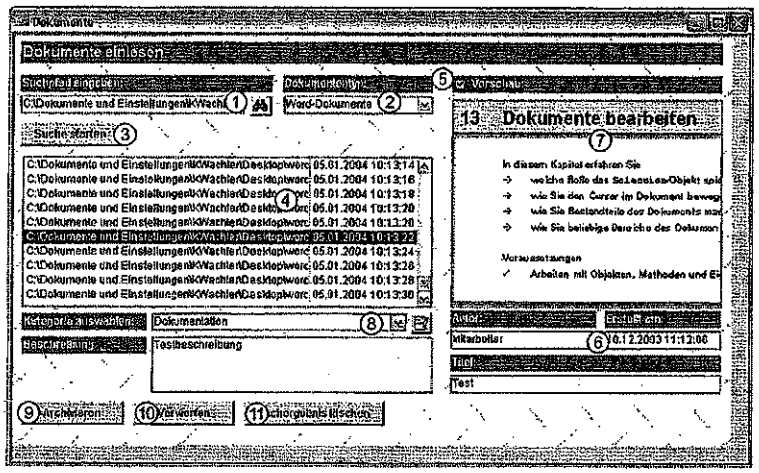
⇒ Aktivieren Sie bei Bedarf das Kontrollfeld VORSCHAU ⑤, um die Vorschaufunktion zu aktivieren.

⇨ Wählen Sie im Listenfeld ④ ein Dokument aus. Sie erhalten danach im Bereich ⑥ weitere Informationen über das Dokument. Bei aktivierter Vorschaufunktion erhalten Sie im Bereich VORSCHAU ⑦ einen Eindruck vom Inhalt des Dokuments.

⇨ Wählen Sie im Listenfeld ⑧ eine geeignete Kategorie aus, und geben Sie einen Beschreibungstext ein.

⇨ Archivieren Sie das Dokument in der Datenbank durch Betätigen der Schaltfläche ARCHIVIEREN ⑨. Um das Dokument aus der Liste der gefundenen Dokumente zu löschen ohne es zu archivieren, betätigen Sie die Schaltfläche VERWERFEN ⑩.

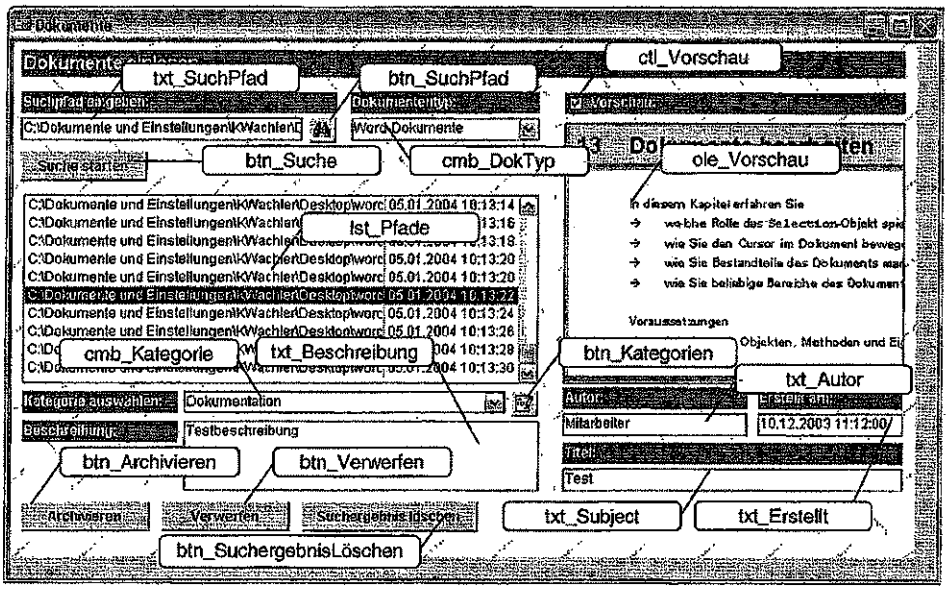
⇨ Mit der Schaltfläche SUCHERGEBNIS LÖSCHEN ⑪ können Sie die Liste der gefundenen Dokumente löschen.



Das Formular DokumenteEinlesen

Steuerelemente und Prozeduren des Formulars

Das Formular enthält Steuerelemente mit der folgenden Benennung:



Notizen

Die folgenden Prozeduren werden im Formularmodul definiert:

btn_Suche_Click	Startet die Suche nach Dokumenten auf der Festplatte
lst_Pfade_Click	Wird ausgeführt, wenn ein Dokument im Listenfeld ausgewählt wird und startet eine Prozedur für die Vorschaufunktion
VorschauWord	Realisiert die Vorschaufunktion für Word-Dokumente und liest Informationen aus dem Dokument
VorschauExcel	Realisiert die Vorschaufunktion für Excel-Dokumente und liest Informationen aus dem Dokument
btn_Archivieren_Click	Archiviert das ausgewählte Dokument in der Datenbank
btn_Verwerfen_Click	Löscht das ausgewählte Dokument aus der Liste der gefundenen Dokumente
Form_Close	Schließt die eventuell im Hintergrund geöffnete Word- oder Excel-Anwendung
btn_Kategorien_Click	Öffnet das Formular KATEGORIENBEARBEITEN
btn_SuchPfad	Öffnet einen Standarddialog zur Verzeichnisauswahl
btn_Suchergebnisloeschen	Leert die Tabelle mit den gefundenen Dokumenten

13.3.2 Dokumente suchen

Die Dokumentensuche wird innerhalb der Ereignisprozedur `btn_Suche_Click` ausgeführt. Dafür findet das `FileSearch`-Objekt, das in allen Office-Anwendungen verfügbar ist, Verwendung.

Die Prozedur wird nach Betätigen der Schaltfläche SUCHE STARTEN ausgeführt. Sie erwartet einen Suchpfad und einen Dokumententyp. Alle gefundenen Dokumente werden der Tabelle `GefundeneDokumente` hinzugefügt. Die Tabelle dient als temporärer Speicher, bevor die Dokumente in der Datenbank archiviert werden.

```
Private Sub btn_Suche_Click()
  Dim DokTyp As Long
①  Dim ADOTab As New ADODB.Recordset
②  If Me.cmb_DokTyp = "Word-Dokumente" Then
    DokTyp = msoFileTypeWordDocuments
  ElseIf Me.cmb_DokTyp = "Excel-Dokumente" Then
    DokTyp = msoFileTypeExcelWorkbooks
  Else
    MsgBox "Bitte wählen Sie einen Dokumenten-Typ aus!", vbExclamation
    Exit Sub
  End If
③  With Application.FileSearch
④    .NewSearch
⑤    .LookIn = Me.txt_SuchPfad
    .SearchSubFolders = True
⑥    .FileType = DokTyp
⑦    If .Execute() > 0 Then
⑧      MsgBox .FoundFiles.Count & " Dokumente gefunden.", vbInformation
⑨      ADOTab.Open "GefundeneDokumente", CurrentProject.Connection, _
                   adOpenDynamic, adLockOptimistic
⑩      For i = 1 To .FoundFiles.Count
        ADOTab.AddNew
        ADOTab!Pfad = .FoundFiles(i)
⑪        ADOTab!Datum = FileSystem.FileDateTime(.FoundFiles(i))
        ADOTab.Update
```

```
        Next
        ADOTab.Close
      Else
⑫       MsgBox "Keine Dateien gefunden.", vbExclamation + vbOKOnly
      End If
    End With
    Me.lst_Pfade.Requery
End Sub
```

① Für den Datenzugriff über ADO-Objekte wird eine Objektvariable `ADOTab` deklariert, die einen Verweis auf eine Datensatzgruppe (Recordset) aufnehmen kann.

② In der `If`-Anweisung wird geprüft, welcher Dokumententyp ausgewählt wurde. Nach dieser Auswahl wird die Variable `DokTyp` für die spätere Suche mit einer Konstante belegt. Falls kein Typ ausgewählt wurde, zeigt die Prozedur ein Meldungsfenster an und bricht die Ausführung ab.

③ Die `With`-Anweisung leitet einen Anweisungsblock für das `FileSearch`-Objekt ein, der mit der Anweisung `End With` abgeschlossen wird. Die Methoden und Eigenschaften können in diesem Block vereinfacht angegeben werden.

④ Die Methode `NewSearch` leitet eine neue Dateisuche ein. Unterverzeichnisse werden mit der Eigenschaft `SearchSubFolders` eingeschlossen.

⑤ Der Suchpfad wird aus dem Textfeld ausgelesen und der Eigenschaft `LookIn` zugewiesen. Der Suchpfad muss ein gültiges Verzeichnis darstellen, in dem die Suche beginnt.

⑥ Die Eigenschaft `FileTyp` bestimmt den Dokumententyp, nach dem gesucht wird. Hier wird die bereits unter ② definierte Variable zugewiesen.

⑦ Die Methode `Execute` startet die Suche und liefert einen Wert größer Null, wenn die Suche erfolgreich war. Die gefundenen Dateinamen werden im Unterobjekt `FoundFiles` alphabetisch nach Namen geordnet gespeichert.

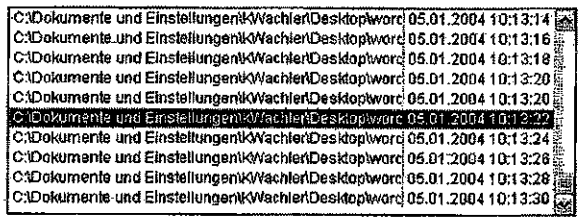
⑧ In einem Meldungsfenster wird mithilfe der `Count`-Eigenschaft die Anzahl der gefundenen Dateien angezeigt.

⑨ Um die gefundenen Dateinamen auf einfache Weise zu verwalten und im Listenfeld `lst_Pfade` anzuzeigen, werden sie temporär in der Tabelle GEFUNDENEDOKUMENTE gespeichert. Dazu wird an dieser Stelle mit der `Open`-Methode des `Recordset`-Objekts eine Datensatzgruppe für diese Tabelle geöffnet. Mit der Objektangabe `CurrentProject.Connection` wird auf die aktuelle ADO-Verbindung der Datenbank verwiesen.

⑩ Nacheinander werden die gefundenen Dateien durchlaufen. Für jede Datei wird mit der `AddNew`-Methode ein neuer Datensatz erzeugt.

⑪ Das `FileSystem`-Objekt von VBA erlaubt den Zugriff auf das Dateisystem. Die Methode `FileDateTime` bestimmt das Dateidatum, das für jede Datei in der Tabelle gespeichert wird.

⑫ Damit die gefunden Dokumente im Listenfeld `lst_Pfade` angezeigt werden, muss es mithilfe der `Requery`-Methode aktualisiert werden.



Das Listenfeld mit den gefundenen Dokumenten

Notizen

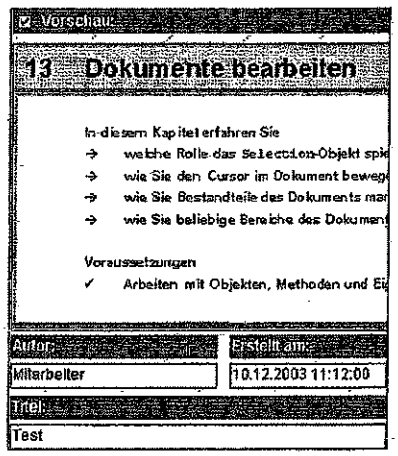
Für den Datenzugriff mit ADO-Objekten unter Access ist kein Verbindungsaufbau mit dem ADO-Connection-Objekt notwendig. Access besitzt bereits eine geöffnete Verbindung, die mit der Objektanweisung `Current Project.Connection` zurückgeliefert wird. Sie können daher sofort mit dem `Recordset`-Objekt eine Datenquelle öffnen.

13.3.3 Vorschaufunktion für Word-Dokumente

Die Vorschau wird über ein gebundenes OLE-Objektfeld als Steuerelemente im Access-Formular realisiert. Die VBA-Prozedur `VorschauWord` öffnet Word im Hintergrund und kopiert den Haupttext des ausgewählten Word-Dokuments in die Zwischenablage. Danach wird er in das OLE-Objektfeld eingefügt, das den Text originalgetreu darstellt.

Gleichzeitig liest die Prozedur verschiedene Informationen aus dem Word-Dokument, beispielsweise den Autor und das Erstellungsdatum.

Da jedes Dokument im Hintergrund in Word geöffnet werden muss, ist die Prozedur sehr zeitaufwändig. Um die Zeit für das Starten der Word-Anwendung gering zu halten, wird Word nur einmal gestartet und später auf die geladene Anwendung zugegriffen. Über ein Kontrollfeld ist außerdem wählbar, ob eine Vorschau angezeigt werden soll. Auch dies kann eine schnellere Ausführung erwirken.



Ein Word-Dokument in der Vorschau

```
Sub VorschauWord(Pfad As String)
  On Error GoTo Fehlerbehandlung
  Dim AppWD As Word.Application, DocWD As Word.Document
  Set AppWD = GetObject(, "Word.Application")
  Set DocWD = AppWD.Documents.Open(Pfad)
  If Me.ctl_Vorschau Then
    DocWD.StoryRanges(wdMainTextStory).Copy
    Me.ole_Vorschau.Action = acOLEPaste
    Me.ole_Vorschau.SizeMode = acOLESizeClip
  End If
  Me.txt_Autor = DocWD.BuiltInDocumentProperties(wdPropertyAuthor)
  Me.txt_Erstellt = DocWD.BuiltInDocumentProperties(wdPropertyTimeCreated)
  Me.txt_Titel = DocWD.BuiltInDocumentProperties(wdPropertyTitle)
  AppWD.DisplayAlerts = wdAlertsNone
  DocWD.Close False
  AppWD.DisplayAlerts = wdAlertsAll
  Exit Sub
Fehlerbehandlung:
  If Err = 429 Then
    Set AppWD = CreateObject("Word.Application")
    Resume Next
  End If
End Sub
```

① Die Prozedur versucht, auf eine vorhandene Word-Instanz zuzugreifen. Falls Word noch nicht geladen wurde, wird dabei ein Fehler erzeugt. Es wird daher vereinbart, dass im Fehlerfall zur Sprungmarke `Fehlerbehandlung` verzweigt wird.

② Mithilfe der `GetObject`-Funktion wird an dieser Stelle versucht, auf eine bereits geladene Word-Anwendung zuzugreifen. Im Erfolgsfall wird `AppWD` ein Verweis auf das `Application`-Objekt zugewiesen. Ist Word nicht geladen, erzeugt VBA einen Fehler und verzweigt zur Fehlerbehandlung.

③ Der Dateipfad zum gewünschten Word-Dokument wurde als Argument übergeben. Die Datei wird hier in Word geöffnet. Die Methode liefert einen Verweis auf das neue `Document`-Objekt zurück.

④ Der Zugriff auf das gerade geöffnete Dokument erfolgt über die Variable `DocWD`. Über das `StoryRange`-Objekt wird der Haupttext des Dokuments in die Zwischenablage kopiert. Dieses Vorgehen ist schneller als das Kopieren des gesamten Dokuments.

⑤ Das OLE-Objektfeld verfügt über die Eigenschaft `Action`, die eine auszuführende Aktion bestimmt. Durch das Zuweisen der vordefinierten Konstante `acOLEPaste` wird der Inhalt der Zwischenablage in das Feld eingefügt. Mit der Eigenschaft `SizeMode` wird eingestellt, dass der Inhalt für eine schnellere Anzeige nicht skaliert, sondern abgeschnitten wird.

⑥ Jedes Dokument verfügt über bestimmte Eigenschaften, die mit der Methode `BuiltInDocumentProperties` angesprochen werden können. Für alle Eigenschaften existieren in Word vordefinierte Konstanten. Die ausgelesenen Eigenschaften werden in den betreffenden Textfeldern angezeigt.

⑦ Das Dokument kann nun wieder geschlossen werden. Zuvor wird mithilfe der Eigenschaft `DisplayAlerts` eine mögliche Meldung über umfangreiche Daten in der Zwischenablage unterdrückt. Durch den Parameter `False` wird eine Aufforderung zum Speichern von Änderungen unterdrückt.

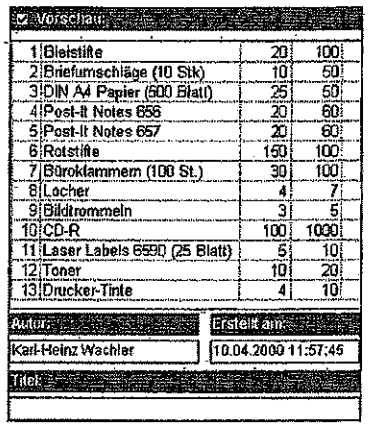
⑧ Kann die Funktion `GetObject` keine geladene Word-Anwendung finden, erzeugt VBA den Fehler 429. In diesem Fall muss die Anwendung mithilfe von `CreateObject` erstmalig geladen werden. Danach wird die Programmausführung bei der nächsten Anweisung fortgesetzt, die auf die Fehler verursachende Anweisung folgt.

13.3.4 Vorschaufunktion für Excel-Dokumente

Auch für Excel-Dokumente wird die Vorschau über das gleiche gebundene OLE-Objektfeld realisiert. Die VBA-Prozedur `VorschauExcel` öffnet Excel im Hintergrund und kopiert den mit Werten belegten Bereich des aktiven Tabellenblatts in die Zwischenablage. Danach wird er in das OLE-Objektfeld eingefügt, das die Tabelle originalgetreu darstellt.

Gleichzeitig liest die Prozedur verschiedene Informationen aus der Excel-Arbeitsmappe, beispielsweise den Autor und das Erstellungsdatum.

Die Prozedur folgt im Wesentlichen dem Aufbau von `VorschauWord`.



Vorschau eines Excel-Dokuments

```
Sub VorschauExcel(Pfad As String)
  On Error GoTo Fehlerbehandlung
  Dim AppXl As Excel.Application, WBXL As Excel.Workbook, _
      SheetXL As Excel.Worksheet
①  Set AppXl = GetObject(, "Excel.Application")
②  Set WBXL = AppXl.Workbooks.Open(Pfad)
③  Set SheetXL = AppXl.ActiveSheet
  If Me.ctl_Vorschau Then
④   SheetXL.UsedRange.Copy
    Me.ole_Vorschau.Action = acOLEPaste
    Me.ole_Vorschau.SizeMode = acOLESizeClip
  End If
⑤  Me.txt_Autor = WBXL.BuiltInDocumentProperties(3)
  Me.txt_Erstellt = WBXL.BuiltInDocumentProperties(11)
  Me.txt_Titel = WBXL.BuiltInDocumentProperties(1)
  AppXl.DisplayAlerts = False
⑥  WBXL.Close False
  AppXl.DisplayAlerts = True
  Exit Sub
Fehlerbehandlung:
  If Err = 429 Then
    Set AppXl = CreateObject("Excel.Application")
    Resume Next
  End If
End Sub
```

① Falls möglich, wird auch in dieser Funktion versucht, auf eine vorhandene Excel-Instanz zuzugreifen. Ist Excel noch nicht gestartet, wird es nach Auswertung des Fehlers neu gestartet.

