Stefan Leibing
Bernd Held

MASTER CLASS

Access-VBA

» Einstieg für Anspruchsvolle

Lerntest und Beispiele

ADDISON-WESLEY [in Kooperation mit] PEARSON Studium

3

Die VBA-Entwicklungsumgebung

Um mit VBA ein Programm zu entwickeln, benötigen Sie die VBA-Entwicklungsumgebung. Glücklicherweise wird diese mit den Microsoft Office-Programmen mitgeliefert, so auch mit Access. Sie müssen die entsprechende Umgebung also nicht erst installieren oder gar gesondert erwerben.

Info

Alle Beispiele aus diesem Kapitel finden Sie auf der mitgelieferten CD im Verzeichnis KAP03 in der Datenbank KAP03.MDB.

3.1 Elemente der Entwicklungsumgebung

3.1.1 Hauptfenster

Wie Ihnen bereits bekannt ist, können Sie über die Tastenkombination Alt + F11 in die Entwicklungsumgebung von Access gelangen. Nach Betätigung der Tastenkombination öffnet sich das Hauptfenster der Entwicklungsumgebung.

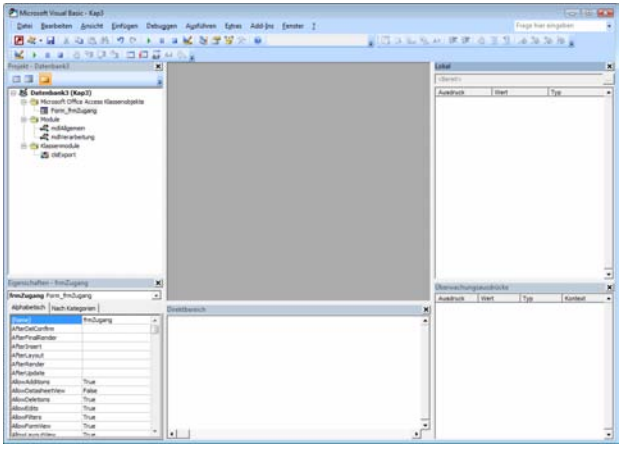


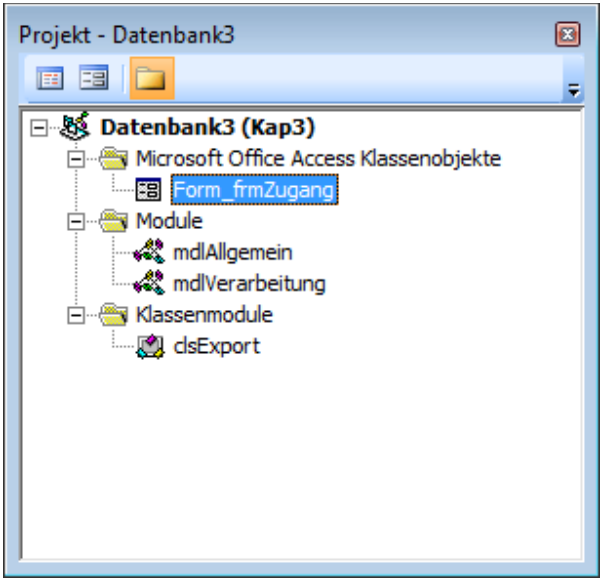
Abbildung 3.1
Das Hauptfenster der Entwicklungsumgebung

In der Entwicklungsumgebung von Access finden Sie eine ganze Reihe von Fenstern und Werkzeugen vor, die auf den nächsten Seiten beschrieben werden.

3.1.2 Projekt-Explorer

Der PROJEKT-EXPLORER stellt in einer Baumansicht diverse, zur Datenbank gehörende Elemente dar. In diesen Elementen kann Programmcode platziert werden.

Abbildung 3.2
Der Projekt-Explorer



Die wichtigsten vom PROJEKT-EXPLORER dargestellten Elemente lassen sich in drei Gruppen einteilen. Diese sind in der dargestellten Abbildung ersichtlich.

Die erste Gruppe, MICROSOFT OFFICE ACCESS KLASSENOBJEKTE, enthält Module mit Programmcode, der jeweils mit einem bestimmten, in der Access-Anwendung definierten Objekt, in der Regel einem Formular, verbunden ist. Im hier gezeigten Beispiel gibt es genau ein solches Formularmodul. Dieses gehört zum in der Datenbank definierten Formular FRMZUGANG.

Wenn Sie mit Formularen arbeiten, werden Sie mit dieser Art von Modulen zwangsläufig in Berührung kommen. Wenn Sie nämlich auf Ereignisse des Formulars reagieren möchten, z.B. auf das Betätigen einer Schaltfläche, platziert Access zu diesem Zweck entsprechende Ereignisprozeduren in diesen Modulen. Diese Ereignisprozeduren können Sie mit Programmcode füllen, damit die von Ihnen gewünschten Aktionen ausgeführt werden, wenn die entsprechenden Ereignisse eintreten. In Kapitel 11 werden Sie dieses Verfahren ausführlich kennenlernen.

Die zweite Gruppe, MODULE, enthält Module, die ähnlich wie zuvor die Formularmodule Programmcode enthalten. Der Unterschied liegt darin, dass diese Module nicht einem bestimmten Formular zugeordnet sind. Sie können sich diese Module wie Textdateien vorstellen, in denen Sie Programmcode platzieren. Nur sind diese Dateien nicht als externe Dateien vorhanden, wie dies in vielen anderen Entwicklungsumgebungen der Fall ist, sondern die Module sind fest in der Access-Datei integriert. Sie können nach Bedarf neue Module erzeugen. Prozeduren, die funktionell zusammengehören, platzieren Sie im Allgemeinen zusammen in einem Modul und geben diesem einen sinnvollen Namen. So schaffen Sie Ordnung in Ihrem VBA-Projekt.

Um ein neues Modul anzulegen, klicken Sie mit der rechten Maustaste im PROJEKT-EXPLORER auf die dort angezeigte Datenbank und wählen aus dem Kontextmenü den Befehl EINFÜGEN/MODUL. Oder wählen Sie im Hauptfenster der Entwicklungsumgebung den Menüpunkt EINFÜGEN/MODUL.

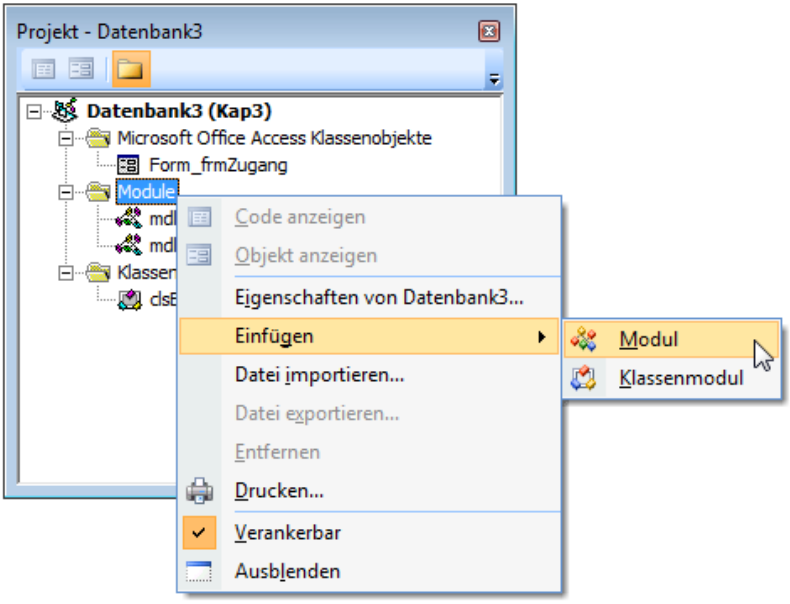
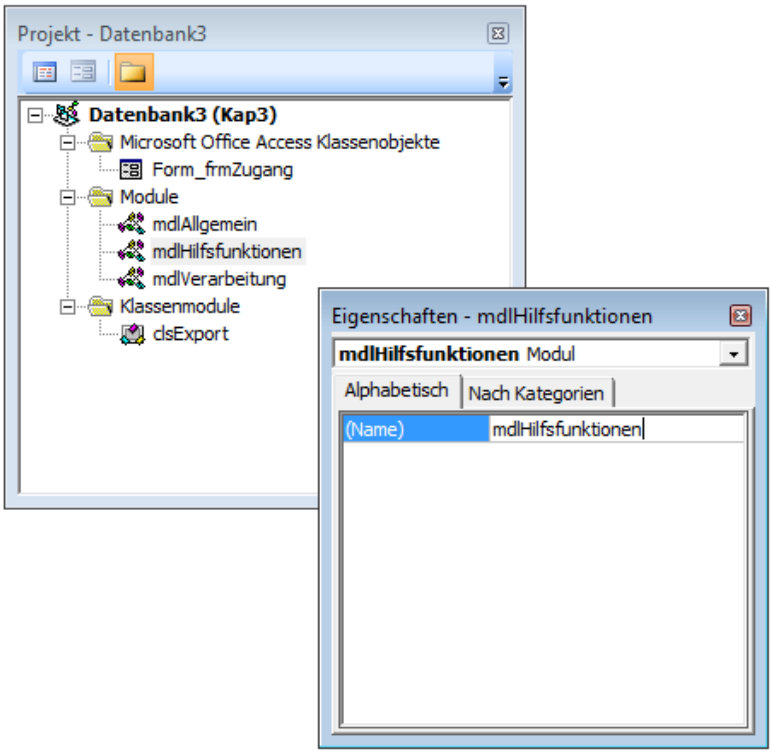


Abbildung 3.3
Ein neues Modul über den Projekt-Explorer einfügen

Hinweis

Direkt nach der Neuanlage eines Moduls empfiehlt sich die Umbenennung desselben. Der standardmäßige Name MODUL1 ist nicht sehr aussagekräftig. Um ein Modul umzubenennen, nutzen Sie das EIGENSCHAFTEN-Fenster (links unten) und geben im Feld (NAME) einen neuen Namen für das Modul ein. Wählen Sie einen Namen, der dem zukünftigen Inhalt des Moduls möglichst nahekommt. In unserem Beispiel nennen wir das neue Modul MDLHILFSFUNKTIONEN.

Abbildung 3.4
Das neue Modul wurde in mdlHilfsfunktionen umbenannt.



Die dritte Gruppe, KLASSENMODULE, enthält wie der Name schon sagt Klassenmodule. Klassenmodule enthalten wie die „gewöhnlichen" Module ebenfalls Programmcode, bieten jedoch in VBA eine Funktionalität, die ansatzweise der Funktionalität von Klassen in objektorientierten Sprachen entspricht. Der Grundgedanke von Klassen ist es, reale oder gedankliche Objekte über Eigenschaften (Variablen) zu beschreiben und diese mit zugehörigen Methoden (Prozeduren) zusammenzufassen. Man stellt dem Verwender der Klasse nur bestimmte, öffentliche Methoden zur Verfügung, um auf die Eigenschaften der Klasse zugreifen zu können, und erhält damit übersichtlichen, zuverlässigen Programmcode. Wenn die Klassen in VBA auch bei Weitem nicht die umfangreichen und nützlichen Mechanismen einer vollwertigen objektorientierten Programmiersprache wie C++ oder C# bieten, ist die Verwendung von Klassenmodulen in VBA auf jeden Fall sehr zu empfehlen.

Ausführliche Informationen über Klassenmodule erhalten Sie in Kapitel 4 dieses Buches.

3.1.3 Code-Fenster

Im Code-Fenster der Entwicklungsumgebung können Sie den Programmcode von Formularen, Modulen und Klassenmodulen ansehen, editieren und testen.

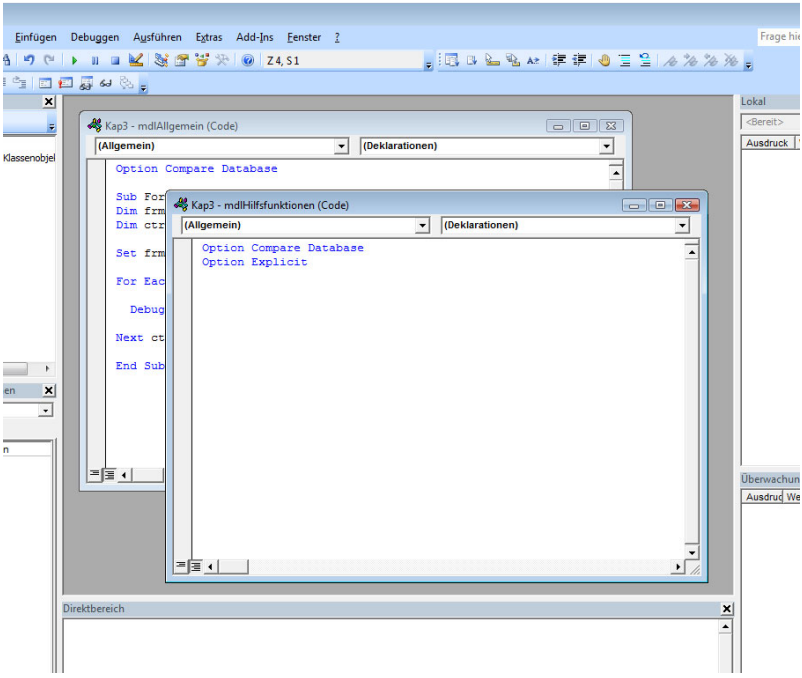


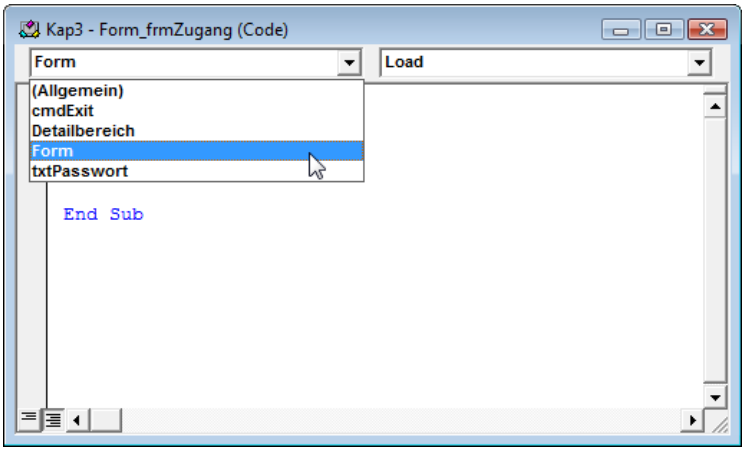
Abbildung 3.5
Die Code-Fenster zur Erstellung von Programmcode

Auf den ersten Blick erkennen Sie, dass sich oberhalb jedes Code-Fensters zwei Kombinationsfelder befinden.

