Seit Visual Basic 6.0 gibt es für alle Programmierer eine gute Nachricht. Durch das neue *Validate*-Ereignis wird die Eingabevalidierung deutlich erleichtert. Vorbei sind die Zeiten, in denen undurchsichtige Nebeneffekte der Ereignisse *GotFocus* und *LostFocus* die Eingabevalidierung zu einem mittleren Alptraum werden ließen. Wann immer ein Eingabefeld im Begriff ist, den Eingabefokus zu verlieren, z.B. weil der Benutzer die ⇆-Taste gedrückt hat, wird ein *Validate*-Ereignis ausgelöst. Innerhalb der Ereignisprozedur wird die Eingabe geprüft. Ist das Ergebnis in Ordnung, passiert nichts. Sollte die Abfrage allerdings gegen eine Validierungsregel verstoßen, wird durch Setzen des *Cancel*-Parameters auf *True* erreicht, daß der Eingabefokus in dem Feld bleibt.

Das folgende Beispiel zeigt die *Validate*-Ereignisprozedur eines Textfeldes, in dem nur numerische Eingaben zugelassen sind.


```
Private Sub txtPreis_Validate(Cancel As Boolean)
    If IsNumeric(txtPreis.Text) = False Then
       With txtPreis
        .SelStart = 0
        .SelLength = Len(.Text)
       End With
       Cancel = True
    End If
End Sub
```

Es ist interessant anzumerken, daß die *IsNumeric*-Funktion z.B. Währungsangaben akzeptiert (der Ausdruck »20 DM« gilt als numerisch).

Soll das *Validate*-Ereignis nicht ausgelöst werden, muß die *CausesValidation*-Eigenschaft auf *False* gesetzt werden.

Früher war es üblich, im Rahmen eines *LostFocus*-Ereignisses auch gleich eine Formatierung des eingegebenen Wertes vorzunehmen. Bei Visual Basic 6.0 (im Zusammenhang mit ADO) ist das überflüssig, denn dafür gibt es die *DataFormat*-Eigenschaft, um die es im nächsten Abschnitt geht.

9.1.1 Abfrage auf NULL

Wird ein Datensatz mit einem leeren Feld aktualisiert oder erstellt, erhält dieses Feld den Wert *NULL*. Auch wenn *NULL*-Werte kein grundsätzliches Problem darstellen, sollten sie nach Möglichkeit vermieden werden, da sie in einem Visual-Basic-Programm die Wahrscheinlichkeit eines Laufzeit-

fehlers erhöhen, denn ein Datenbankfeld mit einem *NULL*-Wert kann nur dann einer Variablen zugewiesen werden, wenn diese den Typ *Variant* besitzt (Variant-Variablen besitzen den Datentyp *Null* als Unterdatentyp – Vartype=1). NULL-Werte lassen sich nach (mindestens) zwei Methoden verhindern:

- ✘ Über eine Eingabevalidierung, die leere Eingabefelder nicht zuläßt.
- ✘ Auf der Ebene der Jet-Engine durch eine Validierungsregel, die bei der Zuweisung eines *NULL*-Wertes einen Laufzeitfehler auslöst.

Die zweite Variante bietet den Vorteil, daß sie den Programmieraufwand reduziert (Laufzeitfehler sollten ohnehin abgefangen werden). Dafür ist die erste Variante etwas flexibler, denn sollten sich die Validierungsregeln eines schönen Tages ändern, ist keine Änderung an der Datenbank erforderlich (erfahrungsgemäß ist es nämlich nirgendwo dokumentiert, welche Felder eine Validierungsregel erhalten haben).

9.2 Die formatierte Anzeige von Datenbankfeldern

Alle Steuerelemente, die an eine OLE DB-Datenquelle gebunden werden können, besitzen eine *DataFormat*-Eigenschaft, über die sich ein bestimmtes Darstellungsformat auswählen läßt. Das ist sehr praktisch, denn wenn ein Datenbankfeld etwa eine Währung darstellt, möchte man in der Regel auch, daß der Feldinhalt in einem Währungsformat dargestellt wird. Bindet man das Datenbankfeld an ein Textfeld, wird der Inhalt ohne *DataFormat*-Eigenschaft entweder als Zeichenkette ohne Formatierung dargestellt, oder der Inhalt muß programmgesteuert »nachformatiert« werden, was bei gebundenen Steuerelementen alles andere als einfach ist und unerwünschte Nebeneffekte nach sich ziehen kann (etwa, wenn sich das dargestellte Format nicht mit dem Format verträgt, das die Datenbank erwartet). Die *DataFormat*-Eigenschaft löst dieses Problem, indem jedes Datenbankfeld ein eigenes Darstellungsformat erhält, ohne daß dies dem Format des Datenbankfeldes in die Quere kommt. Die Auswahl des Formats erfolgt im Eigenschaftsfenster des Steuerelements, in dem Sie die *DataFormat*-Eigenschaft selektieren, das Dialogfeld öffnen und das gewünschte Format auswählen.

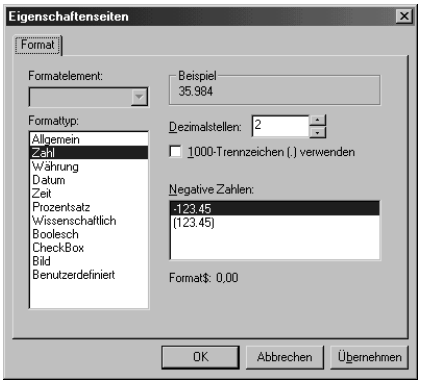


Bild 9.1: In diesem Dialogfeld wird einem Steuerelement ein Format zugewiesen

Hinter der *DataFormat*-Eigenschaft steckt mehr, als man es zunächst vermuten würde. Zuständig für die Formatierung ist das *StdDataFormat*-Objekt, das zur *StdDataFormats*-Auflistung gehört und mit *Changed*, *Format* und *UnFormat* sogar über eigene Ereignisse verfügt.

Ereignis	**Wann wird es ausgelöst?**
Changed	Wenn sich der Inhalt des Steuerelements ändert.
Format	Wenn dem Steuerelement nach dem Auslesen des Datenbankfeldes eine Formatierung zugewiesen wird.
UnFormat	Wenn die Formatierung vor dem Zurückschreiben in die Datenbank wieder aufgehoben wird.

Tabelle 9.1: Die Ereignisse des Data-Format*-Objekts*

9.2.1 Mehr über das *DataFormat*-Objekt

Das *DataFormat*-Objekt ist weit mehr als nur eine oberflächlich hinzugefügte »Schönschreibhilfe«. Wer sich näher mit dem *DataFormat*-Objekt beschäftigt, gerät schnell in die etwas tieferen Gefilde der OLE DB-Programmierung und erhält eine erste Vorahnung davon, daß OLE DB und ADO sehr viel mehr sind als nur eine neue Variante im »Namenspiel« der Abkürzungen und Modebegriffe[1]. Da das *DataFormat*-Objekt in der MSDN-Hilfe gut dokumentiert ist, soll es im folgenden bei einem einfachen Beispiel bleiben.

1 Davon konnte ich Sie hoffentlich bereits in der Einleitung überzeugen.

Damit sich die *DataFormat*-Ereignisse nutzen lassen, müssen in das Projekt sowohl eine Referenz auf die »Microsoft Data Formatting Object Library« als auch auf die »Microsoft Data Binding Collection« eingebunden werden.

Das folgende Beispiel zeigt, wie sich das *DataFormat*-Objekt im Zusammenhang mit einem Textfeld nutzen läßt, in dem der Inhalt des Feldes *Preis* der Tabelle *Fahrzeugdaten* unserer Fuhrpark-Datenbank angezeigt wird. Sie müssen die in dem Beispiel enthaltenen Befehle nicht ausführen, um die automatische Formatierung benutzen zu können. Die in dem Beispiel vorgestellten Schritte sind immer dann notwendig, wenn Sie aus irgendeinem Grund in den Vorgang der automatischen Formatierung eingreifen und den Inhalt einer gebundenen Eigenschaft ändern möchten, bevor dieser in die Datenbank zurückgeschrieben wird.

Schritt 1:

Legen Sie ein neues Standard-Exe-Projekt an, und ordnen Sie darauf ein ADO-Datensteuerelement (*adoData*) an.

Schritt 2:

Stellen Sie über die Eigenschaften *ConnectionString* und *RecordSource* eine Verbindung zur Tabelle *Fahrzeugdaten* der Datenbank *Fuhrpark.mdb* her (siehe Kapitel 4).

Schritt 3:

Ordnen Sie auf dem Formular ein Textfeld (*txtPreis*) an. Setzen Sie die *DataSource*-Eigenschaft auf den Namen des ADO-Datensteuerelements und die *DataField*-Eigenschaft auf den Namen des Feldes.

Schritt 4:

Fügen Sie in den Deklarationsteil des Formulars folgende Befehle ein:

```
Private WithEvents Fo1 As StdDataFormat
Private B As BindingCollection
```

Dadurch werden zwei Objektvariablen definiert. Die Variable *Fo1* erhält die Ereignisse, die später durch das *DataFormat*-Objekt ausgelöst werden. Die Variable ist dagegen für die Bindung zwischen dem Datensteuerelement und dem Textfeld zuständig.

Schritt 5:

Fügen Sie in das *Form_Load*-Ereignis folgende Befehle ein:

```
Private Sub Form_Load()
    Set B = New BindingCollection
    Set B.DataSource = adoData
    Set Fo1 = New StdDataFormat
    With Fo1
     .Type = fmtCustom
     .Format = "Currency"
    End With
    B.Add Object:=txtPreis, PropertyName:="Text", _
      DataField:="Preis", DataFormat:=Fo1
End Sub
```

Als erstes wird ein *BindingCollection*-Objekt instanziert, das für alle Bindungen zwischen Steuerelementen und den über das ADO-Datensteuerelement verbundenen Datenbankfeldern stehen soll. Anschließend wird das *DataFormat*-Objekt instanziert und mit Formatierungsinformationen belegt. Der entscheidende Schritt findet im letzten Befehl statt. Hier wird zur Auflistung der Bindungsobjekte das gebundene Textfeld, die gebundene Eigenschaft (theoretisch können bei einem Steuerelement mehrere Eigenschaften gebunden werden), das Datenbankfeld und zum Schluß das *DataFormat*-Objekt hinzugefügt. Erst dadurch werden die das Textfeld betreffenden Ereignisse an das *DataFormat*-Objekt durchgereicht.

Schritt 6:

Fügen Sie in das *Format*-Ereignis des *Fo1*-Objekts folgende Befehle ein:

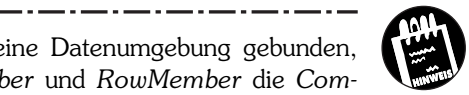
```
Private Sub Fo1_Format _
  (ByVal DataValue As StdFormat.StdDataValue)
    If DataValue > 8000 Then
       txtPreis.ForeColor = vbRed
    Else
       txtPreis.ForeColor = vbBlack
    End If
End Sub
```

Ging alles gut, wird das *Format*-Ereignis immer dann aufgerufen, wenn das Textfeld einen neuen Wert von der Datenbank enthält. Durch die Abfrage des übergebenen Wertes (dieser Wert soll im Textfeld angezeigt werden), läßt sich eine individuelle Formatierung erreichen.

9.3 Das *DataCombo*-Steuerelement zum Darstellen einer n:1-Beziehung

Unter den gebundenen Steuerelementen gibt es einige wenige, die ausschließlich in einer Datenbankanwendung eingesetzt werden. Das *DataCombo*-Steuerelement ist das beste Beispiel. Während sich das *DataGrid* auch als komfortabler Tabellenersatz verwenden läßt, wird man die tiefere Bedeutung des *DataCombo*-Steuerelements erst dann verstehen, wenn man etwas über Beziehungen zwischen Tabellen und den Unterschied zwischen einem Primär- und einem Fremdschlüssel gelernt hat. Anstatt langer Vorreden (das *DataCombo*-Steuerelement wurde bereits in Kapitel 4 eingeführt und wird selbstverständlich in der MSDN-Hilfe ausführlich beschrieben), gleich ein konkretes Beispiel. Stellen Sie sich vor, Sie möchten ein Formular für das Bearbeiten von Datensätzen in der *Fahrzeugdaten*-Tabelle erstellen. Wird ein neues Fahrzeug in den Fuhrpark aufgenommen, müssen Sie neben der Fahrzeugnummer auch die Modellnummer angeben. Doch im allgemeinen möchten Sie dem Anwender nicht zumuten, für jeden zu erfassenden Fahrzeugtyp die Modellnummer auswendig aufsagen zu können. Insbesondere dann nicht, wenn die Tabelle nicht nur mehrere Dutzend, sondern vielleicht mehrere Tausend verschiedene Datensätze umfaßt. Sich die Tabelle *Modelldaten* auszudrucken und auf den Schreibtisch zu legen, kann auch nicht der Sinn der Datenbankprogrammierung sein. Die vermutlich beste Lösung ist es, wenn der Anwender anstelle einer aussagelosen Modellnummer den Modellnamen auswählt und anstelle des Modellnamens, der in der Tabelle *Fahrzeugdaten* aber nicht vorgesehen ist (denken Sie an die Normalisierungsregeln und die Vermeidung von Redundanz), die dazugehörige Modellnummer eingetragen wird. Diese Beziehung zwischen zwei Datensatzgruppen wird über das *DataCombo*-Steuerelement hergestellt. Leider ist das *DataCombo*-Steuerelement weder beim ersten noch beim zweiten Durchlesen der Hilfetexte verständlich. Damit Sie zwischen *BoundColumn*- und *RowSource*-Eigenschaften nicht den Überblick verlieren, macht Bild 9.2 die Verhältnisse beim *DataCombo*-Steuerelement deutlich. Denken Sie daran, daß es hier um die klassische Beziehung zwischen einer Tabelle mit einem Fremdschlüssel (in diesem Beispiel die Tabelle *Fahrzeugdaten*) und einer Tabelle mit einem Primärschlüssel (in diesem Fall die Tabelle *Modelldaten*) geht. Sie bewegen sich durch eine Datensatzgruppe (die Fahrzeugdaten) und möchten, daß über das Feld *ModellNr* (der Fremdschlüssel) ein Datensatz in der Tabelle *Modelldaten* ausgewählt, anstelle des

Primärschlüssels (*ModellNr*) aber ein anderes Feld (*ModellName*) angezeigt wird. Damit das *DataCombo*-Steuerelement seine Aufgaben erfüllen kann, werden stets zwei Datensatzgruppen, die z.B. über zwei Datensteuerelemente zur Verfügung gestellt werden, benötigt. Mit diesem Hintergrundwissen ist die Funktionsweise des *DataCombo*-Steuerelements dann doch nicht so schwer zu verstehen.



Wird das *DataCombo*-Steuerelement an eine Datenumgebung gebunden, müssen über die Eigenschaften *DataMember* und *RowMember* die *Command*-Objekte ausgewählt werden.