② Der als Argument übergebene Dateipfad wird als neue Arbeitsmappe geöffnet und ein Verweis auf das Workbook-Objekt gespeichert.

③ Die Objektvariable `SheetXL` erhält für einen schnellen Zugriff einen Verweis auf das aktive Tabellenblatt.

④ Der mit Werten belegte Bereich des Tabellenblatts wird über das `UsedRange`-Objekt in die Zwischenablage kopiert. Dabei bleiben alle Formatierungen erhalten.

⑤ Ähnlich den Word-Dokumenten enthalten auch Excel-Arbeitsmappen zusätzliche Informationen als Dokumenteigenschaften, z. B. zum Autor. Excel verfügt jedoch nicht über vordefinierte Konstanten, wie sie in Word zur Verfügung stehen. Die einzelnen Informationen werden daher über feste Zahlenwerte angesprochen und ausgelesen.

⑥ Die Arbeitsmappe wird am Ende der Prozedur geschlossen. Mögliche Meldungsfenster werden durch die `DisplayAlerts`-Eigenschaft unterdrückt.

Im Gegensatz zu Word verfügt Excel nicht über vordefinierte Konstanten für das Auslesen von Informationen einer Arbeitsmappe. Die Objektauflistung `BuiltInDocumentProperties` erwartet jedoch in beiden Anwendungen die gleichen Zahlenwerte als Index der betreffenden Eigenschaft. Sie können daher über die Word-Konstante den Zahlenwert für die gewünschte Information bestimmen, beispielsweise besitzt die Konstante `wdPropertyAuthor` für den Autor des Dokuments den Wert 3.

13.3.5 Dokumente in der Datenbank archivieren

Beim Betätigen der Schaltfläche ARCHIVIEREN werden alle Informationen des im Listenfeld ausgewählten Dokuments in der Tabelle DOKUMENTE gespeichert. Anschließend wird der betreffende Eintrag in der Tabelle GEFUNDENEDOKUMENTE gelöscht, und das Listenfeld wird aktualisiert.

Die Prozedur `btn_Archivieren_Click` ist eine Ereignisprozedur, die beim Betätigen der Schaltfläche ARCHIVIEREN aufgerufen wird. Bevor ein Dokument in der Datenbank gespeichert wird, prüft die Prozedur, ob eine Kategorie ausgewählt und eine Beschreibung eingegeben wurde.

Der Zugriff auf die Tabellen erfolgt durch ADO-Objekte:

```
With ADOTab
①  .Open "Dokumente", CurrentProject.Connection, adOpenForwardOnly, _
                                                  adLockOptimistic
②  .AddNew
③  !Pfad = Me.lst_Pfade.Column(1)
   !DatumZugriff = Me.lst_Pfade.Column(2)
④  !Beschreibung = Me.txt_Beschreibung
   !Autor = Me.txt_Autor
   !ThemaID = Me.cmb_Kategorie
   !DatumErstellt = Me.txt_Erstellt
⑤  !DatumArchiv = Now
⑥  .Update
   .Close
⑦  .Open "GefundeneDokumente", CurrentProject.Connection, adOpenForwardOnly, _
         adLockOptimistic
⑧  .Find "ID=" & Me.lst_Pfade
⑨  .Delete
   .Close
End With
```

① `ADOTab` ist eine Objektvariable vom Typ `ADODB.Recordset`. Mithilfe der `Open`-Methode dieses `Recordset`-Objekts wird die Tabelle DOKUMENTE als Datensatzgruppe geöffnet. Da keine Bewegung des Datensatzzeigers notwendig ist, wird die Datensatzgruppe mit der Konstante `adOpenForwardOnly` geöffnet.

② Die `AddNew`-Methode erzeugt einen neuen leeren Datensatz.

③ Aus dem Listenfeld `lst_Pfade` werden der Dokumentenpfad und das Dateidatum ausgelesen. Die `Column`-Methode erlaubt dabei den Zugriff auf bestimmte Spalten des Steuerelements.

④ Die Dokumenteninformationen werden in der Datenbank gespeichert.

⑤ Die VBA-Funktion Now liefert das aktuelle Datum, das als Archivierungsdatum für das Dokument in der Datenbank gespeichert wird.

⑥ Mit dem Aufruf der Update-Methode wird der neue Datensatz in der Tabelle gespeichert. Danach wird die Datensatzgruppe geschlossen.

⑦ Um den Datensatz aus der Tabelle GEFUNDENEDOKUMENTE zu löschen, wird diese Tabelle als Datensatzgruppe geöffnet.

⑧ Das Listenfeld lst_Pfade enthält als versteckte Spalte den ID-Wert des Dokuments. Mit der Find-Methode wird der Datensatzzeiger auf diesen Datensatz positioniert.

⑨ Der Aufruf von Delete löscht den aktuellen Datensatz.

Ist bei einer Datensatzgruppe kein freies Bewegen des Datensatzzeigers in alle Richtungen notwendig, beispielsweise, da Sie einen Datensatz hinzufügen oder die Datensätze nur sequenziell lesen wollen, können Sie als Argument der Open-Methode die Konstante adOpenForwardOnly angeben. Besonders bei umfangreichen Tabellen erreichen Sie dadurch einen schnelleren Zugriff.

13.3.6 Liste der gefundenen Dokumente löschen

Die Dateipfade der gefundenen Dokumente werden in der Tabelle GEFUNDENEDOKUMENTE gespeichert. Bei der Suche im angegebenen Suchpfad kann eine große Anzahl Dokumente gefunden werden. Falls jedoch nur einige wenige davon archiviert werden sollen, ist es uneffektiv, jeden einzelnen Eintrag zu verwerfen.

Mit der Ereignisprozedur für die Schaltfläche SUCHERGEBNIS LÖSCHEN werden alle Datensätze der Tabelle GEFUNDENEDOKUMENTE gelöscht, und die Liste der gefundenen Dokumente wird geleert.

```
Private Sub btn_SuchergebnisLöschen_Click()
①   Dim ADOCnn As ADODB.Connection
②   Set ADOCnn = CurrentProject.Connection
③   ADOCnn.Execute "DELETE FROM GefundeneDokumente"
    ADOCnn.Close
④   Me.lst_Pfade.Requery
    Me.lst_Pfade.SetFocus
End Sub
```

① Um auf die Tabelle zugreifen zu können, wird in diesem Fall ein ADO-Connection-Objekt benötigt.

② Den Verweis auf eine gültige ADO-Verbindung zur aktuellen Datenbank liefert das CurrentProject-Objekt mit der Eigenschaft Connection.

③ Statt alle Datensätze einzeln zu löschen, kann dies mit einer einzigen SQL-Anweisung ausgeführt werden.

④ Mithilfe der Requery-Methode wird das Listenfeld zur Anzeige der gefundenen Dokumente neu initialisiert.

Notizen

13.3.7 Versteckt geöffnete Word- bzw. Excel-Anwendung schließen

Für einen schnelleren Zugriff bei der Vorschaufunktion werden die Anwendungen Word und Excel nicht geschlossen. Da die beiden Anwendungen jedoch viele Ressourcen benötigen, sollten Sie beim Schließen des Formulars beendet werden. Der belegte Speicherplatz wird dabei freigegeben und steht anderen Anwendungen zur Verfügung.

Die Ereignisprozedur `Form_Close` wird beim Schließen des Formulars ausgeführt und schließt die beiden Anwendungen, falls sie versteckt geöffnet wurden.

Word oder Excel können bereits vor der Datenbankanwendung absichtsvoll geöffnet worden sein. In diesem Fall ist die Anwendung sichtbar. Die Prozedur kann dies prüfen und verhindert damit das versehentliche Schließen einer Anwendung, die noch benötigt wird.

```
Private Sub Form_Close()
①   On Error Resume Next
    Dim AppWD As Word.Application, AppXl As Excel.Application
②   Set AppWD = GetObject(, "Word.Application")
③   If AppWD.Visible = False Then
④       AppWD.DisplayAlerts = wdAlertsNone
        AppWD.Quit SaveChanges:=False
        AppWD.DisplayAlerts = wdAlertsAll
⑤       Set AppWD = Nothing
    End If
⑥   Set AppXl = GetObject(, "Excel.Application")
    If AppXl.Visible = False Then
        AppXl.DisplayAlerts = False
        AppXl.Quit
        AppXl.DisplayAlerts = True
        Set AppXl = Nothing
    End If
End Sub
```

① Falls keine Anwendung geladen wurde, würde beim Aufruf der Funktion `GetObject` eine Fehlermeldung erzeugt werden. Durch diese Zeile wird die Fehlermeldung unterdrückt und die nächste Anweisung ausgeführt.

② Hier wird versucht, einen Verweis auf eine vorhandene Word-Instanz zu ermitteln.

③ Mithilfe der `Visible`-Eigenschaft kann geprüft werden, ob die Word-Anwendung sichtbar ist. Falls nicht, wird davon ausgegangen, dass sie durch die Datenbank gestartet wurde und nicht mehr benötigt wird.

④ Die Anwendung wird geschlossen. Mögliche Aufforderungen zum Speichern von Änderungen werden durch das Argument `SaveChanges:=False` und die Eigenschaft `DisplayAlerts` unterdrückt.

⑤ Der durch die Objektvariable belegte Speicher wird freigegeben.

⑥ Die gleichen Schritte sind beim Schließen der Excel-Anwendung notwendig.

Die `Application`-Methode `Quit` kennt in Excel im Gegensatz zu Word kein Argument zum Unterdrücken der Warnungen bei noch nicht gespeicherten Dokumenten. Mit der Eigenschaft `DisplayAlerts` kann dieses Verhalten jedoch auch in Excel erreicht werden. Beachten Sie dabei die unterschiedliche Handhabung der `DisplayAlerts`-Eigenschaft bei Word und Excel.

13.4 Dokumente verwalten und öffnen

13.4.1 Das Formular DokumenteVerwalten

Arbeiten mit dem Formular

Nachdem Sie Dokumente in die Datenbank eingelesen haben, stehen deren Daten für eine Datei- oder Stichwortsuche zur Verfügung. Da die Dokumente nicht auf der Festplatte gesucht werden müssen, erhalten Sie das Suchergebnis deutlich schneller.

Folgender Arbeitsablauf ergibt sich für das Formular DOKUMENTEVERWALTEN:

⇒ Wählen Sie im Listenfeld KATEGORIE ① die gewünschte Kategorie aus, oder lassen Sie das Feld leer.

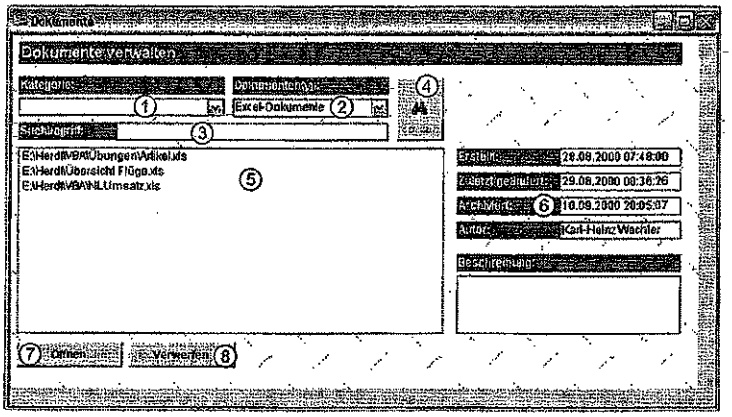
⇒ Wählen Sie im Listenfeld DOKUMENTENTYP ② einen Dokumententyp, oder lassen Sie das Feld leer.

⇒ Geben Sie im Textfeld SUCHBEGRIFF ③ einen Begriff an, nach dem im Dateinamen und Beschreibungstext gesucht werden soll.

⇒ Betätigen Sie die Schaltfläche ④, um die Suche zu starten. Die gefundenen Dokumente werden im Listenfeld ⑤ angezeigt.

⇒ Klicken Sie auf einen Dokumentenpfad im Listenfeld ⑤, um im Bereich ⑥ nähere Informationen zu sehen.

⇒ Betätigen Sie die Schaltfläche ÖFFNEN ⑦, um das ausgewählte Dokument in der zugehörigen Office-Anwendung zu öffnen. Mit der Schaltfläche VERWERFEN ⑧ löschen Sie das Dokument aus der Datenbank.

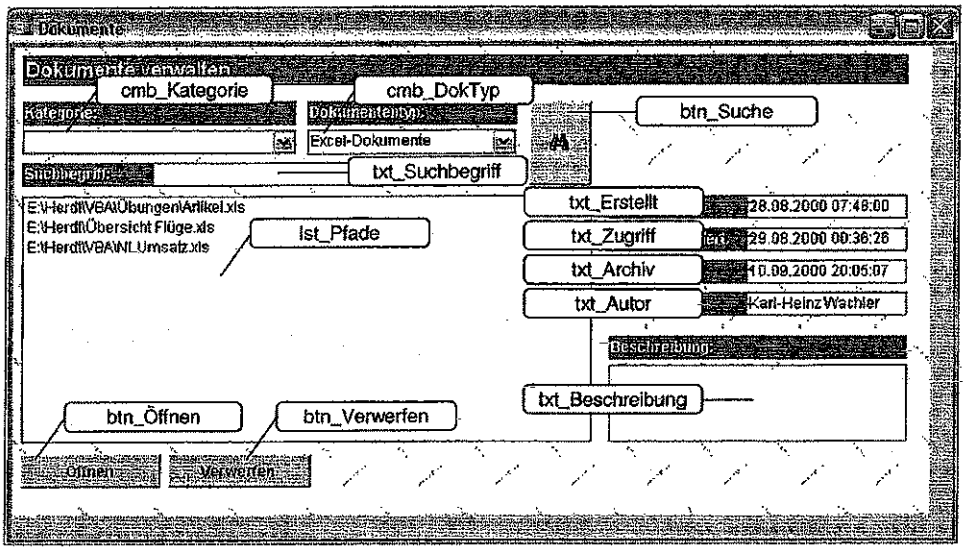


Mit diesem Formular werden die Dokumente verwaltet

Notizen

Steuerelemente und Prozeduren des Formulars

Das Formular enthält Steuerelemente mit der folgenden Benennung:



Die gebundenen Steuerelemente des Formulars, z. B. `txt_Erstellt` und `txt_Autor`, beziehen ihre Daten aus der Abfrage AUSGEWÄHLTESDOKUMENT, die als Datensatzherkunft für das Formular eingetragen ist. Die Abfrage liefert je nach Auswahl im Listenfeld `lst_Pfade` den gewünschten Datensatz in der Tabelle DOKUMENTE.

Die folgenden Ereignisprozeduren werden im Formularmodul definiert:

`btn_Suche_Click`	Die Prozedur realisiert die Suchfunktion nach Dokumenten in der Datenbank.
`btn_Öffnen_Click`	Öffnet das ausgewählte Dokument in der jeweiligen Office-Anwendung
`btn_Verwerfen_Click`	Löscht das ausgewählte Dokument aus der Datenbank
`lst_Pfade_Click`	Aktualisiert das Formular zur Anzeige der Dokumenteninformationen

13.4.2 Dokumente in der Datenbank suchen

Die Prozedur `btn_Suche_Click` wertet die Eingaben in den Steuerelementen `cmb_Kategorie`, `cmb_DokTyp` und `txt_Suchbegriff` aus und erstellt daraus eine SQL-Anweisung, die alle passenden Datensätze in der Tabelle DOKUMENTE zurückliefert.

SQL (Structured Query Language) ist eine standardisierte Sprache für Datenabfragen aus einer Datenbank. Access unterstützt einen großen Teil der sehr umfangreichen Sprache. SQL-Anweisungen sind Abfragen in Access sehr ähnlich. Mit einer SQL-Anweisung können Sie alle Datensätze aus einer Tabelle filtern, auf die ein bestimmtes Suchkriterium passt.

Syntax einer SQL-Anweisung zur Datenabfrage

Eine einfache SQL-Anweisung zum Abfragen von Daten aus einer Tabelle wird nach folgender Syntax gebildet:

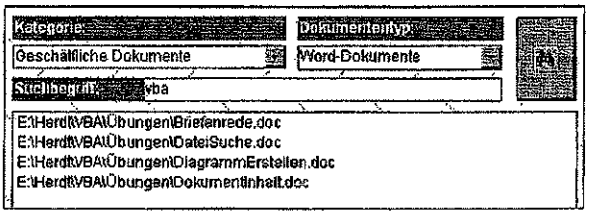
```
SELECT Datenfelder FROM Tabelle WHERE Suchkriterium
```

- ☑ Das Schlüsselwort `SELECT` leitet eine SQL-Anweisung ein. Danach folgen durch Kommata getrennt die Namen der Datenfelder, die aus der Tabelle gelesen werden sollen. Durch die Angabe eines Sterns (*) werden alle Datenfelder zurückgeliefert.
- ☑ Nach dem Schlüsselwort `FROM` folgt der Name der Tabelle.
- ☑ Mit `WHERE` wird das Suchkriterium eingeleitet. SQL unterstützt viele Suchoperatoren, z. B. `>`, `=` oder `Like`. Mehrere Kriterien können über logische Operatoren zusammengefasst werden, z. B. `AND`, `OR`.

Mehr Informationen zu SQL finden Sie in der Online-Hilfe zu Access unter dem Stichwort *Microsoft Jet SQL-Sprachverzeichnis*.

```
Private Sub btn_Suche_Click()
  Dim SQL As String
①  SQL = "SELECT ID,Pfad FROM Dokumente Where (Beschreibung LIKE '*" _
     & Me.txt_Suchbegriff & "*' OR Pfad LIKE '*" & Me.txt_Suchbegriff & "*')"
②  If Not IsNull(Me.cmb_Kategorie) Then
     SQL = SQL & " AND ThemaID = " & Me.cmb_Kategorie
   End If
③  If Not IsNull(Me.cmb_DokTyp) Then
     If Me.cmb_DokTyp = "Word-Dokumente" Then
       SQL = SQL & " AND Pfad Like '*.doc'"
     ElseIf Me.cmb_DokTyp = "Excel-Dokumente" Then
       SQL = SQL & " AND Pfad Like '*.xls'"
     End If
   End If
④  Me.lst_Pfade.RowSource = SQL
   Me.lst_Pfade.Requery
End Sub
```

① Der Beginn der SQL-Anweisung bleibt, unabhängig von den gewählten Suchkriterien, gleich. Hier wird festgelegt, dass die Datenfelder `ID` und `Pfad` aus der Tabelle DOKUMENTE gelesen werden sollen (`SELECT ID,Pfad`). Das Suchkriterium wird mit `Where` eingeleitet. Der gewünschte Suchbegriff wird über den SQL-Operator `Like` in das Suchkriterium eingebunden.



