AUSGABE 04/2025

VISUAL BASIC ENTWICKLER

MAGAZIN FÜR DIE PROGRAMMIERUNG VON MICROSOFT OFFICE UND ANDEREN ANWENDUNGEN MIT VB.NET, VBA UND TWINBASIC

IN DIESEM HEFT:

André Minhorst Verlag

Airtable: Datenbank erstellen

Immer wieder fragen uns Leser, wie sie ihre Access-Datenbank ins Internet stellen können. Die Antwort lautet standardmäßig: Gar nicht. Denn die Access-Datenbanken können erstens nicht einfach über das Internet veröffentlicht werden, und ein Nachbau beispielsweise als Webanwendung übersteigt den Aufwand der Programmierung einer Access-Datenbank um ein Vielfaches. Aber in manchen Fällen geht es gar nicht darum, die vollständige Datenbank über das Web zugänglich zu machen. Oft reicht es aus, wenn Daten über das Internet gesammelt werden oder dort verfügbar gemacht werden können. In diesem Fall kann man beispielsweise Airtable nutzen. Airtable liegt in der Cloud und kann einfach per Browser erstellt und verwaltet werden. Wie der Einstieg gelingt, zeigen wir in diesem Artikel.

Was ist Airtable?

Airtable ist eine cloudbasierte Datenbankplattform, die die einfache Bedienbarkeit von Tabellenkalkulationen mit der Flexibilität relationaler Datenbanken kombiniert.

Anders als klassische Desktoplösungen wie Microsoft Access läuft Airtable vollständig im Browser und erlaubt es, Daten über sogenannte Bases in Tabellen zu verwalten, zu verknüpfen und mit anderen zu teilen.

Durch integrierte Automationen, Ansichten (zum Beispiel Kalender, Kanban, Galerie) und Schnittstellen zu Tools wie Make oder Zapier lässt sich Airtable hervorragend in moderne Cloud-Workflows einbinden. Für Access-Entwickler ist Airtable besonders interessant als leichtgewichtige Online-Datenquelle oder als Synchronisationspunkt zwischen lokalen Access-Datenbanken und webbasierten Anwendungen.

Genau diesen Anwendungszweck wollen wir uns in diesem und anderen Artikeln einmal ansehen. Dazu erstellen wir erst einmal ein Airtable-Benutzerkonto und legen dann eine erste Datenbank mit einer Beispieltabelle an.

Der Anwendungszweck soll sein, über ein Bestellformular Bestellungen für ein bestimmtes Produkt entgegenzunehmen und diese in der Airtable-Datenbank zu speichern.

In weiteren Artikeln schauen wir uns dann an, wie wir diese Daten von Access aus in die lokale Datenbank synchronisieren können.

Anlegen eines Airtable-Benutzerkontos

Airtable finden wir ganz einfach unter dem folgenden Link:

```
https://www.airtable.com
```

Dort sehen wir auf der Startseite direkt einen Button zum Registrieren, der beispielsweise **Sign up for free** heißt.

Airtable funktioniert kostenlos

Nicht nur das Erstellen eines Benutzerkontos ist kostenlos – wir können einfache Lösungen auch mit diesem kostenlosen Benutzerkonto betreiben. Für unser Beispiel reicht dies völlig aus: Wir wollen erst einmal nur eine einfache Tabelle verwalten.

Nach dem Registrierungsvorgang landen wir auf der Startseite von Airtable (siehe Bild 1). Hier können wir mit einer Datenbankvorlage starten, aber wir wählen direkt den Button **Selbst eine App bauen**.