Der Inhalt des linken Kombinationsfeldes ist abhängig davon, welchen Programmcode das Code-Fenster zeigt. Zeigt das Fenster Programmcode, der zu einem Formular gehört, sind im linken Kombinationsfeld die für das Formular existierenden bzw. die mit dem Formular verknüpften Objekte auswählbar.

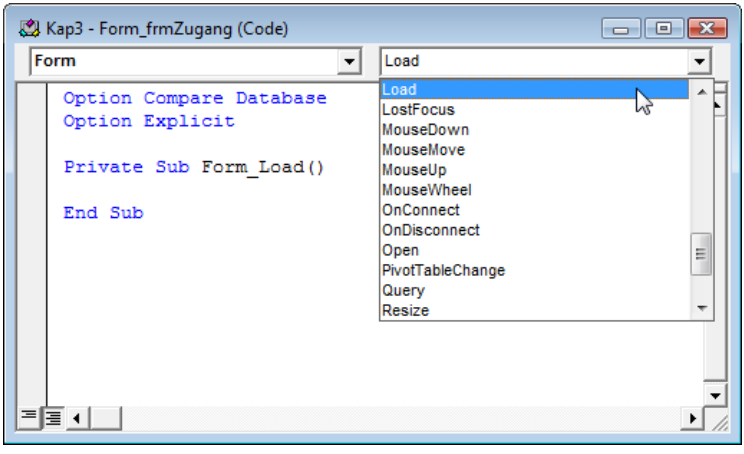
Das sind beispielsweise das Formular-Objekt an sich, der Detailbereich des Formulars und sämtliche auf dem Formular platzierten Steuerelemente. In der folgenden Abbildung sehen Sie beispielsweise die Auswahlmöglichkeit für ein einfaches Formular mit einer Schaltfläche CMDEXIT und einem Textfeld TXTPASSWORT.

Abbildung 3.6
Die auswählbaren Objekte eines einfachen Formulars



Haben Sie im linken Kombinationsfeld ein Objekt ausgewählt, beispielsweise das Objekt FORM, sind im rechten Kombinationsfeld alle Ereignisse auswählbar, die das ausgewählte Objekt auslösen kann. Wenn Sie einen Eintrag auswählen, wird im Code-Fenster automatisch eine Ereignisprozedur platziert, die aufgerufen wird, wenn das entsprechende Ereignis eintritt. In der folgenden Abbildung wurde das Objekt FORM ausgewählt. Im rechten Kombinationsfeld sehen Sie die entsprechenden möglichen Ereignisse des Objekts FORM. Wählen Sie den Eintrag LOAD, wird die Prozedur FORM_LOAD platziert. Diese wird immer dann aufgerufen, wenn das zugehörige Formular aufgerufen wird.

Abbildung 3.7
Die für das Objekt Form zur Verfügung stehenden Ereignisprozeduren



Sie können in dieser Prozedur Programmcode platzieren, der beim Laden des Formulars ausgeführt werden soll.

Info

Die Nutzung von Ereignissen und Ereignisprozeduren wird Ihnen ausführlich in Kapitel 4 und Kapitel 11 beschrieben.

Wenn Sie im linken Kombinationsfeld den Eintrag ALLGEMEIN auswählen – dies ist bei MICROSOFT OFFICE ACCESS KLASSENOBJEKTEN, MODULEN und KLASSENMODULEN möglich –, finden Sie im rechten Kombinationsfeld alle Deklarationen und allgemeinen Prozeduren, die im Code-Fenster enthalten sind. So können Sie mithilfe dieses Kombinationsfeldes beispielsweise einfach zwischen den von Ihnen definierten Prozeduren innerhalb eines Code-Fensters hin und her springen.

Tipp

Eine sehr elegante Möglichkeit, um zwei Passagen im Code-Fenster gleichzeitig anzuzeigen, besteht darin, das Code-Fenster in zwei Bereiche zu teilen. Dazu wählen Sie aus dem Menü FENSTER den Befehl TEILEN. Dadurch wird das Code-Fenster genau in der Mitte horizontal geteilt. Über die zwei vertikalen Bildlaufleisten können Sie in den einzelnen Teilen des Fensters navigieren. Um diese Teilung der Fenster wieder rückgängig zu machen, wählen Sie noch einmal den Befehl TEILEN aus dem Menü FENSTER.

3.1.4 Eigenschaften-Fenster

Das EIGENSCHAFTEN-Fenster haben Sie eben schon in Aktion erlebt, als wir im vorigen Beispiel ein Modul umbenannt haben. Das EIGENSCHAFTEN-Fenster ist nach der Installation von Access standardmäßig in der Entwicklungsumgebung nicht eingeblendet. Dieses Fenster können Sie einblenden, indem Sie aus dem Menü ANSICHT den Befehl EIGENSCHAFTENFENSTER wählen oder alternativ die Taste F4 drücken.

Das EIGENSCHAFTEN-Fenster können Sie dazu benutzen, um beispielsweise in einem Formular bestimmte Eigenschaften festzulegen, ohne eine einzige Zeile programmieren zu müssen.

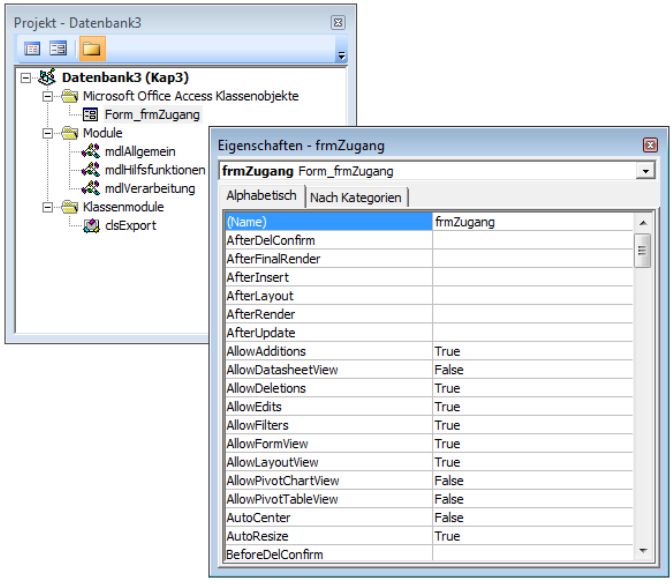
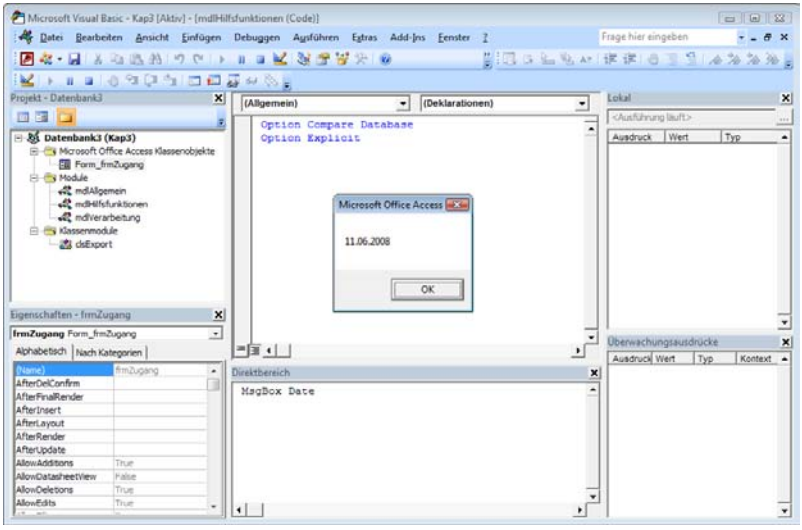


Abbildung 3.8
Über das Eigenschaften-Fenster können z.B. Eigenschaften eines Formulars festgelegt werden.

3.1.5 Direktfenster

Das Direktfenster können Sie nutzen, um Prozeduren oder Codefragmente direkt zu testen. Geben Sie hierzu beispielsweise die Zeile `MsgBox Date` ins Direktfenster ein und bestätigen Sie mit der Taste Enter. Als Ergebnis wird das aktuelle Datum am Bildschirm angezeigt.

Abbildung 3.9
Die Codezeile wurde im Direktfenster ausgeführt.



Wie gesagt, Sie sind bei Ausführung von Programmcode im Direktfenster nicht nur auf eine Zeile Programmcode beschränkt.

Das Direktfenster können Sie auch dazu nutzen, um zur Laufzeit eines Programms zur Kontrolle Ausgaben zu machen. Sie benutzen dazu die Anweisung `Debug.Print`. Die Anweisung `Debug.Print` wird in Abschnitt 3.2.1, Die Anweisung Debug.Print, näher beschrieben.

Das Direktfenster blenden Sie ein, indem Sie in der Entwicklungsumgebung die Tastenkombination Strg + G drücken.

Wenn Sie den Inhalt dieses Fenster löschen möchten, um wieder mehr Übersicht zu erhalten, dann positionieren Sie den Cursor im Fenster, markieren den gesamten Text über die Tastenkombination Strg + A und löschen den Text durch Drücken der Taste Entf.

3.1.6 Überwachungsfenster

Wählen Sie aus dem Menü ANSICHT den Befehl ÜBERWACHUNGSFENSTER, um das Überwachungsfenster anzuzeigen. Mit dem Überwachungsfenster haben Sie beispielsweise die Möglichkeit, zu überprüfen, wann eine bestimmte Variable ihren Inhalt ändert. Genau dann soll der Prozedurablauf unterbrochen werden.

Im folgenden Beispiel wird eine Schleife genau zehn Mal durchlaufen. Bei jedem Schleifendurchlauf wird die Variable `intZ` um den Wert `1` erhöht. Die Aufgabe besteht nun darin, die Prozedur nach der ersten Änderung der Variablen `intZ` zu stoppen. Sehen Sie sich zu diesem Zweck einmal die Prozedur aus Listing 3.1 an.

```
Sub SchleifenTest()
  Dim intZ As Integer

  For intZ = 1 To 10
    Debug.Print "Schleifendurchlauf: " & intZ
  Next intZ

End Sub
```

Listing 3.1
Die Schleife wird genau zehn Mal durchlaufen.

Um nun die Überwachung der Variablen `intZ` einzurichten, befolgen Sie die nächsten Schritte:

1. Blenden Sie das Überwachungsfenster ein, indem Sie aus dem Menü ANSICHT den Befehl ÜBERWACHUNGSFENSTER auswählen.
2. Klicken Sie mit der rechten Maustaste in dieses Fenster und wählen Sie den Befehl ÜBERWACHUNG HINZUFÜGEN aus dem Kontextmenü.

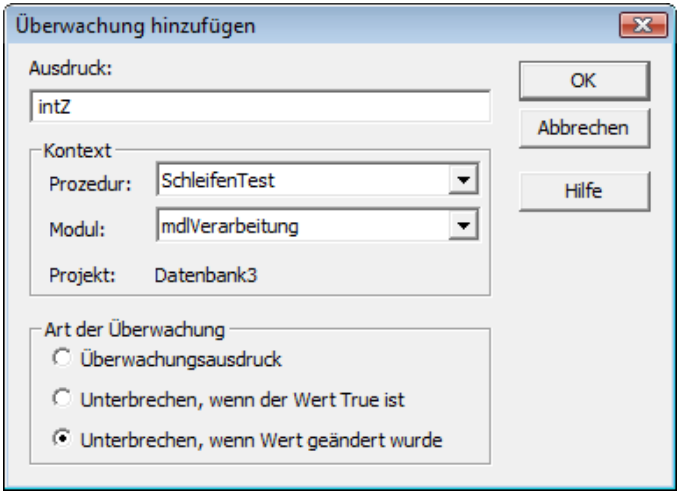
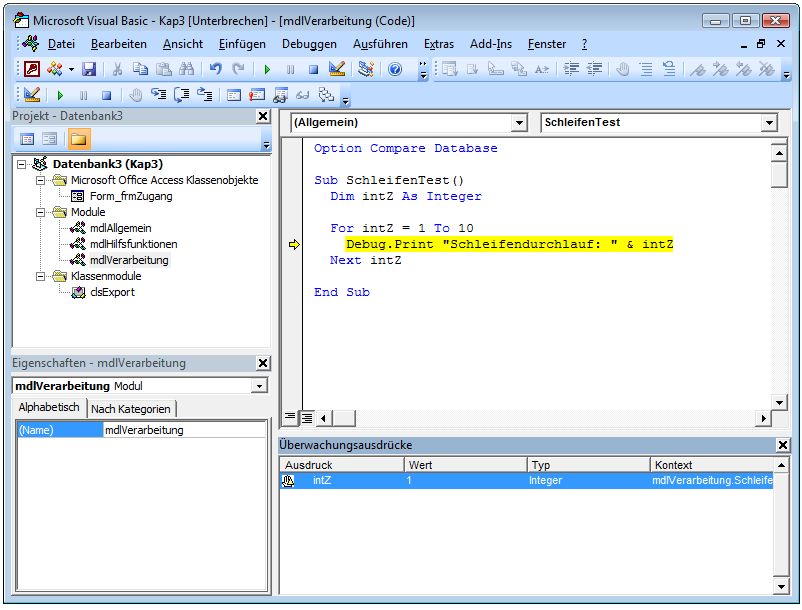


Abbildung 3.10
Eine Prozedur bei Änderung einer Variablen anhalten

3. Geben Sie im Feld AUSDRUCK den Namen der Variablen an, die Sie überwachen möchten.
4. Der Name der Prozedur sowie des Moduls sollten automatisch eingestellt sein, sofern Sie vor dem Aufruf den Mauszeiger innerhalb der Prozedur positioniert haben.
5. Aktivieren Sie die Option UNTERBRECHEN, WENN WERT GEÄNDERT WURDE.
6. Bestätigen Sie Ihre Einstellung mit OK.
7. Positionieren Sie den Mauszeiger innerhalb der Prozedur und drücken Sie die Taste F5, um die Prozedur auszuführen.

Abbildung 3.11
Die Programmausführung wird nach der ersten Änderung der Variablen angehalten.