Die gefundenen Dokumente werden im Listenfeld angezeigt

② Die Funktion `IsNull` prüft, ob im Listenfeld `cmb_Kategorie` ein Wert ausgewählt wurde. Ist dies der Fall, wird er in das Suchkriterium eingebaut.

③ Auch für das Listenfeld `cmb_DokTyp` wird geprüft, ob ein Wert ausgewählt wurde. Der Dokumententyp wird in der SQL-Anweisung anhand der Dateierweiterung bestimmt.

④ Die fertige SQL-Anweisung wird an dieser Stelle im Listenfeld `lst_Pfade` als Datensatzherkunft eingetragen. Danach wird das Listenfeld mithilfe der `Requery`-Methode aktualisiert. Erst jetzt wird die eigentliche Suche ausgeführt. Das Listenfeld enthält danach die gefundenen Datensätze.

Da Access SQL-Anweisungen unterstützt, werden in diesem Fall keine ADO-Objekte benötigt. Die Programmierung ist dadurch wesentlich einfacher.

Notizen

13.4.3 Dokumente öffnen

Beim Betätigen der Schaltfläche ÖFFNEN soll das ausgewählte Dokument in der zugehörigen Office-Anwendung geöffnet werden. Die Anwendung wird dabei nicht versteckt im Hintergrund gestartet, sondern so, dass der Anwender mit ihr arbeiten kann.

```
Private Sub btn_Öffnen_Click()
  Dim App As Object, Pfad As String
  If Not IsNull(Me.lst_Pfade) Then
    Pfad = Me.lst_Pfade.Column(1)
    If InStrRev(Pfad, ".doc") > 0 Then
      Set App = CreateObject("Word.Application")
      App.Documents.Open Me.lst_Pfade.Column(1)
    ElseIf InStrRev(Pfad, ".xls") > 0 Then
      Set App = CreateObject("Excel.Application")
      App.Workbooks.Open Me.lst_Pfade.Column(1)
    End If
    App.Visible = True
    App.Activate
  End If
End Sub
```

① Da sich erst im Verlauf der Prozedur entscheidet, ob Word oder Excel benötigt wird, erfolgt die Deklaration der für die Automation zuständigen Objektvariablen mit später Bindung. Ihr wird daher der Typ `Object` zugewiesen.

② Die `IsNull`-Funktion überprüft, ob im Listenfeld `lst_Pfade` ein Dokument ausgewählt wurde. Nur dann werden die folgenden Anweisungen ausgeführt.

③ Der Dokumentenpfad wird aus dem Listenfeld gelesen.

④ Mithilfe der `InStrRev`-Funktion wird geprüft, ob das Dokument die Dateierweiterung **.doc* besitzt. Ist dies der Fall, handelt es sich um ein Word-Dokument.

⑤ Die `CreateObject`-Funktion startet Word und weist der Variablen `App` einen Verweis auf die Programminstanz zu. Erst zu diesem Zeitpunkt wird der Objekttyp der Variablen `App` festgelegt.

⑥ Falls es sich nicht um ein Word-Dokument handelt, wird ebenfalls mit der `InStr`-Funktion auf die Dateierweiterung **.xls* für Excel-Arbeitsmappen getestet.

⑦ Auch Excel wird mithilfe der Funktion `CreateObject` gestartet. Da die Variable `App` noch keinen festgelegten Objekttyp besitzt, kann sie auch einen Verweis auf die Excel-Programminstanz erhalten.

⑧ Die Office-Anwendung soll nicht unsichtbar im Hintergrund bleiben. Daher wird sie über die `Visible`-Eigenschaft sichtbar gemacht.

⑨ Damit das Anwendungsfenster aktiv im Vordergrund zu sehen ist, wird es mit der Methode `Activate` aktiviert.

Office-Anwendungen, die über Automation-Objekte gestartet werden, sind standardmäßig nicht sichtbar. Mithilfe der Eigenschaft `Visible` des `Application`-Objekts können Sie die Anwendung jedoch jederzeit sichtbar und auf Wunsch wieder unsichtbar machen.

Falls Sie in einer integrierten Anwendung nacheinander auf mehrere verschiedene Office-Anwendungen zugreifen müssen oder wenn sich erst im Verlauf der Ausführung entscheidet, welche Anwendung Sie benötigen, können Sie die späte Bindung für Objektvariablen verwenden. In diesem Fall deklarieren Sie die Objektvariable mit dem Typ `Object`.

13.5 Schnellübersicht

Sie möchten ...	
nach Dateien suchen	`Application.FileSearch`
das Datum einer Datei bestimmen	`FileSystem.FileDateTime(`*`Dateiname`*`)`
den Inhalt der Zwischenablage in ein OLE-Feld einfügen	`Formularname.OLEFeldName.Action = acOLEPaste`
den Anzeigemodus für ein OLE-Feld bestimmen	`Formularname.OLEFeldName.SizeMode = acOLESizeKonstante`
Informationen eines Word- oder Excel-Dokuments auslesen	`Objektname.BuiltInDocumentProperties(`*`Nummer`*`)`
eine Anwendung im Hintergrund sichtbar machen	`ApplicationObjekt.Visible=True`

13.6 Übung

Übung 1

Übungsdatei: *Übung1.mdb* **Ergebnisdatei:** *Übung1-E.mdb*

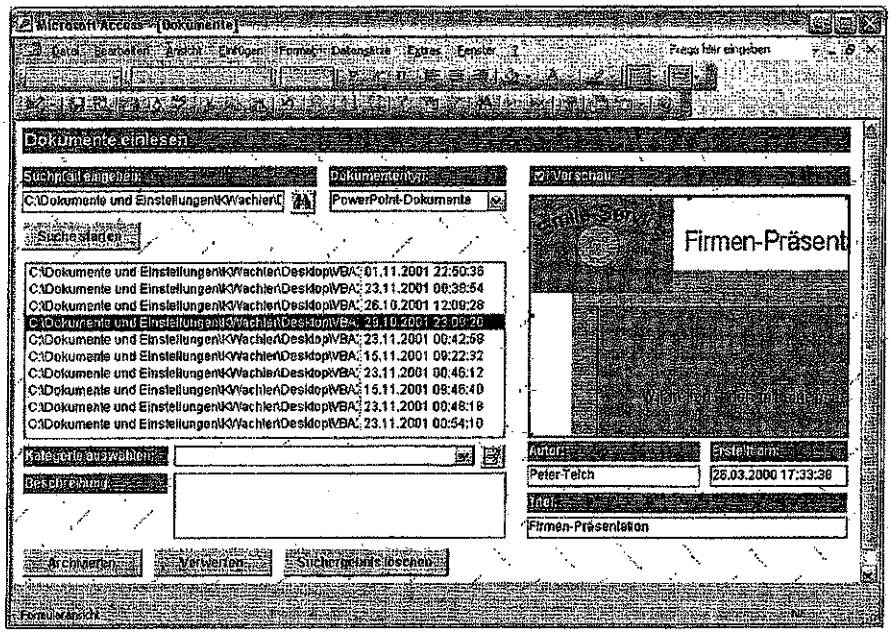
① Die Datenbank soll neben Word- und Excel-Dokumenten auch PowerPoint-Präsentationen verarbeiten können. Öffnen Sie dazu zuerst das Formular DOKUMENTEEINLESEN im Entwurfsmodus.

② Fügen Sie dem Listenfeld DOKUMENTENTYP (`cmb_DokTyp`) einen Eintrag *PowerPoint-Dokumente* hinzu.

③ Verändern Sie die Ereignisprozedur `btn_Suche_Click` so, dass auch nach PowerPoint-Dokumenten gesucht werden kann. Werten Sie dazu die Einstellung im Listenfeld DOKUMENTENTYP aus.

④ Erstellen Sie auf der Basis der Prozedur `VorschauExcel` die Prozedur `VorschauPowerPoint`. Die Prozedur soll die Vorschaufunktion für PowerPoint-Präsentationen realisieren.

Notizen



Eine PowerPoint-Präsentation in der Vorschau

⑤ Damit in PowerPoint eine Präsentation im Hintergrund unsichtbar geöffnet werden kann, müssen Sie der `Open`-Methode für das Argument `WithWindow` die Konstante `msoFalse` zuweisen. PowerPoint erwartet standardmäßig ein sichtbares Fenster und würde sonst mit einer Fehlermeldung abbrechen.

⑥ Kopieren Sie für die Vorschaufunktion die erste Folie (`Slides`-Auflistung) in die Zwischenablage.

⑦ Ermitteln Sie die Dokumenteninformationen mit der `BuiltInDocumentProperties`. Verwenden Sie dabei die gleichen Werte wie in der `VorschauExcel`-Prozedur.

⑧ Verändern Sie die Prozedur `Form_Close` so, dass auch PowerPoint beim Beenden des Formulars geschlossen wird.

⑨ Öffnen Sie das Formular `Dokumentverwalten` im Entwurfsmodus. Fügen Sie dem Listenfeld DOKUMENTENTYP (`cmb_DokTyp`) einen Eintrag *PowerPoint-Dokumente* hinzu.

⑩ Bearbeiten Sie die Ereignisprozedur `btn_Suche_Click`, damit auch nach PowerPoint-Dokumenten mit der Dateiendung **.ppt* gesucht werden kann.

⑪ Nach Betätigen der Schaltfläche ÖFFNEN soll auch eine PowerPoint-Präsentation geöffnet werden. Bearbeiten Sie dazu die passende Ereignisprozedur. Beachten Sie, dass auch hier das Argument `WithWindow` der `Open`-Methode mit der Konstante `msoFalse` belegt werden muss, da das Dokument im Hintergrund geöffnet wird.

Übung 2

Übungsdatei: *Übung1-E.mdb* **Ergebnisdatei:** *Übung2-E.mdb*

① Verwenden Sie die bearbeitete Datenbank aus Übung 1. Es soll auch für das Formular DOKUMENTEVERWALTEN eine Vorschaufunktion eingefügt werden.

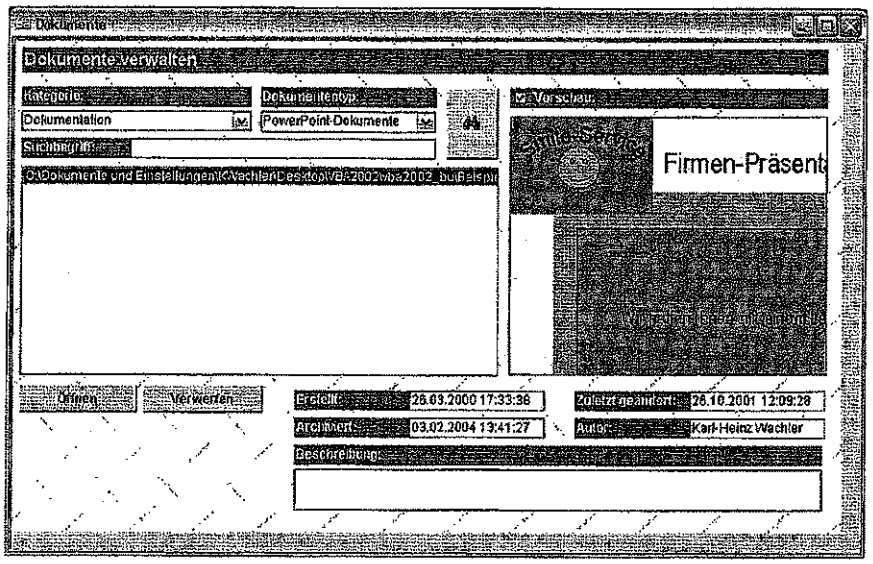
② Öffnen Sie das Formular DOKUMENTEVERWALTEN im Entwurfsmodus, und erstellen Sie ein gebundenes OLE-Objektfeld `ole_Vorschau` und ein Kontrollfeld `ctl_Vorschau`.

③ Fügen Sie die Vorschauprozeduren aus dem Modul des Formulars DOKUMENTEEINLESEN in das Modul des Formulars DOKUMENTEVERWALTEN ein.

④ Das Auslesen der Dokumenteninformationen wird in diesem Formular nicht mehr benötigt. Entfernen Sie die entsprechenden Anweisungen.

⑤ Bearbeiten Sie die Ereignisprozedur für das Auswählen eines Dokuments im Listenfeld. Je nach Typ des ausgewählten Dokuments soll die entsprechende Vorschauprozedur aufgerufen werden. Beachten Sie, dass zu Beginn der Prozedur das Formular mit der `Requery`-Methode aktualisiert wird, damit die Dokumenteninformationen angezeigt werden.

⑥ Erstellen Sie eine Ereignisprozedur, die beim Schließen des Formulars die möglicherweise für die Vorschaufunktion geladenen Anwendungen schließt.



Das Formular DOKUMENTEVERWALTEN mit Vorschaufunktion

Notizen

14 Integrierte Lösungen mit PowerPoint

In diesem Kapitel erfahren Sie

→ welche Möglichkeiten es zur Automation mit PowerPoint gibt

→ wie Sie von PowerPoint auf Word und Excel zugreifen

→ wie Sie eine integrierte Lösung für das Erstellen einer Präsentation realisieren

Voraussetzungen

✓ Programmieren von Automation-Objekten

✓ PowerPoint-Objektmodell

14.1 Automation mit PowerPoint

PowerPoint ist das geeignete Werkzeug für das kreative Gestalten von Informationen. Dabei ist es für das Erstellen von Folien und Bildschirmpräsentationen genauso geeignet wie für einfache Drucksachen.

Da das Gestalten der PowerPoint-Folien die wichtigste Aufgabe der Anwendung ist, sind die Programmiermöglichkeiten gegenüber anderen Office-Anwendungen deutlich eingeschränkt. So können Sie mithilfe von VBA nur wenig Einfluss auf den Ablauf einer Präsentation nehmen. Die meisten der vorhandenen Objekte beziehen sich auf das Aussehen und die Position der Objekte auf den Folien. Das Objektmodell ist daher deutlich kleiner als bei Word, Excel und Access.

Auch wenn PowerPoint weniger Spielraum für VBA-Programme bietet, können integrierte Lösungen mit anderen Office-Anwendungen einige Aufgaben beim Erstellen von Präsentationen vereinfachen. So können Sie beispielsweise Werte aus einer Datenbank dynamisch in einer Präsentation einbauen. Mithilfe von integrierten Lösungen in VBA ist es möglich, im Entwurfs- und teilweise auch im Präsentationsmodus auf die Objekte einer Folie zuzugreifen. Sie können dabei z. B. deren Position, Aussehen oder Inhalt verändern.

PowerPoint wird meist als Automation-Client eingesetzt, um die Funktionen fremder Anwendungen zu nutzen. Integrierte Lösungen eignen sich besonders für das dynamische Erstellen und Verändern von Daten in einer Präsentation.

Folgende Möglichkeiten ergeben sich beispielsweise für integrierte Lösungen mit PowerPoint als Client:

Word als Server	☑	Auslesen von Texten und Einfügen in eine Präsentation
	☑	Bearbeiten größerer Textmengen für den Einsatz in PowerPoint-Folien
Excel als Server	☑	Ermitteln von Daten für dynamische Folieninhalte
	☑	Importieren von Diagrammen zur Visualisierung
Access als Server (ADO-Objekte)	☑	Ermitteln von Daten für dynamische Folieninhalte
	☑	Automatisches Erstellen von Präsentationen aus Datenbankinhalten

In den folgenden Abschnitten wird die Erstellung von integrierten Lösungen mit PowerPoint am Beispiel der Anwendung *SchnellPräsentation.ppt* gezeigt. Die Anwendung nutzt Word und Excel und vereinfacht das Erstellen von Präsentationen, die stets nach einem gleichen Muster gestaltet werden sollen.

14.2 Die Anwendung SCHNELLPRÄSENTATION als integrierte Lösung

Aufgabe der Anwendung

Die in diesem Kapitel beschriebene Anwendung SCHNELLPRÄSENTATION hilft beim Erstellen von Präsentationen, die stets nach einem bestimmten Muster ablaufen sollen. Die Anwendung soll Textpassagen aus Word und Diagramme aus Excel übernehmen und diese Daten in das Folienlayout einfügen. Die Steuerung erfolgt über UserForm-Dialoge, die Eingaben abfragen und den Ablauf über Schaltflächen steuern.

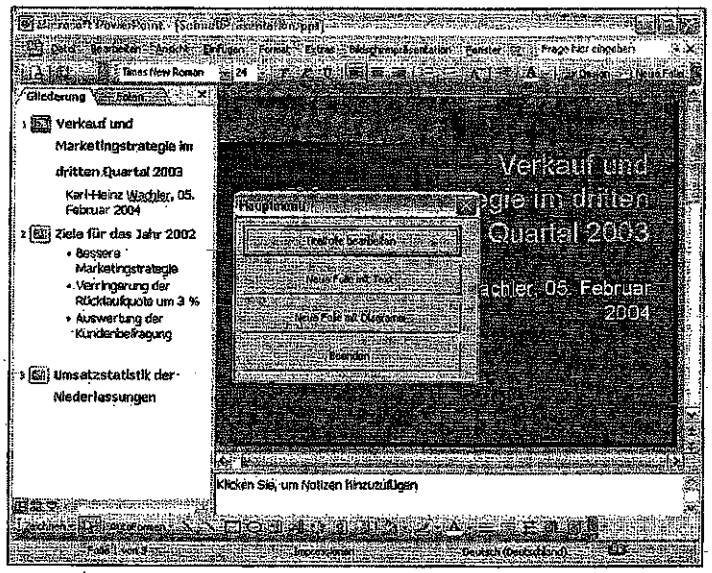
Der Zugriff auf die Anwendungen Word und Excel wird über Automation-Objekte sowie über die Zwischenablage realisiert. Alle Prozeduren sind als Ereignisprozeduren in den Modulen der UserForm-Dialoge ausgeführt.

Mit der Anwendung ist es möglich, eine beliebige Anzahl Folien mit Text oder Diagrammen einzufügen. Zu Beginn muss jedoch bereits eine Titelfolie vorliegen.

Die funktionsfähige Anwendung finden Sie als Datei unter dem Namen *SchnellPräsentation.ppt*.