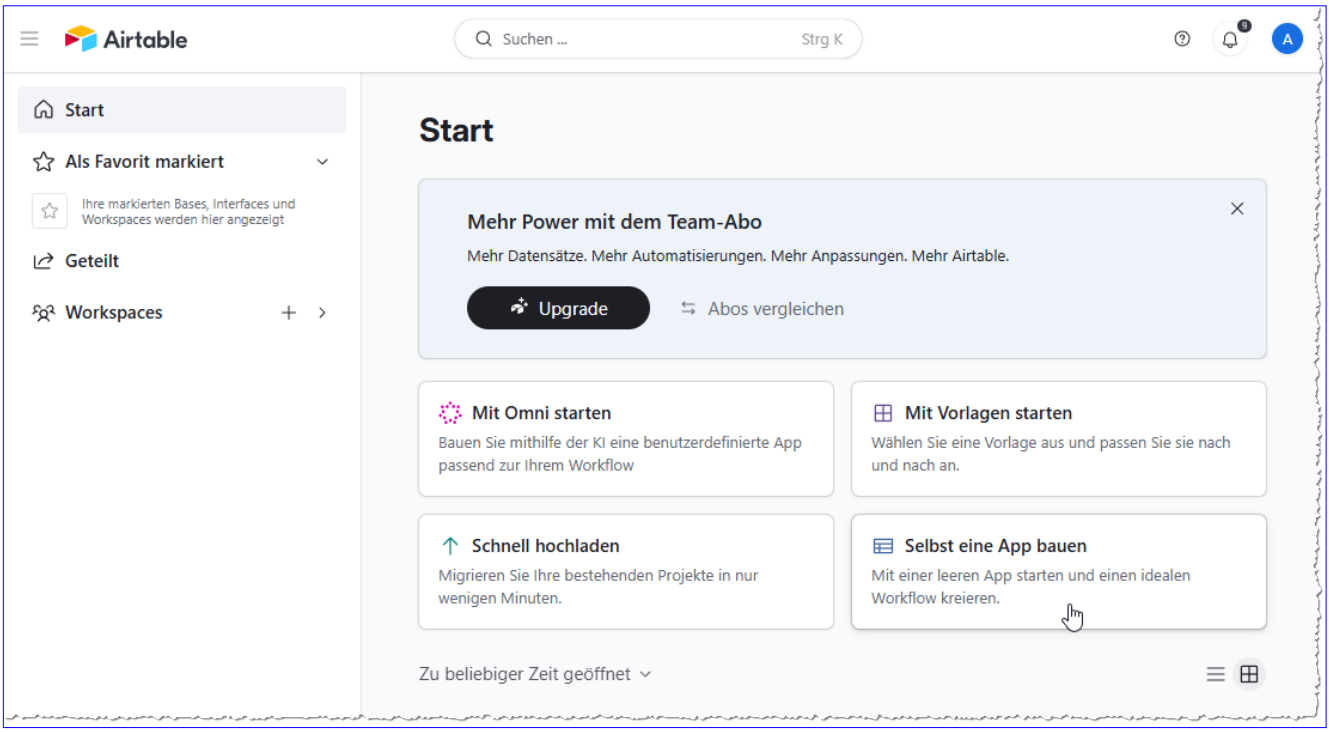

Bild 1: Der Startbildschirm eines jungfräulichen Airtable-Benutzerkontos

Nun erscheint ein Dialog, in dem wir den Namen für unsere Datenbank (in Airtable **Base** genannt) eingeben und eine Themenfarbe auswählen können (siehe Bild 2). Wir legen den Namen **Bestelldatenbank** fest und klicken auf **Weiter**.

Im nächsten Schritt können wir verschiedene Elemente hinzufügen. Damit können wir die genannten Tabellen wie zum Beispiel **Projekte**, **Teams** oder **Aufgaben** hinzufügen.

Wenn wir **Eigene hinzufügen** auswählen, können wir direkt den Namen der ersten Tabelle und einige Felder angeben. Wir wollen die Tabellen jedoch manuell hinzufügen und klicken unten auf **Einrichtung überspringen**.