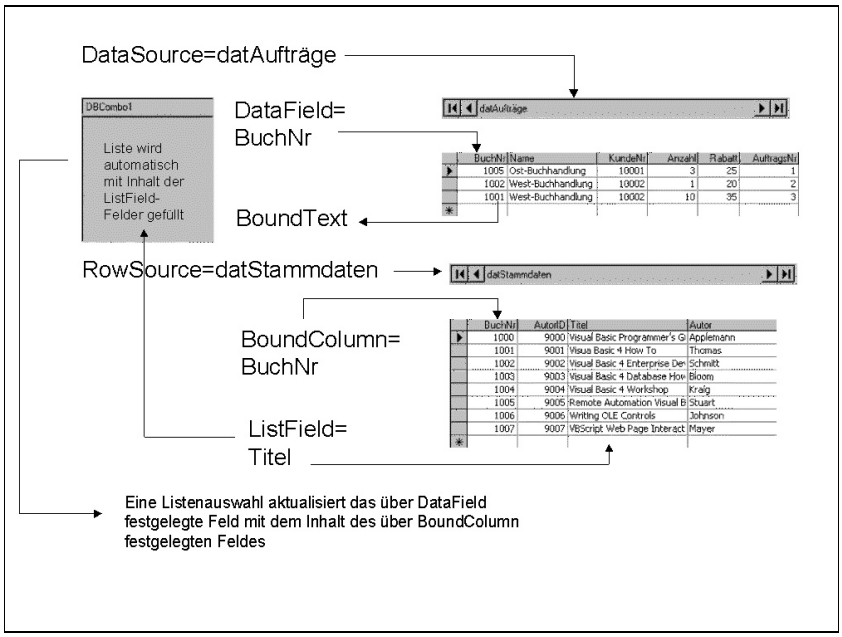


Bild 9.2: Das Data-Combo-*Steuerelement zeigt den Inhalt eines Feldes an, das von einer anderen Datensatzgruppe über ein gemeinsames Feld ausgewählt wird*

Das folgende Beispiel macht den Umgang mit dem *DataCombo*-Steuerelement am Beispiel der Tabellen *Fahrzeugdaten* und *Modelldaten* aus der Fuhrpark-Datenbank deutlich.

Schritt 1:

Legen Sie ein neues Standard-Exe-Projekt an, und ordnen Sie darauf ein ADO-Datensteuerelement (*adoFahrzeugdaten*) an.

Schritt 2:

Ordnen Sie auf dem Formular ein weiteres ADO-Datensteuerelement (*adoModelldaten*) an. Setzen Sie die *Visible*-Eigenschaft auf *False* (das ist allerdings nicht zwingend notwendig).

Schritt 3:

Verbinden Sie das *adoModelldaten*-Steuerelement mit der Tabelle *Modelldaten* der Fuhrpark-Datenbank (siehe Kapitel 4).

Schritt 4:

Verbinden Sie das *adoFahrzeugdaten*-Steuerelement mit der Tabelle *Fahrzeugdaten* der Fuhrpark-Datenbank (siehe Kapitel 4).

Schritt 5:

Ordnen Sie auf dem Formular (wie es in Bild 9.3 zu sehen ist), zwei Textfelder (*txtFahrzeugNr* und *txtPreis*) und ein *DataCombo*-Steuerelement (*dbcModellnamen*) an. Letzteres muß über den Menübefehl PROJEKT/KOMPONENTEN und Auswahl des Eintrags »Microsoft Data List Controls 6.0 (OLED6)« erst zur Werkzeugsammlung hinzugefügt werden.

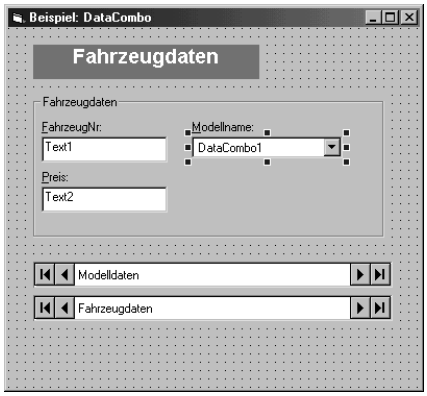


Bild 9.3: Das Formular mit dem DataCombo*-Steuerelement zur Entwurfszeit*

Schritt 6:

Nun müssen beim *DataCombo*-Steuerelement eine Reihe von Eigenschaften gesetzt werden, die in Tabelle 9.2 zusammengestellt sind.

Eigenschaft	Wert	Grund
BoundColumn	*ModellNr*	Über dieses Feld der Tabelle *Modelldaten* (es handelt sich um den Primärschlüssel) wird die Verbindung zur Tabelle *Fahrzeugdaten* hergestellt.
DataField	*ModellNr*	Über dieses Feld der Tabelle *Fahrzeugdaten* (es handelt sich um den Fremdschlüssel) wird die Verbindung zur Tabelle *Modelldaten* hergestellt. Es wird allerdings nicht angezeigt.
DataSource	*adoFahrzeugdaten*	Dies ist die Tabelle mit dem Sekundärschlüssel, deren Datensätze über das sichtbare Datensteuerelement ausgewählt werden.
ListField	*Modellname*	Mit dem Wert dieses Feldes soll die Liste gefüllt werden.
RowSource	*adoModelldaten*	Dies ist die Tabelle mit dem Primärschlüssel. Sie spielt die Rolle der »Look-up-Tabelle«, die zum Nachschlagen des Modellnamens verwendet wird.

Tabelle 9.2: Die Einstellungen beim DataCombo-*Steuerelement*

Die *BoundText*-Eigenschaft muß nicht eingestellt werden. Über sie ist während der Programmausführung der Wert des Fremdschlüssels zugänglich.

Schritt 7:

Starten Sie das Programm. Ging alles gut, sollten Sie sich mit dem ADO-Datensteuerelement in der Tabelle *Fahrzeugdaten* bewegen können. Anstelle der Modellnummer sollte in dem *DataCombo*-Steuerelement der dazugehörige Modellname erscheinen. Doch Vorsicht, durch Auswahl eines anderen Modellnamens wird automatisch die Modellnummer in der Tabelle *Fahrzeugdaten* geändert. Wie sich dies verhindern läßt, wird im Zusammenhang mit den ADO-Objekten in Kapitel 5 erklärt. Das Setzen der *LockType*-Eigenschaft beim *adoFahrzeugdaten*-Steuerelement auf den Wert *1-adLockReadOnly* ist vermutlich keine Lösung, da dies zu einem Laufzeitfehler führt (*Error*-Ereignis beim Datensteuerelement), was wiederum bewirkt, daß der Datensatzzeiger nicht verändert wird. Ein Bewegen innerhalb der Datensatzgruppe ist in diesem Fall nicht möglich.

Das *DataCombo*-Steuerelement ist ein zentrales Element in Formularen zur Datenerfassung. Das vorgestellte Beispiel ist noch nicht komplett. Soll das Formular für das Entleihen eines Fahrzeugs verwendet werden, müssen über mehrere *DataCombo*-Steuerelemente mehrere Beziehungen zwischen der Tabelle *Entleihdaten* und den verschiedenen Stammdatentabellen aufgebaut werden:

Beziehung	Rolle für das Formular
Entleihdaten -> Fahrzeugdaten	Über das Feld *FahrzeugNr* wird eine Beziehung zu den Fahrzeugdaten hergestellt.
Entleihdaten -> Mitarbeiterdaten	Über das Feld *MitarbeiterNr* wird eine Beziehung zur Tabelle *Mitarbeiterdaten* hergestellt.

9.4 Das *HFlexGrid*-Steuerelement zur Anzeige hierarchischer Datensatzgruppen

Hierarchische Datensatzgruppen, was ist denn das schon wieder? Keine Sorge, es ist alles halb so wild, denn eine Hierarchie zwischen zwei Datensatzgruppen haben Sie bereits im letzten Abschnitt kennengelernt. Es ist lediglich eine andere Formulierung für eine Beziehung (etwa n:1), wie sie zwischen Tabellen (und damit Datensatzgruppen) häufig anzutreffen ist. Das Besondere an den hierarchischen Datensatzgruppen, die bereits in Kapitel 6 im Zusammenhang mit dem Datenumgebungs-Designer angesprochen wurden, ist, daß sie sich durch den Umstand auszeichnen, daß ein Feld der Datensatzgruppe A anstelle eines einfachen Wertes ein anderes *Recordset*-Objekt enthält, das aus einer anderen Datensatzgruppe B stammt. Ändert sich der Datensatz in der Datensatzgruppe A, ändert sich dadurch (unter Umständen) auch der Datensatz in der Datensatzgruppe B. Diese hierarchischen Beziehungen werden über das SQL-Kommando *SHAPE* hergestellt, das in diesem Buch allerdings nicht vorgestellt wird, da es bereits in die Kategorie der fortgeschrittenen ADO-Datenbankprogrammierung fällt. Der Umgang mit hierarchischen Datensatzgruppen ist dennoch nicht sonderlich kompliziert, wenn man das Zusammenstellen der Hierarchie dem Datenumgebungs-Designer überträgt und das Ergebnis mit dem *HFlexGrid*-Steuerelement betrachtet, das Sie bereits aus Kapitel 4 kennen. Da im Grunde alles bereits gesagt wurde (und die MSDN-Hilfe wie immer alle Eigenschaften und Methoden ausführlich beschreibt), wird in diesem Abschnitt lediglich gezeigt, wie sich eine Beziehung zwischen einem Mitarbeiter der Tabelle *Mitarbeiter* und der von diesem Mitarbeiter ausgeliehenen Fahrzeuge der Ta-

belle *Ausleihdaten* herstellen läßt. Konkret, wann immer ein neuer Mitarbeiterdatensatz ausgewählt wird, sollen alle Fahrzeuge angezeigt werden, die von diesem Mitarbeiter ausgeliehen wurden.

Das folgende Beispiel geht davon aus, daß die Fuhrpark-Datenbank zwei Tabellen umfaßt: *Mitarbeiterdaten* und *Ausleihdaten*, deren Aufbau in Tabelle 9.3 und Tabelle 9.4 enthalten sind. Diese beiden Tabellen sind Voraussetzung dafür, daß mit der Fuhrpark-Datenbank auch Ausleihvorgänge möglich sind (wenngleich die erforderliche »Logik«, wie z.B. das Überprüfen, ob ein Wagen überhaupt verfügbar ist, in diesem Buch nicht behandelt wird).

Feld	**Datentyp**
AusleihNr	*Long*
FahrzeugNr	*Long*
MitarbeiterNr	*Long*
AusgeliehenAm	*Date/Time*
RückgabeAm	*Date/Time*

Tabelle 9.3: Der Aufbau der Tabelle Ausleihdaten

Feld	**Datentyp**
MitarbeiterNr	*Long*
Name	*Text*
Abteilung	*Text*

Tabelle 9.4: Der Aufbau der Tabelle Mitarbeiterdaten

Schritt 1:

Legen Sie ein neues Standard-Exe-Projekt an, und ordnen Sie auf dem Formular ein Textfeld (*txtMitarbeiterName*) an.

Schritt 2:

Fügen Sie zur Werkzeugsammlung das »Microsoft Hierachical FlexGrid Control 6.0 (OLED6)«-Steuerelement hinzu, und ordnen Sie es auf dem Formular an.

Schritt 3:

Fügen Sie zum Projekt die für die Fuhrpark-Datenbank in Kapitel 6 angelegte Datenumgebung (ihr Name sollte *Fuhrpark.dsr* lauten) hinzu. Diese Datenumgebung wird im folgenden um ein hierarchisches *Command*-Objekt erweitert.

Schritt 4:

Fügen Sie zur Datenumgebung ein neues *Command*-Objekt hinzu, geben Sie ihm den Namen »cmdMitarbeiterdaten«, und verbinden Sie es mit der Tabelle *Mitarbeiterdaten*.

Schritt 5:

Fügen Sie zu *cmdMitarbeiterdaten* ein Unter-*Command*-Objekt hinzu, geben Sie ihm den Namen »cmdEntlieheneFahrzeuge«, verbinden Sie es mit der Tabelle *Ausleihdaten*.

Schritt 6:

Schalten Sie auf die Registerkarte *Beziehung* um, und fügen Sie die bereits aufgeführte Beziehung zwischen *MitarbeiterNr* und *MitarbeiterNr* hinzu.

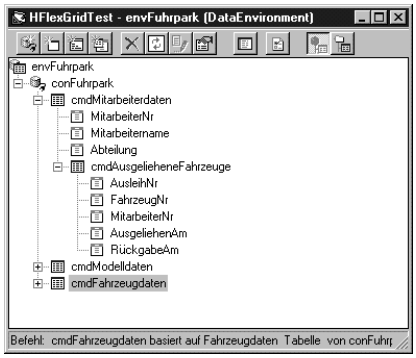


Bild 9.4: Die Datenumgebung zeigt die Beziehung zwischen Mitarbeiterdaten und Ausleihdaten an

Schritt 7:

Setzen Sie die *DataSource*-Eigenschaft des *HFlexGrid*-Steuerelements auf den Namen der Datenumgebung (*envFuhrpark*), und wählen Sie über die *DataMember*-Eigenschaft das *Command*-Objekt *cmdMitarbeiterDaten* aus.

Schritt 8:

Verbinden Sie das Textfeld über seine *DataSource*-Eigenschaft mit *envFuhrpark*, über seine *DataMember*-Eigenschaft mit *cmdMitarbeiterdaten* und über seine *DataField*-Eigenschaft mit dem Feld *Mitarbeitername*.

Schritt 9:

Fügen Sie zu dem Formular zwei Schaltflächen (*cmdVor* und *cmdZurück*) hinzu, mit denen ein Bewegen in der Datensatzgruppe möglich wird. Fügen Sie in die *Click*-Prozeduren die entsprechende Befehle ein:

```
Sub cmdVor_Click ()
    envFuhrpark.rscmdMitarbeiterdaten.MoveNext
End Sub
Sub cmdZurück_Click ()
    envFuhrpark.rscmdMitarbeiterdaten.MovePrevious
End Sub
```

Schritt 10:

Starten Sie das Programm. Wenn Sie sich mit den Schaltflächen in der Datensatzgruppe bewegen, sollte zu jedem Mitarbeiter die von ihm oder ihr ausgeliehenen Fahrzeuge angezeigt werden (was natürlich voraussetzt, daß beide Tabellen bereits einige Datensätze enthalten).

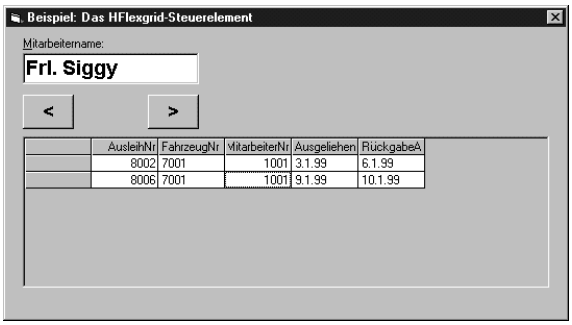


Bild 9.5: Das HFlexGrid-*Steuerelement stellt ein hierarchisches* Command-*Objekt dar*

Natürlich werden in diesem Beispiel die Möglichkeiten des *HFlexGrid*-Steuerelements nicht vollständig genutzt. Wenn Sie das *HFlexGrid* alternativ mit der Datensatzgruppe *cmdMitarbeiterdaten* verbinden oder bereits zu Beginn das *Command*-Objekt *cmdMitarbeiterdaten* mit der rechten Maustaste auf das Formular ziehen und den Eintrag *Hierarchical Flex grid* wählen, wird auf dem Formular ein *HFlexGrid*-Steuerelement angeordnet, in dem alle Datensätze von *cmdMitarbeiterdaten* angezeigt werden. Jeder Datensatz, zu dem in *cmdEntliehenFahrzeuge* Datensätze existieren, wird (wie üblich) mit einem »+«-Zeichen gekennzeichnet.