Arbeiten mit der Anwendung

- ☑ Starten Sie die VBA-Anwendung über die Schaltfläche SCHNELLPRÄSENTATION in der Menüleiste.
- ☑ Das Hauptmenü wird angezeigt und erlaubt Ihnen die Auswahl der auszuführenden Tätigkeit.
- ☑ Bearbeiten Sie die Titelfolie, um Titel, Autor und Datum einzugeben.
- ☑ Erstellen Sie beliebig viele Textfolien nach Eingabe von Text oder Import aus Word.
- ☑ Erstellen Sie beliebig viele Diagrammfolien nach Auswahl einer Excel-Datei und Import eines Diagramms.
- ☑ Beenden Sie die Anwendung.



Das Hauptmenü der Anwendung

Notizen

Bestandteile der Anwendung

Die Anwendung besteht aus mehreren UserForm-Dialogen und deren Ereignisprozeduren im jeweiligen Modul.

frm_Hauptmenü	Das Hauptmenü erlaubt die Auswahl der verschiedenen Funktionen der Anwendung. Es wird durch die Schaltfläche BEENDEN geschlossen.
frm_Titelfolie	Über diesen Dialog werden die Texte der Titelfolie bearbeitet. Der Autor und das Datum werden dabei durch die Anwendung als Voreinstellung eingetragen.
frm_Text	Der Dialog erlaubt das Erstellen einer Folie mit Titel und einer Textpassage. Der Text kann im Formular eingegeben oder nach Betätigen einer Schaltfläche aus Word importiert werden.
frm_Diagramm	Dieser Dialog erstellt eine Folie, die aus einer Titelzeile und einem Diagramm besteht. Nach Auswahl einer Excel-Datei und eines Tabellenblatts wird ein Diagramm aus Excel übernommen und in die neue Folie eingefügt.

Zusätzlich wird durch den UserForm-Dialog `frm_Diagramm` ein Standarddialog zur Dateiauswahl eingeblendet.

Anwendungsstart über Menübefehl

Die Anwendung soll über einen Eintrag in der Menüleiste ständig verfügbar sein, damit ein Anwender sie ohne Aufruf der VBA-Entwicklungsumgebung starten kann.

Für diesen Zweck erstellen Sie zuerst eine Prozedur, die den UserForm-Dialog des Hauptmenüs aufruft:

```
Sub StarteAnwendung()
  frm_Hauptmenü.Show
End Sub
```

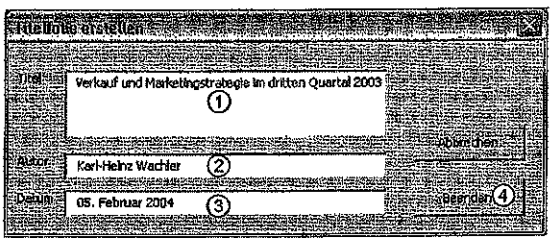
① Die Methode `Show` des UserForm-Dialogs blendet den Dialog ein.

Über den Menübefehl EXTRAS - ANPASSEN können Sie danach eine neue Schaltfläche in der Menüleiste einfügen und mit der Prozedur `StarteAnwendung` verknüpfen.

14.3 Bearbeiten der Titelfolie

Arbeiten mit dem UserForm-Dialog `frm_Titelfolie`

Der UserForm-Dialog erlaubt das Bearbeiten der ersten Folie der Präsentation. Es wird dabei vorausgesetzt, dass die Folie bereits zwei Textobjekte enthält.



Der UserForm-Dialog zum Bearbeiten der Titelfolie

Folgender Arbeitsablauf ist notwendig:

⇨ Öffnen Sie den UserForm-Dialog durch Betätigen der Schaltfläche TITELFOLIE BEARBEITEN im Hauptmenü.

⇨ Geben Sie den Titel der Präsentation im Textfeld Titel ① ein. Der Text wird als Überschrift in der ersten Folie eingefügt.

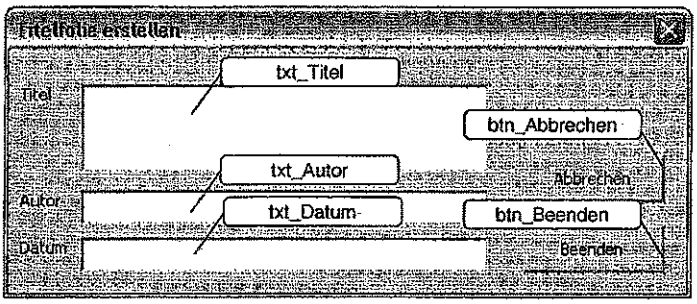
⇨ Geben Sie den Autor der Präsentation im Textfeld ② ein. Das Textfeld enthält als Voreinstellung den aus den Dokumenteneigenschaften ermittelten Autor.

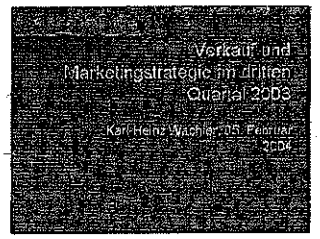
⇒ Das Textfeld DATUM ③ enthält das aktuelle Datum. Passen Sie den Eintrag gegebenenfalls an.

⇒ Beenden Sie den Dialog mit der Schaltfläche BEENDEN ④. Mit der Schaltfläche ABBRECHEN beenden Sie den Dialog ohne Aktion.

Steuerelemente des UserForm-Dialogs

Der UserForm-Dialog enthält Steuerelemente mit der folgenden Benennung:





Die nach den Vorgaben bearbeitete Titelfolie

Die Ereignisprozeduren

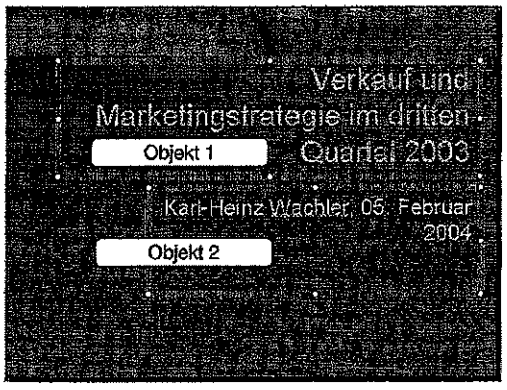
Die Ereignisprozeduren werden beim Öffnen des Formulars oder dem Betätigen einer der Schaltflächen ausgeführt.

```
Private Sub UserForm_Activate()
① Me.txt_Autor = ActivePresentation.BuiltInDocumentProperties(3)
② Me.txt_Datum = Format(Date, "dd. mmmm yyyy")
③ Me.txt_Titel.SetFocus
End Sub

Private Sub btn_Beenden_Click()
  With ActivePresentation
④   .Slides(1).Shapes(1).TextFrame.TextRange.Text = Me.txt_Titel
⑤   .Slides(1).Shapes(2).TextFrame.TextRange.Text = Me.txt_Autor & _
                                                    ", " & Me.txt_Datum
  End With
⑥ Me.Hide
End Sub

Private Sub btn_Abbrechen_Click()
⑦ Me.Hide
End Sub
```

① Über die Eigenschaft `BuiltIndocumentProperties` wird auf die Dokumenteneigenschaften der aktiven Präsentation zugegriffen. Der Wert 3 als Argument bezieht sich auf den Eintrag des Autors. PowerPoint definiert dafür den Namen des standardmäßigen Anwenders. Der Name des Autors wird dem Textfeld `txt_Autor` zugewiesen.

② Die VBA-Funktion `Date` liefert das aktuelle Systemdatum, das mit der Funktion `Format` in das gewünschte Datumsformat umgewandelt wird. Anschließend erfolgt die Zuweisung zum Textfeld `txt_Datum`.

③ Das Textfeld zur Eingabe des Titels wird mithilfe der Methode `SetFocus` ausgewählt. Der Cursor befindet sich danach in diesem Feld.



Die zwei Objekte der Titelfolie

④ Mithilfe des `ActivePresentation`-Objekts wird auf die erste Folie zugegriffen. Das `Shapes`-Objekt erlaubt dabei das Ansprechen der einzelnen Objekte auf der Folie. Da die Folie nur zwei Objekte enthält, werden sie mit den Indexwerten 1 und 2 angesprochen. Das `TextRange`-Objekt verweist auf den Textinhalt der Objekte, der mit der Eigenschaft `Text` verändert werden kann.

⑤ Autor und Datum werden aus den entsprechenden Textfeldern ausgelesen und dem zweiten Objekt auf der Folie zugewiesen.

⑥ Nach dem Einfügen der Texte wird das Formular geschlossen.

⑦ Beim Betätigen der Schaltfläche ABBRECHEN wird das Formular ohne weitere Aktion geschlossen und dadurch zum Hauptmenü gewechselt.

Jedes Objekt einer PowerPoint-Folie entspricht einem `Shape`-Objekt der `Shapes`-Auflistung. Allen Objekten, z. B. AutoFormen, kann über den Menüpunkt EINFÜGEN - TEXTFELD bzw. dem Kontextmenüpunkt TEXT HINZUFÜGEN ein Textrahmen zugeordnet werden. Diesem Textrahmen entspricht das Objekt `TextFrame`, das dem `Shape`-Objekt untergeordnet ist. Auch wenn Sie in PowerPoint ein Textfeld einfügen, greifen Sie darauf über die Anweisung `Shapes(Index).TextFrame` zu.

14.4 Erstellen einer Textfolie mit Word-Unterstützung

Arbeiten mit dem UserForm-Dialog `frm_Text`

Der UserForm-Dialog erlaubt das Eingeben eines Titels und einer Textpassage. Es wird eine neue Folie vom Typ AUFZÄHLUNG mit zwei Objekten eingefügt, denen die Texte zugeordnet werden. Auf Wunsch ist die Übernahme eines in Word markierten Textes möglich.

Folgender Arbeitsablauf ist notwendig:

⇒ Betätigen Sie im Hauptmenü die Schaltfläche NEUE FOLIE MIT TEXT, um den UserForm-Dialog `frm_Text` zu öffnen.

⇒ Geben Sie im Feld TEXT ① den gewünschten Text ein.

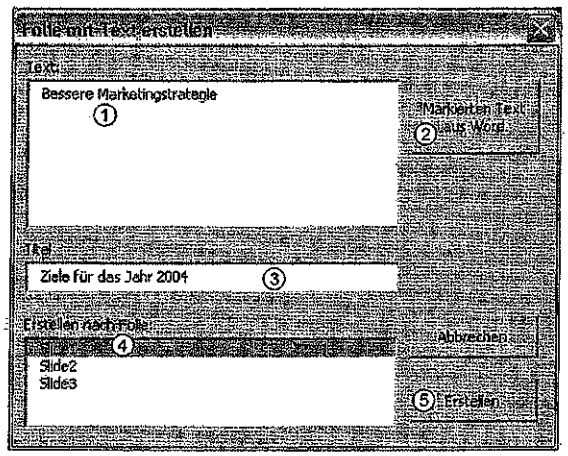
oder Öffnen Sie ein Word-Dokument, und markieren Sie den gewünschten Text. Betätigen Sie danach die Schaltfläche MARKIERTEN TEXT AUS WORD ②.

⇒ Geben Sie in das Feld TITEL ③ die Überschrift der Folie ein.

⇒ Wählen Sie im Listenfeld ERSTELLEN NACH FOLIE ④ die Folie aus, nach der die neue Textfolie eingefügt werden soll.

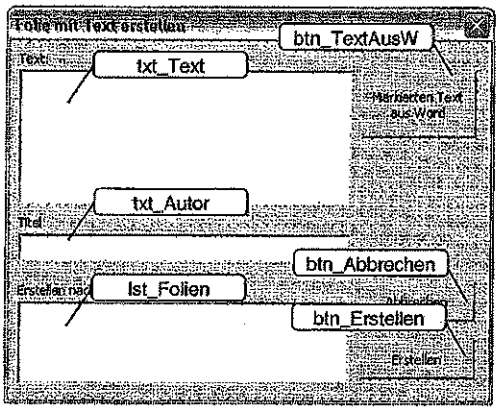
⇒ Mit der Schaltfläche ERSTELLEN ⑤ fügen Sie die neue Folie ein. Der Dialog wird dabei nicht geschlossen.

⇒ Erstellen Sie eine weitere Textfolie, oder beenden Sie den Dialog mit der Schaltfläche ABBRECHEN.



Dialog zum Erstellen einer neuen Textfolie

Steuerelemente des UserForm-Dialogs



Anzeige der vorhandenen Folien

Das Listenfeld `lst_Folien` wird beim Öffnen des UserForm-Dialogs mit den Namen der bereits vorhandenen Folien der Präsentation gefüllt. Sie können in diesem Listenfeld auswählen, an welcher Stelle die neue Folie eingefügt werden soll.

```
Private Sub UserForm_Activate()
  Dim Folie As Slide
  Me.lst_Folien.Clear
  For Each Folie In ActivePresentation.Slides
      Me.lst_Folien.AddItem Folie.Name
  Next
  If Me.lst_Folien.ListCount > 0 Then
     Me.lst_Folien.Selected(Me.lst_Folien.ListCount - 1) = True
  End If
End Sub
```

① Die Methode `Clear` des Listenfelds löscht alle eventuell vorhandenen Einträge.

② Die `For-Each`-Schleife durchläuft alle Elemente der `Slides`-Auflistung. Jedes Element entspricht einem `Slide`-Objekt und damit einer Folie der Präsentation.

③ Die `AddItem`-Methode fügt dem Listenfeld einen neuen Eintrag hinzu. Dazu wird der Name der aktuellen Folie ermittelt.

④ Damit standardmäßig immer die letzte Folie ausgewählt ist, wird dieser Eintrag des Listenfelds mithilfe der `Selected`-Eigenschaft markiert. Die Eigenschaft `ListCount` liefert dabei die Anzahl der Einträge im Listenfeld. Die Indexzählung beginnt bei Null, daher wird der Wert 1 abgezogen. Davor wird mit der Abfrage der `ListCount`-Eigenschaft überprüft, ob es überhaupt Einträge im Listenfeld gibt.

Alle vorhandenen Einträge des Listenfelds werden zu Beginn der Prozedur entfernt. Dies ist notwendig, da der Inhalt aller Steuerelemente beim Ausblenden des Dialogs mit der `Hide`-Methode nicht gelöscht wird. Beim erneuten Anzeigen werden die vorhandenen Elemente des Listenfelds wieder angezeigt.

Text aus Word importieren

Nach dem Betätigen der Schaltfläche MARKIERTEN TEXT AUS WORD wird der Text aus dem aktuellen Word-Dokument ausgelesen und in das Textfeld TEXT des UserForm-Dialogs eingetragen. Word muss zu diesem Zweck bereits geöffnet sein. Die Prozedur prüft, ob Text im aktiven Dokument markiert wurde. Ist dies nicht der Fall, wird eine Fehlermeldung angezeigt.

```
Private Sub btn_TextAusWord_Click()
  On Error GoTo Fehlerbehandlung
  Dim AppWd As Word.Application
  Set AppWd = GetObject(, "Word.Application")
  If AppWd.Selection.Type = wdSelectionIP Then
      MsgBox "Bitte markieren Sie einen Textabschnitt!", vbExclamation
  End If
  Me.txt_Text = AppWd.Selection.Text
  Set AppWd = Nothing
  Exit Sub
Fehlerbehandlung:
  If Err = 429 Then
    MsgBox "Bitte starten Sie Word, und markieren Sie einen Text!", _
           vbExclamation
  End If
End Sub
```

① Mit dieser Anweisung wird die Fehlerkontrolle aktiviert. Im Fall eines Fehlers soll zur Sprungmarke FEHLERBEHANDLUNG verzweigt werden.

② Der Zugriff auf eine aktive Word-Instanz erfolgt mit der `GetObject`-Anweisung. Ist Word noch nicht aktiv, wird ein Fehler erzeugt.

③ Auf die aktive Markierung im Word-Dokument kann über das `Selection`-Objekt zugegriffen werden. Die Eigenschaft `Type` bestimmt die Art der Markierung. Hier wird geprüft, ob es sich um eine Einfügemarke handelt (`wdSelectionIP`). Ist dies der Fall, wurde kein Text markiert, und eine Fehlermeldung wird eingeblendet.

Fehlermeldung, falls kein Text ausgewählt wurde

④ Der Text der aktiven Markierung wird mithilfe der Texteigenschaft ausgelesen und dem Textfeld zugewiesen.

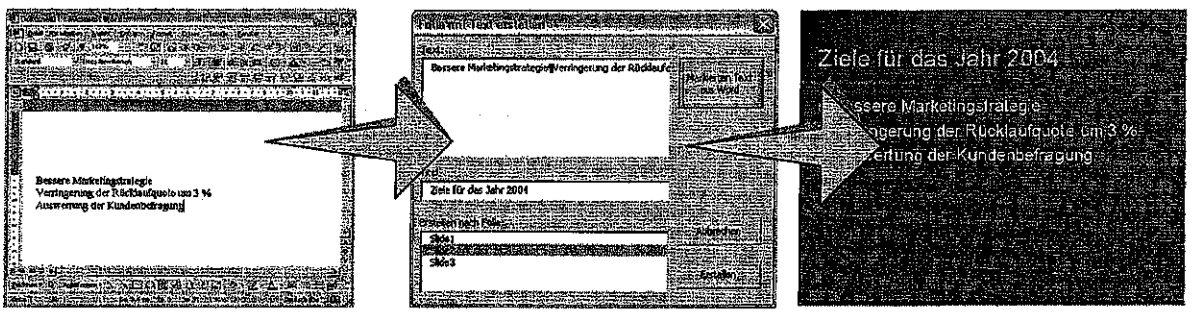
⑤ Das Automation-Objekt `AppWD` zum Zugriff auf Word wird nicht mehr benötigt. Durch Zuweisen von `Nothing` wird der belegte Speicher freigegeben.

⑥ Ist Word noch nicht aktiv, erzeugt die `GetObject`-Anweisung den Fehler 429. Falls dieser Fehler auftritt, wird eine Fehlermeldung angezeigt.

Textfolie einfügen

Nach dem Betätigen der Schaltfläche ERSTELLEN wird die neue Folie vom Typ *Aufzählung* eingefügt. Dazu werden die Eingaben im UserForm-Dialog verwendet. Die Position der neuen Folie richtet sich nach der Auswahl im Listenfeld.

Beim Importieren eines Textabschnitts aus Word werden auch Absatzzeichen beachtet. Für jeden Absatz erzeugt PowerPoint einen neuen Aufzählungspunkt.