Da die Variable `intZ` zu Beginn der Schleife vom Initialisierungswert `0` auf den Wert `1` gesetzt wird, findet gleich hier die erste Änderung des Variableninhalts statt. Genau danach stoppt die Programmausführung. Der gelbe Balken zeigt Ihnen die Codezeile an, die als Nächstes ausgeführt wird, wenn die Programmausführung fortgesetzt wird.

In diesem Zustand können Sie sich recht einfach den Inhalt von anderen Variablen einer Prozedur anzeigen lassen. Dabei positionieren Sie den Mauszeiger über der Variable, deren Inhalt Sie sehen möchten. Anschließend wird der Wert der Variable in einem kleinen Pop-up-Fenster angezeigt, sofern dieses Feature nicht von Ihnen in den Optionen deaktiviert wurde.

Den Inhalt der Variable `intZ` können Sie in diesem Fall auch dem Überwachungsfenster entnehmen.

Sie können sich auch generell den Inhalt einer Variablen im Überwachungsfenster anzeigen lassen, also auch ohne Definition einer Bedingung, die zum Anhalten des Programms führt.

Um Variablen zum Überwachungsfenster hinzuzufügen, können Sie diese ganz einfach im Code-Fenster markieren und mit der Maus per Drag & Drop ins Überwachungsfenster ziehen.

Um unser angehaltenes Programm weiter auszuführen, drücken Sie abermals die Taste F5. Die Prozedur stoppt nun bei jedem weiteren Schleifendurchlauf und zeigt den geänderten Inhalt der Variable im Überwachungsfenster an.

Das Überwachungsfenster wird selbstverständlich nur während des Testens eines Programms eingesetzt. Ist Ihr Test beendet, entfernen Sie den Überwachungsausdruck, indem Sie mit der rechten Maustaste in das Überwachungsfenster klicken und aus dem Kontextmenü den Befehl ÜBERWACHUNG ENTFERNEN auswählen.

Das gerade vorgestellte Beispiel stoppt bei der ersten Änderung der Variablen `intZ`. Im folgenden Beispiel definieren Sie die Haltebedingung so, dass angehalten wird, wenn die Variable `intZ` den Wert `5` annimmt. Um diese Einstellung vorzunehmen, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste ins Überwachungsfenster und wählen Sie den Befehl ÜBERWACHUNGSAUSDRUCK HINZUFÜGEN.

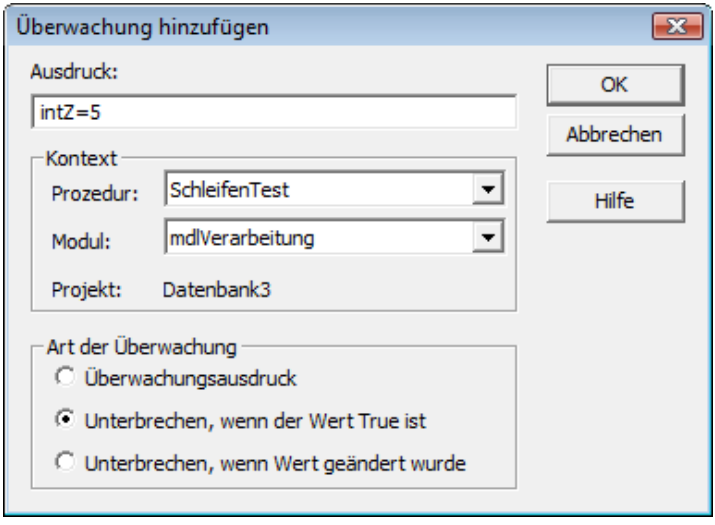
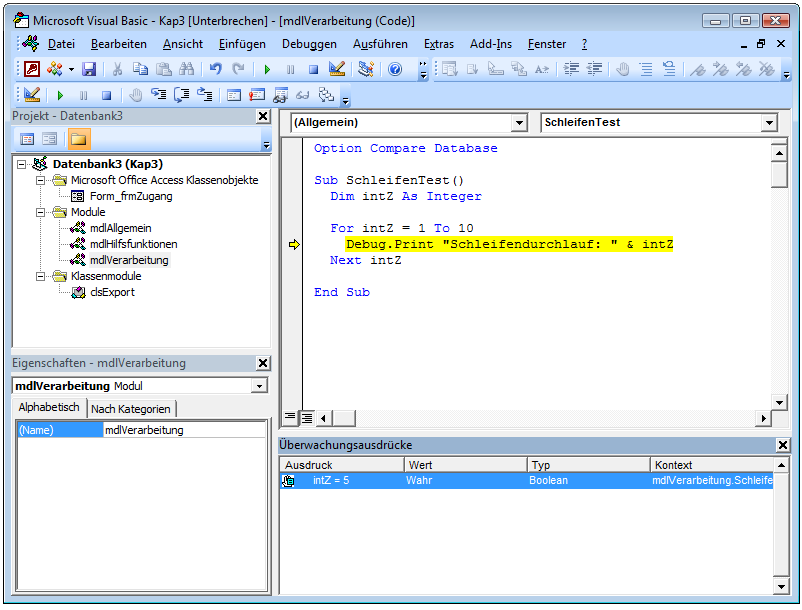


Abbildung 3.12
Die Programmausführung unterbrechen, wenn die Variable einen bestimmten Wert annimmt

2. Geben Sie im Feld AUSDRUCK die Formel INTZ = 5 ein.
3. Aktivieren Sie die Option UNTERBRECHEN, WENN DER WERT TRUE IST.
4. Bestätigen Sie mit OK.

Führen Sie anschließend die Prozedur mit F5 aus und kontrollieren Sie nach dem Stoppen der Programmausführung das Überwachungsfenster.

Abbildung 3.13
Die Programmausführung stoppt, wenn die Bedingung intZ=5 erfüllt ist.



Hinweis

Wenn Sie den Überwachungsausdruck nicht mehr benötigen, dann löschen Sie diesen wieder, indem Sie mit der rechten Maustaste im Überwachungsfenster auf den jeweiligen Eintrag klicken und den Befehl ÜBERWACHUNG ENTFERNEN aus dem Kontextmenü wählen.

3.1.7 Lokal-Fenster

Wenn sich das Programm im Haltemodus befindet, es also beispielsweise über eine im Überwachungsfenster definierte Haltebedingung angehalten wurde, zeigt das LOKAL-Fenster Ihnen immer automatsch alle innerhalb der aktuellen Prozedur lokal deklarierten Variablen und deren Inhalte an. Sie haben daher die Möglichkeit, die Werte von Variablen übersichtlich zu prüfen. Sie können sogar die Werte der Variablen ändern. Dies ist beim Testen von Programmen immer wieder sehr hilfreich.

Das LOKAL-Fenster wird in der Entwicklungsumgebung standardmäßig nicht angezeigt. Über das Menü ANSICHT können Sie dieses Fenster einblenden.

Im folgenden Beispiel werden alle Steuerelemente eines Formulars in einer Schleife abgefragt. Dabei werden die Namen der Steuerelemente im Direktbereich der Entwicklungsumgebung ausgegeben. Wenn Sie die Prozedur schrittweise durchlaufen, können Sie über das LOKAL-Fenster Informationen über die Eigenschaften der Steuerelemente abfragen, wie beispielweise Farbeinstellungen oder die Rahmenart ermitteln.

Um ein Beispiel zu sehen, befolgen Sie die nächsten Arbeitsschritte:

1. Wechseln Sie mit der Tastenkombination Alt + F11 in die Entwicklungsumgebung.
2. Geben Sie in ein neues oder bestehendes Modul folgende Prozedur ein bzw. greifen Sie auf das fertige Beispiel der Beispiel-CD zurück.

Listing 3.2
Alle Steuerelemente eines Formulars ermitteln

```
Sub FormularSteuerelementTypen()
  Dim frm As Form
  Dim ctrl As Control

  Set frm = Form_frmZugang

  For Each ctrl In frm.Controls

    Debug.Print "Typ: " & ctrl.ControlType & " Name: " & ctrl.Name

  Next ctrl

End Sub
```

3. Führen Sie den Programmcode der Prozedur schrittweise aus, indem Sie den Mauszeiger auf die erste Zeile der Prozedur setzen und für jeden Schritt die Taste F8 drücken.
4. Betätigen Sie die Taste F8 so oft, bis im Direktfenster der Text txtPASSWORT angezeigt wird.
5. Wählen Sie im LOKAL-Fenster den Eintrag für die Variable CTRL und klicken Sie auf das +-Zeichen, um den Eintrag aufzuklappen und die darunterliegenden Elemente sehen zu können.

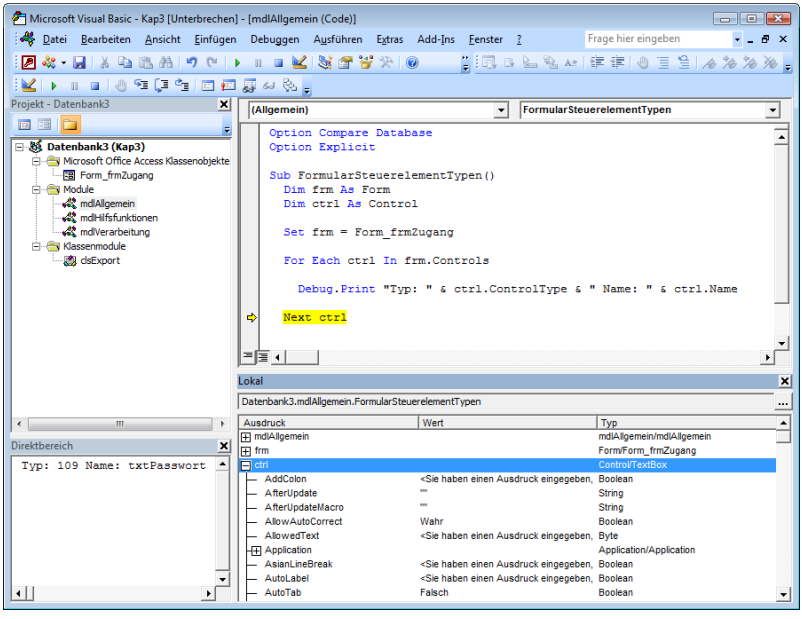


Abbildung 3.14
Die Eigenschaften eines TextBox-Steuerelements im Lokal-Fenster ansehen

Was sehen Sie nun, indem Sie den Eintrag `ctrl` im LOKAL-Fenster markiert haben? Nun, unser kleines Beispiel enthält im Prinzip zwei Variable. Dies ist zum einen eine Variable `frm` vom Typ `Form`. Eine Variable vom Typ `Form` repräsentiert grundsätzlich ein Formular. Über die Anweisung `Set frm = Form_frmZugang` referenzieren wir über diese Variable unser in der Datenbank definiertes Formular `frmZugang`.

Die zweite Variable der Prozedur ist die Variable `ctrl` vom Typ `Control`. Eine Variable vom Typ `Control` repräsentiert **grundsätzlich** ein Steuerelement. Wie Sie sehen, wird in unserem Beispiel der Variablen `ctrl` in einer `For Each…Next`-Schleife, bei jedem Schleifendurchlauf jeweils ein Element der zur Variablen `frm` gehörenden Auflistung `Controls` zugewiesen. Die Auflistung `Controls` enthält alle auf dem Formular platzierten Steuerelemente oder besser gesagt die zu den Steuerelementen gehörenden Objekte.

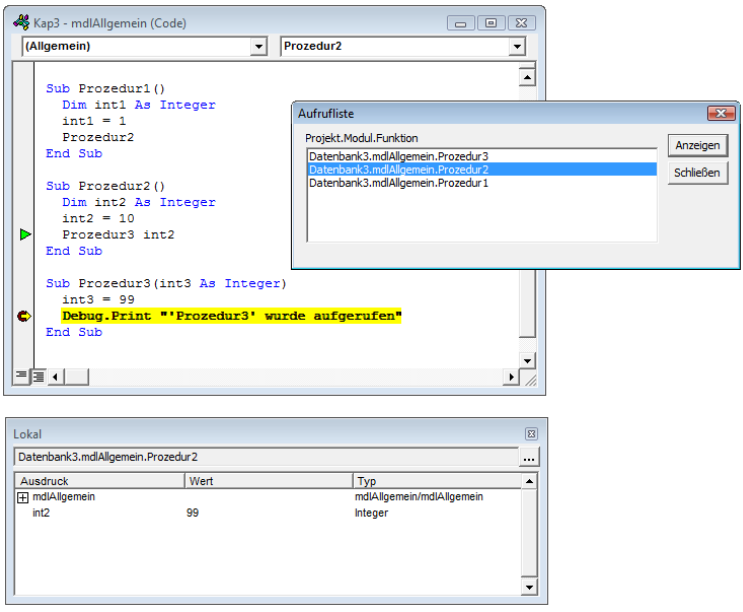
Sie haben nun an der Stelle der Programmausführung angehalten, an der die Variable `ctrl` das Textfeld mit Namen `txtPasswort` referenziert. Durch Aufklappen des Eintrags `ctrl` im LOKAL-Fenster sehen Sie alle Eigenschaften des Textfeldes `txtPasswort`. Wie Sie sehen, besitzt solch ein `Control`-Objekt eine ganze Menge Eigenschaften.

3.1.8 Aufrufliste

Ein weiteres nützliches Fenster ist der Dialog AUFRUFLISTE. Dieser Dialog zeigt in einer Liste die aktuelle Prozedur und die übergeordneten Prozeduren, die diese Prozedur aufgerufen haben, hierarchisch an.

Sehen Sie sich dazu einmal die folgende Abbildung an.

Abbildung 3.15
Verwendung der Aufrufliste



Sie sehen in der Abbildung drei Prozeduren mit den Namen `Prozedur1`, `Prozedur2` und `Prozedur3`. In `Prozedur3` wurde ein Haltepunkt gesetzt. Die `Prozedur1` wurde in diesem Fall unter Zuhilfenahme der Taste F5 aufgerufen. Das Programm ist den gesetzten Haltepunkt in `Prozedur3` angelaufen und ist in den Haltemodus gewechselt.

Die AUFRUFLISTE wurde über den Menüpunkt ANSICHT/AUFRUFELISTE aufgerufen. In der AUFRUFLISTE sehen Sie, dass die aktuelle `Prozedur3`, die ganz oben in der Liste steht, von `Prozedur2` aufgerufen wurde. `Prozedur2` wiederum wurde von `Prozedur1` aufgerufen.