9.5 Ein Formular zum Bearbeiten der *Modelldaten*-Tabelle

In diesem Abschnitt wird ein Formular vorgestellt, mit dem das Erfassen neuer Datensätze in der Tabelle *Modelldaten* möglich wird. Bild 9.6 zeigt das Formular, in dem ein Datensatz dargestellt ist. Neben Textfeldern für jedes einzelne Feld (allerdings ohne spezielle Formatierungseinstellungen) enthält das Formular insgesamt neun Schaltflächen, mit denen sich die wichtigsten Datenbankoperationen erledigen lassen:

- ✘ Bewegen in der Datensatzgruppe (Vor, Zurück, Anfang und Ende)
- ✘ Hinzufügen eines neuen Datensatzes
- ✘ Aktualisieren des aktuellen oder neu hinzugefügten Datensatzes
- ✘ Löschen des aktuellen Datensatzes
- ✘ Suchen nach einem oder mehreren Datensätzen
- ✘ Den aktuellen Datensatz neu anzeigen

Bild 9.6: Das Formular zum Bearbeiten eines Modelldatensatzes

Die Anbindung an die Datenbank wird über die Datenumgebung *Fuhrpark.dsr* (Kapitel 6) durchgeführt, mit der jedes einzelne Textfeld über seine Eigenschaften *DataSource*, *DataMember* und *DataField* verbunden wird. Der Zugriff auf die Tabelle *Modelldaten* erfolgt über das *Command*-Objekt

rscmdModelldaten, das zu Beginn der Objektvariablen *RsModelldaten* zugewiesen wird:

```
Private WithEvents RsModelldaten As ADODB.Recordset
```

Damit *RsModelldaten* auch auf die Ereignisse des Recordset-Objekts reagieren kann, wird das Schlüsselwort *WithEvents* verwendet. Die Instanzierung erfolgt in *Form_Load*:

```
Set RsModelldaten = envFuhrpark.rscmdModelldaten
If RsModelldaten.State = adStateClosed Then
   RsModelldaten.Open
End If
```

Auch wenn es nicht zwingend notwendig ist, wird vor dem Öffnen des *Recordset*-Objekts geprüft, ob es nicht bereits geöffnet ist.

9.5.1 Das Bewegen in der Datensatzgruppe

Das Bewegen in der Datensatzgruppe bedarf keiner tiefergehenden Analysen, denn es besteht aus dem Aufruf der dafür zuständigen Methode. Dennoch ist es lehrreich, eine dieser Aktionen exemplarisch vorzustellen:

```
Private Sub cmdVor_Click()
    RsModelldaten.MoveNext
    If RsModelldaten.EOF = True Then
       RsModelldaten.MoveLast
       Status "Ende der Datensatzgruppe erreicht"
    End If
    AnzahlDatensätzeAnzeigen
End Sub
```

Zuerst wird der Datensatzzeiger nach vorne bewegt, anschließend wird geprüft, ob damit das Ende der Datensatzgruppe überschritten wurde.

9.5.2 Hinzufügen eines neuen Datensatzes

Für das Hinzufügen eines Datensatzes ist die *AddNew*-Methode zuständig. Es wird ein neuer Datensatz an die Datensatzgruppe angefügt, und alle gebundenen Steuerelemente werden automatisch gelöscht. Das eigentliche Hinzufügen geschieht über die *Update*-Methode (oder indem eine Aktion durchgeführt wird, wie z.B. ein erneuter Aufruf von *AddNew*, der ein Update erforderlich macht). Da zwischen dem Entschluß, einen neuen Datensatz hinzufügen zu wollen, und dem Aufruf der *Update*-Methode erst einmal

die Daten eingegeben werden müssen, gibt es für das Aktualisieren eine eigene Schaltfläche.

Die hinzuzufügenden Daten können bereits beim Aufruf der *AddNew*-Methode (als Feldvariable vom Typ *Variant*) übergeben werden. In diesem Fall ist der Aufruf der *Update*-Methode nicht erforderlich. Das funktioniert aber (anscheinend) nicht bei gebundenen Steuerelementen, da deren Inhalt mit dem Aufruf der *AddNew*-Methode automatisch gelöscht wird.

Da nach dem Anklicken der *Hinzufügen*-Schaltfläche und damit verbunden dem Aufruf der *Update*-Methode vermieden werden soll, daß der Anwender (unbeabsichtigt natürlich) die anderen Schaltflächen betätigt, werden diese deaktiviert, was den Programmcode etwas umfangreicher erscheinen läßt, als es notwendig wäre:

```
Private Sub cmdHinzufügen_Click()
If UpdateModus = False Then
       RsModelldaten.AddNew
       UpdateModus = True
       cmdHinzufügen.Caption = "A&bbrechen"
       fraNavigation.Visible = False
       cmdLöschen.Enabled = False
       cmdSuchen.Enabled = False
       txtModellNr.Text = LetzteModellNr + 1
       txtModellname.SetFocus
    Else
       RsModelldaten.CancelUpdate
       Status "Hinzufügen abgebrochen"
       UpdateModus = False
       cmdHinzufügen.Caption = "&Hinzufügen"
       fraNavigation.Visible = True
       cmdLöschen.Enabled = True
       cmdSuchen.Enabled = True
    End If
End Sub
```

Übrigens kann der »Hinzufügen-Zustand« beim *Recordset*-Objekt auch über die *EditMode*-Eigenschaft abgefragt werden. Der Aufruf der *Update*-Methode erfolgt nach dem Anklicken der *Aktualisieren*-Schaltfläche:

```
Private Sub cmdAktualisieren_Click()
' Durchgeführte Änderungen in Datenbank übernehmen
    RsModelldaten.Update
```

```
    If UpdateModus = True Then
       Status "Datensatz wurde hinzugefügt"
       LetzteModellNr = txtModellNr.Text
       AnzahlDatensätzeAnzeigen
       cmdHinzufügen.Caption = "&Hinzufügen"
       fraNavigation.Visible = True
       cmdLöschen.Enabled = True
       cmdSuchen.Enabled = True
       txtModellname.SetFocus
       UpdateModus = False
    End If
End Sub
```

Damit die Modellnummer, die der neu hinzugefügte Datensatz erhält, nicht aus der Datenbank abgefragt werden muß (das Feld *ModellNr* ist ein Autozähler-Feld), wird diese in einer separaten Variablen gespeichert.

9.5.3 Aktualisieren des Datensatzes

Da das Aktualisieren des aktuellen Datensatzes auch unabhängig vom Hinzufügen eines Datensatzes erforderlich sein kann, gibt es dafür eine eigene Schaltfläche, in deren *Click*-Prozedur, wie im letzten Abschnitt beschrieben, die *Update*-Methode ausgeführt wird.

9.5.4 Löschen des aktuellen Datensatzes

Das Löschen eines Datensatzes erfolgt über den Aufruf der *Delete*-Methode, wobei es im allgemeinen sinnvoll ist, vom Benutzer eine Bestätigung zu verlangen:

```
Private Sub cmdLöschen_Click()
    Dim Antwort As VbMsgBoxResult
    Antwort = MsgBox(Prompt:="Aktuellen Datensatz löschen?", _
     Buttons:=vbYesNo + vbQuestion, _
     Title:="Bitte Löschoperation bestätigen!")
    If Antwort = vbYes Then
       RsModelldaten.Delete
       Status "Datensatz wurde gelöscht"
    End If
    cmdVor_Click
End Sub
```

Da durch das Löschen des Datensatzes der aktuelle Datensatzzeiger unbestimmt ist, wird am Ende die *cmdVor_Click*-Prozedur aufgerufen, die eine *MoveNext*-Operation durchführt und dabei gleichzeitig jenen Fall prüft, daß das Ende der Datensatzgruppe erreicht wird. Nicht geprüft wird jener Fall, in dem der letzte Datensatz gelöscht wird und sowohl *EOF* als auch *BOF* den Wert *True* annehmen und eine *MoveNext*-Operation daher nicht möglich ist.

9.5.5 Suchen nach einem oder mehreren Datensätzen

Wie sich das Suchen nach einem Datensatz in einer Datensatzgruppe über die *Find*-Methode durchführen läßt, wurde bereits in Kapitel 5 beschrieben. In diesem Abschnitt lernen Sie in erster Linie ein passendes Dialogfeld kennen, in dem die Suchwerte eingegeben werden müssen. Bild 9.7 zeigt das Dialogfeld, in dem für jeden Suchbegriff ein eigenes Feld existiert. Damit der Anwender eine maximale Flexibilität erhält, ist die Suche nach jedem Datenbankfeld möglich. Doch was ist, wenn mehrere Begriffe gesucht werden sollen? Nun, für diesen Fall müssen die Suchbegriffe mit dem AND-Operator (SQL) verknüpft werden. Da die *Find*-Methode bei ADO aber nur einen Suchbegriff zuläßt, wird auf die *Filter*-Methode ausgewichen. Das bedeutet konkret, daß keine echte Suche stattfindet, sondern vielmehr, daß in der Datensatzgruppe alle Datensätze ausgeblendet werden, die nicht dem Suchkriterium entsprechen.

Übrigens kann die Suche auch Vergleiche einschließen, denn wenn z.B. in das Textfeld für das Suchkriterium »Geschwindigkeit« (bezogen auf das obige Beispiel) das Kriterium »> 200« eingegeben wird, wird dies bei der Suche automatisch berücksichtigt. Und so sieht der komplette Suchstring aus:

```
Dim sSuchkriterium As String
Dim AktuellePosition As Double
SuchkriteriumHinzufügen Suchstring:=sSuchkriterium, _
  Suchfeld:="ModellNr", Suchkriterium:=txtModellNr.Text
SuchkriteriumHinzufügen Suchstring:=sSuchkriterium, _
  Suchfeld:="ModellName", Suchkriterium:=txtModellname _
  .Text, LikeModus:=True
SuchkriteriumHinzufügen Suchstring:=sSuchkriterium, _
  Suchfeld:="HerkunftslandKürzel",
Suchkriterium:=txtLand.Text
```

```
SuchkriteriumHinzufügen Suchstring:=sSuchkriterium, _
  Suchfeld:="Leistung", Suchkriterium:=txtLeistung.Text
SuchkriteriumHinzufügen Suchstring:=sSuchkriterium, _
  Suchfeld:="Zylinder", Suchkriterium:=txtZylinder.Text
SuchkriteriumHinzufügen Suchstring:=sSuchkriterium, _
  Suchfeld:="Geschwindigkeit", _
  Suchkriterium:=txtGeschwindigkeit.Text
AktuellePosition = mRsModelldaten.Bookmark
mRsModelldaten.Filter = sSuchkriterium
```

Ein wenig umfangreich, doch hoffentlich leicht verständlich. Das Ergebnis ist ein Recordset *mRsModelldaten*, der nur noch jene Datensätze enthält, die dem Suchkriterium entsprechen.

Die Funktion *Suchkriterium Hinzufügen* macht nichts anderes, als die Zeichenkette des Suchkriteriums zusammenzubauen.

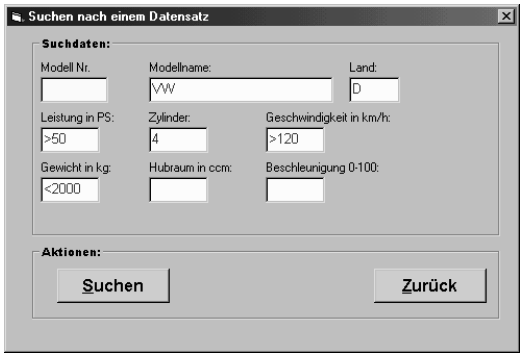


Bild 9.7: In diesem Formular findet die Suche nach einem Datensatz in der Tabelle Modelldaten *statt*

9.5.6 Den aktuellen Datensatz neu anzeigen

Haben Sie Änderungen in verschiedenen Eingabefeldern durchgeführt, möchten aber nun den Originalinhalt wiederherstellen, muß dieser von der Datenbank über die *Resync*-Methode abgerufen werden.

9.5.7 Eine Anmerkung zum Schluß

Was sich in diesem Abschnitt nach einer relativ problemlosen Umsetzung anhören mag, hat in der Praxis etliche Probleme aufgeworfen, die teilweise nicht ganz einfach zu lösen sind. Nicht, daß es sich um schwierige Bereiche der Datenbankprogrammierung handelt, es fehlt einfach an Dokumentation,

die das manchmal etwas »gewöhnungsbedürftige« Verhalten der ADO-Ereignisse (also allgemein der Bindung an Steuerelemente auf der Basis von OLE DB) erklärt. Auch der Datenumgebungs-Designer macht es einem in einigen Punkten nicht gerade einfach, indem man z.B. grundsätzlich (zumindest am Anfang) nachprüfen muß, welche Art von *Recordset*-Objekt (u.a. was die *Locking*- und *Cursortype*-Eigenschaften betrifft) nun erzeugt wurde. Geben Sie daher bitte nicht frustriert auf, wenn es nicht auf Anhieb funktioniert. Selbst den »Profis« gelingt so etwas nicht auf Anhieb[1]. Sollte es mit den gebundenen Steuerelementen überhaupt nicht klappen, gibt es immer noch die Möglichkeit, diese ungebunden zu betreiben und das Darstellen der Datenbankfelder sowie das Übertragen der geänderten Werte per Programmcode in das Recordset zu speichern. Das ist zwar mehr Programmieraufwand, bietet aber den Vorteil, daß Sie (und nicht der OLE DB-Provider) die Kontrolle über die ADO-Objekte besitzen und festlegen können, wann was in die Datenbank übertragen wird.

9.6 Das Jahr-2000-Problem

Das magische Datum 1.1.2000 wurde (zu unserer großen Erleichterung) offenbar ohne große Störfälle überschritten. Die Frage, wie man eine Datenbankanwendung »Jahr-2000-fähig« macht, bzw. sicherstellt, daß es zu keiner falschen Zuordnung des Jahrhunderts kommen kann, ist jedoch nach wie vor aktuell. Auch im Jahr 2000 und danach (streng genommen bis in alle Ewigkeit) kann es passieren, daß wenn bei einem Datum lediglich die letzten beiden Jahreszahlen gespeichert werden das Jahrhundert falsch zugeordnet wird. Allerdings dürfen die Benutzer davon ausgehen, daß zumindest die Standardsoftware und das Betriebssystem (Office 2000 und Windows 2000 sind in dieser Beziehung sicher) nichts durcheinander bringt.

Visual Basic 6.0 sowie Microsoft Access 97 sind relativ Jahr-2000-sicher. In beiden Fällen wird die Datumskonvertierung nicht von der Anwendung, sondern von der Systemdatei *Oleaut32.dll* vorgenommen (die Versionsnummer muß größer 2.2 sein – zu erkennen auf der Registerkarte *Version* der Datei). Beide Programme arbeiten mit einem »Zeitfenster«, das von der vernünftigen Annahme ausgeht, daß eine Jahreszahl zwischen 30 und 99 dem 20. Jahrhundert, eine Jahreszahl zwischen 0 und 29 dagegen dem 21. Jahrhundert zugeordnet werden kann. Starten Sie dazu einmal Visual Basic, und geben Sie die folgende kleine Schleife in das Direktfenster ein:

1 Den Quellcode für das in diesem Abschnitt beschriebene Formular finden Sie auf der Webseite.

```
For j = 28 to 31 : ?Year(DateSerial(j, 1, 1)) : Next
2028
2029
1930
1931
```

Bis zur 29 stellt Visual Basic »20«, danach »19« der Jahreszahl voraus (dieses Zeitfenster läßt sich bei Windows 98 in den Ländereinstellungen der Systemsteuerung verschieben[1]). Daran ist zunächst einmal nichts Verwerfliches. Man muß dieses Verhalten als Programmierer allerdings kennen und gleichzeitig wissen, daß es Situationen geben kann (ein Darlehen für den Kauf einer Wohnung oder eines Hauses, das vom Jahr 1999 an gerechnet etwa in 35 Jahren abläuft), in denen diese automatische Ergänzung nicht funktionieren wird. Das bedeutet konkret, daß, wenn in einem Eingabefeld die Jahreszahl 31 eingegeben und vom Programm nicht ergänzt wird, diese als 1931 in der Datenbank gespeichert wird.