Aus den Eingaben im UserForm-Dialog erzeugt die Prozedur eine PowerPoint-Folie

Für das zweite Objekt der Folie, das Textfeld mit der Aufzählung, legt die Prozedur zusätzlich eine Textanimation fest.

```
Private Sub btn_Erstellen_Click()
  Dim NeueFolie As Slide
①  Set NeueFolie = ActivePresentation.Slides.Add(Me.lst_Folien.ListIndex + _
                   2, ppLayoutText)
②  NeueFolie.Shapes(1).TextFrame.TextRange.Text = Me.txt_Titel
   NeueFolie.Shapes(2).TextFrame.TextRange.Text = Me.txt_Text
③  NeueFolie.Shapes(2).AnimationSettings.TextUnitEffect = ppAnimateByParagraph
   NeueFolie.Shapes(2).AnimationSettings.EntryEffect = ppEffectFlyFromBottom
④  NeueFolie.Select
⑤  UserForm_Activate
End Sub
```

① Die `Slides`-Auflistung erlaubt mit der `Add`-Methode das Hinzufügen neuer Folien. Dazu wird als erstes Argument die Position der neuen Folie und als zweites Argument der Layouttyp als Konstante übergeben. Die Folienposition wird über die Auswahl im Listenfeld ermittelt.

② Beiden Objekten wird der entsprechende Text aus dem UserForm-Dialog zugewiesen. Dazu wird auf das `TextFrame`-Objekt zugegriffen.

③ Jedem Objekt der Folie kann eine Animation zugewiesen werden. Dazu dient das `AnimationSettings`-Objekt. Hier wird zuerst festgelegt, dass der Text absatzweise animiert wird. Die Eigenschaft `EntryEffect` legt danach den Animationseffekt beim Einblenden fest (hier Text von unten).

④ Mithilfe der `Select`-Methode wird die neue Folie aktiviert und angezeigt.

⑤ Da die Präsentation nun eine neue Folie enthält, muss das Listenfeld `lst_Folien` aktualisiert werden. Zu diesem Zweck wird die Ereignisprozedur `UserForm_Activate` aufgerufen.

Für das Wechseln zu einer bestimmten Folie existieren verschiedene Methoden, je nach aktivem PowerPoint-Modus. Ist eine Bildschirmpräsentation aktiv, erfolgt der Zugriff auf einzelne Folien über das Objekt `SlideShowWindows(Index).View`. Dieses Objekt stellt die Methode `GoToSlide` zur Verfügung. Im Entwurfsmodus existiert für diesen Zweck nur die Methode `Select`, die eine Folie auswählt und gleichzeitig als aktive Folie anzeigt.

14.5 Erstellen einer Diagrammfolie mit Excel-Unterstützung

Arbeiten mit dem UserForm-Dialog `frm_Diagramm`

Der UserForm-Dialog erlaubt die Auswahl eines Tabellenblatts mit Diagramm oder eines Diagrammblatts einer Excel-Arbeitsmappe. Das Diagramm wird über die Zwischenablage, zusammen mit dem eingegebenen Folientitel, in eine neue PowerPoint-Folie vom Typ DIAGRAMM eingefügt.

Notizen

Folgender Arbeitsablauf ist notwendig:

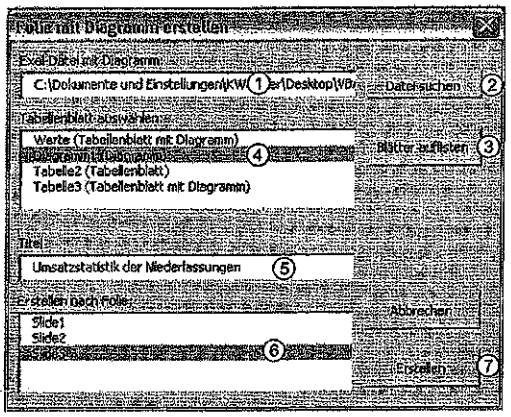
⇒ Wählen Sie im Hauptmenü die Schaltfläche NEUE FOLIE MIT DIAGRAMM, um den UserForm-Dialog frm_Diagramm zu öffnen.

⇒ Geben Sie im Textfeld EXCEL-DATEI MIT DIAGRAMM ① den Dateipfad einer Excel-Arbeitsmappe an.

oder Öffnen Sie mit der Schaltfläche DATEI SUCHEN ② den Dialog zum Suchen einer Excel-Arbeitsmappe.

⇒ Betätigen Sie die Schaltfläche BLÄTTER AUFLISTEN ③, um alle Tabellenblätter und Diagramme der Arbeitsmappe im Listenfeld TABELLENBLATT AUSWÄHLEN ④ zu sehen.

⇒ Wählen Sie im Listenfeld ④ das gewünschte Tabellenblatt mit Diagramm bzw. das Diagramm, das Sie importieren möchten.



Dialog zum Erstellen einer Diagrammfolie

⇒ Geben Sie im Listenfeld TITEL ⑤ die Überschrift der Folie ein.

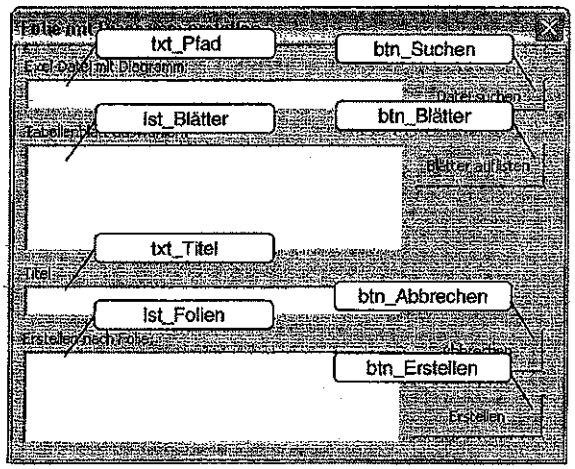
⇒ Wählen Sie im Listenfeld ERSTELLEN NACH FOLIE ⑥ die Folie, nach der Sie die neue Folie einfügen möchten.

⇒ Betätigen Sie die Schaltfläche ERSTELLEN ⑦, um die neue Folie mit den gewählten Daten zu erstellen.

⇒ Fügen Sie weitere Folien ein, oder beenden Sie den Dialog mit der Schaltfläche ABBRECHEN.

Die Anzeige der vorhandenen Folien erfolgt auf die gleiche Weise, wie im UserForm-Dialog `frm_Text`. Auch hier wird das Listenfeld `lst_Folien` beim Anzeigen des Dialogs gefüllt.

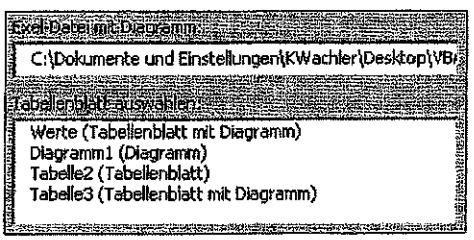
Steuerelemente des UserForm-Dialogs



Anzeige der Tabellenblätter und Diagramme

Nach Betätigen der Schaltfläche BLÄTTER AUFLISTEN wird die durch den Dateipfad im Textfeld `lst_Pfad` spezifizierte Excel-Arbeitsmappe geöffnet, und die vorhandenen Tabellenblätter und Diagramme werden ausgelesen.

Neben dem Namen soll das Listenfeld anzeigen, ob es sich um ein Tabellenblatt oder Diagramm handelt. Für alle Tabellenblätter soll außerdem ermittelt werden, ob sie ein Diagramm enthalten.



Die Tabellenblätter und Diagramme werden im Listenfeld angezeigt

Die Ereignisprozedur für die Schaltfläche BLÄTTER AUFLISTEN öffnet eine Arbeitsmappe im Hintergrund und ermittelt die Namen der Tabellenblätter und Diagramme.

```
Private Sub btn_Blätter_Click()
①   Dim TabBlatt As Object, Anzeige As String
②   If Dir(Me.txt_Pfad) = "" Then
      MsgBox "Die angegebene Datei existiert nicht!", vbCritical
      Exit Sub
    End If
③   Set MappeXl = GetObject(Me.txt_Pfad)
④   Me.lst_Blätter.Clear
⑤   For Each TabBlatt In MappeXl.Sheets
⑥     Anzeige = TabBlatt.Name
⑦     If TypeName(TabBlatt) = "Worksheet" Then
⑧       If TabBlatt.ChartObjects.Count > 0 Then
          Anzeige = Anzeige & " (Tabellenblatt mit Diagramm)"
        Else
          Anzeige = Anzeige & " (Tabellenblatt)"
        End If
      Else
        Anzeige = Anzeige & " (Diagramm)"
      End If
⑨     Me.lst_Blätter.AddItem Anzeige
    Next
End Sub
```

① Für den Zugriff auf die einzelnen Tabellenblätter und Diagramme wird die Objektvariable `TabBlatt` benötigt. Da für Tabellenblätter und Diagramme unterschiedliche Objekttypen (`WorkSheet` und `Chart`) existieren, wird die Variable mit dem allgemeinen Typ `Object` deklariert.

② Die `Dir`-Funktion prüft, ob die angegebene Datei vorhanden ist. Sie liefert einen gültigen Dateipfad im Erfolgsfall oder eine leere Zeichenkette bei einem Fehler.

③ Die Arbeitsmappe wird mithilfe der Funktion `GetObject` geöffnet. Die Objektvariable `MappeXL`, die in allen Prozeduren zur Verfügung steht, erhält einen Verweis auf die geöffnete Arbeitsmappe.

④ Alle eventuell vorhandenen Einträge des Listenfelds werden gelöscht.

⑤ Die `Sheets`-Auflistung der Arbeitsmappe enthält alle `WorkSheet`- und `Chart`-Objekte der Arbeitsmappe. Die `For-Each`-Schleife durchläuft alle Elemente der Auflistung. Der Objekttyp der Variablen `TabBlatt` wird erst an dieser Stelle für jeden Durchlauf festgelegt.

⑥ Zuerst wird der Name des Tabellenblatts oder Diagramms ermittelt.

⑦ Die VBA-Funktion `TypeName` ermittelt den Objektnamen, z. B. `Worksheet`. Daraus lässt sich schließen, ob es sich um ein Tabellenblatt oder ein Diagramm handelt. Die jeweilige Bezeichnung wird im Listenfeld als Information angezeigt.

⑧ Die `ChartObjects`-Auflistung eines Tabellenblatts verweist auf die vorhandenen Diagramme. Falls ein Diagramm vorhanden ist, enthält die Auflistung mindestens ein Element. In diesem Fall wird der Anzeige im Listenfeld eine entsprechende Information hinzugefügt.

⑨ Dem Listenfeld wird an dieser Stelle die Anzeige als neuer Eintrag hinzugefügt.

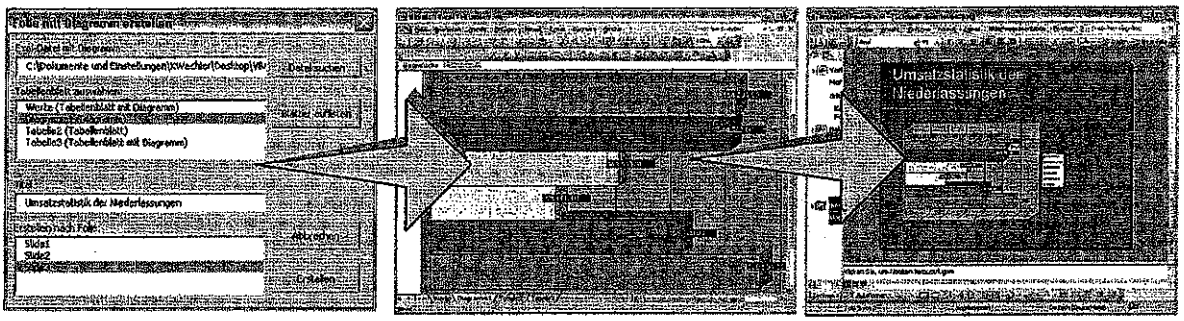
Auf die geöffnete Arbeitsmappe soll in verschiedenen Prozeduren zugegriffen werden. Die Objektvariable `MappeXL` wird daher bereits im Deklarationsteil des Moduls deklariert und steht damit in allen Prozeduren zur Verfügung.

Einige Auflistungen in den Objektmodellen der Office-Anwendungen können verschiedene Objekte beinhalten. Die `Sheets`-Auflistung von Excel unterscheidet beispielsweise zwischen `WorkSheet`- (Tabellenblatt) und `Chart`-Objekten (Diagrammen). Objektvariablen zum Zugriff auf solche Auflistungen können Sie mit dem Typ `Object` deklarieren.

Diagramm importieren und neue Folie erstellen

Mit dem Betätigen der Schaltfläche ERSTELLEN erzeugt die Ereignisprozedur eine neue PowerPoint-Folie vom Typ DIAGRAMM. Danach wird das erste Diagramm des ausgewählten Tabellenblatts über die Zwischenablage in diese Folie eingefügt.

Die Prozedur verwendet für das Skalieren des Diagramms die Maße des bereits vorhandenen Objekts der Folienvorlage. Dieses Objekt wird nach dem Ermitteln der Abmessungen gelöscht und durch das kopierte Diagramm ersetzt.



Die Prozedur kopiert das Diagramm aus einem im Hintergrund geöffneten Excel-Tabellenblatt

```
Private Sub btn_Erstellen_Click()
  Dim NeueFolie As Slide, Links As Long, Oben As Long, Breite As Long
  Dim Höhe As Long, TabBlatt As Object
①  Set NeueFolie = ActivePresentation.Slides.Add(Me.lst_Folien.ListIndex + 2, _
                                                  ppLayoutChart)
  With NeueFolie
②    .Shapes(1).TextFrame.TextRange.Text = Me.txt_Titel
③    .Select
④    Links = .Shapes(2).Left
     Oben = .Shapes(2).Top
     Breite = .Shapes(2).Width
     Höhe = .Shapes(2).Height
⑤    .Shapes(2).Delete
⑥    Set TabBlatt = MappeXl.Sheets(Me.lst_Blätter.ListIndex + 1)
⑦    TabBlatt.Activate
⑧    If TypeName(TabBlatt) = "Worksheet" Then
       If TabBlatt.ChartObjects.Count > 0 Then
⑨        TabBlatt.ChartObjects(1).Copy
       End If
     Else
⑩      TabBlatt.ChartArea.Copy
     End If
⑪    .Shapes.Paste
⑫    .Shapes(2).Left = Links
     .Shapes(2).Top = Oben
     .Shapes(2).Width = Breite
     .Shapes(2).Height = Höhe
  End With
  UserForm_Activate
End Sub
```

① Über die Add-Methode der Slides-Auflistung erfolgt das Hinzufügen einer neuen Folie. Dazu wird als erstes Argument die Position der neuen Folie und als zweites Argument der Layouttyp als Konstante (ppLayoutChart) übergeben. Die Folienposition wird über die Auswahl im Listenfeld ermittelt.

② Die im UserForm-Dialog angegebene Überschrift wird dem ersten Objekt der Folie zugewiesen.

③ Die Select-Methode wählt die neue Folie als aktiv aus.

④ Die Abmessungen und die Position des zweiten Objekts der Folie werden für die spätere Formatierung des Diagramms gespeichert.

⑤ Danach wird das Objekt über die `Delete`-Methode gelöscht.

⑥ Der Objektvariablen `TabBlatt` wird ein Verweis auf das im Listenfeld ausgewählte Tabellenblatt übergeben.

⑦ Einige Objektmethoden können erst verwendet werden, wenn das betreffende Tabellenblatt mithilfe der `Activate`-Methode aktiviert ist.

⑧ Da das Diagramm auf einem Tabellenblatt mit einer anderen Anweisung kopiert werden muss als ein Diagramm auf einem eigenen Blatt, erfolgt hier eine Unterscheidung anhand des Objekttyps.

⑨ Die `ChartObjects`-Auflistung enthält alle Diagramme eines Tabellenblatts. Mit der `Copy`-Methode wird das erste Diagramm in die Zwischenablage kopiert.

⑩ Handelt es sich bei dem Objekt der `Sheets`-Auflistung bereits um ein `Chart`-Objekt, kann der Diagrammbereich über das `ChartArea`-Objekt kopiert werden.

⑪ Die `Shapes`-Auflistung der Folienobjekte verfügt über die Methode `Paste`, die den Inhalt der Zwischenablage, in diesem Fall das Excel-Diagramm, als neues Objekt in die Folie einfügt.

⑫ Mit den gespeicherten Werten für Position und Größe wird am Ende der Prozedur das neue Diagramm formatiert.

Viele Objektmethoden können nur auf aktivierte Elemente angewendet werden. So ist beispielsweise in Excel das Kopieren eines Diagramms nur im aktiven Tabellenblatt möglich. Versuchen Sie, eine solche Aktion ohne vorherige Aktivierung auszuführen, kann es zu unerwarteten Ergebnissen oder einer Fehlermeldung kommen. In Excel können Sie ein Tabellenblatt mit der `Activate`-Methode aktivieren. In PowerPoint ist dafür die `Select`-Methode zuständig.

Wenn der UserForm-Dialog geschlossen wird, erzeugt VBA das `Deactivate`-Ereignis. Da zu diesem Zeitpunkt auf die Excel-Objekte kein Zugriff mehr notwendig ist, kann der Speicherplatz freigegeben werden.

```
Private Sub UserForm_Deactivate()
  On Error Resume Next
  MappeXl.Close SaveChanges:=False
  Set MappeXl = Nothing
End Sub
```

14.6 Suchfunktion für Excel-Arbeitsmappen

Arbeiten mit dem `FileDialog`-Objekt

Der UserForm-Dialog `frm_Diagramm` erlaubt die direkte Eingabe eines Dateipfads. Bei einem langen und komplizierten Verzeichnispfad ist dies jedoch sehr aufwändig und unbequem.

In Office XP können Sie auf einfache Weise die standardmäßigen Windows-Dialogfenster zur Datei- und Verzeichnisauswahl verwenden. Dazu existiert das `FileDialog`-Objekt, das allen `Application`-Objekten der Office-Anwendungen zur Verfügung steht. Es erlaubt die Anzeige eines Dialogfensters zur Datei- oder Verzeichnisauswahl. Dabei können einige Parameter des Fensters, wie z. B. der Text in der Titelleiste, frei gewählt werden.