Sie können nun über die AUFRUFLISTE in eine der aufgelisteten Ebenen springen, indem Sie auf einen Eintrag in der Liste doppelklicken. Im Code-Fenster wird dann am linken Seitenrand ein grüner Pfeil in der Prozedur sichtbar, die dem Eintrag in der AUFRUFLISTE entspricht, den Sie per Doppelklick ausgewählt haben. Der grüne Pfeil markiert genau den Punkt in der Prozedur, an dem in die nächste Prozedur der AUFRUFLISTE gesprungen wurde.

Bitte beachten Sie, dass das LOKAL-Fenster beim Wechsel in eine andere Ebene gleichzeitig auch den angezeigten Inhalt ändert. Es werden jeweils die lokalen Variablen der Prozedur angezeigt, in der sich der grüne Pfeil befindet. Die Variablen werden mit ihren aktuellen Werten angezeigt. Es werden also nicht die Werte angezeigt, die die Variablen hatten, als die Prozedur der nächsten Ebene aufgerufen wurde, wie vielleicht der ein oder andere vermuten mag. So wird beispielweise, wie in der Abbildung ersichtlich, für die Variable `int2` der Wert `99` angezeigt. Diesen Wert hat die Variable erst innerhalb der `Prozedur3` erhalten.

Wenn Sie nun über die Taste F5 das Programm fortsetzen, wird an der Stelle fortgesetzt, an der sich momentan der Programmzeiger befindet. In unserem Beispiel wäre das der Haltepunkt in `Prozedur3`. Die momentane Position des grünen Zeigers spielt für die Fortsetzung des Programms keine Rolle.

Tipp

Wenn Sie beim Test Ihres Programms einen Haltepunkt in einer Prozedur gesetzt haben, Ihr Programm den Haltepunkt angelaufen hat und Sie wissen möchten, in welcher Folge von verschachtelten Aufrufen die Prozedur aufgerufen wurde, dann nutzen Sie dazu die AUFRUFLISTE.

3.1.9 Verweise

Ein ganz wichtiger Punkt in der Softwareentwicklung ist es, dass Sie nicht alle Funktionalität, die Sie benötigen, von Grund auf selbst entwickeln müssen, sondern beim Entwickeln Ihrer Programme eigentlich immer in irgendeiner Form auch auf Funktionalität zurückgreifen, die von anderen Softwareentwicklern bzw. -herstellern zur Verfügung gestellt wird. Diese Funktionalität wird grundsätzlich in Form von Bibliotheken bereitgestellt.

Bibliotheken müssen Sie einbinden, um sie nutzen zu können. Einbinden bedeutet im Prinzip, dass Sie eine Verbindung zur Bibliothek herstellen. Dies können Sie auf zwei Arten machen.

Sie können die Bibliothek zur Laufzeit einbinden, und zwar erst in dem Moment, in dem ein Objekt erstellt wird, das auf einer Klasse basiert, die in einer Bibliothek definiert ist. Man spricht dann von *später Bindung,* im Englischen *Late Binding* genannt. Sie können die Bibliotheken auch schon von vornherein in das Projekt einbinden. Man spricht dann von *früher Bindung,* im Englischen *Early Binding* genannt.

Die Unterschiede zwischen den beiden Verfahren der Einbindung von Bibliotheken werden Ihnen in Kapitel 4 erläutert.

In diesem Abschnitt möchten wir Ihnen zeigen, wie Sie eine Bibliothek nach dem Verfahren der *frühen Bindung* einbinden und Funktionalität in dieser Bibliothek nutzen.

Um eine Bibliothek einzubinden, öffnen Sie den Dialog VERWEISE. Diesen öffnen Sie im Menü EXTRAS mithilfe des Befehls VERWEISE. Über die Einbindung von bestimmten Bibliotheken erhalten Sie beispielsweise Zugriff auf VBA-Befehle anderer Office-Programme. Dabei können Sie auch die HILFE-Funktion in VBA nutzen, um mehr über die Funktionen und Objekte anderer Office-Programme zu erfahren.

Im folgenden Listing soll beispielsweise die Anwendung Excel gestartet werden. Dazu sind bei dieser Technik folgende Schritte notwendig:

1. Wechseln Sie in die Entwicklungsumgebung von Access.
2. Wählen Sie aus dem Menü EXTRAS den Befehl VERWEISE.

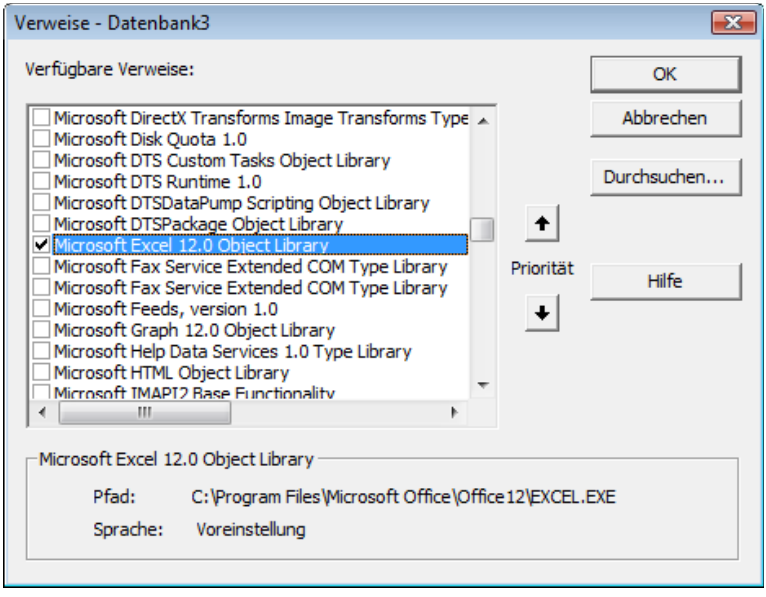


Abbildung 3.16
Die Excel-Bibliothek über den Dialog Verweise einbinden

3. Aktivieren Sie in der Liste VERFÜGBARE VERWEISE die Bibliothek MICROSOFT EXCEL OBJECT LIBRARY in der verfügbaren Version.
4. Bestätigen Sie mit OK.

Geben Sie nun den folgenden Quellcode ein, um Excel zu starten und die Datei UMSATZ.XLS zu öffnen.

Listing 3.3
Excel starten und Mappe öffnen

```
Sub ExcelStartenMappeOeffnen()
  Dim appXL As Excel.Application

  On Error GoTo Fehler

  Set appXL = New Excel.Application
  appXL.Visible = True
  appXL.Workbooks.Open ("C:\Eigene Dateien\Umsatz.xls")

  'Weitere Befehle

  Exit Sub
Fehler:
  MsgBox Err.Description
End Sub
```

Über die Zeile `Set appXL = New Excel.Application` wird ein neues Objekt vom Typ `Excel.Application` erzeugt. Dabei wird auf die zuvor eingebundene Excel-Bibliothek zugegriffen. Durch Erzeugung des Objekts wird eine neue Instanz einer Excel-Applikation gestartet.

Über die Objektvariable `appXL` haben Sie anschließend eine Referenz auf das neu erzeugte `Excel.Application`-Objekt und können dadurch auf Eigenschaften und Methoden dieses Objekts zugreifen. Dadurch, dass Sie der Eigenschaft `Visible` den Wert `True` zuweisen, wird das Excel-Hauptfenster sichtbar. Die Klasse `Excel.Application` besitzt ein Element mit dem Namen `Workbooks`. Dieses Objekt besitzt auch eine Methode `Open`. Über diese Methode können Sie eine bestimmte Datei öffnen. Schlägt das Öffnen fehl, da beispielsweise die angegebene Datei nicht existiert, dann wird ein Laufzeitfehler ausgelöst. In unserem Beispiel wird dieser Fehler mit einer `On Error`-Anweisung abgefangen.

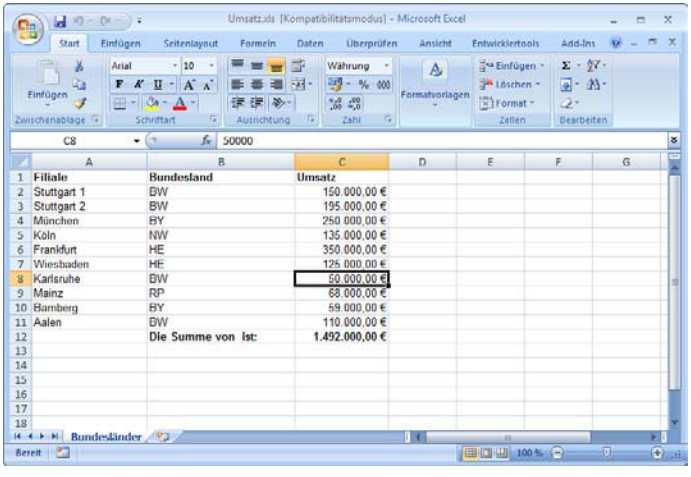


Abbildung 3.17
Excel wurde gestartet und eine Datei wurde geöffnet.

Achtung

Wenn Sie Ihre Anwendung verteilen möchten, d.h. diese auf verschiedenen PCs laufen soll, müssen Sie beim Einbinden von Bibliotheken berücksichtigen, dass die eingebundenen Bibliotheken auf dem jeweiligen Ziel-PC auch vorhanden sein müssen. Viele der im Dialog VERWEISE angezeigten Bibliotheken sind nämlich nicht Teil von Access, Office oder Windows, sondern können Teil der Installation eines anderen Programms sein. Wenn Sie also Bibliotheken verwenden, die nicht standardmäßig Teil von Microsoft Office oder Windows sind, müssen diese unter Umständen auf dem jeweiligen Ziel-PC erst installiert werden, damit Ihr Programm lauffähig ist.

Info

Sie werden früher oder später feststellen, dass die unterschiedlichen Office-Versionen zwar gleichnamige Bibliotheken mitliefern, diese Bibliotheken aber unterschiedliche Versionen besitzen.

So hat die in unserem vorigen Beispiel eingebundene Bibliothek MICROSOFT EXCEL OBJECT LIBRARY in Office 2007 die Version 12.0. In Office 2003 besitzt diese Bibliothek aber beispielweise die Version 11.0. Speichern Sie nun eine Access-Datenbank, in der Sie die Bibliothek MICROSOFT EXCEL OBJECT LIBRARY in der Version 12.0 eingebunden haben, mit Access 2007 im Access 2003-Format ab und öffnen diese Datei später in Access 2003, dann erhalten Sie beim Öffnen der Datei eine Fehlermeldung. Sie müssen dann in der Entwicklungsumgebung mithilfe des Dialogs VERWEISE manuell die MICROSOFT EXCEL OBJECT LIBRARY in der Version 11.0 einbinden.

Speichern Sie jedoch in Access 2003 eine Datenbank mit eingebundener Bibliothek MICROSOFT EXCEL OBJECT LIBRARY in der Version 11.0 ab und öffnen diese Datei mit Access 2007, dann wechselt Access 2007 automatisch auf die höhere Version 12.0, ohne sich zu beschweren. Dies ist möglich, da Bibliotheken im Allgemeinen abwärts kompatibel gehalten werden. Leider gelingt das aber nicht immer völlig.

Tipp

Möchten Sie Fehlermeldungen vermeiden, die darauf basieren, dass eine von Ihnen weitergegebene Access-Datenbank eingebundene Bibliotheken auf einem PC zwar vorfindet, aber nicht in der von Ihnen über den Dialog VERWEISE eingebundenen Version, dann sollten Sie generell auf das frühe Einbinden von Bibliotheken verzichten. Stattdessen sollten Sie grundsätzlich das Verfahren der *späten Bindung* einsetzen. Dieses Verfahren wird ausführlich in Kapitel 4 beschrieben.

Info

Die im Dialog VERWEISE gezeigten und von VBA nutzbaren Bibliotheken basieren auf der von Microsoft entwickelten *COM*-Technologie. COM steht für *Component Object Model* und ist eine Technologie, die es unter anderem Bibliotheken ermöglicht, ihre Schnittstellen zu veröffentlichen und damit für andere Programme nutzbar zu machen.

Eine Bibliothek oder besser gesagt deren Schnittstelle wird bei der Installation der Bibliothek in der *Windows-Registry* registriert und ist somit für andere Programme auffindbar und nutzbar. In den meisten Fällen besteht eine Bibliothek aus einer sogenannten *DLL* (Dynamic Link Library).

3.1.10 Objektkatalog

Wie Sie im vorigen Abschnitt gesehen haben, steht Ihnen in VBA eine riesige Menge an Bibliotheken zur Verfügung, die Ihnen in Summe eine nahezu unüberschaubare Menge an Klassen mit deren Methoden und Eigenschaften zur Verfügung stellen.

Hilfe zu den Inhalten der zentralen Bibliotheken von Office-VBA erhalten Sie über die Online-Hilfe der VBA-Entwicklungsumgebung. Wenn Sie jedoch Hilfe zu den Bibliotheken benötigen, die nicht zu den Bibliotheken des Office-VBA-Pakets gehören, werden Sie in der Online-Hilfe nicht fündig.

Der OBJEKTKATALOG bietet Ihnen dahingehend Unterstützung, dass er Ihnen die Möglichkeit gibt, den Inhalt der Bibliotheken anzusehen, die Sie momentan über den Dialog VERWEISE in das Projekt eingebunden haben. Über den OBJEKTKATALOG können Sie sehen, welche Klassen in den jeweiligen Bibliotheken definiert sind und welche Methoden und Eigenschaften diese Klassen besitzen.

Um den OBJEKTKATALOG von Access aufzurufen, wählen Sie aus dem Menü ANSICHT den Befehl OBJEKTKATALOG. Alternativ dazu können Sie auch die Taste F2 drücken.

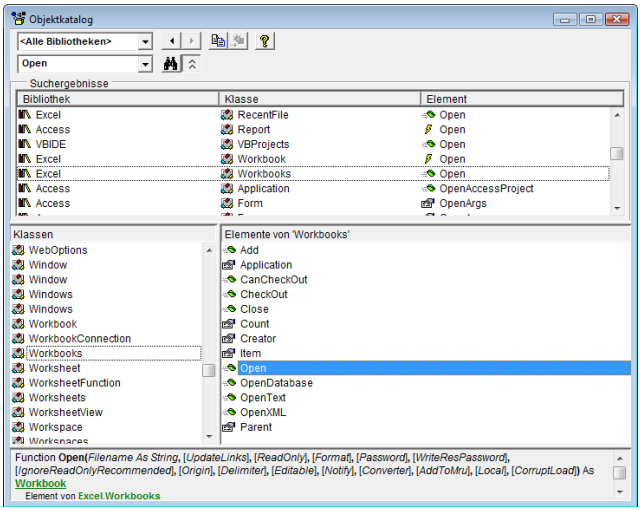


Abbildung 3.18
Der Objektkatalog

Im folgenden Beispiel suchen wir zur Demonstration einmal die Methode `Open`, mit der wir in Listing 3.3 eine Excel-Datei geöffnet haben.