Eine Rolle für die Datumsverarbeitung spielen daher die Datumsformate. Während es Microsoft Access 2.0, das die Datumskonvertierung noch selbst in die Hand nahm, dem Anwender praktisch unmöglich machte, eine vierstellige Jahreszahl des 21. Jahrhunderts einzugeben, sieht es mit den Nachfolgeversionen, die diese wichtige Angelegenheit an *Oleaut32.dll* übertragen, ein wenig besser aus. Doch auch Microsoft Access 97 bietet kein Standarddatumsformat, das völlig Jahr-2000-sicher ist. Beim kurzen und beim mittleren Datumsformat wird das Jahrhundert nur zweistellig gespeichert. Das lange Datumsformat kommt für viele Dialogfelder nicht in Frage, da es einfach zu lang ist. Doch auch hier gibt es keine Jahr-2000-Garantie, denn ein Anwender kann über die Systemsteuerung im Rahmen der Ländereinstellungen aus der vierstelligen eine zweistellige Jahreszahl machen (über eine API-Funktion ist es allerdings möglich, die Einstellungen der Systemsteuerung zu überschreiben).

Datumsformat	**Beispiel**	**Jahr-2000-sicher?**
Short Date	01.01.20	Nein
Medium Date	01. Jan. 20	Nein
Long Date	Mittwoch, 1. Januar 2020	Bedingt
dd.mm.yyyy benutzerdefiniertes Format)	1.1.2020	Ja

Tabelle 9.5: Die wichtigsten Datumsformate und ihre Jahr-2000-Sicherheit

1 Genauer gesagt, ab der Version 2.3.0 der Automationsbibliothek *Oleaut32.dll*.

Die Kurzübersicht in Tabelle 9.5 zeigt, daß keines der Standarddatumsformate Jahr-2000-sicher ist. Auch das lange Datumsformat nicht, da es über die Systemsteuerung manipulierbar ist. Probieren Sie es einmal aus, indem Sie das Datumsformat in den Ländereinstellungen auf zwei Jahresziffern kürzen und im Direktfenster von Microsoft Access 97 den Befehl

```
MsgBox Format("1-1-20", "Long Date")
```

ausführen. Das Jahrhundert wird in diesem Fall nicht mehr angezeigt.

Natürlich besteht ein wichtiger Unterschied zwischen der Anzeige eines Datums und der Speicherung des Datums in einer Datenbank. Hier erfolgt die Abspeicherung der Jahreszahl fast immer vierstellig, es stellt sich nur die Frage, ob das richtige Jahrhundert vorangestellt wird. Und genau an diesem Punkt kommt der Programmierer ins Spiel, der keinen Raum für Mehrdeutigkeiten lassen darf. Die Konsequenz aus diesen (am Anfang sicherlich etwas verwirrenden) Umständen lautet daher:

- ✘ Stellen Sie sicher, daß alle Datumseingaben ausnahmslos vierstellig erfolgen.
- ✘ Vermeiden Sie konsequent jegliche Situationen, in denen das Jahrhundert »interpretiert« werden muß.
- ✘ Speichern Sie Datumsangaben nicht in Stringvariablen, verwenden Sie konsequent den Datentyp *Date*, und führen Sie keine eigenen Datumsumrechnungen durch, sondern verwenden Sie die dafür vorgesehenen Funktionen (z.B. *DateDiff* und *DateAdd*).
- ✘ Führen Sie Datumsumrechnungen an einer zentralen Stelle des Programms durch (ideal dafür ist eine Klasse, da sich die Funktionalität besser kapseln läßt) und nicht an verschiedenen Stellen des Programms.

Bei neuen Projekten dürfte es relativ einfach sein, diese einfachen, aber enorm wichtigen Regeln einzuhalten. Das gilt auch für die erste Regel, da die meisten Anwender sogar Wert darauf legen dürften, in diesem und den kommenden Jahren das vollständige Datum zu sehen, auch wenn es zusätzliche Eingaben erforderlich macht. Bei bereits existierenden Projekten müssen alle Stellen überprüft werden, bei denen Datumsangaben im Spiel sind (das ist leicht gesagt, doch alles andere als trivial in der Umsetzung). Das betrifft nicht nur Datenbankfelder, sondern auch Validierungsregeln und Default-Werte, die für einzelne Felder vereinbart wurden (oft vergißt man, daß im Rohzustand der Datenbank zum Ausprobieren einzelne Default-Werte gesetzt wurden), Abfragen, *DataFormat*-Einstellungen und natürlich den VBA-Code. Es gibt aber auch viele subtile Stellen in einem Programm, wie z.B. beim Importieren von Textdaten und anderen Daten (etwa aus Word

oder einer anderen Anwendung per Automation) oder die Übernahme von Daten aus der Zwischenablage. Nirgends gibt es eine Garantie, daß die Jahresangaben wirklich vierstellig vorliegen. Die Folgen eines Datumswirrwarrs beschränken sich nicht auf irreführende Anzeigen oder falsche Rechenresultate, die sich durch Validierungsregeln wieder auffangen lassen und manchmal auch nur zur Belustigung beitragen, da die Resultate so falsch sind, daß der Fehler sofort offensichtlich ist. Werden Datumsfelder als Schlüssel oder Teilschlüssel einer Tabelle verwendet (etwa in Kombination mit einer Auftragsnummer), kann eine falsche Jahreszahl die Integrität der Datenbank gefährden, da Beziehungen und Abfragen auch einmal nicht mehr funktionieren oder gar fehlerhafte Daten in die Datenbank geschrieben werden. Wird ein solcher Fehler Wochen oder vielleicht sogar Monate später entdeckt, ist das Chaos perfekt. Es gibt also genügend Gründe, sich Gedanken zu machen.

Leider ist es aus Platzgründen nicht möglich, alle Aspekte dieses vielschichtigen Problems zu beleuchten, zumal es mit der reinen ADO-Programmierung nur indirekt etwas zu tun hat. Hier noch einmal die wichtigsten Regeln zusammengefaßt:

- ✘ Arbeiten Sie grundsätzlich mit vierstelligen Jahreszahlen.
- ✘ Zeigen Sie Jahreszahlen grundsätzlich vierstellig an.
- ✘ Testen Sie Ihr Programm ausführlich, und stellen Sie die interne Uhr des PC vor, um zu prüfen, wie es sich unter »realen« Bedingungen verhält (Jahr-2000-Unstimmigkeiten werden sich allerdings nicht sofort bemerkbar machen).
- ✘ Nicht Jahr-2000-sichere Programme können auch im Jahr 2000 und danach geschrieben werden, da es die Notwendigkeit, mit Jahresangaben aus dem 20. Jahrhundert arbeiten zu müssen, noch eine Weile geben wird (und nicht vergessen, daß das Jahr 2000 ein Schaltjahr ist, so daß falsch programmierte Schaltjahrroutinen auf einmal keinen Dienstag, den 29. Februar 2000 kennen).
- ✘ Nehmen Sie das Problem nicht auf die leichte Schulter. Es betrifft jeden Datenbankprogrammierer (die einen sicher mehr, die anderen weniger).
- ✘ Machen Sie sich mit der Jahr-2000-Problematik vertraut. Informationen zu diesem Thema gibt es inzwischen in ausreichender Menge.

Gerade der letzte Punkt ist von besonderer Bedeutung. Der angehende Datenbankprogrammierer sollte sich auf den vielen Webseiten eingehend mit dieser Problematik befassen und sich vor allem die Zeit dafür nehmen, auch wenn es für viele vielleicht kein sehr aufregendes Thema ist. Programmierer

besitzen die Verantwortung dafür, daß ihr Code nicht nur fehlerfrei, sondern auch in ein paar Jahren noch funktionstüchtig ist. Die Fehler der Vergangenheit sollten sich nicht wiederholen.

9.7 Zusammenfassung

Zur Datenbankprogrammierung mit Visual Basic gehört nicht nur der Umgang mit den ADO-Objekten. In einer Datenbankanwendung spielt (natürlich) auch die Benutzeroberfläche eine wichtige Rolle, in der Datensätze bearbeitet, hinzugefügt, gelöscht oder gesucht werden können. Visual Basic ist nicht Microsoft Access (das wurde bereits mehrfach erwähnt). Daher müssen diese Dinge im allgemeinen von Grund auf programmiert werden, was Vorteile (der Programmierer hat auf alles Einfluß) und Nachteile (der Programmierer hat sehr viel mehr Arbeit) hat. Die Oberfläche einer Datenbankabwendung ist kein Fremdkörper oder gar ein lästiger Zustand, sie ist ein integraler Bestandteil der Anwendung, die u.a. mit einer durchdachten Eingabevalidierung und einer »intelligenten« Benutzerführung maßgeblich zur Leistungsfähigkeit und Datensicherheit beiträgt.

9.8 Wie geht es weiter?

Langsam aber zielstrebig nähern wir uns dem Ende dieser Einführung. Die Themen werden daher etwas knapp, was ein durchaus positiver Aspekt ist. In Kapitel 10 geht es um die etwas fortgeschrittenere ADO-Programmierung. Hier lernen Sie Möglichkeiten der ADO-Objekte kennen, die bislang noch nicht vorgestellt wurden. Kapitel 11 zeigt, wie sich mit dem Datenreportgenerator schöne Datenbankreports erstellen lassen.

9.9 Fragen

Frage 1:

Welche Rolle spielt das *Validate*-Ereignis für die Eingabevalidierung?

Frage 2:

In einem Textfeld, das über seine Eigenschaften *DataSource* und *DataField* an ein Datenbankfeld gebunden ist, soll ein Währungsbetrag dargestellt werden. Wie läßt sich erreichen, daß der Inhalt automatisch in einem Währungsformat dargestellt wird, und welche Rolle spielt dabei die Systemsteuerung?

Frage 3:

Aus welchem Grund muß (bzw. kann) das *DataCombo*-Steuerelement mit zwei ADO-Datensteuerelementen verbunden werden? Was ist, wenn lediglich die Eigenschaften *RowSource* und *ListField* einen Wert erhalten?

Frage 4:

Welchen besonderen Vorteil bietet das *HFlexGrid*-Steuerelement gegenüber dem *DataGrid*-Steuerelement?

Frage 5:

Ein Programmierer möchte die Tabelle *Modelldaten* nach den Kriterien Modellname und Geschwindigkeit durchsuchen und versucht es mit der *Find*-Methode:

```
RsModellDaten.Find "ModellName = " & Chr(39) & _
txtModellname.text & Chr(39) & " AND Geschwindigkeit = " & _
txtGeschwindigkeit.text
```

Warum kann dieser Ansatz nicht funktionieren?

Frage 6:

Gibt es bei Visual Basic 6.0 bzw. beim Zugriff auf die Jet-Engine ein Jahr-2000-Problem?

Die Antworten zu den Fragen finden Sie in Anhang D.

Kapitel 10

ADO für (etwas) Fortgeschrittenere

In diesem Kapitel geht es um die etwas fortgeschritteneren Themen der Datenbankprogrammierung mit Visual Basic, die für die bisherigen Kapitel ein wenig zu speziell waren. Neue ADO-Objekte oder bislang nicht vorgestellte Datenzugriffswerkzeuge sind allerdings nicht dabei, denn die kennen Sie bereits aus den vorangegangenen Kapiteln. Vielmehr geht es um Themen, die bei der Umsetzung von Datenbankprojekten eine Rolle spielen können und deren Verständnis ein gewisses Grundlagenwissen über ADO-Objekte voraussetzt. Dieses Kapitel kann natürlich nur einen kleinen Ausschnitt der vielen Problemlösungen bieten, die im Programmieralltag benötigt werden. Wer auf der Suche nach einer Lösung ist, sollte in jedem Fall die MSDN-Hilfe und die einschlägigen Adressen im Internet (siehe Anhang E) berücksichtigen. Alleine aufgrund der rasanten Entwicklung bei den ADOs kann ein Handbuch nur einen Teil des erforderlichen Wissens vermitteln.

Sie lesen in diesem Kapitel etwas zu folgenden Themen:

- ✘ Zugriff auf Binärfelder
- ✘ Zugriff auf Bitmaps in der Jet-Engine
- ✘ *Recordset*-Objekte zur Laufzeit anlegen
- ✘ *Recordset*-Objekte in lokalen Dateien speichern
- ✘ Das ADO-Datensteuerelement zur Laufzeit initialisieren
- ✘ Zugriff auf Excel-Tabellen und Textdateien über ODBC

10.1 Zugriff auf Binärfelder

Ein Binärfeld ist ein Datenbankfeld, das beliebige binäre Daten enthält und damit keinen bestimmten Datentyp wie z.B. *Integer* oder *String* besitzt. In einem Binärfeld werden u.a. allgemeine Dateiinhalte, Bitmaps oder andere Multimediaelemente gespeichert. Da diese Datenelemente häufig sehr umfangreich sind, spricht man auch von *Binary Large Objects*, kurz BLOBs. Da weder Microsoft Access noch der Microsoft SQL-Server in der Lage ist, die verschiedenen BLOB-Typen direkt zu unterstützen, muß man sich mit einem allgemeinen Feldtyp begnügen. Enthält bei der Jet-Engine ein Feld den Typ *Binary*, kann sein Inhalt nicht direkt über die *Value*-Eigenschaft angesprochen werden. Statt dessen gibt es die Methoden *GetChunk* (Lesen) und *AppendChunk* (Schreiben) des *Field*-Objekts.


```
Variable = Field.GetChunk(Größe)
```

In Klammern wird die Anzahl der zu lesenden Bytes übergeben. Der Rückgabewert ist vom Typ *Variant* und kann z.B. einer Stringvariablen oder einer Feldvariable (die ebenfalls vom Typ *Variant* sein muß) zugewiesen werden. Ist das Feld kleiner als der beim Aufruf von *GetChunk* übergebene Wert, werden entsprechend weniger Bytes gelesen.

Die *GetChunk*-Methode kann auch für das Lesen von Memofeldern benutzt werden, wenngleich deren Wert auch über die *Value*-Eigenschaft zur Verfügung steht. Bei Verwendung der *GetChunk*-Methode muß beachtet werden, daß sich mit dem Lesen auch die interne Leseposition um die Anzahl der gelesenen Bytes verschiebt, so daß beim erneuten Aufruf der *GetChunk*-Methode ein *NULL*-Wert zurückgegeben wird, wenn keine zu lesenden Bytes mehr vorhanden sind. Erst ein Aufruf der *Requery*-Methode oder ein erneutes Ansteuern des Datensatzes setzt die Leseposition auf den Anfang zurück. Auch das Lesen eines anderen Feldes im gleichen Datensatz bewirkt, daß beim erneuten Lesen des vorherigen Feldes dieses wieder vom Anfang gelesen wird.

Bei sehr großen Feldern, etwa hochauflösenden Bitmap-Bildern, deren Größe mehrere Mbyte betragen kann, ist es sinnvoll, den Feldinhalt portionsweise zu lesen. Dabei wird in einer *Do*-Schleife der Feldinhalt mit der *GetChunk*-Methode blockweise gelesen, bis diese einen kleineren Wert als die Blockgröße zurückgibt.

Das Gegenstück zur *GetChunk*-Methode ist die *AppendChunk*-Methode, die einen beliebigen Datenblock in Gestalt einer *Variant*-Variable in ein *Field*- oder *Parameter*-Objekt überträgt.


```
Objekt.AppendChunk Daten
```

Auch bei der *AppendChunk*-Methode ist es möglich, die Daten portionsweise zu übertragen. Allerdings gibt es keine Möglichkeit, die Anzahl der Bytes festzulegen. Statt dessen setzt eine *AppendChunk*-Methode das Schreiben dort fort, wo eine vorherige *AppendChunk*-Methode aufgehört hat. Wie bei der *GetChunk*-Methode setzt das Schreiben eines anderen Feldes des gleichen Datensatzes den internen Positionszeiger für das vorherige Feld zurück. Bereits vorhandene Daten werden dadurch überschrieben.