Dateisuche nach Excel-Arbeitsmappen

Nach dem Betätigen der Schaltfläche DATEI SUCHEN wird ein Dialogfenster eingeblendet, das die Auswahl einer Excel-Arbeitsmappe erlaubt. Dazu werden ein Titelleisten-Text und ein spezieller Dateifilter definiert. Nach dem Schließen des Fensters wird der gewählte Dateiname übernommen.

```
Private Sub btn_Suchen_Click()
  Dim fd As FileDialog
①  Set fd = Application.FileDialog(msoFileDialogFilePicker)
  With fd
②    .Filters.Add "Excel-Dateien", "*.xls", 1
    .Filters.Add "Alle Dateien", "*.*", 2
③    .InitialView = msoFileDialogViewDetails
    .Title = "Excel-Arbeitsmappe zum Diagramm-Import auswählen"
```

```
④      .Show
⑤      If .SelectedItems.Count > 0 Then
         Me.txt_Pfad = .SelectedItems(1)
       End If
     End With
End Sub
```

① An dieser Stelle wird eine Instanz des `FileDialog`-Objekts erstellt, auf die über die Objektvariable `fd` zugegriffen werden kann. Mit dem übergebenen Parameter wird der Typ des Dialogfensters festgelegt.

② Der erste Dateifilter wird auf die Dateierweiterung **.xls* für Excel-Dateien eingestellt. Danach wird ein weiterer Filter für die Anzeige aller Dateien erstellt. Den Filter kann der Benutzer im Dialogfenster auswählen, um die Anzeige einzuschränken.

③ Als Starteinstellung für die Anzeige der Dateien wird die Detailansicht gewählt. Zusätzlich definiert die `Title`-Eigenschaft den Text in der Titelleiste des Fensters.

④ Die `Show`-Methode zeigt das Dialogfenster am Bildschirm an.

⑤ Mithilfe der `Count`-Eigenschaft der `SelectedItems`-Auflistung wird überprüft, ob eine Datei ausgewählt wurde. Ist dies der Fall, wird der entsprechende Dateiname in das Eingabefeld des aufrufenden UserForm-Dialogs eingetragen.

14.7 Schnellübersicht

Sie möchten ...	
eine Schaltfläche in der Menüleiste erstellen	EXTRAS - ANPASSEN
einem Objekt auf einer Folie Text zuordnen	`Folienobjekt.Shapes(Index).TextFrame.TextRange.Text`
eine Markierung in Word prüfen	`Applicationobjekt.Selection.Type`
einen markierten Text in Word auslesen	`Applicationobjekt.Selection.Text`
einen Animationseffekt einstellen	`Folienobjekt.Shapes(Index).AnimationSettings. TextUnitEffect`
eine Folie auswählen und aktivieren (Entwurfsmodus)	`Folienobjekt.Select`
auf Diagramme in einem Excel-Tabellenblatt zugreifen	`Tabellenblattobjekt.ChartObjects(Index)`
ein Tabellenblatt aktivieren	`Tabellenblattobjekt.Activate`

14.8 Übung

Übung 1

Übungsdatei: *Übung1.ppt* Ergebnisdatei: *Übung1 E.ppt*

① Öffnen Sie die Präsentation *Übung1.ppt* und starten Sie die VBA-Entwicklungsumgebung.

② Der UserForm-Dialog `Frm_Titelfolie` soll im Textfeld `Titel` bereits den Text der Titelfolie anzeigen. Damit wird das nachträgliche Bearbeiten der Titelfolie vereinfacht. Bearbeiten Sie dazu die Ereignisprozedur, die beim Aktivieren des Dialogs aufgerufen wird.

③ Lesen Sie den zum ersten Objekt auf der Folie gehörenden Text aus, und weisen Sie ihn dem Textfeld im Dialog zu.

Übung 2

Übungsdatei: *Übung1-E.ppt* **Ergebnisdatei:** *Übung2-E.ppt*

① Falls Word nicht aktiv ist, wenn der markierte Text importiert werden soll, wird ein Meldungsfenster eingeblendet. Verändern Sie die Ereignisprozedur so, dass der Benutzer gefragt wird, ob Word gestartet werden soll.

② Verändern Sie die Anweisungen zur Fehlerbearbeitung: Zeigen Sie ein Meldungsfenster an, das die Schaltflächen JA und NEIN enthält.

Word wird auf Wunsch des Anwenders gestartet

③ Wird das Meldungsfenster mit der Schaltfläche JA beendet, starten Sie Word mithilfe der `CreateObject`-Funktion.

④ Verwenden Sie die `Visible`-Eigenschaften, um die Anwendung sichtbar zu machen.

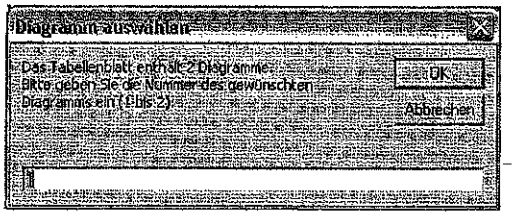
⑤ Geben Sie den Speicherplatz des Automation-Objekts frei, ohne Word dabei zu schließen.

Übung 3

Übungsdatei: *Übung2-E.ppt* **Ergebnisdatei:** *Übung3-E.ppt*

① Die Ereignisprozedur `btn_Erstellen_Click` im UserForm-Dialog `frm_Diagramm` kopiert stets das erste Diagramm eines Tabellenblatts. Die Prozedur soll so geändert werden, dass der Anwender das gewünschte Diagramm auswählen kann.

② Deklarieren Sie die Variablen `DiaNr` für die Auswahl einer Diagrammnummer und `Meldung` für das Erstellen eines Meldungstextes. `DiaNr` soll außerdem standardmäßig den Wert 1 enthalten.



Eingabe der Nummer des gewünschten Diagramms

③ Prüfen Sie mit einer `If`-Anweisung, ob mehr als ein Diagramm im Tabellenblatt enthalten ist.

④ Erstellen Sie einen Meldungstext, der den Anwender auf die Anzahl der vorhandenen Diagramme hinweist. Der Anwender soll außerdem aufgefordert werden, die Nummer des gewünschten Diagramms einzugeben.

⑤ Zeigen Sie einen Eingabedialog an, der den Meldungstext enthält. Als Vorgabe soll der Wert 1 eingetragen sein. Den Rückgabewert des Eingabedialogs speichern Sie in der Variablen `DiaNr`.

⑥ Wenn der Eingabedialog mit der Schaltfläche ABBRECHEN beendet wird, enthält der Rückgabewert eine leere Zeichenkette. Prüfen Sie dies, und brechen Sie die Prozedur in diesem Fall ab.

⑦ Verändern Sie die Anweisung aus ⑥ so, dass gleichzeitig geprüft wird, ob die Diagrammnummer höher ist als die Zahl der verfügbaren Diagramme. Brechen Sie auch in diesem Fall die Prozedur ab.

⑧ Kopieren Sie das ausgewählte Diagramm in die Zwischenablage.

Notizen

15 Integrierte Lösungen mit Outlook

In diesem Kapitel erfahren Sie

→ welche Möglichkeiten es zur Automation mit Outlook gibt
→ wie Sie Notizen, E-Mails und Aufgaben in Outlook erstellen
→ wie Sie in Word Serien-E-Mails mit Outlook versenden

Voraussetzungen

✓ Programmieren von Automationsobjekten
✓ Outlook-Objektmodell

15.1 Automation mit Outlook

Outlook ist eine sehr umfangreiche Anwendung. Neben der Organisation der elektronischen Kommunikation gehören dazu auch Funktionen wie Kalender, Aufgabenplanung, Notizen und Adressbücher. Frühere Outlook-Versionen konnten mit VBA nur schwer bzw. gar nicht programmiert werden. Seit Outlook 2002 ist jedoch der programmierbare Zugriff auf alle Objekte der Anwendung möglich.

Die VBA-Programmierung in Outlook ist aufgrund der komplexen und umständlichen Objektstruktur schwieriger als in anderen Office-Anwendungen. Prinzipiell haben Sie mit VBA die Möglichkeit, auf alle Outlook-Elemente zuzugreifen, beispielsweise auf Kontakte, E-Mails oder Termine, Sie können z. B. Kontakte auslesen und verändern oder E-Mails senden.

Da Outlook im Gegensatz zu den anderen Office-Anwendungen keine Dokumente besitzt, wird der VBA-Code der Anwendung zugeordnet und in der Datei *VbaProject.OTM* gespeichert. Der Speicherort ist vom verwendeten Betriebssystem abhängig.

Da Outlook alle Objekte auch als Automation-Server zur Verfügung stellt, können Sie die vielfältigen Möglichkeiten von Outlook in anderen Office-Anwendungen nutzen. So ist es z. B. möglich, ein Word-Dokument als E-Mail zu senden oder die in Outlook gespeicherten Kontakte in eine Datenbank aufzunehmen.

Bei der Erstellung integrierter Lösungen mit Outlook bietet sich die Verwendung von Outlook als Automation-Server an. Sie können so mithilfe von VBA die verschiedenen Möglichkeiten von Outlook in anderen Anwendungen nutzen.

Folgende beispielhafte Möglichkeiten ergeben sich für integrierte Lösungen mit Outlook als Server:

Word als Client	☑	Auslesen von Kontaktadressen für Word-Serienbriefe
	☑	Verwenden der Terminfunktion von Outlook zur Wiedervorlage von Dokumenten
Access als Client	☑	Auslesen und Verändern der Kontaktinformationen
	☑	Versenden von Serien-E-Mails auf Basis einer Datenbank
Excel als Client	☑	Auslesen der Kontakte oder Termine zum Erstellen von Tabellen

In den folgenden Abschnitten wird die Erstellung von integrierten Lösungen mit zwei Beispielen beschrieben, die Outlook aus Word heraus verwenden. Dabei werden Notizzettel erstellt und Serien-E-Mails versendet.

15.2 Notizen in Word erstellen

Aufgabe der Anwendung *NotizZettel.doc*

Diese VBA-Anwendung erstellt aus dem markierten Text eines Word-Dokuments einen Outlook-Notizzettel, damit z. B. eine wichtige Information zur späteren Verwendung vorgemerkt werden kann.

Outlook verfügt über den Ordner NOTIZEN, in dem kurze Textinformationen gespeichert werden können. Die VBA-Anwendung erstellt z. B. nach dem Betätigen einer Schaltfläche automatisch eine neue Notiz und zeigt diese am Bildschirm an.

VBA-Code der Anwendung

Die Sub-Prozedur verwendet ein Automationsobjekt zur Kommunikation mit Outlook und erstellt aus dem markierten Text eine neue Notiz, die sofort am Bildschirm angezeigt wird.

```
Sub NotizErstellen()
On Error GoTo Fehlerbehandlung
  Dim AppOL As Outlook.Application, NotizOL As Outlook.NoteItem
  Dim NotizText As String
  Set AppOL = GetObject(, "Outlook.Application")
  Set NotizOL = AppOL.CreateItem(olNoteItem)
  If Selection.Type = wdNoSelection Or Selection.Type = wdSelectionIP Then
    Selection.WholeStory
  End If
  NotizText = Selection.Range.Text
  NotizOL.Body = NotizText
  NotizOL.Display
Fehlerbehandlung:
  If Err = 429 Then
    Set AppOL = CreateObject("Outlook.Application")
  End If
  Resume Next
End Sub
```

① Für den Zugriff auf Outlook wird eine Objektvariable für das `Outlook.Application`-Objekt erstellt. Zusätzlich wird eine Objektvariable benötigt, die eine einzelne Notiz in Outlook repräsentiert.

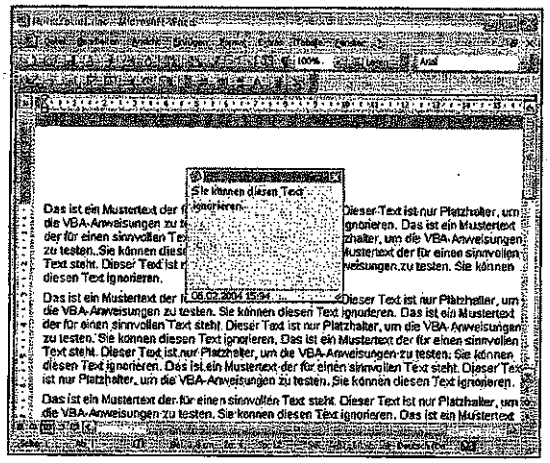
② Mithilfe der Funktion `GetObject` wird versucht, auf eine bereits gestartete Outlook-Anwendung zuzugreifen. Ist Outlook nicht geöffnet, wird ein Fehler erzeugt, den die Sprungmarke `Fehlerbehandlung` abfängt, und Outlook startet.

③ Die `CreateItem`-Methode ermöglicht das Erstellen neuer Outlook-Elemente. In diesem Fall wird eine neue Notiz erzeugt und ein entsprechender Verweis mithilfe der Objektvariablen `NotizOL` gespeichert.

④ Zuerst erfolgt eine Überprüfung, ob eine Textstelle im aktuellen Dokument markiert wurde. Ist dies nicht der Fall, wird der gesamte Text des Dokuments markiert. Danach wird der markierte Text in die Variable `NotizText` eingelesen.

⑤ Der Text der Notiz wird mit der Eigenschaft `Body` festgelegt.

⑥ Die `Display`-Methode zeigt die erstellte Notiz am Bildschirm an.



Das Word-Dokument mit der erstellten Notiz

15.3 Serien-E-Mails mit Word und Outlook versenden

15.3.1 Mit der Anwendung SERIEN-E-MAIL arbeiten

Aufgabe der Anwendung

Die in diesem Abschnitt beschriebene Anwendung SERIEN-E-MAIL erlaubt das Versenden von Serien-E-Mails an ausgewählte Empfänger aus den Outlook-Kontakten. Standardmäßig wird dabei das aktive Word-Dokument als Dateianhang verschickt. Die Anwendung soll nach der Auswahl eines Kontakteordners die darin gespeicherten Adressen anzeigen. Von diesen Adressen können beliebig viele als Empfänger der Serien-E-Mail ausgewählt werden. Nach der Eingabe eines Betreffs und eines Nachrichtentexts wird mithilfe von Outlook an alle Empfänger eine E-Mail mit dem Word-Dokument als Dateianhang gesendet.

Word arbeitet in der Anwendung als Automation-Client mit Outlook als Automation-Server. Die Steuerung der Anwendung erfolgt über einen UserForm-Dialog. Das Modul dieses Dialogs enthält alle Prozeduren der Anwendung. Die funktionsfähige Anwendung finden Sie unter dem Namen *Serien-E-Mail.doc*.

Word verfügt mit dem Menübefehl DATEI - SENDEN AN - VERTEILEREMPFÄNGER bereits über eine Standardfunktion zum Versenden des aktuellen Dokuments an mehrere Empfänger. Im Gegensatz zu dieser vorgegebenen Funktion können Sie die VBA-Anwendung SERIEN-E-MAIL beliebig erweitern und an eigene Anforderungen anpassen, um z. B. personalisierte E-Mails zu versenden oder gleichzeitig mit dem Senden eine Aufgabe im Kalender zu erstellen.

Anwendungsstart über Menübefehl

Die Anwendung wird über den Menübefehl SERIEN-E-MAIL gestartet.

Der Menübefehl zum Starten der Anwendung

Arbeiten mit der Anwendung

Für das Arbeiten mit der Anwendung SERIEN-E-MAIL ist folgender Arbeitsablauf notwendig:

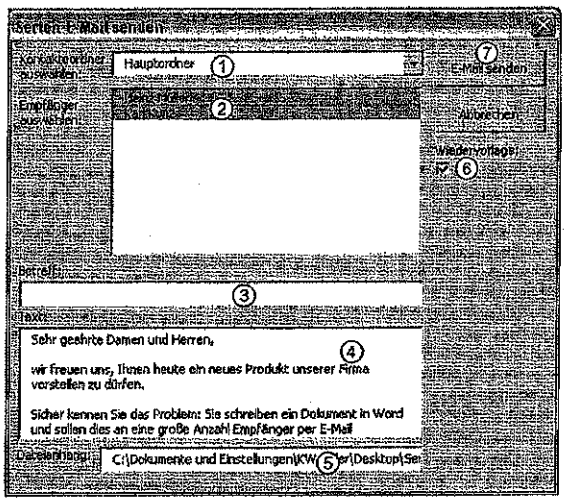
⇒ Wählen Sie den Menübefehl SERIEN-E-MAIL, um die Anwendung zu starten. Der UserForm-Dialog SERIEN-E-MAIL SENDEN wird angezeigt.

⇒ Wählen Sie im LISTENFELD KONTAKTEORDNER AUSWÄHLEN ① den gewünschten Kontakteordner von Outlook aus. Die darin gespeicherten Kontakte werden im Listenfeld EMPFÄNGER AUSWÄHLEN ② angezeigt.

⇒ Wählen Sie im Listenfeld ② die Empfänger der E-Mail.

⇒ Geben Sie im Textfeld BETREFF ③ den Betreff der E-Mail ein.

⇒ Geben Sie im Textfeld TEXT ④ den Nachrichtentext ein. Mit der Tastenkombination STRG ENTER ↵ erzeugen Sie einen Zeilenumbruch.



Die Benutzeroberfläche der Anwendung

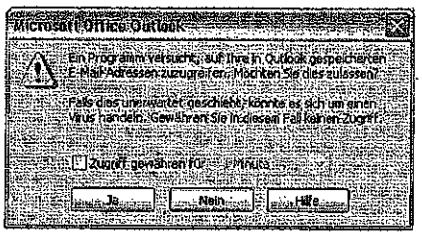
⇒ Verändern Sie bei Bedarf den Pfad für den Dateianhang im Textfeld DATEIANHANG ⑤.

⇒ Auf Wunsch kann automatisch eine Aufgabe im Kalender von Outlook erstellt werden, die als Wiedervorlage auf das versendete Dokument und die Empfänger hinweist. Aktivieren Sie dazu das Optionsfeld WIEDERVORLAGE ⑥.

⇒ Betätigen Sie die Schaltfläche E-MAIL SENDEN ⑦, um das Versenden der E-Mails zu starten. Mit der Schaltfläche ABBRECHEN beenden Sie den Dialog, ohne eine Aktion auszuführen.

Outlook-Sicherheitswarnungen

Da viele neue Viren in der Vergangenheit im Hintergrund über Outlook E-Mails versendet haben, hat Microsoft seit Outlook 2002 die Sicherheitsbestimmungen für den Zugriff auf Outlook-Elemente per VBA verstärkt. Aus diesem Grund warnt Outlook bei jedem externen Zugriff auf den Kontakteordner oder beim Versenden von E-Mails.