1. Geben Sie dazu im Suchfenster den Begriff `Open` ein und klicken Sie auf die neben dem Suchfenster liegende Schaltfläche mit dem Fernglas. In der Liste SUCHERGEBNISSE erscheinen anschließend sämtliche Klassen und Elemente, in denen der Suchbegriff enthalten ist.
2. Scrollen Sie in dieser Liste nach unten. Dort finden Sie einen Eintrag für die Methode OPEN der in der Bibliothek EXCEL definierten Klasse WORKBOOKS. Markieren Sie diese Zeile. In der Liste KLASSEN des Dialogs werden dann alle Klassen angezeigt, die in der Bibliothek enthalten sind, die in der Liste SUCHERGEBNISSE ausgewählt wurde.
3. Wählen Sie in der Liste KLASSEN die Klasse WORKBOOKS aus. Anschließend erscheinen in der Liste ELEMENTE VON sämtliche Elemente der ausgewählten Klasse. Sie finden dort auch die Methode OPEN.
4. Markieren Sie in der Liste ELEMENTE VON den Eintrag OPEN. Sie erhalten dann im unteren Teilbereich des Dialogs eine Information über die Syntax der Methode OPEN mit allen verfügbaren Parametern.

Sie können übrigens im oberen Drop-down-Feld, das standardmäßig die Auswahl <ALLE BIBLIOTHEKEN> zeigt, auch von vornherein gezielt eine Bibliothek auswählen.

3.1.11 Optionen

In der Entwicklungsumgebung haben Sie die Möglichkeit, den Visual Basic-Editor in einigen Bereichen Ihren Wünschen entsprechend anzupassen. Dazu wählen Sie aus dem Menü EXTRAS den Befehl OPTIONEN. Anschließend öffnet sich der Dialog OPTIONEN.

Registerkarte Editor

Wechseln Sie auf die Registerkarte EDITOR. Dort können Sie die Einstellungen für das Code- und das Projektfenster festlegen.

Im Gruppenfeld CODE-EINSTELLUNGEN finden Sie die folgenden Einstellungsmöglichkeiten:

- AUTOMATISCHE SYNTAXÜBERPRÜFUNG: Mit dieser Einstellung sorgen Sie dafür, dass Ihr Editor nach der Eingabe einer Codezeile automatisch eine Syntaxprüfung vornimmt. Wird bei Verlassen einer Codezeile ein Syntax-Fehler detektiert, werden Sie über einen Dialog auf den Fehler aufmerksam gemacht. Fortgeschrittene Anwender neigen dazu, diese Option zu deaktivieren, da dieses Feature schnell lästig werden kann, wenn Sie bei der Programmerstellung mal schnell die aktuelle Zeile verlassen, um Code von einer anderen Zeile zu kopieren.

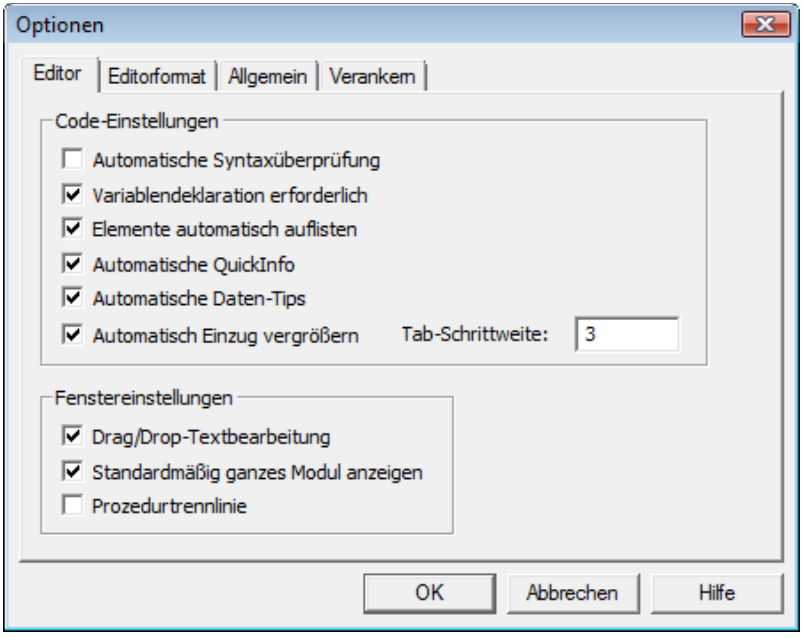


Abbildung 3.19
Editor-Einstellungen vornehmen

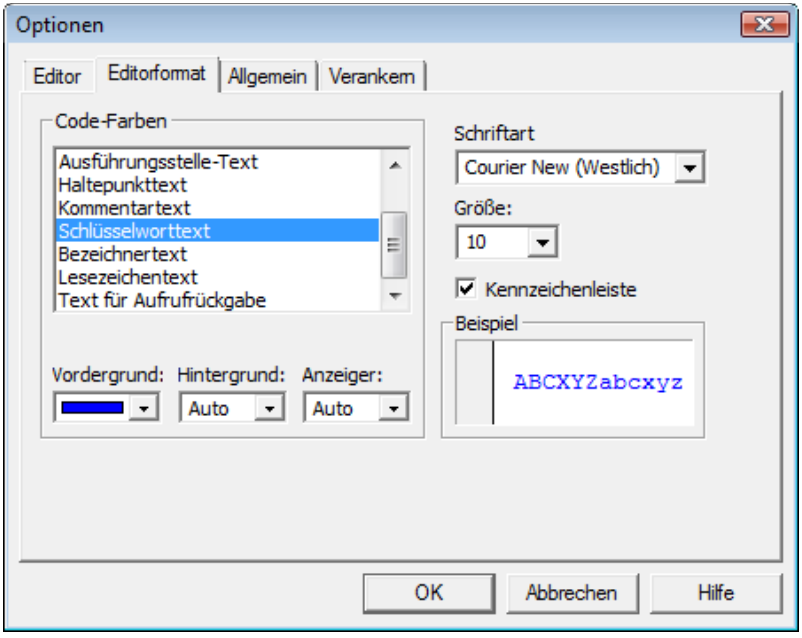
- VARIABLENDEKLARATION ERFORDERLICH: Wenn Sie diese Option aktivieren, wird die Anweisung `Option Explicit` in allen neu erzeugten Modulen automatisch hinzugefügt. Bei Vorhandensein der Anweisung `Option Explicit` am Anfang eines Moduls müssen alle im Code verwendeten Variablen deklariert werden. Wird die Deklaration unterlassen, weist Sie Access automatisch darauf hin. Eine Prozedur wird dann nicht ausgeführt, wenn nicht alle darin verwendeten Variablen deklariert sind. Diese Option sollten Sie auf jeden Fall aktivieren!
- ELEMENTE AUTOMATISCH AUFLISTEN: Zeigt eine Liste mit den Informationen an, die die Anweisung an der aktuellen Einfügemarke logisch vervollständigen würden.
- AUTOMATISCHE QUICKINFO: Wird diese Option aktiviert, werden bei der Eingabe eines Befehls die dazugehörigen Funktionen bzw. Parameter angezeigt.
- AUTOMATISCHE DATEN-TIPS: Diese Option ist lediglich im Haltemodus verfügbar und zeigt die Werte der Variablen an, auf der der Mauszeiger sich gerade befindet.
- AUTOMATISCH EINZUG VERGRÖSSERN: Zur besseren Übersichtlichkeit sollten Sie Ihren Quellcode auf jeden Fall einrücken. Diese Option ist Ihnen dabei behilflich. Sie haben die Möglichkeit, für die erste Codezeile einen Tabulator festzulegen. Alle nachfolgenden Zeilen beginnen an der Tabulatorposition.

- TAB-SCHRITTWEITE: In diesem Eingabefeld stellen Sie die Tab-Schrittweite auf einen Wert zwischen 1 und 32 Leerzeichen ein. Üblich ist eine Tab-Schrittweite von 3 bis 4 Zeichen.
- Im Gruppenfeld FENSTEREINSTELLUNGEN können Sie unter anderem die Drag-and-drop-Möglichkeit im Code-Fenster ausschalten, automatisch eine Trennlinie zwischen den einzelnen Prozeduren ziehen lassen und das Erscheinungsbild von neuen Modulen beeinflussen.

Registerkarte Editorformat festlegen

Wenn Sie zur Registerkarte EDITORFORMAT wechseln, können Sie die Darstellung des Quellcodes im Code-Fenster anpassen.

Abbildung 3.20
Die Codeformatierung über die Registerkarte Editorformat anpassen



Im Listenfeld CODE-FARBEN werden die Textelemente angezeigt, für die Farben angepasst werden können. Darunter befinden sich drei Drop-down-Felder, in denen Sie die Farben für den Vorder- bzw. Hintergrund der einzelnen Elemente sowie das Kennzeichen in der Kennzeichenleiste bestimmen können. Darüber hinaus haben Sie die Möglichkeit, die Schriftart sowie deren Größe zu bestimmen.

Registerkarte Allgemein

Wechseln Sie nun zur Registerkarte ALLGEMEIN. Dort werden die Einstellungen für die Fehlerbehandlung und die Kompilierungseinstellungen für das aktuelle Visual Basic-Projekt festgelegt.

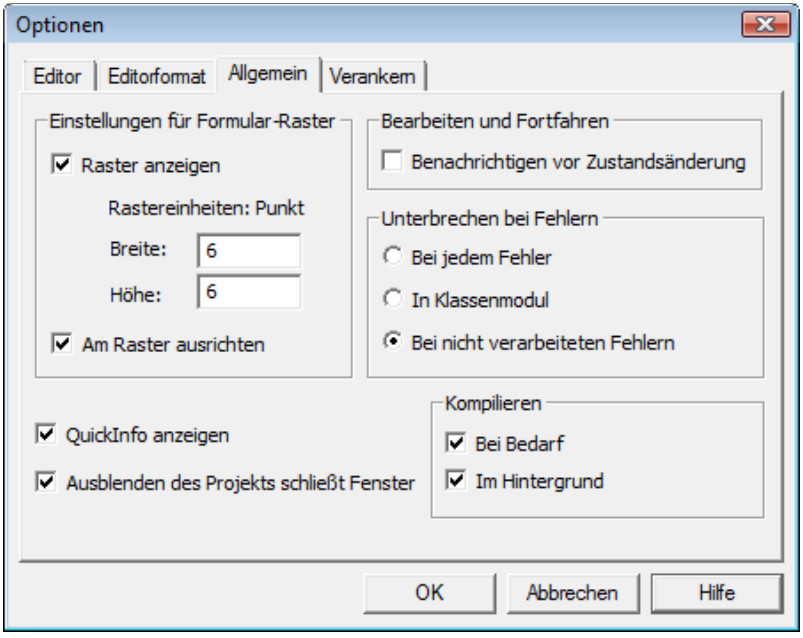


Abbildung 3.21
Allgemeine Einstellungen vornehmen

Im Gruppenfeld EINSTELLUNGEN FÜR FORMULAR-RASTER können Sie die Darstellungsart des Formulars beim Bearbeiten festlegen. Sie können ein Raster anzeigen, die Rastereinheiten sowie die Rasterung selbst für das Formular festlegen und eingefügte Steuerelemente automatisch am Raster ausrichten lassen.

Die Einstellung QUICKINFO ANZEIGEN bezieht sich lediglich auf die QuickInfos für die Symbolschaltflächen und kann deaktiviert werden.

Aktivieren Sie das Kontrollkästchen AUSBLENDEN DES PROJEKTS SCHLIESST FENSTER, wenn Projekt-, Objekt- oder Modulfenster automatisch geschlossen werden sollen, sobald ein Projekt im Projekt-Explorer ausgeblendet wird.

Im Gruppenfeld BEARBEITEN UND FORTFAHREN bestimmen Sie, ob eine Benachrichtigung erfolgen soll, wenn durch eine angeforderte Aktion alle Variablen auf Modulebene für ein laufendes Projekt zurückgesetzt werden.

Das Gruppenfeld UNTERBRECHEN BEI FEHLERN bestimmt, wie Fehler in der Visual Basic-Entwicklungsumgebung verarbeitet werden sollen. Das Einstellen dieser Option wirkt sich auf alle Instanzen von Visual Basic aus, die nach dem Ändern dieser Einstellung gestartet werden.

Sie haben dabei folgende drei Möglichkeiten:

- BEI JEDEM FEHLER: Bei jedem Fehler wird für das Projekt der Haltemodus aktiviert, unabhängig davon, ob eine Fehlerbehandlungsroutine aktiviert ist oder sich der Code in einem Klassenmodul befindet. Die Zeile, in der Fehler auftreten, wird dann mit einer gelben Hintergrundfarbe hinterlegt.

- In Klassenmodul: Mit dieser Einstellung werden alle nicht verarbeiteten Fehler in einem Klassenmodul mit dem Haltemodus gestoppt.
- Bei nicht verarbeiteten Fehlern: Wenn eine Fehlerbehandlungsroutine vorhanden ist, wird der Fehler behandelt, ohne den Haltemodus zu aktivieren. Sollte keine Fehlerbehandlungsroutine vorhanden sein, bewirkt der Fehler, dass der Haltemodus für das Projekt aktiviert wird.

Im Gruppenfeld Kompilieren legen Sie fest, ob ein Projekt vor dem Start vollständig kompiliert oder ob der Code bei Bedarf kompiliert wird, wodurch die Anwendung schneller gestartet werden kann. Sie können hier auch bestimmen, ob Access Leerlaufzeiten dazu benutzen kann, um Teile des Codes im Hintergrund zu kompilieren, was ebenfalls die Zeitspanne bis zum Start des VBA-Programms reduziert.

Info

Kompilierung wird der Vorgang der Umwandlung von Programmcode in eine vom VBA-Code-Interpreter ausführbare Form genannt. Dabei findet gleichzeitig eine Syntaxüberprüfung des Programmcodes statt. Im Falle eines bei der Kompilierung aufgespürten Syntaxfehlers erfolgt eine Fehlermeldung.