Der Einsatz von *GetChunk/AppendChunk* kann auch beim Zugriff auf Memofelder einer Access-Datenbank sinnvoll sein.

Die Problematik, die beim Lesen größerer Felder auftreten kann, beschreibt der KnowledgeBase-Artikel: »Q194975 – HOWTO: Sample Functions Demonstrating GetChunk and AppendChunk«. Dieser Artikel steht im Internet unter der Adresse *http://support.microsoft.com/support/kb/articles/q194/9/75.asp* zur Verfügung.

Leider hat der Visual Data Manager gewisse Schwierigkeiten mit *Binary*-Feldern und zeigt beim Aufruf von Datensätzen mit *Binary*-Feldern Fehlermeldungen an, da der Feldinhalt nicht dargestellt werden kann. Die Inhalte der übrigen Felder werden jedoch angezeigt. Auch ist es möglich, Datensätze zu aktualisieren und zu löschen.

10.1.1 Die *ActualSize*-Eigenschaft

Im Zusammenhang mit den Methoden *GetChunk* und *AppendChunk* ist die *ActualSize*-Eigenschaft eines *Field*-Objekts von Bedeutung. Während die *DefinedSize*-Eigenschaft die Anzahl in Bytes in Abhängigkeit der Felddefinition zurückgibt, steht *ActualSize* für die tatsächliche Größe.

Wurde ein Feld vom Typ *Text* mit einer Länge von maximal 50 Zeichen definiert, gibt die *DefinedSize*-Eigenschaft diesen Wert zurück. Über *ActualSize* erhält man dagegen die Anzahl der Zeichen des aktuellen Wertes.

Kann ADO die tatsächliche Größe eines BLOB-Feldes nicht bestimmen, gibt *ActualSize* den Wert *adUnknown* zurück.

Das folgende Beispiel zeigt, wie ein Datenfeld mit einer Reihe von Zufallszahlen in einer Access-Datenbank gespeichert und anschließend wieder gelesen wird. Bei der Access-Datenbank handelt es sich um eine winzig kleine Testdatenbank, die lediglich aus einer Tabelle mit den beiden Feldern *BlobNr* und *Blob* besteht[1].

a) das Schreiben des Feldes *Blob* mit *AppendChunk*

Zuerst wird das zu schreibende Feld mit Zufallszahlen gefüllt und in einen String umgewandelt:

```
For n = 1 To AnzahlZahlen
        z = Int(Rnd * 255) + 1
        lstZahlen.AddItem Item:=z
        ZahlenString = ZahlenString & Format(CStr(z), "000")
Next
```

Dieser Abschnitt dient lediglich der Vorbereitung eines BLOB und hat noch nichts mit dem Datenzugriff zu tun. Anschließend wird die Tabelle *Blob* geöffnet und ein Zugriff auf die Felder *BlobNr* und *Blob* durchgeführt:

```
With Rs
    .AddNew
    .Fields("BlobNr").Value = BlobNr
    .Fields("Blob").AppendChunk ZahlenString
    .Update
End With
```

1 Es gibt natürlich keinen zwingenden Grund, die Bezeichnung »Blob« in diesem Zusammenhang zu verwenden. Vielleicht ist dieser Begriff auch nur eine ironische Anspielung auf den B-Movie-Klassiker »The Blob« aus dem Jahre 1958 (dt. Titel: »Das Ding, das aus der Tiefe kam«).

b) das Lesen des Feldes *Blob* mit *GetChunk*

Beim Lesen eines Feldinhalts soll die Nummer des Datensatzes zuvor über eine *InputBox*-Methode abgefragt werden, wobei dem Benutzer die maximal mögliche Nummer angezeigt wird:

```
With Rs
     .ActiveConnection = Cn
     .CursorType = adOpenStatic
     .LockType = adLockReadOnly
     .Source = "Blobs"
     .Open
     BlobNr = InputBox("Blob Nr. (max. " & Rs.RecordCount & ")")
     .Move NumRecords:=BlobNr - 1, Start:=1
End With
```

Anschließend tritt die *GetChunk*-Methode in Aktion, die den kompletten Inhalt des Feldes in die Variable *ZahlenString* überträgt:

```
AnzahlByte = Rs.Fields("Blob").ActualSize
ZahlenString = Rs.Fields("Blob").GetChunk(AnzahlByte)
For n = 0 To AnzahlZahlen - 1
    lstZahlenNachher.AddItem _
    Item:=Mid(ZahlenString, n * 3 + 1, 3)
Next
```

Da durch *GetChunk* die zuvor gespeicherten Zahlen zeichenweise gelesen werden und davon ausgegangen wird, daß jede Zahl drei Zeichen umfaßt (dafür war beim Schreiben die *Format*-Methode zuständig), sorgt die *Mid*-Methode dafür, daß jeweils drei Zeichen in das Listenfeld *lstZahlenNachher* übertragen werden.

10.2 Zugriff auf Bitmaps in der Jet-Engine

Beim Zugriff auf in einer Access-Datenbank gespeicherte Bitmap-Bilder von einem Visual-Basic-Programm aus gilt es, ein kleines Problem zu lösen, da es zwei verschiedene Feldformate gibt. Welches Format für die Speicherung einer Bitmap verwendet wird, hängt davon ab, ob die Bitmap vom Visual-Basic-Programm oder innerhalb von Microsoft Access eingefügt wird. Im letzteren Fall wird sie in Gestalt eines OLE-Objekts in ein Feld vom Typ *Binary* eingetragen. Dieses OLE-Objekt, bei dem der Bitmap ein sogenannter *OLE-Header* (der Informationen über das Objekt enthält) vorausgeht, kann aber in einem Visual-Basic-Programm weder von der *Picture*-Eigenschaft

eines Bildfeldes noch einer Anzeige gelesen werden, da beide Eigenschaften eine Bitmap in einem der unterstützten (Binär-)Formate (etwa BMP oder GIF) erwarten. Keine Probleme beim Auslesen gibt es dagegen, wenn die Bitmap von einem gebundenen Bildfeld oder einer Anzeige in einem Visual-Basic-Programm aus in die Access-Datenbank übertragen wird, da die Steuerelemente über ihre *Picture*-Eigenschaft das Binärformat verwenden. Um eine Bitmap aus einer Access-Datenbank in einem Visual-Basic-Programm darstellen zu können, gibt es zwei Möglichkeiten:

- ✘ Die Verwendung des OLE-Steuerelements anstelle eines Bildfeldes oder einer Anzeige.
- ✘ Das Einlesen des OLE-Objekts über die *GetChunk*-Methode und das »Heraustrennen« der Bitmap aus dem OLE-Rahmen.

Alternative 1 ist zwar einfach zu lösen, da auch das OLE-Steuerelement gebunden ist. Doch leider kann es nicht an eine OLE DB-Datenquelle, sondern nur an eine DAO-Datenquelle gebunden werden (dazu muß das »normale« Datensteuerelement auf einem Formular angeordnet werden), was es als universelle Lösung ausscheiden läßt. Außerdem ist ein OLE-Steuerelement aus verschiedenen Gründen kein vollwertiger Ersatz für ein Bildfeld, so daß diese Alternative nur in Ausnahmefällen in Frage kommen dürfte. Alternative 2 funktioniert auch mit ADO, setzt aber ein wenig Programmierung voraus. Als erstes wird das Feld über die *GetChunk*-Methode (je nach Größe) entweder »portionsweise« oder in einem Rutsch in eine *Variant*-Variable gelesen. Da der OLE-Header beim Einlesen ausgelassen wird, enthält die Variable lediglich die Bitmap. Diese Variable wird in einer temporären Datei gespeichert, damit ihr Inhalt im nächsten Schritt über die *LoadPicture*-Methode der *Picture*-Eigenschaft eines Bildfeldes oder einer Anzeige zugewiesen werden kann.

Das folgende Beispiel ist leider etwas umfangreicher. Es besteht aus einem Formular, auf dem eine Anzeige (*imgBild*) und zwei Schaltflächen (*cmdMovePrevious* und *cmdMoveNext*) angeordnet sind. Auch wenn es im Rahmen des Buches angebrachter wäre, die Fuhrpark-Datenbank zu verwenden, greift es auf die beliebte Nwind-Datenbank zu (passen Sie bitte den Verzeichnispfad an), damit dieses Beispiel auch jene Leser nachvollziehen können, die mit der Umsetzung der Fuhrpark-Datenbank noch nicht so weit vorangekommen sind.

```
Option Explicit
Const OBJECT_SIGNATURE = &H1C15
Const CHECKSUM_SIGNATURE = &HFE05AD00
Const OBJECT_HEADER_SIZE = 20 ' Größe der OBJECTHEADER-Struktur
```

```
Private Cn As ADODB.Connection
Private Rs As ADODB.Recordset

Private Type PT
    Width As Integer
    Height As Integer
End Type

' Diese Struktur enthält den OLE-Vorspann
Private Type OBJECTHEADER
    Signature As Integer
    HeaderSize As Integer
    ObjectType As Long
    NameLen As Integer
    ClassLen As Integer
    NameOffset As Integer
    ClassOffset As Integer
    ObjectSize As PT
    OleInfo As String * 256
End Type

Private Declare Function GetTempFileName Lib "Kernel32" _
  Alias "GetTempFileNameA" (ByVal lpszPath As String, _
  ByVal lpPrefixString As String, ByVal wUnique As Long, _
  ByVal lpTempFileName As String) As Long

Private Declare Sub CopyMemory Lib "Kernel32" _
  Alias "RtlMoveMemory" (lpvDest As Any, lpvSource As Any, _
  ByVal cbCopy As Long)

Private Sub cmdMoveNext_Click()
    Rs.MoveNext
    BildFeldLesen
End Sub

Private Sub cmdMovePrevious_Click()
    Rs.MovePrevious
    BildFeldLesen
End Sub
```

```
Private Sub Form_Load()
    Set Cn = New ADODB.Connection
    With Cn
     .Provider = "Microsoft.Jet.OLEDB.3.51"
     .ConnectionString = _
     "Data Source=C:\Eigene Dateien\Nwind.mdb"
     .Open
    End With
    Set Rs = New ADODB.Recordset
    With Rs
     .ActiveConnection = Cn
     .CursorType = adOpenStatic
     .Source = "Personal"
     .Open
    End With
' Diese Form der Bindung ist mit OLE-Felder leider nicht
möglich
'    With imgBild
'     Set .DataSource = Rs
'     .DataField = "Foto"
'    End With
End Sub

Public Function CopyOLEBitmapToFile(OleField As ADODB.Field) _
  As String
    Dim tmpFileName As String
    Dim DateiNr As Integer
    Dim Buffer As String
    Dim ObjHeader As OBJECTHEADER
    Dim ObjectOffset As Long
    Dim BitmapOffset As Long
    Dim BitmapHeaderOffset As Long
    Dim bFeld() As Byte
    Dim bFeldBmp() As Byte
    Dim i As Long
    Dim RetVal As Long
' Erst einmal den OLE-Vorpann des OLE-Feldes einlesen
    bFeld() = OleField.GetChunk(OleField.ActualSize)
' Aus bFeld in die OLE_HEADER-Struktur kopieren
    CopyMemory ObjHeader, bFeld(0), OBJECT_HEADER_SIZE
' Offset der Objektinformation berechnen
    ObjectOffset = ObjHeader.HeaderSize + 1
```

```
' Alle Objektinformationen in Stringvariablen übertragen
    For i = ObjectOffset To ObjectOffset + 512
        Buffer = Buffer & Chr(bFeld(i))
    Next i
' Lautet der Klassenname "PBrush"?
    If Mid(Buffer, 12, 6) = "PBrush" Then
' Handelt es sich auch um eine Bitmap?
    BitmapHeaderOffset = InStr(Buffer, "BM")
     If BitmapHeaderOffset > 0 Then
' Wenn ja, dann den Beginn der Bitmap berechnen
       BitmapOffset = ObjectOffset + BitmapHeaderOffset - 1
' Das Feld bFeldBmp nimmt die Bitmap auf
       ReDim bFeldBmp(0 To UBound(bFeld) - BitmapOffset)
' Bitmap in das Feld kopieren
' CopyMemory Ziel, Quelle, Anzahl
       CopyMemory lpvDest:=bFeldBmp(0), _
        lpvSource:=bFeld(BitmapOffset), _
        cbCopy:=UBound(bFeld) - BitmapOffset + 1
' Temporäre Datei anlegen
       tmpFileName = Space(256)
       RetVal = GetTempFileName(0, "", -1, tmpFileName)
       DateiNr = FreeFile
' Inhalt des Feldes bFeldBmp in Datei speichern
       Open tmpFileName For Binary As DateiNr
        Put #DateiNr, , bFeldBmp
       Close
     End If
    End If
' Dateiname als Funktionswert übergeben
    CopyOLEBitmapToFile = Trim(tmpFileName)
End Function

Sub DisplayOleBitmap(ctlPict As Control, _
  OleField As ADODB.Field)
 Const DT_LONGBINARY = 11
 Dim RetVal As Long
 Dim DateiNr As Integer
 Dim OleFileName As String
 If OleField.Type = adLongVarBinary Then
    OleFileName = CopyOLEBitmapToFile(OleField)
    If OleFileName <> "" Then
       ctlPict.Picture = LoadPicture(OleFileName)
```

```
      Kill OleFileName
    End If
 End If
End Sub

Private Sub BildFeldLesen()
  Screen.MousePointer = 11
  DisplayOleBitmap imgBild, Rs.Fields("Foto")
  Screen.MousePointer = vbDefault
End Sub
```

Wenn Sie das Beispiel für kompliziert halten, haben Sie recht (wenngleich es aufgrund des Umstandes, daß der Feldinhalt in einem Stück gelesen wird, noch relativ überschaubar ist). Mehrere Dutzend Programmzeilen und sogar zwei API-Funktionen sind eindeutig zuviel für einen doch recht simplen Vorgang. Eine Lösung wäre es sicherlich, ein ActiveX-Steuerelement zu programmieren, das eine Anzeige enthält und eine Bindung zu einem OLE-Feld so durchführt, daß sie für die Anwender völlig transparent ist.

10.3 *Recordset*-Objekte zur Laufzeit anlegen

Ein *Recordset*-Objekt kann auch unabhängig von einer Datenquelle und damit von einem *Connection*-Objekt angelegt und mit Inhalten gefüllt werden. Damit läßt sich eine komplette auf einem *Recordset*-Objekt basierende Datensatzgruppe aus Programmdaten erstellen, ohne daß dafür eine Datenbank existieren muß.

Das folgende Beispiel legt ein *Recordset*-Objekt an, fügt eine Reihe von Datensätzen hinzu und gibt den Inhalt eines Feldes aus.

```
Option Explicit
Private Rs As ADODB.Recordset

Private Sub Form_Load()
    Set Rs = New ADODB.Recordset
    With Rs
     .CursorLocation = adUseClient
     .LockType = adLockBatchOptimistic
     .Fields.Append "Name", adVarChar, 30
     .Fields.Append "Telefon", adVarChar, 12
     .Open
```

```
    .AddNew
    .Fields("Name") = "Captain Kirk"
    .Fields("Telefon") = "0211-466713"
    .AddNew
    .Fields("Name") = "Spok"
    .Fields("Telefon") = "0228-112233"
    .UpdateBatch
    .MoveFirst
  End With

  MsgBox Prompt:="Der Gewinner heißt: " & _
    Rs.Fields("Name").Value
  Rs.Close
End Sub
```

Der Sperrtyp *adLockBatchOptimistic* ist in diesem Beispiel sinnvoll, aber nicht zwingend notwendig.