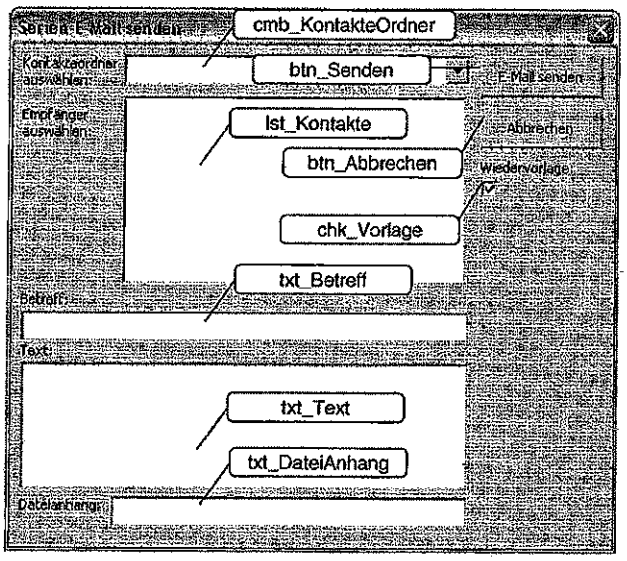


Outlook-Warnung beim Zugriff per Automation

Diese Warnungen verhindern zwar das Verbreiten von E-Mail-Viren, jedoch behindern Sie auch die Automation und Programmierung von Outlook. Leider gibt es bis zum aktuellen Zeitpunkt keine Möglichkeit, diese Warnungen zu deaktivieren. Es ist auch nicht möglich, die Warnungen, beispielsweise mit der `SendKeys`-Anweisung, automatisch zu bestätigen.

Steuerelemente und Prozeduren des UserForm-Dialogs

Die Anwendung besteht nur aus dem UserForm-Dialog `frm_SerienEmail`. Es enthält folgende Steuerelemente:



Damit im Listenfeld `lst_Kontakte` eine Mehrfachauswahl möglich ist, muss im Eigenschaftenfenster für die Eigenschaft `MultiSelect` der Wert `fmMultiSelectMulti` ausgewählt sein. Auch das Textfeld `txt_Text` benötigt das Bearbeiten der Eigenschaften, damit die Eingabe von mehreren Zeilen möglich ist. Für die Eigenschaft `MultiLine` muss der Wert `True` gewählt werden.

Notizen

Folgende Prozeduren sind im Modul des UserForm-Dialogs enthalten:

UserForm_Activate	Initialisiert den UserForm-Dialog, startet Outlook und zeigt die verfügbaren Kontakteordner an
cmb_KontakteOrdner_Change	Zeigt nach Auswahl eines Kontakteordners die darin enthaltenen Kontakte an
btn_Senden_Click	Startet nach dem Betätigen der Schaltfläche das Senden der E-Mails und wertet die ausgewählten Empfänger aus
E-Mail Senden	Sendet die E-Mail an einen Empfänger
UserForm_Deactivate	Schließt beim Beenden des Dialogs Outlook und gibt den Speicherplatz des Automation-Objekts frei
btn_Abbrechen_Click	Beendet den Dialog ohne weitere Aktion
WiedervorlageErstellen	Erstellt eine neue Aufgabe im Outlook-Kalender

Die Prozeduren benötigen für die Funktion einen Verweis auf die Microsoft-Outlook 11.0-Objektbibliothek.

15.3.2 Initialisieren der Anwendung und Bestimmen der Kontakteordner

Beim Start des UserForm-Dialogs wird die Ereignisprozedur `UserForm_Activate` ausgeführt. Sie startet Outlook und liest die vorhandenen Kontakteordner aus. Dazu werden alle Ordner bestimmt, die dem Standardordner für Kontakte in Outlook untergeordnet sind.

```
Private Sub UserForm_Activate()
On Error GoTo Fehlerbehandlung
  Dim KontakteOL As Outlook.MAPIFolder, OrdnerOL As Outlook.MAPIFolder
①  Set AppOL = GetObject(, "Outlook.Application")
②  Set NameSpaceOL = AppOL.GetNamespace("MAPI")
③  Set KontakteOL = NameSpaceOL.GetDefaultFolder(olFolderContacts)
④  Me.cmb_KontakteOrdner.AddItem "Hauptordner"
⑤  For Each OrdnerOL In KontakteOL.Folders
⑥    Me.cmb_KontakteOrdner.AddItem OrdnerOL.Name
⑦  Next
⑧  Me.cmb_KontakteOrdner.ListIndex = 0
⑨  Me.txt_Anhang = ActiveDocument.FullName
  Exit Sub
Fehlerbehandlung:
  If Err = 429 Then
    Set AppOL = CreateObject("Outlook.Application")
  End If
  Resume Next
End Sub
```

① Für den Zugriff auf die Kontakteordner werden zwei Objektvariablen vom Typ `MAPIFolder` benötigt.

② Mithilfe der `GetObject`-Funktion wird versucht, auf ein bereits gestartetes Outlook zuzugreifen. Ist Outlook noch nicht aktiv, wird an der Sprungmarke `Fehlerbehandlung` der Fehler abgefangen und Outlook gestartet. Die Variable `AppOL` wird in mehreren Prozeduren verwendet und wurde daher im Deklarationsteil des Moduls definiert.

③ Für den Zugriff auf die in Outlook verfügbaren Elemente wird stets ein `NameSpace`-Objekt benötigt. Die Methode `GetNamespace` liefert den benötigten Verweis. Auch die Variable `NameSpaceOl` wurde im Deklarationsteil des Moduls definiert.

④ Über das `NameSpace`-Objekt kann auf die Kontakteordner zugegriffen werden. Mit der Methode `GetDefaultFolder` wird hier ein Verweis auf den Standard-Kontakteordner erstellt.

⑤ Für den Standard-Kontakteordner wird im Listenfeld `lst_KontakteOrdner` der Eintrag HAUPTORDNER erzeugt.

⑥ Die Auflistung `Folders` enthält alle Unterordner als `MAPIFolder`-Objekte. Mit einer `For-Each`-Schleife werden diese nacheinander durchlaufen.

⑦ Für jeden Unterordner wird dem Listenfeld der Name des Ordners als neuer Eintrag hinzugefügt.

⑧ Standardmäßig soll der Eintrag HAUPTORDNER im Listenfeld ausgewählt sein. Dazu wird der Eigenschaft `ListIndex` der Indexwert dieses Eintrags zugewiesen (0).

⑨ Dem Textfeld `txt_Anhang` wird der Dateipfad des aktuellen Word-Dokuments zugewiesen.

Für den Zugriff auf Outlook-Elemente wird im Gegensatz zu den anderen Office-Anwendungen nicht nur ein `Application`-Objekt, sondern auch ein `NameSpace`-Objekt benötigt.

Ein `NameSpace`-Objekt wird stets nach folgendem Muster erzeugt:

```
Set Objektvariable = Application.GetNamespace("MAPI")
```

Als Argument für die `GetNamespace`-Funktion ist ausschließlich die Zeichenkette `MAPI` erlaubt.

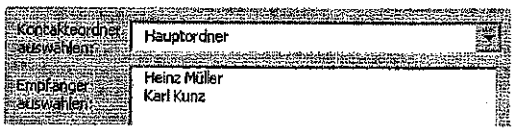
15.3.3 Kontaktinformationen auslesen

Wird im Listenfeld `cmb_KontakteOrdner` ein Kontakteordner ausgewählt, ermittelt die zugehörige Ereignisprozedur alle in diesem Ordner gespeicherten Kontakte und zeigt die Namen im Listenfeld `lst_Kontakte` an.

```
Private Sub cmb_KontakteOrdner_Change()
①  Dim OrdnerOL As Outlook.MAPIFolder, KontaktOL As Outlook.ContactItem
   Me.lst_Kontakte.Clear
②  If Me.cmb_KontakteOrdner.ListIndex > 0 Then
③    Set OrdnerOL = NameSpaceOL.GetDefaultFolder(olFolderContacts)
   Else
④    Set OrdnerOL = NameSpaceOL.GetDefaultFolder(olFolderContacts). _
                                  Folders(Me.cmb_KontakteOrdner.ListIndex)
   End If
⑤  For Each KontaktOL In OrdnerOL.Items
⑥    Me.lst_Kontakte.AddItem KontaktOL.FirstName & " " & KontaktOL.LastName
   Next
End Sub
```

① Für den Zugriff auf den Kontakteordner wird eine Objektvariable vom Typ `MAPIFolder` benötigt. Für den Zugriff auf Kontakteinträge lautet der Objekttyp `ContactItem`.

② Hier wird überprüft, ob der erste Eintrag des Listenfelds ausgewählt wurde. Dieser Eintrag (Indexwert 0) steht für den Hauptordner.



Im Listenfeld werden Vor- und Nachname angezeigt

Notizen

③ Der Zugriff auf den Hauptordner erfolgt über die `Get DefaultFolder`-Methode und mit dem Argument `olFolderContacts`. Die Variable `OrdnerOl` erhält einen Verweis auf diesen Ordner.

④ Falls nicht der Hauptordner ausgewählt wurde, muss ein Unterordner ausgelesen werden. Die `Folders`-Auflistung erlaubt den Zugriff auf diese Ordner. Der Indexwert der Auswahl im Listenfeld bestimmt dabei, um welchen Ordner es sich handelt.

⑤ Nacheinander werden alle Kontakteinträge des gewählten Ordners durchlaufen. Die Einträge sind in der `Items`-Auflistung enthalten. Jeder Kontakteintrag besitzt den Objekttyp `ContactItem`.

⑥ Die Eigenschaften `FirstName` und `LastName` liefern den Vor- und Nachnamen des Kontakteintrags. Beide Werte werden verbunden und als neuer Eintrag dem Listenfeld `lst_Kontakte` hinzugefügt.

Die `NameSpace`-Methode `GetDefaultFolder` ermöglicht den Zugriff auf alle Standardordner in Outlook, beispielsweise den Posteingang, Termine oder Kontakte. Alle Standardordner können weitere Unterordner enthalten, die in der `Folders`-Auflistung enthalten sind.

```
Set Ordnerobjekt = NameSpaceObjekt.GetDefaultFolder(Ordnerkonstante).Folders
```

15.3.4 E-Mails an ausgewählte Empfänger senden

Mit dem Betätigen der Schaltfläche E-MAIL SENDEN werden in der zugehörigen Ereignisprozedur die ausgewählten Empfänger aus dem Listenfeld `lst_Kontakte` ausgelesen. Für jeden Empfänger wird mithilfe der Prozedur `EmailSenden` eine eigene E-Mail erstellt und verschickt. Falls gewünscht, wird danach über die Prozedur `WiedervorlageErstellen` eine neue Aufgabe im Kalender erstellt.

Empfänger in der Ereignisprozedur bestimmen

```
Private Sub btn_Senden_Click()
  Dim Zähler As Long, Index As Long, KontaktOL As Outlook.ContactItem
  Dim OrdnerOL As Outlook.MAPIFolder, EmailAdressen As String, Email As String
④ If Me.txt_Anhang <> "" And Dir(Me.txt_Anhang) = "" Then
    MsgBox "Sie haben keinen gültigen Dateipfad eingegeben!", _
    vbExclamation Me.txt_Anhang.SetFocus
    Exit Sub
  End If
  If Me.cmb_KontakteOrdner.ListIndex = 0 Then
②   Set OrdnerOL = NameSpaceOL.GetDefaultFolder(olFolderContacts)
  Else
③   Set OrdnerOL = NameSpaceOL.GetDefaultFolder(olFolderContacts). _
                                Folders(Me.cmb_KontakteOrdner.ListIndex)
  End If
④ For Index = 0 To Me.lst_Kontakte.ListCount - 1
⑤   If Me.lst_Kontakte.Selected(Index) = True Then
⑥     Set KontaktOL = OrdnerOL.Items(Index + 1)
⑦     Email = KontaktOL.Email1Address
⑧     EmailSenden Email, Me.txt_Betreff, Me.txt_Text, Me.txt_Anhang
      EmailAdressen = EmailAdressen & vbCrLf & Email
      Zähler = Zähler + 1
    End If
  Next
  Me.Caption = Me.Caption & " - " & Zähler & " E-Mails verschickt!"
⑨ If Me.chk_Vorlage.Value = True Then
    WiedervorlageErstellen EmailAdressen
  End If
End Sub
```

① Zu Beginn der Ereignisprozedur wird überprüft, ob ein Pfad für den Dateianhang eingegeben wurde und ob dieser gültig ist. Die `Dir`-Funktion liefert eine leere Zeichenkette, wenn die angegebene Datei nicht existiert. In diesem Fall wird die Prozedur mit einer Warnung abgebrochen.

② Das Listenfeld `cmb_KontakteOrdner` bestimmt, welcher Ordner für das Auslesen der Adressen verwendet wird. Ist der Eintrag *Hauptordner* ausgewählt, liefert die Eigenschaft `ListIndex` den Wert 0, und der Standardordner wird geöffnet.

③ Für alle anderen Ordner muss das Ordnerobjekt aus der `Folders`-Auflistung ermittelt werden.

④ Um zu ermitteln, welche Empfänger ausgewählt wurden, müssen alle Einträge des Listenfelds `lst_Kontakte` nacheinander durchlaufen werden. Für diesen Zweck wird hier eine `For-Next`-Schleife initialisiert. Die Eigenschaft `ListCount` liefert die Anzahl der Listenelemente.

⑤ Die Eigenschaft `Selected` liefert den Wert `True`, wenn der betreffende Listeneintrag ausgewählt wurde.

⑥ Der Zugriff auf einzelne Kontakte des Ordners erfolgt über die `Items`-Auflistung. Der entsprechende Indexwert wird aus dem Listenfeld `lst_Kontakte` ermittelt. Da die Zählung im Listenfeld bei 0 und in der `Items`-Auflistung bei 1 beginnt, muss der Indexwert um den Wert 1 erhöht werden.

⑦ Die E-Mail-Adresse wird über die Eigenschaft `EmailAdresse` ermittelt und in der Variablen `Email` gespeichert.

⑧ Die Prozedur `EmailSenden` erstellt eine E-Mail und verschickt sie. Zu diesem Zweck wird die E-Mail-Adresse zusammen mit anderen Argumenten übergeben.

⑨ Ist das Erstellen einer Wiedervorlage erwünscht, wird hier die entsprechende Prozedur aufgerufen. Als Argument erhält sie die Liste mit den bearbeiteten E-Mail-Adressen.

E-Mails senden

```
① Private Sub EmailSenden(Email As String, Betreff As String, _
                          Text As String, Optional DateiAnhang As String)
②   Dim NachrichtOL As Outlook.MailItem
③   Set NachrichtOL = AppOL.CreateItem(olMailItem)
    With NachrichtOL
④     .Recipients.Add Email
⑤     .Subject = Betreff
⑥     .Body = Text
      If DateiAnhang <> "" Then
⑦       .Attachments.Add Source:=DateiAnhang, Position:=Len(Text) + 20
      End If
⑧     .Send
    End With
End Sub
```

① Die Prozedur erwartet mehrere Argumente, um die Nachricht zu erstellen: E-Mail-Adresse, Betreff, Nachrichtentext und einen Pfad für den Dateianhang. Mit dem Schlüsselwort `Optional` wird festgelegt, dass das Argument `DateiAnhang` auch entfallen kann.

② Jede E-Mail-Nachricht besitzt den Objekttyp `MailItem`.

Notizen

③ Die Methode `CreateItem` des Outlook-`Application`-Objekts erstellt ein neues Outlook-Element. Mit der Konstante `olMailItem` wird festgelegt, dass eine E-Mail-Nachricht erstellt werden soll.

④ Die `Recipients`-Auflistung der E-Mail-Nachricht enthält die Empfänger der Nachricht. Mit der `Add`-Methode wird hier die übergebene E-Mail-Adresse hinzugefügt.

⑤ Mithilfe der `Subject`-Eigenschaft wird die Betreffzeile festgelegt.

⑥ Die `Body`-Eigenschaft verweist auf den Nachrichtentext.

⑦ Falls ein Dateipfad übergeben wurde, kann er als so genanntes Attachment (Dateianhang) eingetragen werden. Dazu dient die `Add`-Methode der `Attachments`-Auflistung. Das Argument `Source` bestimmt den Dateipfad. Mit dem Argument `Position` kann festgelegt werden, wo sich das Symbol des Dateianhangs im Nachrichtentext befinden soll.

⑧ Die Methode `Send` sendet die vorbereitete E-Mail-Nachricht.

Falls zum Zeitpunkt des Sendens keine Internetverbindung besteht, speichert Outlook die gesendeten Nachrichten zunächst im Ordner POSTAUSGANG und sendet sie zum nächstmöglichen Zeitpunkt.

Mit der `CreateItem`-Methode können Sie alle in Outlook möglichen Elemente erzeugen, beispielsweise E-Mails, Kontakte oder Notizen. Jedes Element verfügt über einen eigenen Objekttyp, z. B. `MailItem` oder `ContactItem`.

Wiedervorlage als Aufgabe erstellen

```
Sub WiedervorlageErstellen(EmailAdressen)
  Dim AufgabeOl As Outlook.TaskItem
①
  Set AufgabeOl = AppOL.CreateItem(olTaskItem)
②
  AufgabeOl.Subject = "Wiedervorlage Serien-E-Mail-Dokument " & _
③                     ActiveDocument.Name
  AufgabeOl.Body = "Das Dokument " & ActiveDocument.Name & " wurde am " & _
④                  Now() & " als Serien-E-Mail versendet." & _
                   vbCrLf & "Empfänger:" & vbCrLf & EmailAdressen & vbCrLf
  AufgabeOl.Attachments.Add Source:=ActiveDocument.Path & "\" & _
⑤                  ActiveDocument.Name, Position:=Len(AufgabeOl.Body) + 20
⑥ AufgabeOl.Display
End Sub
```

① Alle Aufgaben im Kalenderordner besitzen den Objekttyp `TaskItem`.

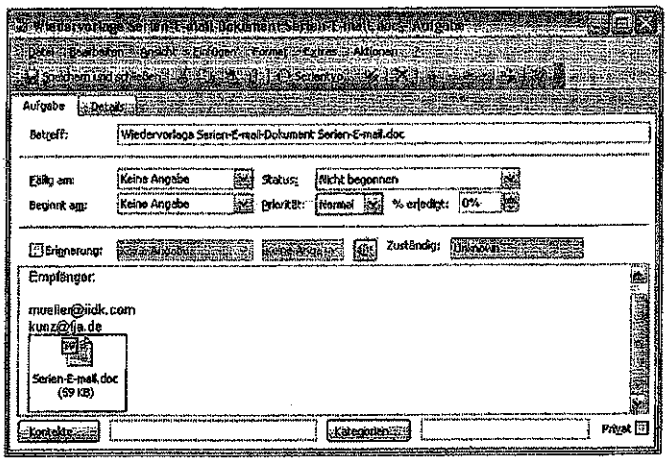
② Mit der Methode `CreateItem` und dem Parameter `olTaskItem` wird eine neue Aufgabe erstellt und der Objektvariablen `AufgabeOl` der entsprechende Verweis zugewiesen.