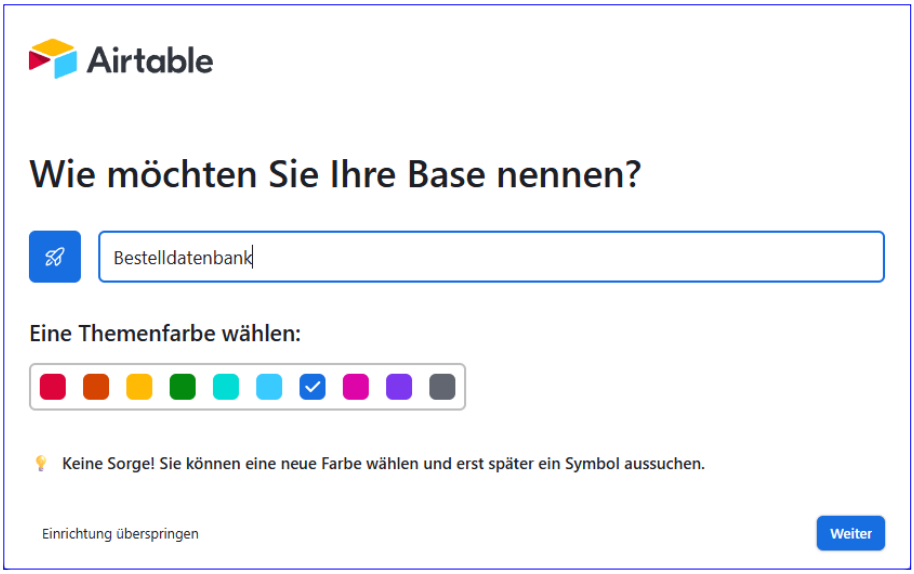

Bild 2: Anpassen von Datenbankname und Themenfarbe

Daten importieren oder Tabelle erstellen

Nachdem wir dies bestätigt haben, wird unsere **Base** erstellt. Danach erscheint die Übersicht aus Bild 3. Hier sehen wir im linken Bereich einige Möglichkeiten, mit denen wir gleich zu Beginn Daten aus anderen

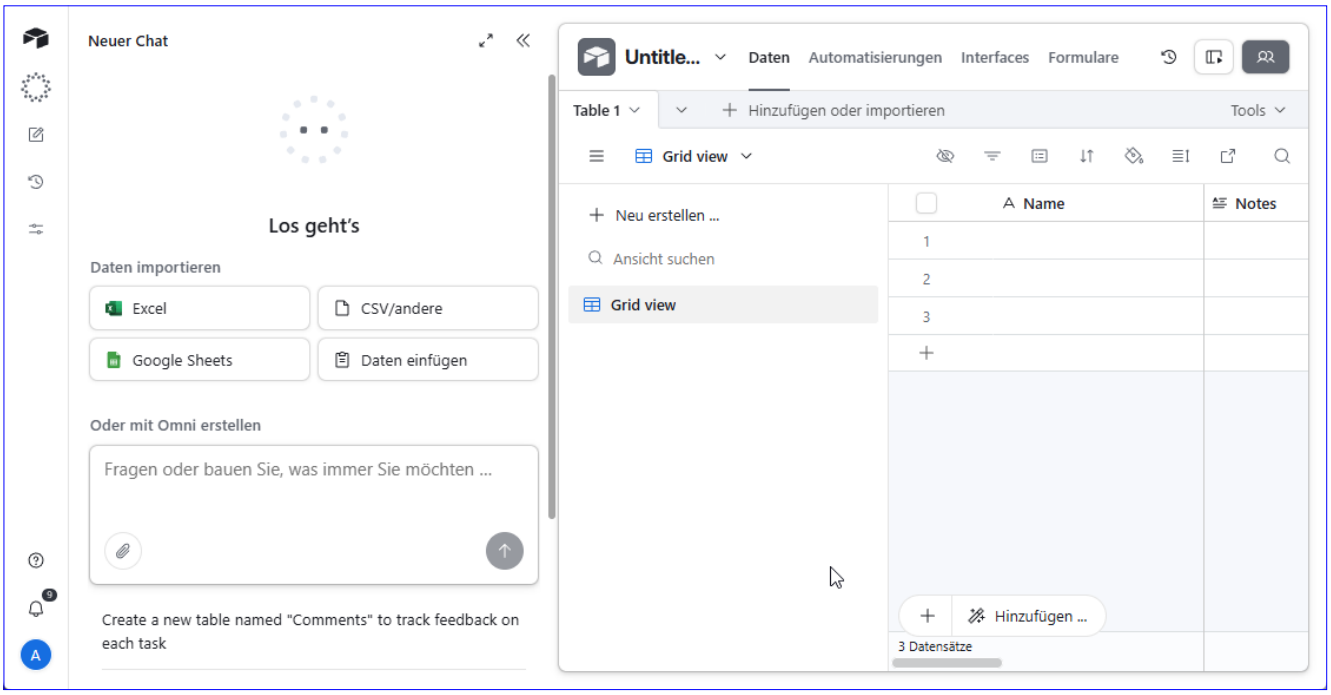

Bild 3: Übersicht für die neue Datenbank

Datenquellen importieren können – zum Beispiel aus Excel, CSV-Dateien oder Google Sheets. Diese benötigen wir nicht und wenden uns dem Erstellen einer eigenen Tabelle zu. Dazu minimieren wir den Bereich auf der linken Seite.