Tipp

Wenn Sie in der Gruppe Kompilieren die Option Bei Bedarf aktiviert haben und gleichzeitig die Option Automatische Syntaxüberprüfung auf der Registerkarte Editor deaktiviert haben, werden Sie auf Syntaxfehler innerhalb einer Prozedur erst aufmerksam gemacht, wenn die betreffende Prozedur innerhalb des Programms aufgerufen wird, da der Programmcode der Prozedur erst kurz vor dem Aufruf derselben kompiliert wird. Dies ist natürlich ungünstig, da Sie die komplette Syntaxüberprüfung Ihres Programmcodes erst dann bewerkstelligt hätten, wenn sämtliche Prozeduren Ihres Programms mindestens einmal aufgerufen wurden. Glücklicherweise gibt es da eine einfachere Lösung.

Wählen Sie aus dem Menü Debuggen den Menüpunkt Kompilieren von. Dabei wird der gesamte Programmcode Ihres Projekts kompiliert und sämtliche Syntaxfehler im Programmcode werden gefunden.

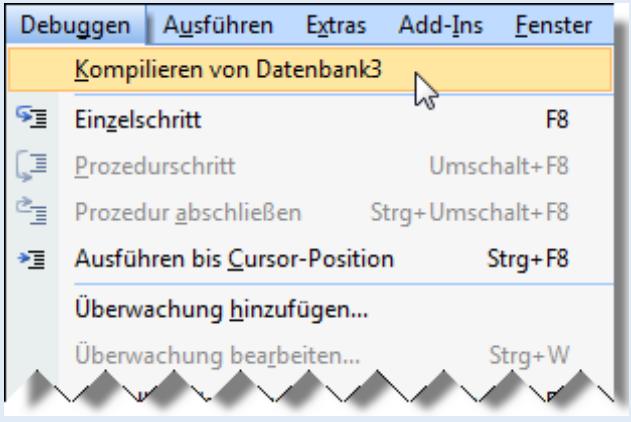


Abbildung 3.22
Über die Funktion Kompilieren von finden Sie sämtliche Syntaxfehler im Projekt.

Registerkarte Verankern

Auf der Registerkarte VERANKERN legen Sie fest, welche Fenster des VBA-Editors verankerbar sein sollen.

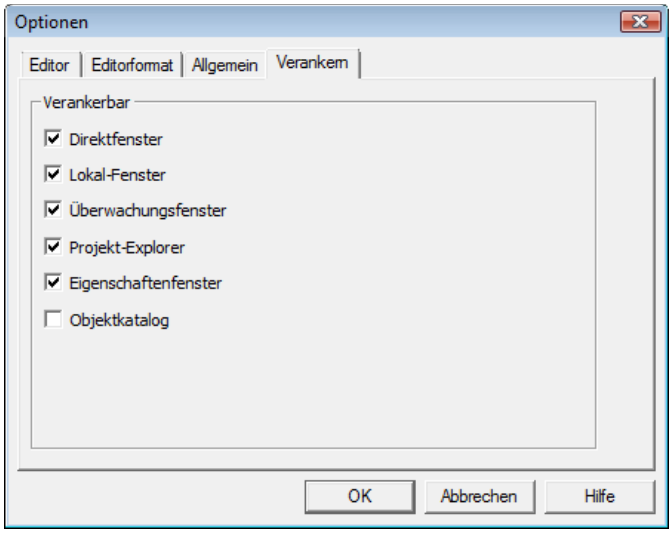


Abbildung 3.23
Verankerungsoptionen der Editor-Fenster

Ein Fenster ist verankert, wenn es mit einer Kante eines anderen verankerbaren Fensters oder eines Anwendungsfensters verbunden ist. Ein verankerbares Fenster wird beim Verschieben automatisch ausgerichtet. Verankerbare Fenster können Sie wie die Code-Fenster auch mitten im VBA-Editor platzieren. Wählen Sie die Fenster aus, die verankerbar sein sollen, und deaktivieren Sie die Kontrollkästchen für die anderen Fenster.

3.1.12 Suchen

Sie werden während der Programmentwicklung sehr häufig nach bestimmtem Code, Codefragmenten oder einfach irgendwelchen Zeichenketten suchen müssen.

Wenn Sie beispielsweise auf der Suche nach einer bestimmten Methode sind, dann können Sie eine Suche über alle Module wie folgt durchführen:

1. Wählen Sie in der Entwicklungsumgebung aus dem Menü BEARBEITEN den Befehl SUCHEN. Der Dialog SUCHEN öffnet sich.

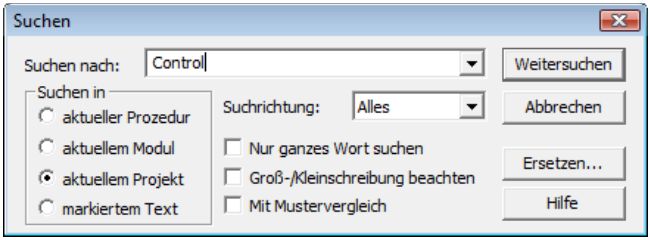
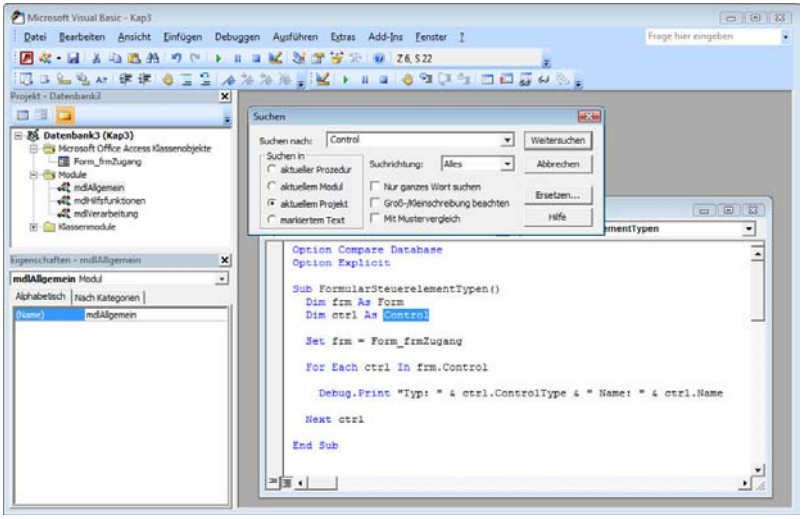


Abbildung 3.24
Befehle oder beliebige Zeichenketten im Quellcode suchen

2. Geben Sie im Feld SUCHEN NACH den Suchbegriff ein.
3. Aktivieren Sie die Option AKTUELLEM PROJEKT, um die Suche über alle Module hinweg durchzuführen.
4. Klicken Sie auf die Schaltfläche WEITERSUCHEN.
5. Klicken Sie so oft auf diese Schaltfläche, bis Sie zur gewünschten Stelle gelangen.

Abbildung 3.25
Fundstelle für Fundstelle „abklappern“



3.1.13 Tastenkombinationen

Neben den bekannten Tastenkombinationen Strg + C, um Codeteile zu kopieren, und Strg + V, um diese kopierten Codeteile an anderer Stelle des Moduls wieder einzufügen, gibt es eine ganze Reihe weiterer Tastenkombinationen, die Ihnen die Arbeit erleichtern.

Entnehmen Sie aus der folgenden Tabelle einige der gängigsten Tastenkombinationen für das zügige Arbeiten in der Entwicklungsumgebung.

Tabelle 3.1
Die Tastenkombinationen des VBA-Editors

Tastenkombination	Beschreibung
F7	Code-Fenster anzeigen
F2	OBJEKTKATALOG anzeigen
STRG + F	Suchen
STRG + H	Ersetzen
F3	Weitersuchen
⇧ + F3	Vorheriges suchen
STRG + ↓	Nächste Prozedur
STRG + ↑	Vorherige Prozedur

Tastenkombination	Beschreibung
⇧ + F2	Definition anzeigen
STRG + BILD ↓	Einen Bildschirm nach unten
STRG + BILD ↑	Einen Bildschirm nach oben
STRG + ⇧ + F2	Zur letzten Position wechseln
STRG + POS1	Anfang des Moduls
STRG + ENDE	Ende des Moduls
STRG + →	Ein Wort nach rechts
STRG + ←	Ein Wort nach links
ENDE	Zum Zeilenende wechseln
POS1	Zum Zeilenanfang wechseln
STRG + Z	Letzten Befehl rückgängig machen
STRG + C	Code kopieren
STRG + X	Code ausschneiden
STRG + V	Code einfügen
STRG + Y	Aktuelle Zeile löschen
STRG + ENTF	Bis zum Wortende löschen
⇥	Einzug vergrößern
⇧ + ⇥	Einzug verkleinern
STRG + ⇧ + F9	Alle Haltepunkte löschen
⇧ + F10	Kontextmenü anzeigen
STRG + P	Modul drucken
STRG + E	Modul/Formular exportieren
STRG + S	Modul speichern

Tabelle 3.1 (Forts.) Die Tastenkombinationen des VBA-Editors

3.1.14 Die Symbolleiste Bearbeiten

Die Symbolleiste BEARBEITEN stellt Ihnen Funktionalität für die Bearbeitung von Programmcode zur Verfügung. Diese Symbolleiste ist standardmäßig nach dem allerersten Starten der Entwicklungsumgebung nicht eingeblendet. Blenden Sie diese ein, indem Sie im Menü ANSICHT den Befehl SYMBOLLEISTEN/BERBEITEN aufrufen.

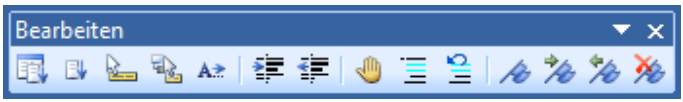


Abbildung 3.26 Die Symbolleiste Bearbeiten

Auf die wichtigsten Funktionen dieser Symbolleiste soll im Folgenden kurz eingegangen werden:

Einzug vergrößern bzw. verkleinern

Mit der Funktion EINZUG VERGRÖSSERN können Sie einzelne Zeilen oder auch mehrere Zeilen blockweise nach rechts einrücken.

Analog zur vorherigen Funktion können Sie mit der Funktion EINZUG VERKLEINERN eingerückte Programmteile wieder nach links rücken und pro Klick jeweils den markierten Text um einen Tabstopp zurückversetzen.

Haltepunkt ein/aus

Wenn Sie ein Programm testen, können Sie durch Platzieren von Haltepunkten den Programmablauf gezielt an bestimmten Stellen anhalten. Das Programm wird dann in den Haltemodus versetzt, sobald die Programmausführung einen Haltepunkt erreicht. Sie können dann beispielsweise den Inhalt verschiedener Variablen überprüfen und anschließend z.B. durch Betätigung der Taste F5, das Programm einfach an dieser Stelle fortsetzen.

Block auskommentieren

Mit der Funktion BLOCK AUSKOMMENTIEREN können Sie mehrere Zeilen auf einmal auskommentieren. Dies ist sehr praktisch, wenn Sie beispielsweise mehrere Programmzeilen zum Test vorübergehend deaktivieren möchten.

Möchten Sie auskommentierte Zeilen wieder aktivieren, dann markieren Sie die entsprechende(n) Zeile(n) und klicken auf das Symbol AUSKOMMENTIERUNG DES BLOCKS AUFHEBEN.

Eigenschaften/Methoden anzeigen

Möchten Sie die zur Verfügung stehenden Eigenschaften und Methoden eines Objekts angezeigt bekommen, platzieren Sie die Einfügemarke hinter dem auf eine Objektvariable folgenden Punkt und klicken auf das Symbol EIGENSCHAFTEN/METHODEN ANZEIGEN.

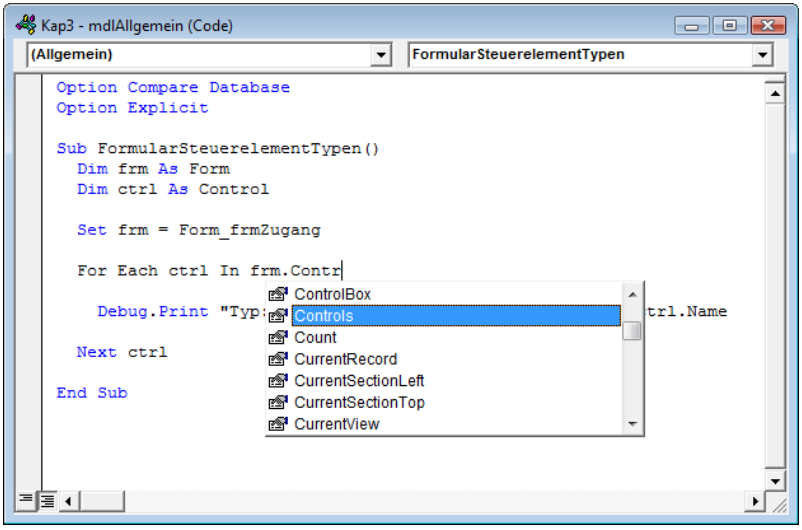


Abbildung 3.27
Alle Eigenschaften und Methoden einer Objektvariablen werden angezeigt.

Alternativ lautet die Tastenkombination für das Aufrufen der zur Verfügung stehenden Methoden und Eigenschaften: Strg + J.

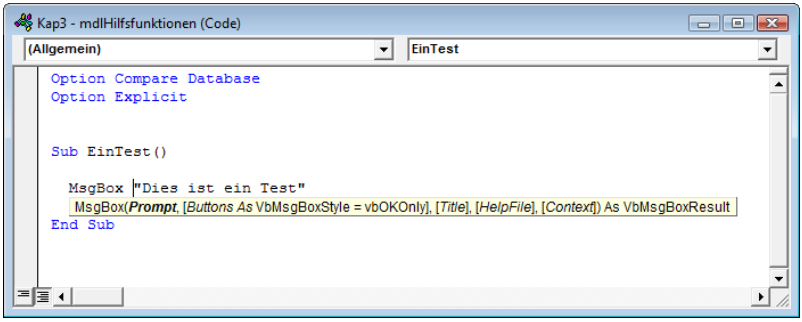
Sie erhalten dieses Informationsfenster übrigens auch automatisch, sobald Sie hinter dem Namen einer Objektvariablen einen Punkt eingeben. Dies funktioniert allerdings nur, wenn Sie im Dialog OPTIONEN auf der Registerkarte EDITOR die Option ELEMENTE AUTOMATISCH AUFLISTEN aktiviert haben. Diese Funktion ist standardmäßig allerdings ohnehin aktiviert.