③ Über die `Subject`-Eigenschaft wird der Aufgabenbetreff festgelegt.

④ Hier wird der Text der Aufgabe festgelegt. Dazu werden der Name des versendeten Dokuments, das aktuelle Datum und die E-Mail-Adressen der Empfänger eingefügt.

⑤ Mit der `Add`-Methode der `Attachments`-Auflistung wird das aktuelle Dokument der Aufgabe als Anhang hinzugefügt. Dadurch kann es später schnell geöffnet werden.

⑥ Die Aufgabe wird noch nicht gespeichert, sondern am Bildschirm angezeigt. So kann vom Anwender ein Fälligkeitsdatum festgelegt werden.



Die Prozedur erstellt automatisch eine Aufgabe

15.4 Schnellübersicht

Sie möchten ...	
eine Mehrfachauswahl in einem Listenfeld einstellen	Eigenschaft `MultiSelect`
mehrere Zeilen in einem Textfeld eingeben	Eigenschaft `MultiLine`
einen Verweis auf das Outlook-NameSpace-Objekt erstellen	`Set NameSpaceObjekt=Application.GetNamespace("MAPI")`
auf einen Standardordner in Outlook zugreifen	`Set OrdnerObjekt = NameSpaceObjekt.GetDefaultFolder(Ordnerkonstante)`
auf einen Unterordner in Outlook zugreifen	`OrdnerObjekt.Folders(Index)`
auf Elemente eines Ordners zugreifen	`OrdnerObjekt.Items(Index)`
eine E-Mail erstellen	`Set MailItemObjekt=Application.CreateItem(olMailItem)`
einen E-Mail-Empfänger festlegen	`MailItemObjekt.Recipients.Add Email-Adresse`
den Betreff und den Text einer Nachricht festlegen	`MailItemObjekt.Subject = Text` `MailItemObjekt.Body = Text`
einen Dateianhang einfügen	`MailItemObjekt.Attachments.Add Dateipfad, Position`
eine E-Mail senden	`MailItemObjekt.Send`
eine Aufgabe erstellen	`Set TaskItemObjekt=Application.CreateItem(olTaskItem)`
eine Notiz erstellen	`Set TaskItemObjekt=Application.CreateItem(olNoteItem)`

Notizen

15.5 Übung

Übung 1

Übungsdatei: *Übung1.doc* **Ergebnisdatei:** *Übung1-E.doc*

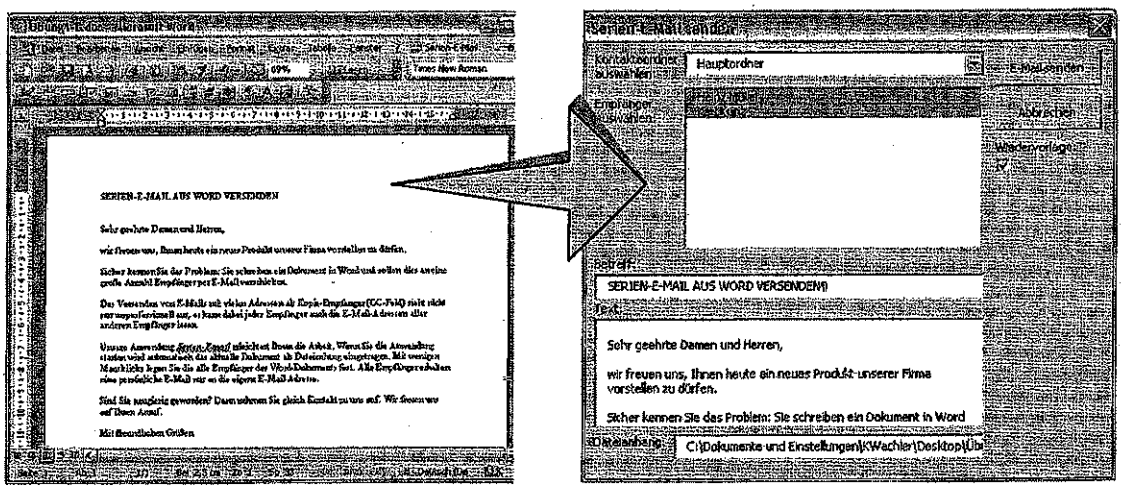
① Öffnen Sie das Dokument *Übung1.doc*, und starten Sie die VBA-Entwicklungsumgebung.

② Beim Start des UserForm-Dialogs soll automatisch die erste Zeile des Dokuments als Betreff und der weitere Text als Nachrichtentext angezeigt werden. Bearbeiten Sie dazu die geeignete Ereignisprozedur.

③ Es wird davon ausgegangen, dass die Betreffzeile mit einem Zeilenumbruch abgeschlossen wird. Nach einer Leerzeile beginnt der Nachrichtentext.

④ Erstellen Sie eine Anweisung zum Auslesen der Betreffzeile. Verwenden Sie dafür die Paragraphs-Auflistung.

⑤ Lesen Sie den Text nach der Leerzeile bis zum Ende aus. Verwenden Sie dazu das Range-Objekt mit einer geeigneten Startposition.



Betreffzeile und Nachrichtentext werden automatisch erkannt und als Voreinstellung angezeigt

Übung 2

Übungsdatei: *Übung1-E.doc* **Ergebnisdatei:** *Übung2-E.doc*

① Es soll eine Prozedur erstellt werden, die die Anrede im Nachrichtentext der E-Mail automatisch erstellt. Dazu wird in den Kontaktinformationen Outlooks das Feld TITEL ausgewertet.

② Erstellen Sie im Modul eine neue `Function`-Prozedur `Anrede`. Der Prozedur soll als Argument ein Outlook-Kontakt übergeben werden. Als Rückgabewert liefert die Prozedur die Anrede als Zeichenkette.

③ Werten Sie in einem `If`-Ausdruck das Feld ANREDE (`ContactItem`-Eigenschaft `Title`) aus. Erstellen Sie je nach Inhalt dieses Feldes eine passende Anrede, beispielsweise *Sehr geehrte Frau*. Fügen Sie danach den Nachnamen ein.

④ Falls kein Titel vorhanden ist, soll eine neutrale Anrede ohne Namen verwendet werden.

⑤ Fügen Sie der Anrede ein Komma und zwei Zeilenumbrüche hinzu, um die Anrede abzuschließen und eine Leerzeile zu erzeugen. Übergeben Sie die Anrede als Rückgabewert.

⑥ Verändern Sie die Ereignisprozedur für die Schaltfläche E-MAIL SENDEN so, dass die Anrede in den Nachrichtentext eingefügt wird.

Notizen

A

Access, Automation 150
Access, Berichte ansprechen 85
Access, Besonderheiten 18
Access, Formulare ansprechen 85
Access, Steuerelemente ansprechen 86
Access-Basic 10
Access-Datenbanken, Zugriff 84
Access-Objektmodell 84
ActiveX 95
ActiveX-Datenobjekte 84, 118
ADO 95, 135
ADO .NET 118
ADO, Datenzugriff 119
ADO, Verbindung erstellen 120
ADO-Objekte 150
ADO-Objektmodell 84, 118
ADO-Objektmodell aktivieren 118
ADO-Objektmodell, Hierarchie 119
Animation 175
Animationseffekt 175
Anweisungen eingeben 22
Anwendungsfenster 62
Anwendungsfenster, Anzeigemodus 63
Anwendungsfenster, Größe ändern 63
Anwendungsfenster, Position ändern 63
Anwendungsfenster, Titelleiste 64
Anwendungsname ermitteln 64
Anzeige 109
`Application` 50, 51, 62
Arbeitsmappen öffnen 80
Arbeitsmappen schließen 80
Arbeitsmappen, Zugriff 80
Arbeitsspeicher freigeben 101
Argumente 36
Arrays 41
Arrays deklarieren 41
Arrays, dynamische 42
Arrays, mehrdimensionale 41
`As` 41
Auflistung 56
Auflistung, einzelnes Objekt 56
Auflistung, Namen verwenden 57
Auflistung, Zugriff 57
Auswahl, einseitige 43
Auswahl, mehrstufige 43
Auswahl, zweiseitige 43
Automation 95
Automation mit VBA 96
Automationsserver 95
Automatische Direkthilfe 25
Automatische Syntaxüberprüfung 24

B

Befehlsschaltfläche 109
Benutzerabfragen 70
Berichtsmodule 33
Bezeichnungsfelder 108
Bibliotheken 59
Bildlaufleisten 109
Bildschirmaktualisierung unterdrücken 104
Binden 96
Binden, frühes 97
Binden, spätes 97, 164
`BOF` 124
`Boolean` 40
`BuiltInDocumentProperties` 158
`ByRef` 36
`ByVal` 36

C

`Chart` 177
Codefenster 22
COM 94, 95
COM+ 95
Connection 119
`Const` 39
`Copy` 135
`CreateItem` 190
`CreateObject` 99
`Currency` 40
Cursor 123
Cursortyp 123

D

DAO 95
DAO-Objektmodell 118
`Date` 40
Dateiauswahl 68
Dateien, suchen 67
Dateisuche 179
Datenbank, Verbindung öffnen 121
Datenbank-Engine 84
Datenbankfenster 18
Datenbankzugriff 135
Datenfelder 41, 123
Datenfelder deklarieren 41
Datenfelder, dynamische 42
Datenfelder, mehrdimensionale 41
Datenfelder, Zugriff 125
Datenquelle 123
Datensätze ändern 128
Datensätze auslesen 123
Datensätze hinzufügen 127
Datensätze löschen 129
Datensätze suchen 126
Datensatzgruppe 122
Datensatzzeiger 123, 159
Datentypen 40
Datenzugriff 94
Datenzugriff mit ADO 119
Datenzugriffsobjekte 84, 118
DCOM 95
DDE 95
Definition anzeigen 28
Deklaration, Konstanten 39
Deklaration, Variablen 38
`Delete` 129
Diagrammfunktion von Excel nutzen 132
Dialoge 106
Dialoge, benutzerdefinierte 106
Dialoge, integrierte 106
Dialoge, vordefinierte 106
Dialogfenster 70
`Dim` 38, 55
Direktfenster 30
Direkthilfe, automatische 25
`Document` 50
`Documents` 76
Dokumente suchen 154
Dokumente, Sichtbarkeit 77
Dokumenteneigenschaften 171
Dokumentenfenster 64
Dokumentenfenster aktivieren 65
Dokumentenfenster schließen 67
Dokumentenfenster, -aktuelles Fenster 65
Dokumentenfenster, Fensterzustand 65
Dokumentenfenster, Größe 66
Dokumentenfenster, neues 66
Dokumentenfenster, Position 66
Dokumentenmanagement-system 151
Dokumentenmodul 33, 51
Dokumentenverwaltung 151
`Double` 40
`Do-While/Until`-Schleife 45
Drehfelder 109
Druckfunktion, Word 141

E

Eigenschaften 50, 52
Eigenschaften, Steuerelemente 110
Eigenschaften, Syntax 52
Eigenschaftenfenster 21
Eingabedialog 70
Eingabefelder 114
Eingabehilfen 24, 26
Elemente automatisch auflisten 25
Elemente erstellen 91
Elemente, Zugriff 91
E-Mail-Viren 185
`Empty` 38
`EOF` 124
Ereignisprozeduren 35, 107, 111, 114
Ereignisprozeduren erstellen 113
Ereignisse 114
`Error` 86, 99, 100, 102, 119, 156, 157, 174, 186
Excel, Automation 140
Excel, Zellen programmieren 82
Excel-Objektmodell 80
`Exit` 46

F

Fallauswahl 43
`Field` 119
`FileDialog` 68, 179
`FileSearch` 67, 154
`FileSystem` 69
Folien wechseln 175
Folien, Zugriff 87, 88

Folieninhalt ... 88
For Each ... 58
Formatierung des VBA-Codes ... 24
Formen ... 88
Formen formatieren ... 88
Formen hinzufügen ... 88
Form ... 50
Forms ... 85
Formulare ... 106
Formulare, Access ... 106
Formularmodule ... 33
For-Next ... 45
Function-Prozeduren ... 35
Function-Prozeduren, Aufbau ... 36
Funktionen, vordefinierte ... 47

G

GetObject ... 99
Get-Prozedur ... 37
Gültigkeitsbereich ... 26

H

Haltepunkt ... 24

I

If-Then ... 43
If-Then-Else ... 43
If-Then-ElseIf ... 43
If ... 44
InputBox ... 70
Installation ... 6
Integer ... 40
Integration ... 12, 94
Integrierte Lösungen ... 11

J

Jet-Datenbank-Engine ... 121

K

Kalenderordner ... 190
Klassenmodule ... 33
Kombinationsfelder ... 114
Kommentare ... 24
Konstanten ... 25, 39
Konstanten anzeigen ... 26
Konstanten, integrierte ... 39
Konstantendeklaration ... 39
Kontakteordner ... 145, 184, 186
Kontrollfelder ... 108, 115
Kontrollstrukturen ... 43

L

Lesezeichen löschen ... 29
Lesezeichen setzen ... 29
Lesezeichen wechseln ... 29
Let-Prozedur ... 37
Listenfeld ... 108, 114
Listenfeld, Mehrfachauswahl ... 185
Lokalfenster ... 30
Long ... 40

M

Makros ... 10
Markierung ... 78
MDAC ... 121
Me ... 51
Mehrfachauswahl ... 185
Meldungen unterdrücken ... 103
Meldungsfenster ... 72
Meldungsfenster, Rückgabewert ... 73
Menübefehl erstellen ... 170
Menüleiste ... 15, 170
Methoden ... 50, 53
Methoden, Syntax ... 53
Modul ... 32
Modul bearbeiten ... 18, 34
Modul benennen ... 34
Modul erstellen ... 18
Modul exportieren ... 34
Modul importieren ... 34
Modul löschen ... 34
Modulfenster ... 22
Modulfenster teilen ... 28
Modulfenster, Ansicht ändern ... 28
Modultypen ... 33
MsgBox ... 70, 72
Multiseiten ... 109

N

NameSpace ... 90, 187
New ... 99, 100
Nothing ... 101
Notiz erstellen ... 183

O

Objekt ... 50
Objektauflistungen ... 56
Objektbibliotheken einbinden ... 59
Objektinstanz erzeugen ... 99
Objektkatalog ... 58
Objektklasse ... 97
Objektmodell ... 51
Objektmodell, Word ... 76
Objektvariablen ... 54
Objektvariablen zuweisen ... 55
Objektvariablen, Syntax ... 55
OCX ... 95
ODBC ... 94
Office-Anwendungen ... 12
Office-Objekte ... 50
OLE-Automatisierung ... 95
OLE-Objektfeld ... 156
Optionsfelder ... 109
Ordner, Zugriff ... 90
Ordnerauswahl ... 68
Outlook automatisieren ... 90
Outlook, Automation ... 182
Outlook, Elemente ... 91, 182
Outlook, Zugriff auf ... 144
Outlook-Objektmodell ... 90
Outlook-Ordner ... 90, 144

P

Parameter ... 25, 36
Parameterinfo ... 26
Paste ... 135
PowerPoint automatisieren ... 87
PowerPoint, Automation ... 168
PowerPoint, Textfeld ... 172
PowerPoint-Objektmodell ... 87
Präsentationen öffnen ... 87
Präsentationen schließen ... 87
Präsentationen, Zugriff ... 87
Presentation ... 50
PrintOut ... 143
Private ... 38
Programmierung, strukturierte ... 32
Programmschema, modulorientiertes ... 32
Projekt-Explorer ... 20
Property-Prozeduren ... 35
Provider ... 121
Prozedur ... 20, 34
Prozedur ändern ... 27
Prozedur ausführen ... 27
Prozedur erzeugen ... 26
Prozedur starten ... 23
Prozedur testen ... 27
Prozedur, Argumente ... 36
Prozedur, Definition anzeigen ... 28
Prozedurdefinition ... 36
Prozedurtypen ... 34
Public ... 38

Q

QuickInfo ... 24, 26

R

Rahmen ... 109
Range ... 50, 79
Recordset ... 119, 122
Recordset ... 122
Recordset, AddNew ... 127
Recordset, Delete ... 129
Recordset, Open ... 123
Recordset, Update ... 127
Register ... 109

S

Schaltflächen ... 109, 114
Schleifen ... 45
Schleifenabbruch ... 46
Schlüsselwörter ... 24
Select-Case ... 43
Selection ... 78
Select ... 179
Serien-E-Mails ... 184
Set-Anweisung ... 55
Set-Prozedur ... 37

Shape 88, 172
Sheets 177
Sicherheitswarnungen 185
Single 40
SQL 162
SQL-Anweisung 123, 162
SQL-Anweisung, Syntax 162
Standardmodul 33
Static 38
Steuerelemente 153
Steuerelemente hinzufügen 109
Steuerelemente, Eigenschaften 110
Steuerelemente, UserForm-Dialoge 108
String 40
Sub-Prozeduren 35
Sub-Prozeduren, Aufbau 35
Syntaxüberprüfung 24
Syntaxüberprüfung, automatische 24

T

Tabellenblatt aktivieren 179
Tabellenblatt, Zugriff 80, 81
Table 50
Textbereiche, Zugriff 79
Textfelder 108
Textmarken 137

U

Übungsdateien 8
Umschaltfelder 109
UsedRange 145
UserForm-Dialog 33, 62, 106, 111
UserForm-Dialog anzeigen 113
UserForm-Dialog ausblenden 114
UserForm-Dialog erstellen 108
UserForm-Dialog testen 110
UserForm-Fenster 108

V

Variablen 37
Variablendeklaration 38
Variablendeklaration, explizite 38
Variablendeklaration, implizite 38
Variablennamen 37
Variant 40
VBA 10
VBA-Dialoge 106
VBA-Elemente 62
VBA-Entwicklungsumgebung 10
VBA-Entwicklungsumgebung öffnen 18
Verweis 59
Verweis erstellen 96
Verzweigungen 43
Visual-Basic-Editor 18
Visual-Studio-Tools für Office 2003 ..... 12, 118
Vorschaufunktion 156

W

Warnungen, Outlook 185
Werkzeugsammlung 108
Wiederholung, bedingte 45
Wiederholung, zählergesteuerte 45
Window 64
Word als Client 132
Word, Dokumente öffnen 76
Word, Dokumente schließen 76
Word, markierter Bereich 78
Word, Textbereiche 79
Word-Basic 10
Workbook 50
Wort vervollständigen 26

Z

Zellen adressieren 83
Zellen, Zugriff 82
Zwischenablage 94, 135