Erste Tabelle erstellen

Damit kommen wir zum Bereich mit der neuen Tabelle **Table 1**. Diese wollen wir zunächst in Bestellungen umbenennen, was über das Aufklappmenü gelingt. Danach widmen wir uns den Feldern der Tabelle. Hier sind bereits einige Beispielfelder vorgegeben, die wir nacheinander löschen. Dazu nutzen wir das Menü aus Bild 4, in dem wir auch einige weitere Befehle finden.

Das Feld »Name« kann nicht gelöscht werden

Hier stellen wir auch fest, dass wir das erste Feld der Tabelle namens **Name** nicht löschen können.

Der Hintergrund ist, dass dieses Feld immer vorhanden sein muss. Es wird als Wert für eventuelle Nachschlagefelder von anderen Tabellen aus verwendet. Wenn wir also eine weitere Tabelle namens **Kunden** erstellen wollten, könnten wir von dieser aus die Tabelle **Bestellungen** verknüpfen und würden beim Auswählen den Inhalt des Feldes **Name** sehen.

Der Vorteil dieses Feldes ist, dass wir ihm nicht zwingend einen festen Wert für jeden Datensatz zuweisen müssen.

Wir können stattdessen auch eine Funktion nutzen, die verschiedene andere Felder aggregiert oder auch Zeichenketten oder Funktionen enthält. So können wir immer einen aussagekräftigen Wert als Identifikation für den Datensatz verwenden.

Wo ist das Primärschlüsselfeld?

Eingefleischte Access-Programmierer werden das Primärschlüsselfeld vermissen. Das Feld **Name** ist es nicht – es kann durchaus auch mehrere gleichlautende Werte aufnehmen. Der Primärschlüssel wird intern verwaltet und enthält eine eindeutige Zeichenkette.