10.4 *Recordset*-Objekte können sich selbst speichern

Der Inhalt eines *Recordset*-Objekts muß nicht unbedingt in einer Datenbank landen. Über die *Save*-Methode ist es kein Problem, den Inhalt eines *Recordset*-Objekts in einer gewöhnlichen Datei zu speichern. Dafür dürfte es im (harten) Datenbankalltag nur wenige Anlässe geben. Gedacht ist diese Variante eher für den Fall, daß ein *Recordset*-Objekt auf einer HTML-Seite zum Einsatz kommt. Hier ist es sehr praktisch, wenn eine Datensatzgruppe lokal gespeichert wird, da diese beim nächsten Aufruf der Seite nicht mehr vom Server abgerufen werden muß.

Der folgende Befehl schließt an das Beispiel aus dem letzten Absatz an, indem er das angelegte *Recordset*-Objekt in einer Datei mit dem Namen *Kirk.dat* speichert:

```
Rs.Save FileName:="Kirk.dat", PersistFormat:=adPersistXML
```

Als Format stehen das XML- und das ADTG-Format (*Advanced Data Tablegram*) zur Auswahl[1]. Eingelesen wird der Inhalt der Datei ganz einfach dadurch, daß diese beim Öffnen eines *Recordset*-Objekts über den *Source*-Parameter angegeben wird:

1 ADTG ist ein Binärformat, das für die Übertragung von Recordset-Objekten über ein Netzwerk im Rahmen der Remote Data Service (RDS) verwendet wird.

```
Private Sub cmdLoadRs_Click()
    Set Rs = New ADODB.Recordset
    Rs.Open Source:="Kirk.dat"
    MsgBox Prompt:="Der Captain heißt: " & _
      Rs.Fields("Name").Value
End Sub
```

10.5 Zugriff auf ISAM-Datenbanken

In die Kategorie ISAM-Datenbanken fallen im Zusammenhang mit der Jet-Engine alle Datenbanken, die lediglich aus einer Aneinanderreihung von Datensätzen bestehen und daher auch nur datensatzweise angesprochen werden können[1]. In die Kategorie ISAM-Datenbanken, die allerdings nicht »hochoffiziell« ist, fallen z.B. xBase-Datenbanken, Paradox-Datenbanken, Excel-Tabellen und reine Textdateien, die einen tabellarischen Aufbau besitzen. Aber auch die MAPI-Ablage, die z.B. Microsoft Outlook für die Postablagen benutzt, wird durch die Jet-Engine wie eine ISAM-Datenbank angesprochen. Der Zugriff auf diese ISAM-Datenbanken kann auf zwei verschiedene Weisen erfolgen:

- ✘ Über ODBC-Treiber.
- ✘ Über die ISAM-Treiber (mit OLE DB ab Jet Engine 4.0).

Für welche der beiden Möglichkeiten Sie sich entscheiden, hängt eher von Nebensächlichkeiten ab. Allerdings kann sich der Zugriff auf ein und dieselbe Datenbank bei Verwendung eines ISAM-Treibers von dem Zugriff per ODBC-Treiber in Kleinigkeiten, wie z.B. in der Frage, ob Groß-/Kleinschreibung eine Rolle spielt oder wie Trennzeichen behandelt werden, unterscheiden. Das Beste ist es daher vermutlich, beide Zugriffsmethoden an einem konkreten Beispiel zu vergleichen.

10.5.1 Der Zugriff auf eine Excel-Tabelle per ODBC

Eine Excel-Tabelle besitzt von Anfang an den Aufbau einer Datenbanktabelle. Es liegt daher nahe, sie auch als solche zu behandeln. Beim Zugriff auf eine Excel-Arbeitsmappe (also eine XLS-Datei) spielt jedes Tabellenblatt die Rolle einer Tabelle. Als kleine Besonderheit gilt es zu beachten, daß an den Tabellennamen ein $-Zeichen gehängt werden muß. Heißt ein Tabel-

1 Daher vermutlich auch die Bezeichnung »Indexed Sequential Access Method«.

lenblatt z.B. Tabelle1, muß der *Source*-Eigenschaft des *Recordset*-Objekts der Wert »Tabelle1$« zugewiesen werden. Da der Zugriff via ODBC erfolgt, müssen die typischen ODBC-Informationen zur Verfügung gestellt werden. Dazu gehören mindestens:

- ✘ Der Name des ODBC-Treibers
- ✘ Der Name bzw. vollständige Pfad der Excel-Datei.

Optional, aber üblich benötigt man:

- ✘ Den Namen, über den die ODBC-Verbindung im Programm angesprochen werden soll.

Diese Angaben können beim Öffnen des *Connection*-Objekts entweder direkt übergeben oder der *ConnectionString*-Eigenschaft zugewiesen werden. Am einfachsten ist es in der Regel, über das Unterprogramm *ODBC (32 Bit)* der Systemsteuerung alle diese Angaben unter einem Namen abzuspeichern. Das Ergebnis ist ein *Data Source Name* (DSN), der für alle getroffenen Einstellungen steht, und der wahlweise in der Registry als System- oder Benutzer-DSN oder als eigenständige Datei vorliegt. Damit muß der *ConnectionString*-Eigenschaft lediglich der DSN-Name angegeben werden.

Das folgende Beispiel führt einen Zugriff auf die Excel-Arbeitsmappe *Fussball.xls*, die ein Tabellenblatt mit dem Namen *Spieler* enthält, per ODBC durch.

Schritt 1:

Öffnen Sie in der Systemsteuerung das Unterprogramm *OBDC (32bit)*.

Schritt 2:

Aktivieren Sie die Registerkarte *System-DSN*. Alle hier angelegten DSNs stehen systemweit zur Verfügung.

Schritt 3:

Klicken Sie auf *Hinzufügen*, wählen Sie den Eintrag »Microsoft Excel-Treiber (*.xls)« aus, und klicken Sie auf *Fertig stellen*.

Schritt 4:

Geben Sie bei Datenquellennamen den Namen des DSN an, z.B. »Fußballtabelle«. Klicken Sie auf *Arbeitsmappe auswählen*, und wählen Sie die XLS-Datei aus. Klicken Sie auf *OK*, um den DSN zu erstellen, der daraufhin in der Liste der Datenquellennamen geführt wird. Klicken Sie auf *OK*, um das Unterprogramm zu beenden.

Schritt 5:

Legen Sie ein neues Visual-Basic-Projekt an, fügen Sie eine Referenz auf die Microsoft ActiveX Data Objects ein, und geben Sie in *Form_Load* folgende Befehle ein:

```
Set Cn = New ADODB.Connection
With Cn
  .Provider = "MSDASQL"
  .ConnectionString = "Data Source=Fußballtabelle"
  .Open
End With
Set Rs = New ADODB.Recordset
  With Rs
  .ActiveConnection = Cn
  .CursorType = adOpenStatic
  .Source = "[Spieler$]"
  .Open
End With
```

Sie verfügen nun über ein *Recordset*-Objekt, über das Sie auf die in der Tabelle enthaltenen Datensätze (Zeilen) und deren Felder, z.B. per SQL, zugreifen können. Eine passende Übung finden Sie bei den Fragen zu diesem Kapitel.

10.5.2 Der Zugriff auf eine Textdatei per ODBC

Auch eine (normale) Textdatei kann die Rolle einer »Datenbank« spielen. Voraussetzung ist lediglich, daß sie eine bestimmte Aufteilung besitzt. In diesem Fall entspricht jede Textdatei genau einer Tabelle. Am einfachsten ist es, wenn die erste Zeile die Namen der Felder und die folgenden Zeilen für jedes Feld ein Textelement und damit die Datensätze enthalten. Eine Zeile ergibt sich durch Zeichen bis zum nächsten Zeilenumbruch (d.h. der Kombination aus *Chr(13)* und *Chr(10)*). Alle Felder in einer Zeile werden durch ein Trennzeichen, z.B. ein Komma, getrennt. Sollte die Textdatei einen anderen Aufbau besitzen oder spezielle Trennzeichen verwenden, muß dies in einer Steuerungsdatei (meist besitzt sie den Namen *Schema.ini*) festgelegt werden.

Im folgenden Beispiel wird per ODBC auf eine Textdatei mit dem Namen *Fussball.txt* zugegriffen, die folgenden Aufbau besitzt:

Spieler, Alter, Tore
Franz Beckenbauer, 52, 73
Jürgen Grabowski, 48, 52
Paul Breitner, 46, 28
Gerd Müller, 55, 92
usw.

Damit das folgende Beispiel funktionieren kann, muß (wie im letzten Abschnitt beschrieben) zuvor in der Systemsteuerung ein DSN mit dem Namen »Fussball« angelegt werden, in dem der ODBC-Treiber »Microsoft Text-Treiber (*.txt;*.csv)« ausgewählt und der Pfadname der Datei eingestellt wird. Beachten Sie dabei, daß Sie (anders als beim Excel-Zugriff) lediglich das Verzeichnis auswählen müssen (deaktivieren Sie gegebenenfalls die Option *Aktuelles Verzeichnis verwenden*). Alle in diesem Verzeichnis befindlichen Textdateien werden dadurch zu Tabellen, die später über die Verbindung zur Verfügung stehen.

```
Set Cn = New ADODB.Connection
With Cn
.ConnectionString = "DSN=Fussball"
.Open
End With
Set Rs = New ADODB.Recordset
With Rs
    .ActiveConnection = Cn
    .Source = "Fussball.txt"
    .CursorType = adOpenStatic
    .Open
    End With
    Rs.MoveLast
MsgBox Prompt:=Rs.RecordCount & " Datensaetze"
Cn.Close
```

Das Beispiel macht deutlich, daß sich der Zugriff auf eine Textdatei grundsätzlich nicht von dem Zugriff auf eine SQL-Serverdatenbank unterscheidet. Lediglich die Verbindungszeichenfolge und die *Source*-Eigenschaft müssen die richtigen Werte erhalten.

10.5.3 Der Zugriff auf eine Textdatei per ISAM-Treiber

Die Jet-Engine kann über ihre ISAM-Treiber direkt auf ISAM-Datenbanken, wie xBase-Datenbanken oder Textdateien, zugreifen. Voraussetzung ist, daß der dafür benötigte ISAM-Treiber nicht nur vorhanden ist, sondern auch lokalisiert werden kann (sowohl bei Visual Basic als auch bei Microsoft Access 97 kann im Setup ausgewählt werden, ob die ISAM-Treiber installiert werden sollen). Fehlt der ISAM-Treiber oder kann dieser aus irgendwelchen Gründen nicht gefunden werden, erscheint beim Zugriff auf die Datenbank die Fehlermeldung »Installierbares ISAM nicht gefunden«.

Die Konfigurationsdaten der ISAM-Treiber der Jet-Engine werden in der Registry im Zweig *\HKEY_LOCAL_MACHINE\Software\Microsoft\Jet \3.5* (bzw. 4.0) abgelegt. Die genaue Schreibweise, die bei der Auswahl eines ISAM-Treibers angegeben werden muß, ist im Schlüssel *ISAM Formats* zu finden.

Das folgende Beispiel zeigt, wie der Zugriff auf die Textdatei *Lieblingsgetränke.txt* im Verzeichnis *C:\Eigene Dateien* über den ISAM-Treiber der Jet-Engine 4.0 vonstatten geht. Das Beispiel funktioniert (offenbar) nur mit ADO 2.1 und der Jet-Engine 4.0, die beide nicht nur durch Office 2000, sondern auch durch MDAC 2.1 installiert werden. Bei der Jet-Engine 3.51 ist stets die lapidare Fehlermeldung »Installierbares ISAM nicht gefunden« die Folge.

```
Option Explicit

Private Cn As ADODB.Connection
Private Rs As ADODB.Recordset

Const DBPfad = "C:\Eigene Dateien\"

Private Sub cmdVor_Click()
    Rs.MoveNext
    If Rs.EOF = True Then
       Rs.MoveLast
    End If
End Sub
```

```
Private Sub cmdZurück_Click()
    Rs.MovePrevious
    If Rs.BOF = True Then
       Rs.MoveFirst
    End If
End Sub

Private Sub Form_Load()
    Set Cn = New ADODB.Connection
    With Cn
     .Provider = "Microsoft.Jet.OLEDB.4.0"
     .ConnectionString = "Data Source=" & DBPfad & ";Extended
Properties=Text;"
     .Open
    End With
    Set Rs = New ADODB.Recordset
    With Rs
     .ActiveConnection = Cn
     .CursorType = adOpenKeyset
'     .CursorType = adOpenForwardOnly
'     .LockType = adLockOptimistic
     .LockType = adLockReadOnly
     .Source = "Lieblingsgetränke.txt"
     .Open
    End With

    With txtAlter
     Set .DataSource = Rs
     .DataField = "Geburtstag"
    End With
    With txtGetränk
     Set .DataSource = Rs
     .DataField = "Lieblingsgetränk"
    End With

    With txtName
     Set .DataSource = Rs
     .DataField = "Name"
    End With
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    Cn.Close
    Set Cn = Nothing
End Sub
```

Änderungen an der Textdatei, etwa über DELETE oder ADDNEW, und an einzelnen Felder sind beim ISAM-Treiber nicht möglich. Dafür lassen sich einfache SQL-Abfragen durchführen:

```
With Rs
    .Source = "SELECT MAX(Geburtstag) FROM
[Lieblingsgetränke.txt]"
    .Open ActiveConnection:=Cn
End With
```

Sollte der ISAM-Zugriff bei Ihnen aus irgendeinem Grund nicht funktionieren, so ist dies nicht weiter tragisch. Zum einen ist es möglich, diesen per ODBC durchzuführen. Zum anderen hält Sie niemand davon ab, dafür auf die »guten alten« DAOs zurückzugreifen (mehr dazu in Anhang C).

10.6 Weitere Tips zu ADO

In diesem Abschnitt werden einige Teilbereiche beschrieben, bei denen es in der Programmierpraxis zu »Problemen« kommen kann[1].

10.6.1 ADO-Datensteuerelement zur Laufzeit initialisieren

Um ein Programm flexibel zu gestalten, kann es erforderlich sein, das ADO-Datensteuerelement erst in *Form_Load* mit einer Datenquelle zu verbinden.

Das folgende Beispiel zeigt, wie das ADO-Datensteuerelement vollständig zur Laufzeit mit der Tabelle *Modelldaten* in der Datenbank *Fuhrpark.mdb* verbunden wird.

```
Private Sub Form_Load()
    With adoData
     .ConnectionString = _
       "Provider=Microsoft.Jet.OLEDB.3.51;" & _
       "Data Source=C:\Eigene Dateien\Fuhrpark.mdb"
```

1 Meist handelt es sich, das weiß ich aus eigener Erfahrung, um Verständnisschwierigkeiten, die sich bei längerem Nachdenken und häufigerem Ausprobieren lösen. Meist heißt allerdings nicht immer.

```
     .CommandType = adCmdTable
     .RecordSource = "Modelldaten"
    End With
    With txtModellname
    Set .DataSource = adoData
     .DataField = "Modellname"
    End With
End Sub
```

Achten Sie auf den *Set*-Befehl, der für die *DataSource*-Eigenschaft erforderlich ist. Befindet sich auf dem Formular z.B. ein *DataGrid*, kann es ebenfalls zur Laufzeit gebunden werden:

```
With adoData
     Set .DataSource = adoData
End With
```

Sollte es mit dieser »frühen Bindung« Probleme geben, ist es einen Versuch wert, das ADO-Datensteuerelement zunächst über seine *Enabled*-Eigenschaft zu deaktivieren, es zu einem späteren Zeitpunkt zu reaktivieren, um dann die Bindung an eine Datenquelle durchzuführen.