Parameterinfo

Wenn Sie den Namen einer Funktion, einer Prozedur oder der Methode eines Objekts im Code-Fenster eingeben und danach die Taste Leer betätigen, wird automatisch ein Fenster angezeigt, in dem die Parameter der Funktion aufgelistet werden. Sie können dann ganz genau sehen, welches Argument als Nächstes von Ihnen erwartet wird.

Möchten Sie bei bereits eingegebenem Code diese Info nochmals abrufen, setzen Sie die Einfügemarke hinter den Namen der Funktion und klicken in der Symbolleiste BEARBEITEN auf das Symbol PARAMETERINFO.

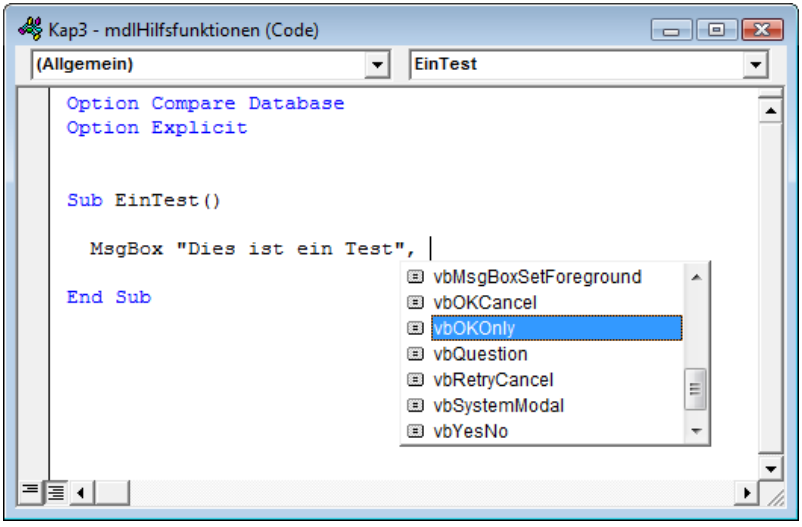
Abbildung 3.28
Die Parameterinfo für die Funktion MsgBox anzeigen



Konstanten anzeigen

Einige Funktionen, Prozeduren oder Methoden erwarten als Argumente bestimmte Konstanten. Als Beispiel sei hier die Prozedur `MsgBox` genannt, bei der Sie über Konstanten bestimmen können, welche Schaltflächen und welches Symbol der angezeigte Dialog besitzen soll. Wenn die Einfügemarke sich an einer Position befindet, an der eine Konstante als Argument erwarten wird, können Sie sich über die Schaltfläche KONSTANTEN ANZEIGEN eine Liste der zur Verfügung stehenden Konstanten anzeigen lassen.

Abbildung 3.29
Die zur Verfügung stehenden Konstanten werden angezeigt und können einfach ausgewählt werden.



QuickInfo

Wenn Sie die Einfügemarke auf den Namen einer Variablen, einer Prozedur, einer Funktion oder eine Methode setzen und anschließend auf das Symbol QUICKINFO klicken, werden Ihnen in einem Fenster Informationen zur Syntax angezeigt.

Lesezeichen setzen/zurücksetzen

Wenn Sie in einem umfangreichen Projekt arbeiten, hilft es Ihnen, dass Sie Lesezeichen in Ihrem Code setzen können. Dazu setzen Sie die Einfügemarke auf die Stelle im Code, an der Sie ein Lesezeichen setzen möchten, und klicken auf das Symbol LESEZEICHEN SETZEN/ZURÜCKSETZEN. Am Beginn der Zeile wird dann ein hellblaues abgerundetes Viereckssymbol eingefügt. Betätigen Sie das Symbol LESEZEICHEN SETZEN/ZURÜCKSETZEN erneut, wird das Lesezeichen wieder entfernt. Mithilfe von Lesezeichen können Sie schnell von einer Stelle im Programmcode zu einer anderen springen.

Sobald Sie ein oder mehrere Lesezeichen in Ihrem Quellcode platziert haben, werden weitere Symbole in der Symbolleiste BEARBEITEN aktiviert. Mit diesen können Sie zum nächsten bzw. vorherigen Lesezeichen springen. Wenn Sie die gesetzten Lesezeichen nicht mehr benötigen, können Sie mit einem Klick alle Lesezeichen auf einmal entfernen.

Ganzes Wort

Hinter diesem Symbol verbirgt sich eine Erleichterung bei der Eingabe von Befehlen. Geben Sie hierfür beispielsweise die ersten Buchstaben einer Funktion ein. Der VBA-Editor ergänzt dann automatisch die restlichen Buchstaben der Funktion, sofern er diese erkennt. Geben Sie beispielsweise einmal die Buchstaben `Ms` ein und klicken danach auf das Symbol GANZES WORT. Der VBA-Editor schlägt daraufhin die Funktion `MsgBox` vor.

Schneller geht es allerdings, wenn Sie die ersten Buchstaben einer Funktion eingeben und dann die Tastenkombination Strg + Leer drücken.

3.1.15 Die Symbolleiste Debuggen

Die Symbolleiste DEBUGGEN stellt Ihnen Funktionalität zur Verfügung, die Sie für das Testen von Programmcode benötigen. Diese Symbolleiste ist nach dem allerersten Starten der Entwicklungsumgebung standardmäßig nicht eingeblendet. Um diese Symbolleiste einzublenden, wählen Sie in der Entwicklungsumgebung aus dem Menü ANSICHT den Befehl SYMBOLLEISTEN.

Abbildung 3.30
Die Symbolleiste Debuggen

Entnehmen Sie der folgenden Tabelle die Bedeutung der einzelnen Symbole:

Tabelle 3.2
Die Symbole der Symbolleiste Debuggen

Symbol	Bedeutung
	Aktiviert und deaktiviert den Entwurfsmodus. Im Entwurfsmodus wird der Programmcode des Projekts nicht abgearbeitet und Ereignisse werden nicht ausgelöst.
	Führt die aktuelle Prozedur aus, falls sich der Cursor in einer Prozedur befindet, und öffnet anderenfalls einen Dialog zum Auswählen einer Prozedur.
	Beendet die Ausführung eines Programms und wechselt in den Haltemodus. Stattdessen können Sie auch die Taste Esc drücken, um den Ablauf der Prozedur zu stoppen.
	Beendet den Programmablauf. Es wird allerdings nicht in den Haltemodus gewechselt. Hierbei werden die AUFRUFLISTE sowie die Inhalte der Variablen auf Modulebene gelöscht.
	Setzt oder entfernt einen Haltepunkt in der aktuellen Zeile. Alternativ dazu genügt auch das Drücken der Taste F9, um einen Haltepunkt zu setzen bzw. zu entfernen. Dort, wo Haltepunkte gesetzt sind, wechselt das Programm in den Haltemodus, sobald der Programmablauf die entsprechende Stelle im Programmcode erreicht.
	Führt jeweils genau eine Anweisung im Code aus. Alternativ können Sie hier mit der Taste F8 arbeiten, um einen Code Zeile für Zeile zu durchlaufen. Wenn der Ausführungspunkt momentan auf einem Prozeduraufruf steht, wird in die Prozedur gesprungen.
	Führt im Code-Fenster jeweils eine Prozedur oder eine Anweisung im Code aus. Wenn der Ausführungspunkt momentan auf einem Prozeduraufruf steht, wird die komplette Prozedur ausgeführt.
	Führt die restlichen Zeilen einer Prozedur aus, in der sich der aktuelle Ausführungspunkt befindet.

Symbol	Bedeutung
	Blendet das LOKAL-Fenster ein.
	Blendet das Direktfenster ein. Alternativ dazu können Sie auch die Tastenkombination Strg + G drücken, um das Direktfenster einzublenden.
	Blendet das Überwachungsfenster ein.
	Zeigt das Dialogfeld AKTUELLEN WERT ANZEIGEN mit dem aktuellen Wert des ausgewählten Ausdrucks an.
	Zeigt das Dialogfeld AUFRUFLISTE an, in dem die derzeit aktiven Prozeduraufrufe (Prozeduren in der Anwendung, die aufgerufen, aber nicht abgeschlossen wurden) angezeigt werden.

Tabelle 3.2 (Forts.) Die Symbole der Symbolleiste Debuggen

3.1.16 Online-Hilfe

Wenn Sie bei der Programmentwicklung Hilfe zu einem bestimmten Objekt oder einer Funktion benötigen, können Sie die Online-Hilfe der Entwicklungsumgebung nutzen, sofern das besagte Objekt oder die Funktion in einer der mit Microsoft Office mitgelieferten Bibliotheken enthalten ist.

Möchten Sie beispielsweise mehr Informationen über eine verwendete VBA-Funktion erhalten, dann setzen Sie dort, wo Sie die Funktion verwenden, den Mauszeiger mitten in den Namen der Funktion und drücken die Taste F1.

Bei der folgenden Abbildung wurde in der Online-Hilfe nach dem Befehl `IsDate` recherchiert.

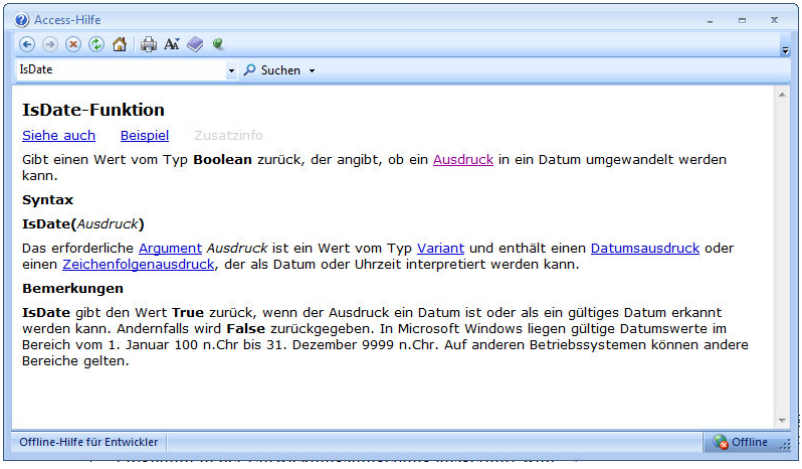


Abbildung 3.31 Schnelle Hilfe und nähere Erläuterungen aus der Online-Hilfe

3.2 Das Objekt Debug

Das Objekt `Debug` liefert Ihnen zwei sehr nützliche Methoden, die Sie bei Entwicklung und Test eines Programms nutzen können.

3.2.1 Die Anweisung Debug.Print

Über die Anweisung `Debug.Print` ist es möglich, Text im Direktfenster der Entwicklungsumgebung auszugeben. Sie können damit beispielsweise den Inhalt von Variablen ausgeben:

```
Debug.Print " Es wurden " & intAnzahl & " Datensätze gelesen."
```

Oder einfach verfolgen, ob eine Prozedur im Programmverlauf aufgerufen wurde:

```
Debug.Print "Prozedur XYZ wurde aufgerufen"
```

Die Anweisung `Debug.Print` werden Sie in den Code-Beispielen dieses Buchs sehr häufig vorfinden, da sie eine einfache Methode bietet, Informationen im Entwicklungsumfeld darzustellen.

3.2.2 Die Anweisung Debug.Assert

Eine ebenfalls sehr hilfreiche Funktionalität können Sie sich über die Anweisung `Debug.Assert` zunutze machen. Der Anweisung `Debug.Assert` folgt grundsätzlich ein boolescher Ausdruck. Wenn Sie diese Anweisung im Programmcode platzieren, definieren Sie eine Bedingung, die aus Ihrer Sicht immer gelten muss. Ist dies einmal nicht der Fall, wechselt das Programm in den Haltemodus. Der Ausführungspunkt steht dabei auf der `Assert`-Bedingung, die verletzt wurde. Sehen Sie sich dazu einmal das Beispiel in der folgenden Abbildung an.

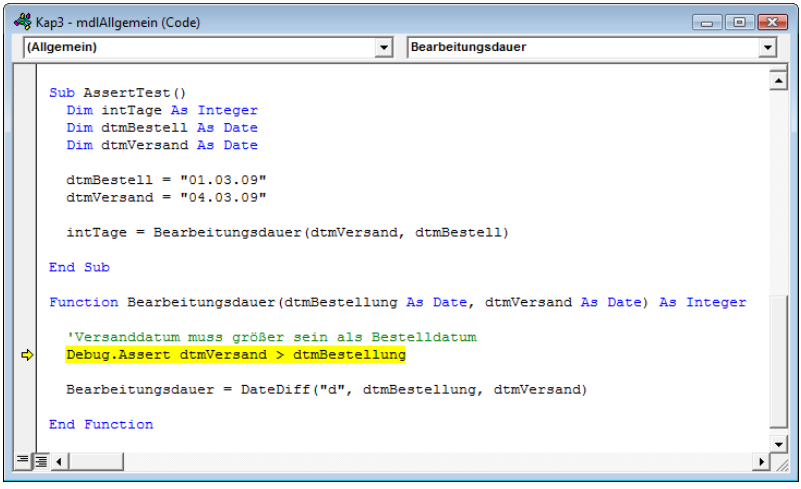


Abbildung 3.32
Die Assert-Überprüfung hat zugeschlagen.

Die im Beispiel gezeigte Funktion `Bearbeitungsdauer` berechnet die Differenz zwischen Bestelldatum und Versanddatum in Tagen. Dabei gilt natürlich immer, dass das Versanddatum nach dem Bestelldatum liegt. Zu Beginn der Funktion `Bearbeitungsdauer` wird über die Anweisung `Debug.Assert dtmVersand > dtmBestellung` bei jedem Funktionsaufruf überprüft, ob diese Bedingung erfüllt ist.

Im gezeigten Beispiel wurde die Funktion `Bearbeitungsdauer` über die Prozedur `AssertTest` bewusst mit vertauschten Argumenten aufgerufen, um einen Fehler zu simulieren. Die Abbildung zeigt, was nach Ausführen der Prozedur `AssertTest` passiert. Das Programm bleibt bei der `Assert`-Anweisung in der Prozedur `Bearbeitungsdauer` stehen und wechselt in den Haltemodus.

3.3 Fehlersuche

3.3.1 Fehlerarten

Grundsätzlich gibt es drei verschiedene Arten von Fehlern, mit denen Sie als Softwareentwickler konfrontiert werden. Es handelt sich um die drei folgenden Fehlerarten:

- Syntaxfehler
- Logische Fehler
- Laufzeitfehler

Alle drei Fehlerarten haben einen unterschiedlichen Ursprung, und Sie müssen ihnen auf unterschiedliche Weise begegnen.