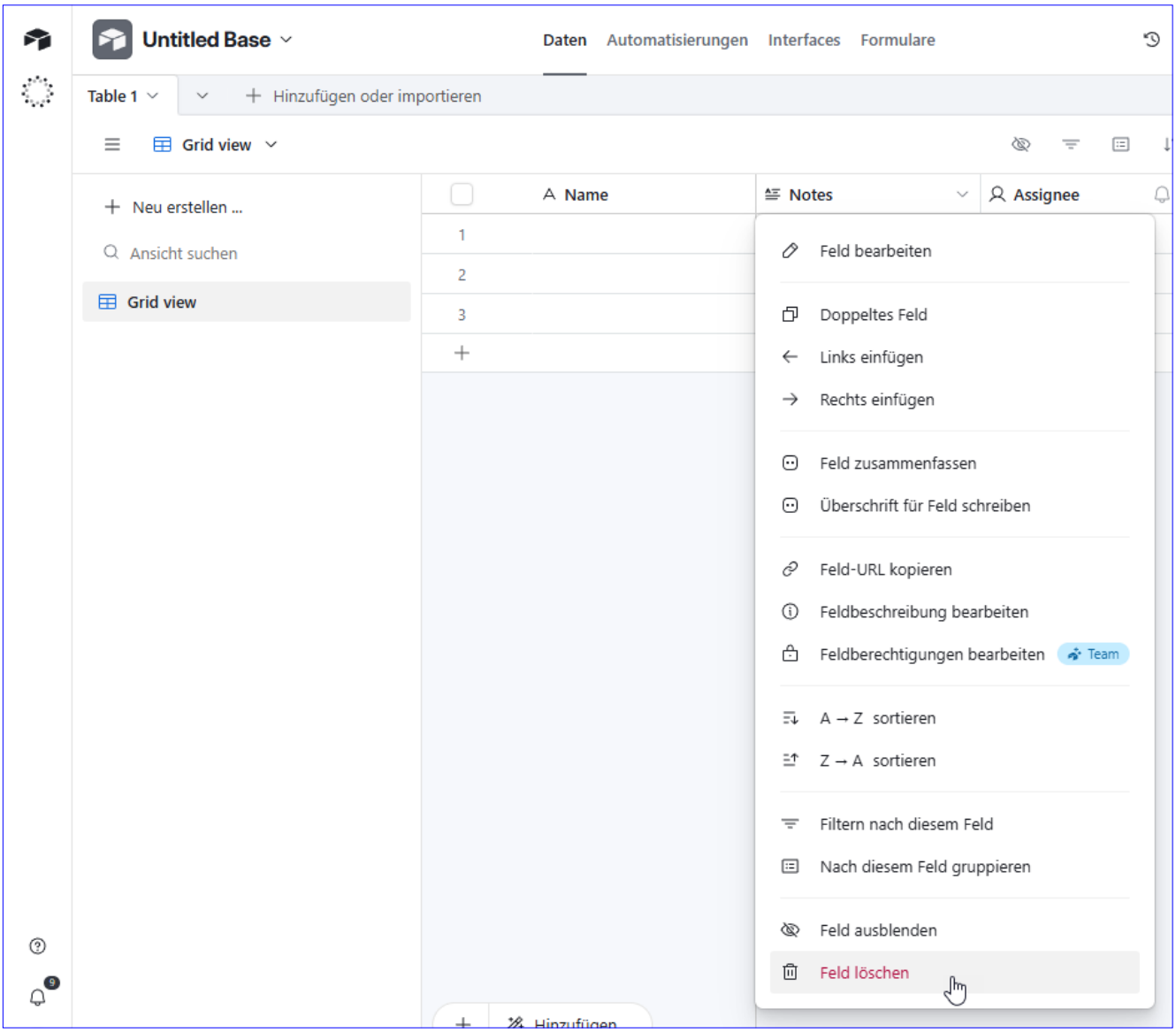

Bild 4: Löschen eines Feldes

Felder anlegen

Damit legen wir die benötigten Felder an. Unsere Tabelle soll alle für eine Bestellung notwendigen Felder aufnehmen können.

Hier machen wir einmal eine Ausnahme und normalisieren die Tabelle nicht – wir nehmen einfach die Kundendaten und die Daten bezüglich des bestellten Produkts in einer einzigen Tabelle auf. Das ist in diesem Fall kein Problem, denn die Tabelle soll uns nur als Zwischenspeicher für die über das Onlineformular aufgenommenen Bestellungen dienen.

In einem späteren Artikel werden wir dann beschreiben, wie wir die aufgenommenen Informationen aus dieser Tabelle in die entsprechend normalisierten Tabellen einer Access-Datenbank übernehmen.

Starten wir also mit dem Anlegen der Felder. Als Erstes möchten wir ein Datumsfeld mit dem Bestelldatum

anlegen. Nachdem wir zuvor alle Beispielfelder gelöscht haben, können wir nun mit einem Klick auf das **Plus**-Symbol in der Spalte direkt neben dem Feld **Name** beginnen.

Es öffnet sich eine Auswahl, die im oberen Bereich einige Feldagenten anzeigt, also eine Hilfe zum Erstellen von Feldern (siehe Bild 5).

Wir wollen jedoch direkt ein Datumsfeld erstellen, wozu wir aus der Liste darunter den Eintrag **Datum** auswählen.

Danach erscheint ein weiterer Dialog mit den Eigenschaften des Datumsfeldes (siehe Bild 6).