10.6.2 Update des aktuellen Datensatzes erzwingen

Beim Arbeiten mit gebundenen Steuerelementen kann es erforderlich sein, den Inhalt der Steuerelemente mit dem aktuellen Inhalt der Datenbank zu aktualisieren. Dies läßt sich mit dem Befehl

```
Rs.Move 0
```

am schnellsten erreichen, wobei *Rs* für das *Recordset*-Objekt steht, über das die Verbindung hergestellt wird. Der Vorteil gegenüber *MoveFirst* ist, daß die aktuelle Position des Datensatzzeigers erhalten bleibt. Unter Umständen kann es erforderlich sein, vor der Ausführung der *Move*-Methode den Zustand der *EditMode*-Eigenschaft abzufragen:

```
If Rs.EditMode = adEditNone Then
    Rs.Move 0
End If
```

Dieser »Trick« kann auch dann hilfreich sein, wenn trotz Änderungen in einem gebundenen Steuerelement dessen Inhalt beim Aufruf der *Update*-Methode nicht in die Datenbank übernommen wird.

10.6.3 *Recordset*-Objekte auf *Nothing* setzen

Wird ein *Recordset*-Objekt in einem laufenden Programm nicht mehr benötigt, sollte es mit dem Befehl

```
Set Rs = Nothing
```

wobei *Rs* für das *Recordset*-Objekt steht, »vernichtet« werden, da dadurch die Datenbankverbindung freigegeben wird. Dies ist gerade bei großen Datenbankanwendungen, die auf eine SQL-Server-Datenbank zugreifen, ein entscheidendes Kriterium. So wird z.B. bei jedem Aufruf der *Execute*-Methode eines *Connection*-Objekts eine neue Verbindung angelegt.

10.6.4 Einzelne Spalten im *DataGrid* sperren

Möchte man nicht alle Spalten in einem *DataGrid* sehen (z.B. soll häufig der Primärschlüssel nicht angezeigt werden), kann dies über die *Visible*-Eigenschaft der Spalte erreicht werden.

Das folgende Beispiel macht die erste Spalte unsichtbar und sperrt die zweite Spalte gegen Eingaben:

```
With adoData
     Set .DataSource = adoData
     .Columns(0).Visible = False
     .Columns(1).Locked = True
End With
```

Soll eine Spalte angezeigt, aber nicht bearbeitet werden, muß die *Locked*-Eigenschaft auf *False* gesetzt werden. Dies kann häufiger notwendig sein, als Sie es vielleicht vermuten würden, denn wenn es nicht verhindert wird, wird jede unbedachte Änderung im *DataGrid* in die Datenbank übernommen (das nicht OLE DB-fähige *FlexGrid*-Steuererlement ist nur Read-Only, was sich in diesem Zusammenhang als Vorteil auswirken kann).

10.6.5 Ausliefern einer ADO-Anwendung

Damit ein Visual-Basic-Programm, das mit den ADO-Objekten arbeitet, auch auf einem Windows-95-PC lauffähig ist, müssen eine Reihe von Systemdateien vorhanden sein. Außerdem muß auf dem PC DCOM (Distributed COM) installiert sein, was bei Windows-95-PCs nicht selbstverständlich ist. Am einfachsten ist es daher, die Datei *Mdac_typ.exe* (d.h. die Microsoft Data Access Components) mit auszuliefern, was der Verpackungs- und Weitergabeassistent erledigen kann. Einziger Nachteil: Die Datei ist mit

ca. 6 bis 7 Mbyte (je nach aktueller Version) nicht gerade klein. Es ist wichtig, zu beachten, daß vor der Installation von *Mdac_typ.exe* unbedingt sichergestellt werden muß, daß die Systemerweiterung *DCOM* (in Gestalt von *Dcom95.exe* für Windows 95 bzw. *Dcom98.exe* für Windows 98 – letztere befindet sich u.a. auf der Visual-Basic-6-CD-ROM) bereits installiert wurde. Muß dies im Rahmen der Installation der Anwendung geschehen, ist ein Neustart des Systems erforderlich[1].

Arbeiten Sie stets mit der aktuellsten ADO-Version (dies wäre die Version 2.1 – Stand: 4/99), da dies jene Version ist, die (vermutlich) am stabilsten läuft[2].

Auf der Microsoft-Webseite *http://support.microsoft.com/support/kb/articles/q217/7/54.asp* ist kurz beschrieben, wie im Zusammenspiel mit dem Verpackungs- und Weitergabeassistenten sichergestellt werden kann, daß dieser MDAC 2.1 und nicht MDAC 2.0 installiert.

10.7 Zusammenfassung

Die Programmierung mit ADO besitzt viele Facetten. Ist das Prinzip der ADO-Objekte und der Umgang mit *Connection*-, *Recordset*- und *Command*-Objekten erst einmal klar, wird schnell deutlich, daß es in der Praxis viele kleine Probleme und Problemchen zu lösen gilt. Das notwendige Wissen befindet sich nur zum Teil in der MSDN-Hilfe. Zu einem erfahrenen Datenbankprogrammierer wird man, indem man die Hilfe aufmerksam gelesen hat, über Praxiserfahrung verfügt und weiß, daß es wichtig ist, am Informationsaustausch in den verschiedenen Newsgroups im Internet (siehe Anhang E) teilzunehmen.

1 Der KnowlegeBase-Artikel Q191094 geht kurz auf diesen Aspekt ein – er kann im Internet unter *http://support.microsoft.com/support/kb/articles/q191/0/94.asp* abgerufen werden.

2 Wie ein Blick in die verschiedenen Newsgroups zeigt, lassen sich gewisse »Inkompatibilitäten« dabei nicht vermeiden.

10.8 Wie geht es weiter?

Das »Boot-Camp« (also die »harte« Grundausbildung) haben Sie absolviert. Mit Ihrem jetzigen Wissen sollten Sie gerüstet sein, um kleinere Datenbankprojekte beginnen zu können. Zum Ausklang erwartet Sie in Kapitel 11 ein angenehmes Thema, bei dem die Programmierung nur eine Nebenrolle spielt. Es geht um das Anfertigen von Datenreports.

10.9 Fragen

Frage 1:

Welche Möglichkeiten bietet die Jet-Engine, um beliebige binäre Daten zu speichern, und wie greift man auf diese Daten zu?

Frage 2:

Auf welche Weise lassen sich MP3-Dateien in einer Access-Datenbank speichern?

Frage 3:

Ein Anwender hat eine Datenbank mit Microsoft Access erstellt und in eine Tabelle mehrere Gif-Dateien eingefügt. Was muß beachtet werden, damit diese Bilder in einem Visual-Basic-Programm angezeigt werden können?

Frage 4:

Woran kann die Ausführung eines Visual-Basic-Programms, das per ADO auf Datenbanken zugreift, unter Windows 95 scheitern?

Frage 5:

Eine Programmiererin möchte das Ergebnis einer Datenbankabfrage auf einer Diskette speichern und später von einem Visual-Basic-Programm weiterverarbeiten. Wie ist das am einfachsten mit ADO möglich?

Frage 6:

Eine Excel-Tabelle mit dem Namen *Torschützen.xls* enthält eine Tabelle mit dem Namen *Spieler* mit einer Statistik der Torschützenkönige der Fußballbundesliga, wobei die erste Spalte *Name* den Namen des Spielers, die zweite Spalte *Alter* sein Alter und die dritte Spalte *Tore* die Anzahl der Tore in einer Saison enthält. Wie läßt sich am einfachsten feststellen, welcher Spieler die meisten Tore geschossen hat?

Die Antworten zu den Fragen finden Sie in Anhang D.

Kapitel 11

Datenreports

Erst durch einen Datenreport wird eine Datenbankanwendung komplett. Ein Datenreport ist eine Abfrage in »Schönschrift«, die nicht nur in einem Fenster auf dem Bildschirm erscheint, sondern optisch (nahezu) beliebig gestaltet und vor allem auch ausgedruckt werden kann. Damit ein Datenreport nicht umständlich programmiert werden muß (was gerade in Visual Basic aufgrund der nur rudimentären Druckerausgabeunterstützung kein allzu großes Vergnügen sein dürfte), gibt es komfortable Programme, die dem Datenbankprogrammierer genau diese Arbeit abnehmen: die Datenreportgeneratoren. Bereits Visual Basic 3.0 wurde mit dem *Crystal Report Writer*, einem sehr komfortablen Reportgenerator, ausgestattet, wobei die erstellten Reports mittels eines Zusatzsteuerelements über ein Visual-Basic-Programm aufgerufen und ausgedruckt werden konnten. Seit der Version 6.0 gibt es mit dem Datenreport-Designer einen eigenen Reportgenerator als Alternative, der in diesem Kapitel vorgestellt wird. Dieser bietet zwar nicht ganz den Komfort eines »richtigen« Reportgenerators, ist dafür aber als Designer (relativ) nahtlos in Visual Basic integriert.

Sie lesen in diesem Kapitel etwas zu folgenden Themen:

- ✘ Übersicht über den Datenreport-Designer
- ✘ Ein Datenreport Schritt für Schritt erstellen

11.1 Übersicht über den Datenreport-Designer

Der Datenreport-Designer ist ein Designer, der unter Umständen erst zur Entwicklungsumgebung hinzugefügt werden muß:

- ✘ Rufen Sie im PROJEKT-Menü den Eintrag KOMPONENTEN auf.
- ✘ Öffnen Sie die Registerkarte *Designer*, und wählen Sie den Datenreport-Generator (*Data Report*) aus.

Anschließend kann der Datenreport-Designer über das PROJEKT-Menü aufgerufen werden (Befehl DATA REPORT HINZUFÜGEN). Wie bei allen Designer ist der Report Teil des Projekts und muß in einer separaten DSR-Datei gespeichert werden.

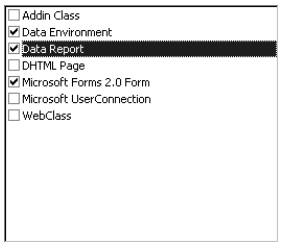


Bild 11.8: Der Datenreport-Designer muß gegebenenfalls erst einmal hinzugefügt werden

11.1.1 Allgemeines zur Bedienung

Der Datenreport-Designer lehnt sich bezüglich seiner Bedienung an den integrierten Berichtgenerator von Microsoft Access an, wenngleich dieser mehr Steuerelemente und aufgrund des Umstandes, daß auch ActiveX-Steuerelemente eingebunden werden können, einen deutlichen Vorteil besitzt (beim Datenreport-Designer ist man auf lediglich fünf Steuerelemente beschränkt). Wie nicht anders zu erwarten ist, können Datenbankfelder aus einer Datenumgebung auf den Detailbereich eines Datenreports gezogen werden. Voraussetzung ist allerdings, daß die Eigenschaften *DataSource* und *DataMember* gesetzt wurden. Angezeigt wird ein Datenreport entweder programmgesteuert, über seine *Show*-Methode oder indem der Datenreport in den Projekteigenschaften als Startobjekt eingestellt wird.

Der Datenreport-Designer ist in vielen Aspekten mit einem herkömmlichen Formular vergleichbar und besitzt u.a. auch die Ereignisse *Initialize*, *Resize* und *Terminate*.

11.1.2 Hinzufügen eines Datenreports

Ein Datenreport wird über den Eintrag DATA REPORT HINZUFÜGEN im PROJEKT-Menü zu einem Projekt hinzugefügt. Sie erhalten ein leeres Designfenster, das am Anfang aus fünf Teilbereichen besteht:

- ✘ Berichtskopf (Bereich4)
- ✘ Seitenkopf (Bereich2)
- ✘ Detail (Bereich1)
- ✘ Seitenfuß (Bereich3)
- ✘ Berichtsfuß (Bereich5)

Außerdem kann ein Bericht über das Kontextmenü um einen Gruppenfuß (Bereich6) ergänzt werden. Diese Aufteilung mag zunächst etwas verwirrend erscheinen, ist aber notwendig, denn schließlich möchte man erreichen, daß bestimmte Angaben (wie z.B. die Seitenzahl oder das Datum) am Ende jeder Seite (Seitenfuß), andere Angaben nur zu Beginn des Reports (Berichtskopf) oder am Ende des Reports (Berichtsfuß) erscheinen. Datenbankfelder werden nur im Detailbereich angeordnet. Dies ist jener Bereich, der auf jeder Seite des Berichts gleich dargestellt wird.

Damit Sie zu Beginn eines Projekts nicht Datenreport und Datenumgebung hinzufügen müssen, bietet Visual Basic den Projekttyp »Datenprojekt« an.

Jeder Bereich im Datenreportfenster stellt ein eigenes Objekt dar, dessen Eigenschaften im Eigenschaftsfenster eingestellt werden.

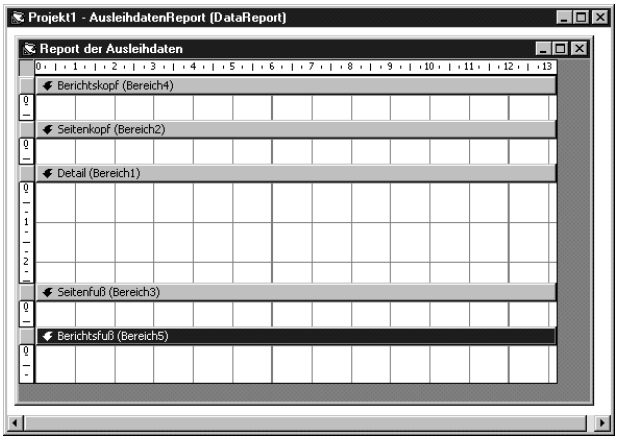


Bild 11.9: Nach dem Hinzufügen eines Datenreports erscheint ein leerer Designer

Anders als in Microsoft Access enthalten die Menüs keine Formatierungsbefehle für den Report. Spezielle Kommandos stehen lediglich im Kontextmenü zur Verfügung.

11.1.3 Das Hinzufügen von Datenbankfeldern

Voraussetzung dafür, daß im Datenreport Datenbankfelder angeordnet werden können, ist, daß die Eigenschaften *DataSource* und *DataMember* entsprechend belegt wurden. Auf diese Weise wird festgelegt, welche Datensatzgruppe in dem Datenreport dargestellt werden soll. Sollen im Datenreport Felder mehrerer Tabellen oder Abfragen erscheinen, muß in der Datenumgebung ein entsprechendes *Command*-Objekt angelegt worden sein.

Um in einem Datenreport ein Datenbankfeld anzuzeigen, gibt es zwei Möglichkeiten:

- ✘ Anordnen eines (zunächst ungebundenen Textfeldes) und Setzen der Eigenschaften *DataMember* und *DataField*.
- ✘ Ziehen eines Feldes aus dem Datenumgebungs-Designer in den Detailbereich.

Am einfachsten ist es, ein komplettes *Command*-Objekt aus dem Datenumgebungs-Designer in den Detailbereich zu ziehen und anschließend die einzelnen Felder zu justieren. Das Justieren der Felder bedarf ein wenig Übung, wobei die Rasterlinien am Anfang eine große Hilfe sind (anders als bei Microsoft Access werden Namens- und Inhaltsfeld nicht automatisch zusammen verschoben).

11.1.4 Anordnen von Steuerelementen

Um auch in den Kopf- und Fußbereichen etwas darstellen zu können, etwa eine Überschrift oder die Anzahl der Seiten, müssen die entsprechenden Steuerelemente aus der Werkzeugsammlung des Reports in den gewünschten Bereich gezogen werden. Auch dieser jedem Visual-Basic-Programmierer bestens vertraute Vorgang erfordert beim Datenreport-Designer eine gewisse Übung (am einfachsten ist es daher, ein Steuerelement über das Kontextmenü des Bereichs einzufügen). Besonders nachteilig wirkt sich der Umstand aus, daß sich, anders als bei Microsoft Access, ein versehentliches Verschieben nicht über den RÜCKGÄNGIG-Befehl zurücknehmen läßt. Alle verfügbaren Steuerelemente werden in der Werkzeugsammlung angeboten.