Syntaxfehler

Syntaxfehler treten dann auf, wenn im Programmcode beispielsweise ein Funktionsname falsch geschrieben, ein erforderliches Schlüsselwort vergessen oder bei einem Prozeduraufruf ein Argument mit falschem Datentyp übergeben wurde, um nur einige Beispiele zu nennen.

Syntaxfehler sind recht einfach zu finden, da Sie von der Entwicklungsumgebung beim Kompilieren des Programmcodes durch eine ordentliche Fehlermeldung auf den Fehler hingewiesen werden.

Wählen Sie aus dem Menü DEBUGGEN den Menüpunkt KOMPILIEREN VON, um den gesamten Programmcode des Projekts auf Syntaxfehler zu überprüfen.

Logische Fehler

Logische Fehler sind Fehler im Design des Programmcodes. Mit anderen Worten, das entwickelte Programm lässt sich zwar ausführen, es erfüllt jedoch nicht die gewünschte Funktionalität. Sie nutzen die Werkzeuge der Entwicklungsumgebung, um das Programm zu testen und zu analysieren und um herauszufinden, wo der Fehler liegt.

Laufzeitfehler

Laufzeitfehler sind, wie der Name schon sagt, Fehler, die zur Laufzeit des Programms auftreten. Ein typisches Beispiel wäre eine Datei, die sich nicht öffnen lässt, weil sie wider Erwarten nicht vorhanden ist.

In solchen Fällen erzeugt Access eine Fehlermeldung. Fangen Sie die Fehlermeldung nicht ab, führt diese zum Abbruch des Programms. Laufzeitfehler können Sie glücklicherweise mit der `On Error`-Anweisung abfangen und behandeln. Diesen Mechanismus haben Sie bereits kennengelernt.

Laufzeitfehler können auch die Folge eines logischen Fehlers sein. Hier ein Beispiel: Tritt bei der Ausführung von Programmcode eine Division durch null auf – ein typischer Laufzeitfehler –, könnte die Ursache des Fehlers darin liegen, dass Sie es versäumt haben, eine Eingabe des Anwenders auf gültige Werte hin zu überprüfen. In diesem Fall sollten Sie dann nicht den Laufzeitfehler abfangen, sondern Ihren Programmcode um eine Überprüfung des Eingabewertes des Anwenders ergänzen.

3.3.2 Grundsätzliches zur Fehlersuche

Sobald Sie eigene Programme entwickeln, kommen Sie früher oder später unweigerlich in die Situation, dass Ihr Programm syntaktisch in Ordnung ist, die gewünschte Funktionalität jedoch ausbleibt, oder Ihr Programm etwas völlig Falsches tut.

Sie müssen dann mithilfe der Werkzeuge, die Ihnen die Entwicklungsumgebung zur Verfügung stellt, nach dem Fehler suchen. In der Sprache der Softwareentwickler wird dieser Vorgang übrigens *Debuggen* genannt.

Wir haben Ihnen in diesem Kapitel viele dieser Werkzeuge vorgestellt. Anbei haben wir in einer Liste nochmals kurz die wichtigsten Punkte bei der Fehlersuche zusammengefasst, wenn Sie momentan noch gar nicht wissen, wie Sie anfangen sollen:

- Versuchen Sie anhand des Fehlerbildes, die Fehlerursache näher einzukreisen.
- Setzen Sie über die Symbolleiste DEBUGGEN oder die Taste F9 Haltepunkte an Stellen im Programmcode, an denen die Fehlerquelle begründet sein könnte.
- Durchlaufen Sie den Code schrittweise mithilfe der Symbolleiste DEBUGGEN.
- Beobachten und analysieren Sie die Werte von Variablen im LOKAL-Fenster.
- Modifizieren Sie Werte von Variablen im LOKAL-Fenster, wenn es Ihnen bei der Fehlersuche nützlich ist.
- Machen Sie Ausgaben im Direktfenster über die Anweisung `Debug.Print`, um den Programmverlauf zu kontrollieren.

- Kommentieren Sie einzelne Codepassagen temporär aus, wenn Sie damit den Fehler weiter einkreisen können.
- Schreiben Sie ggf. kleine Testprogramme, um eine Prozedur gezielt mit bestimmten Aufrufparametern zu testen, wenn Sie den Verdacht haben, dass die betreffende Prozedur unter bestimmten Bedingungen einen Fehler aufweist. In der Softwareentwicklung wird so etwas übrigens *Unit Test* genannt.

An dieser Stelle haben wir nur die wichtigsten Aspekte der Fehlersuche erläutert. Wir empfehlen Ihnen, mit den Beispielen dieses Buchs zu arbeiten, um den Umgang mit den Debug-Werkzeugen der Entwicklungsumgebung zu trainieren!

3.4 Weitere Funktionen

3.4.1 Import und Export von Quellcode

Möchten Sie Quellcode sichern oder in einem Quellcodeverwaltungsprogramm wie Microsoft Source Safe, CVS oder Subversion verwalten, dann können Sie den Quellcode von Modulen über das Kontextmenü in Dateien exportieren. Um beispielsweise den Quellcode des Moduls MDLVERARBEITUNG zu exportieren, verfahren Sie wie folgt:

1. Klicken Sie im PROJEKT-Explorer mit der rechten Maustaste auf das Modul MDLVERARBEITUNG.
2. Wählen Sie aus dem Kontextmenü den Befehl DATEI EXPORTIEREN.

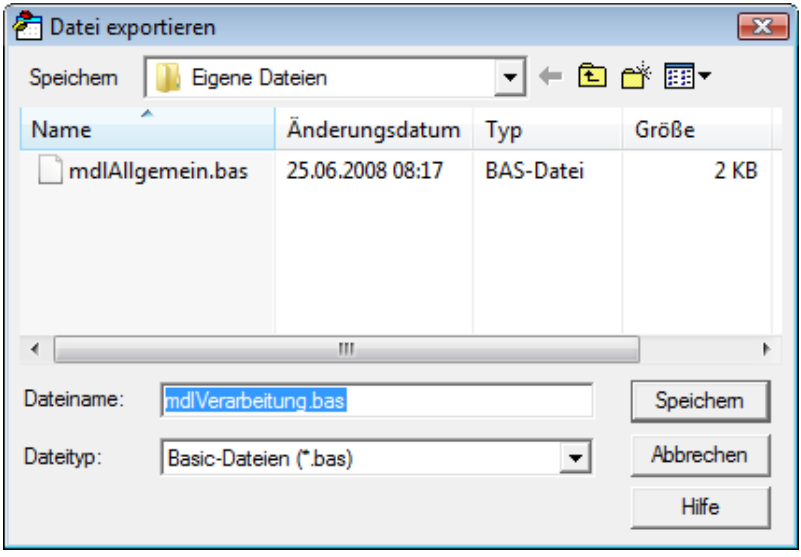
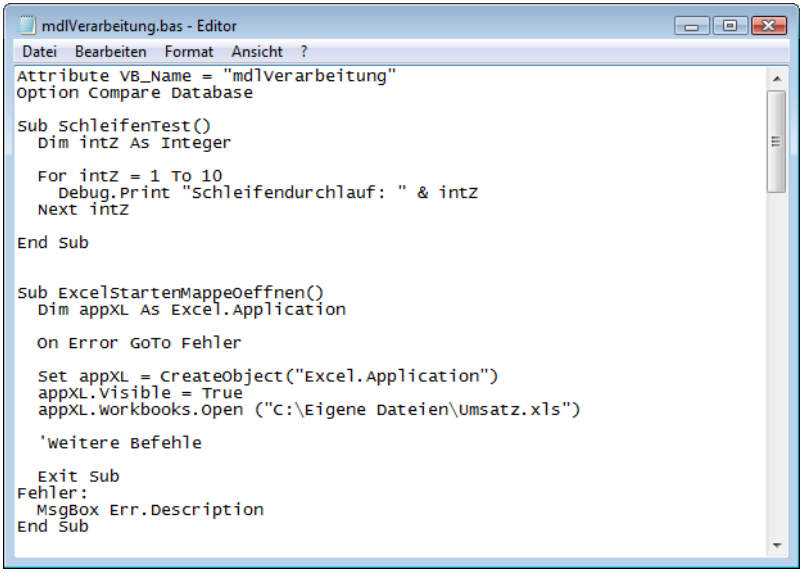


Abbildung 3.33
Den Quellcode eines Moduls in eine Datei exportieren

3. Wählen Sie einen Zielordner aus und klicken Sie auf die Schaltfläche SPEICHERN.

Diese gespeicherte Datei kann beispielsweise mit einem herkömmlichen Texteditor geöffnet werden und enthält den kompletten Quellcode des Moduls.

Abbildung 3.34
Das exportierte Modul geöffnet im Windows Editor



Der umgekehrte Vorgang, also ein Import eines Moduls, ist ebenso möglich und wird beispielsweise dann ausgeführt, wenn Sie Quellcode von einer Datenbank in eine andere Datenbank übertragen möchten.

Um einen Import durchzuführen, verfahren Sie wie folgt:

1. Nach dem Export eines Moduls öffnen Sie die Datenbank, die den Quellcode erhalten soll.
2. Wechseln Sie über die Tastenkombination Alt + F11 in die Entwicklungsumgebung von Access.
3. Klicken Sie innerhalb des PROJEKT-Explorers die rechte Maustaste und wählen Sie den Befehl DATEI IMPORTIEREN aus dem Kontextmenü.
4. Wählen Sie das gewünschte Modul aus dem Dialog DATEI IMPORTIEREN aus und bestätigen Sie mit einem Klick auf die Schaltfläche ÖFFNEN.

Hinweis

Ebenso können Sie umfangreiche Formulare auf diese Weise sichern und in eine andere Datenbank importieren. Bei den Formularen werden dabei alle darauf befindlichen Steuerelemente sowie sämtlicher Quellcode, der notwendig ist, um das Formular zu steuern, exportiert. Sie müssen daher einmal entworfene und programmierte Formulare nicht erneut entwerfen, um diese auch in anderen Datenbanken zur Verfügung zu haben.

3.4.2 Quellcode schützen

Der Zugang zum Quellcode kann in VBA geschützt werden. Damit können Sie Ihr Know-how schützen.

Um ein VBA-Projekt zu schützen, gehen Sie wie folgt vor:

1. Wechseln Sie mit der Tastenkombination Alt + F11 in die Entwicklungsumgebung von Access.
2. Wählen Sie aus dem Menü EXTRAS den Befehl EIGENSCHAFTEN VON NAME DES PROJEKTS.
3. Im Dialog PROJEKTEIGENSCHAFTEN wechseln Sie auf die Registerkarte SCHUTZ.

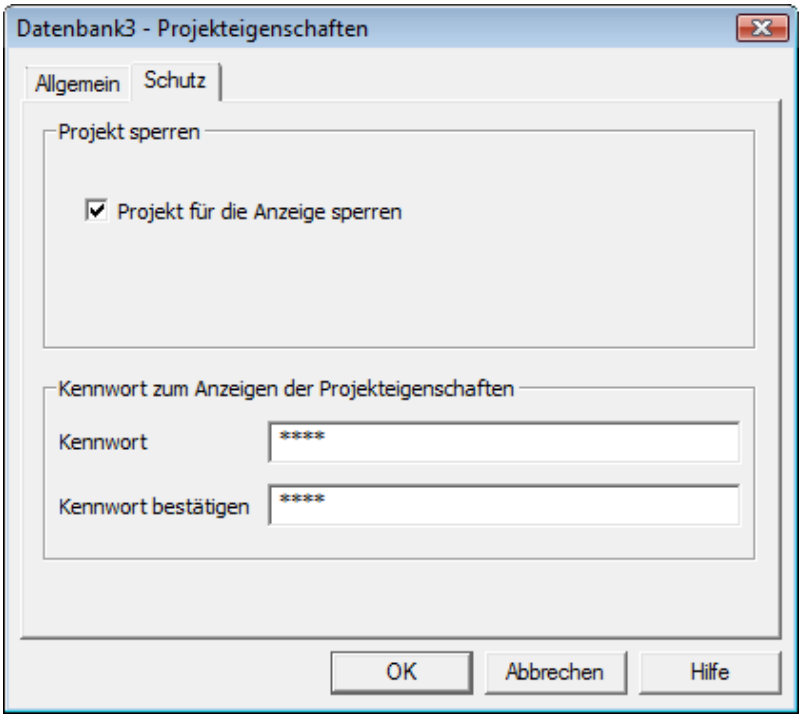


Abbildung 3.35
Den Zugriff auf den Quellcode eines Projekts verhindern

4. Aktivieren Sie das Kontrollkästchen PROJEKT FÜR DIE ANZEIGE SPERREN.
5. Vergeben Sie im Feld KENNWORT ein Passwort und bestätigen dieses im Feld KENNWORT BESTÄTIGEN.
6. Klicken Sie auf OK.

Nach dieser Einstellung speichern Sie die Datenbank. Der Zugangsschutz ist erst nach erneutem Öffnen der Datenbank aktiv. Ohne die Eingabe des Passwortes ist dann ein Zugang zum Programmcode standardmäßig nicht mehr möglich.

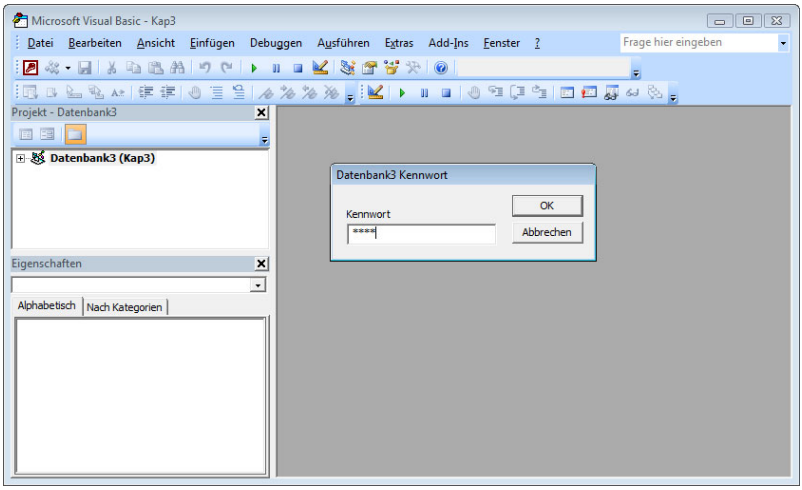


Abbildung 3.36
Der Zugang zum Quellcode ist nur über das Passwort möglich.