Zusätzliche Steuerelemente gibt es leider nicht[1]. Die wichtigsten Eigenschaften dieser *Rpt*-Steuerelemente sind *DataMember*, *DataField* und *DataFormat*.

Bild 11.10: Die Werkzeugsammlung enthält alle Steuerelemente, die auf einem Report angeordnet werden können

Über die *CanGrow*-Eigenschaft wird eingestellt, ob sich die Größe des Steuerelements automatisch an seinen Inhalt anpassen soll.

11.1.5 Einfügen von Seitenzahlen und Datum

Am Ende einer Seite (Seitenfuß-Bereich) sollen in der Regel Seitenzahlen und andere Angaben, wie z.B. das Datum, erscheinen. Diese »Angaben« (es handelt sich um Bezeichnungsfelder mit einem besonderen Wert der *Caption*-Eigenschaft) werden durch Anklicken des Bereichs mit der rechten Maustaste und Auswahl eines Eintrags im Untermenü STEUERELEMENT EINFÜGEN aufgenommen. Klicken Sie das eingefügte Elemente erneut mit der rechten Maustaste an, erscheint ein Kontextmenü mit einer Reihe von Ausrichtungsmöglichkeiten.

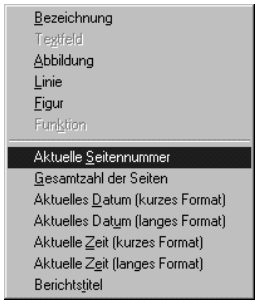


Bild 11.11: Die einzufügenden Elemente werden aus dem Kontextmenü eines Bereichs ausgewählt

1 Ein wenig mehr Mühe hätte man sich bei Microsoft in diesem Punkt schon geben können.

Soll am Seitenende die Angabe »n von m Seiten« erscheinen, kann dies am einfachsten durch Einfügen eines Bezeichnungsfeldes, dessen *Caption*-Eigenschaft den Wert » %p von %P« erhält, erreicht werden.

11.1.6 Allgemeine Tips für das Erstellen eines Reports

- ✘ Alle Felder werden durch Anklicken des Berichts mit der rechten Maustaste und Auswahl des Eintrags ALLES AUSWÄHLEN selektiert.
- ✘ Um mehrere Felder zu selektieren, kann ein Rahmen um diese Felder gezogen werden.
- ✘ Der Rand des Reports wird über die Eigenschaften *LeftMargin*, *RightMargin*, *BottomMargin* und *TopMargin* eingestellt.
- ✘ Um ein oder mehrere Felder zu löschen, müssen diese zunächst selektiert und über den Befehl LÖSCHEN im Kontextmenü entfernt werden.
- ✘ Soll der Vorspann des Reports (Seitenkopf) als separate Seite erscheinen, muß die *ForcePageBreak*-Eigenschaft des Objekts den Wert *rptPageBreakAfter* erhalten.
- ✘ Um eine feinere Justierung zu ermöglichen, muß die Einstellung AM RASTER AUSRICHTEN im Kontextmenü deaktiviert werden.
- ✘ Ein Datenreport kann über seine *MDIChild*-Eigenschaft MDI-Kindfenster eines MDI-Hauptfensters sein. In diesem Fall wird der Report im Rahmen des Hauptfensters dargestellt.

11.1.7 Eigenschaften und Methoden des Datenreport-Objekts

Das *Datenreport*-Objekt verfügt über eine Reihe von Eigenschaften, Methoden und Ereignissen, von denen die wichtigsten in Tabelle 11.6 zusammengefaßt sind. Um die Ereignisse von einem anderen Modul auswerten zu können, muß eine Variable mit *WithEvents* deklariert werden:

```
Private WithEvents Rp As DataReport
```

Mitglied	Bedeutung
Error-Ereignis	Wird beim Auftreten eines Fehlers aufgerufen. Der Parameter *JobType* gibt die Operation an, die den Fehler ausgelöst hat.
AsyncProcess-Ereignis	Wird während einer asynchronen Operation, wie dem Ausdrucken eines Reports, aufgerufen.
ProcessingTimeOut-Ereignis	Tritt während einer asynchronen Operation in regelmäßigen Abständen auf und gibt dem Programm so Gelegenheit, den Vorgang abzubrechen.
AsyncCount-Eigenschaft	Gibt an, ob eine asynchrone Operation, wie das Exportieren eines Datenreports, noch läuft.
DataMember-Eigenschaft	Legt eine Datensatzgruppe in der über die *DataSource*-Eigenschaft ausgewählten Datenquelle fest oder gibt diese zurück.
DataSource-Eigenschaft	Legt die Datenquelle fest, mit der ein Report verbunden ist, oder gibt diese zurück.
ExportFormats-Eigenschaft	Gibt eine Referenz auf die *ExportFormats*-Auflistung zurück, die für jedes unterstützte Format ein *ExportFormat*-Objekt enthält.
ExportReport-Methode	Exportiert einen Report in das Text- oder HTML-Format.
PrintReport-Methode	Gibt einen Report auf dem Drucker aus.
Refresh-Methode	Bewirkt, daß der Report neu dargestellt wird.
Sections-Eigenschaft	Ermöglicht einen Zugriff auf die einzelnen *Section*-Objekte und damit auch auf den Inhalt einzelner Steuerelemente, die in diesen Bereichen angeordnet sind.
ReportTitle-Eigenschaft	Gibt den Titel des Reports an oder legt ihn fest.

Tabelle 11.6: Die wichtigsten Mitglieder des DataReport*-Objekts*

11.2 Ein Datenreport – Schritt für Schritt erstellt

In diesem Abschnitt wird für die beliebte Fuhrpark-Datenbank ein kleiner Report erstellt. Stellen Sie sich dazu vor, es ist Jahresabschlußzeit und der Leiter des Fuhrparks möchte sehen, welche Fahrzeuge wann von welchen Mitarbeitern ausgeliehen wurden. Da diese Informationen nicht in einer einzigen Tabelle enthalten sind, muß zunächst ein passendes *Command*-Objekt angelegt werden, das später die Grundlage für den Report bilden soll (das *DataReport*-Objekt unterstützt auch hierarchische *Command*-Objekte

– allerdings nicht mit mehr als einem Unter-*Command*-Objekt pro *Command*-Objekt).

Schritt 1:

Legen Sie ein neues Standard-Exe-Projekt an, und fügen Sie die Datenumgebung *Fuhrpark.dsr* hinzu. Falls Sie diese Datenumgebung noch nicht angelegt haben, sollten Sie dies wie in Kapitel 6 beschrieben nachholen. Außerdem wird vorausgesetzt, daß die Fuhrpark-Datenbank bereits über die Tabellen *Mitarbeiterdaten* und *Ausleihdaten* (siehe Kapitel 9) und einige Datensätze verfügt.

Schritt 2:

Legen Sie in der Datenumgebung ein neues *Command*-Objekt mit dem Namen *cmdAusleihstatistik* an. Weisen Sie dem Objekt das folgende SQL-Kommando zu, das sich am einfachsten mit dem SQL-Generator erstellen läßt:

```
SELECT Ausleihdaten.*, Mitarbeiterdaten.Mitarbeitername AS
Name FROM Mitarbeiterdaten, Ausleihdaten WHERE
Mitarbeiterdaten.MitarbeiterNr = Ausleihdaten.MitarbeiterNr
```

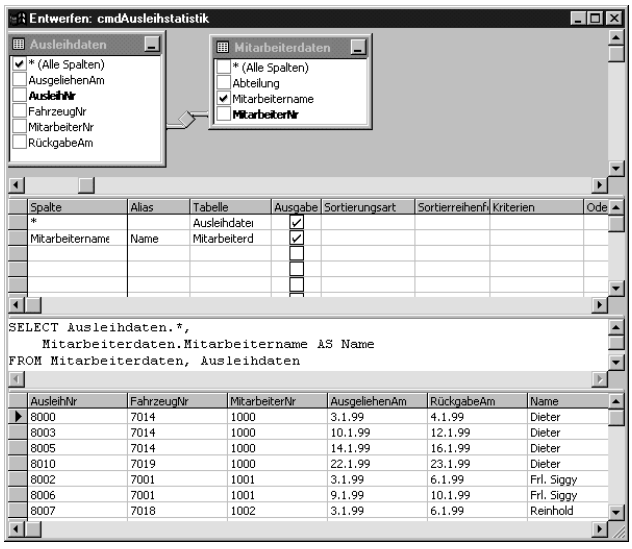


Bild 11.12: Dem Datenreport liegt ein SQL-Kommando zugrunde, das mit Hilfe des SQL-Generators erstellt wurde

Schritt 3:

Fügen Sie über das PROJEKT-Menü einen Datenreport zum Projekt hinzu.

Schritt 4:

Ziehen Sie das *Command*-Objekt *cmdAusleihstatistik* mit der Maus in den Detail-Bereich des Datenreports.

Schritt 5:

Ordnen Sie die Felder nach Ihren Vorstellungen an, und »verschönern« Sie den Report beispielsweise durch Hinzufügen einer Überschrift, Einfügen von Seitenzahlen, Datum, Linien und Ändern der Schriftgrößen.

Schritt 6:

Stellen Sie in den Projekteigenschaften den Reportdesigner als Startobjekt ein, und starten Sie das Programm.

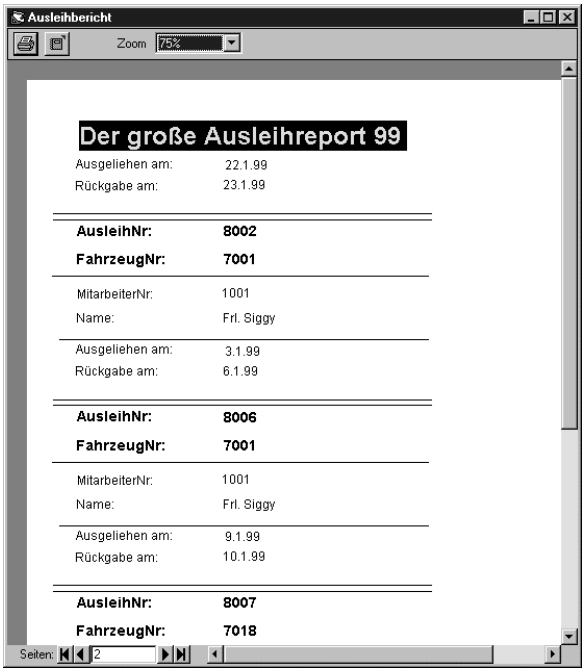


Bild 11.13: Das erste Ergebnis des Datenreports kann sich durchaus sehen lassen, wenngleich es noch ausbaufähig ist

11.2.1 Das programmgesteuerte Erstellen von Datenreports

Alle Dinge, die im Rahmen des Datenreport-Designers durchgeführt werden, lassen sich auch programmgesteuert erledigen.

Binden eines Datenreports

Wie bereits erwähnt, wird ein Datenreport über seine Eigenschaften *DataSource* und *DataField* an eine Datenquelle gebunden.

Das Anzeigen eines Datenreports

Für das Anzeigen eines Datenreports ist die *Show*-Methode zuständig:

```
Ausleihreport.Show
```

Das Ausdrucken eines Datenreports

Diesen Job übernimmt die *PrintReport*-Methode:

```
Ausleihreport.PrintReport
Do While Ausleihreport.AsyncCount > 0
    DoEvents
    Debug.Print "Läuft noch, sorry!"
Loop
Unload Ausleihreport
```

In diesem Beispiel wird über die *AsyncCount*-Eigenschaft geprüft, ob der Ausdruck fertig ist.

Das Exportieren eines Datenreports

Das Exportieren eines Datenreports bedeutet, diesen in einer Text- oder HTML-Datei zu speichern. Diese Aufgabe übernimmt die *ExportReport*-Methode, wobei in den Parametern das Format oder der Umstand, daß ein Auswahldialogfeld angezeigt wird, festgelegt wird:

```
Ausleihreport.ExportReport
```

Zugriff auf einzelne Steuerelemente

Der Zugriff auf einzelne Report-Steuerelemente ist über die *Sections*-Eigenschaft möglich:

```
?Ausleihreport.Sections(2).Controls.(1).Name
```

oder

```
?Ausleihreport.Sections(2).Controls.(2).Caption
```

Dabei lassen sich nur jene Eigenschaften ansprechen, die im Eigenschaftenfenster angezeigt werden (es ist daher auch nicht möglich, auf den Inhalt eines Textfeldes zuzugreifen).

11.2.2 Ausblick auf etwas fortgeschrittenere Themen

Das *DataReport*-Objekt kann noch ein wenig mehr. Dazu gehört z.B. die Darstellung hierarchischer Command-Objekte, wobei in diesem Zusammenhang auch Gruppenbereiche ins Spiel kommen. Wie sich Gruppierungen, hierarchische Datensatzgruppen oder auch per SQL-Kommando berechnete Felder darstellen lassen, wird in der MSDN-Hilfe beschrieben.

11.3 Zusammenfassung

Mit dem Datenreport-Designer steht Visual-Basic-Datenbankprogrammierern ein schlichter, dafür aber leicht zu bedienender und vor allem gut integrierter Reportgenerator zur Verfügung. Er ist sicherlich noch keine echte Alternative zum Crystal Report Generator (von Seagate Software), doch da die Visual Basic 6.0 beiliegende alte Variante weder ADO/OLE DB noch ODBC unterstützt (dieses Feature gibt es nur in der separat zu erwerbenden Professional Edition), dürfte der Datenreport-Designer für angehende Datenbankprogrammierer zunächst die erste Wahl sein.

11.4 Ausblick

Sie sind damit am Ende des Buches und damit am Ende der Einführung in die Datenbankprogrammierung mit Visual Basic angelangt. Beginnend bei den elementaren Grundlagen (was ist eine Datenbank – die Frage mit der in Kapitel 1 alles begann) und den wichtigsten Merkmalen relationaler Datenbanken (erinnern Sie sich noch an die Normalisierungsregeln aus Kapitel 2?) haben Sie die Active Data Objects (die Abkürzung ADO sollte jetzt einen sehr vertrauten Klang besitzen) und die verschiedenen Werkzeuge (Datensteuerelement, Datenumgebungs-Designer und Datenreport-Designer) kennengelernt, die Visual Basic zur Datenbankprogrammierung zur Verfügung stellt.

Die Entwicklung bleibt bekanntlich nicht (nie) stehen. Seit Juni 99 steht für Datenbankprogrammierer mit Microsoft Access 2000 im Rahmen von Office 2000 eine neue Version zur Verfügung, deren herausragendes Merkmal

die *Microsoft Desktop Engine* (MSDE) als Alternative zur Jet-Engine (4.0) ist. Hinter dem unscheinbaren Kürzel verbirgt sich die Desktop-Version des Microsoft SQL-Servers 7.0, die auch unter Windows 95/98 läuft, und damit ein überaus leistungsfähiges DBMS. Programmiert wird die MSDE (selbstverständlich) über die ADOs (2.1), so daß Sie in dieser Beziehung nichts Neues lernen müssen. Neu dürften zur Zeit noch verschiedene Merkmale der MSDE wie Stored Procedures und Triggers sein, die aber schon in zwei bis drei Jahren zum täglichen Brot des Visual-Basic-Programmierers gehören dürften. Der Enterprise Edition von Visual Basic 6.0 ist zwar »nur« der Microsoft SQL Server 6.5 beigelegt, doch es liegt nahe, daß die MSDE ein fester Bestandteil eines »Visual Basic 7.0« sein wird. Haben Sie dieses Buch gründlich durchgearbeitet, sind Sie für die kommende Entwicklung gut vorbereitet.