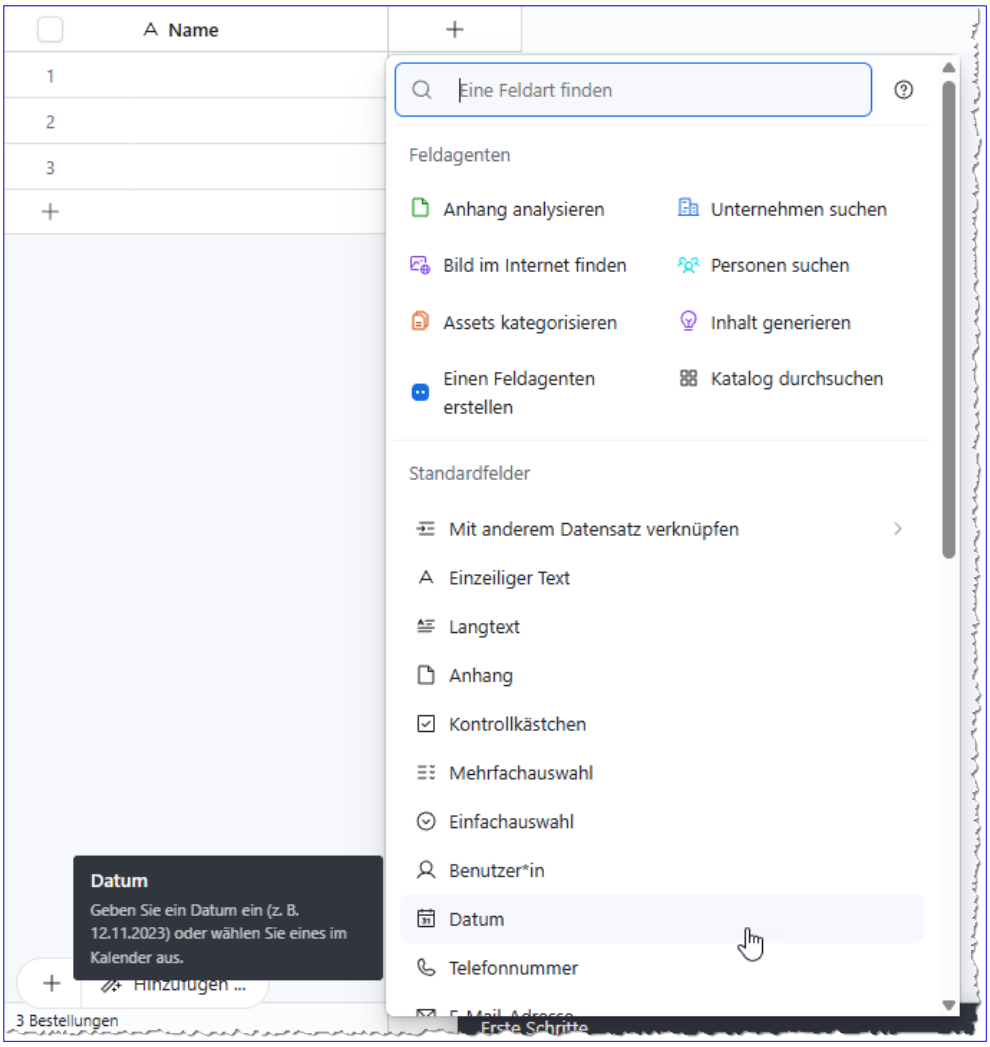

Bild 5: Anlegen eines Feldes

Hier geben wir den Feldnamen ein (**Bestelldatum**), behalten das Datumsformat bei und wählen **Uhrzeit einschließen** aus. Für diese legen wir das Zeitformat auf **24 hour** fest.

Mit einem Klick auf **Feld erstellen** legen wir das Feld en.

Außerdem wollen wir ein Feld für das bestellte Produkt anlegen.

Dazu klicken wir erneut auf das **Plus**-Symbol und wählen diesmal **Einzeiliger Text** aus. Im folgenden Bereich legen wir als Feldnamen **Produkt** fest.

Felder für die Kundendaten erstellen

Danach folgen die Kundendaten. Hier legen wir nacheinander die folgenden Felder jeweils als einzeiligen Text an:

- **Firma**
- **Vorname**
- **Nachname**
- **Straße**
- **PLZ**

- **Ort**
- **Land**

Für das Feld **E-Mail** wählen wir den Datentyp **E-Mail-Adresse** aus.

Vor dem Feld **Vorname** fügen wir noch ein Feld namens **Anrede** ein. Dazu klicken wir auf das Feld **Vorname** und wählen die Funktion **Links einfügen** aus. Für das Feld **Anrede** verwenden wir allerdings den Feldtyp **Einfachauswahl**. Im nun erscheinenden Konfigurationsbereich für das Feld geben wir wie üblich den Feldnamen ein und klicken dann auf Option hinzufügen. Damit können wir die gewünschten Auswahlwerte eingeben (**Herr**, **Frau** und **Divers**) – siehe Bild 7.

Feldeigenschaften nachträglich ändern

Wenn wir den Namen oder andere Eigenschaften eines Feldes nachträglich ändern wollen, klicken wir auf den nach unten zeigenden Pfeil im Spaltenkopf und wählen den Befehl **Feld bearbeiten** aus.

Beispieldaten eingeben

Damit können wir bereits Beispieldaten in den ersten Datensatz eingeben. Airtable zeigt für neue Tabellen standardmäßig drei leere Datensätze an.

Diese können wir erst einmal löschen, indem wir diese durch einen Klick auf das Kontrollkästchen oben im Feld Name markieren und mit der rechten Maustaste das Kontextmenü öffnen. In diesem wählen wir den

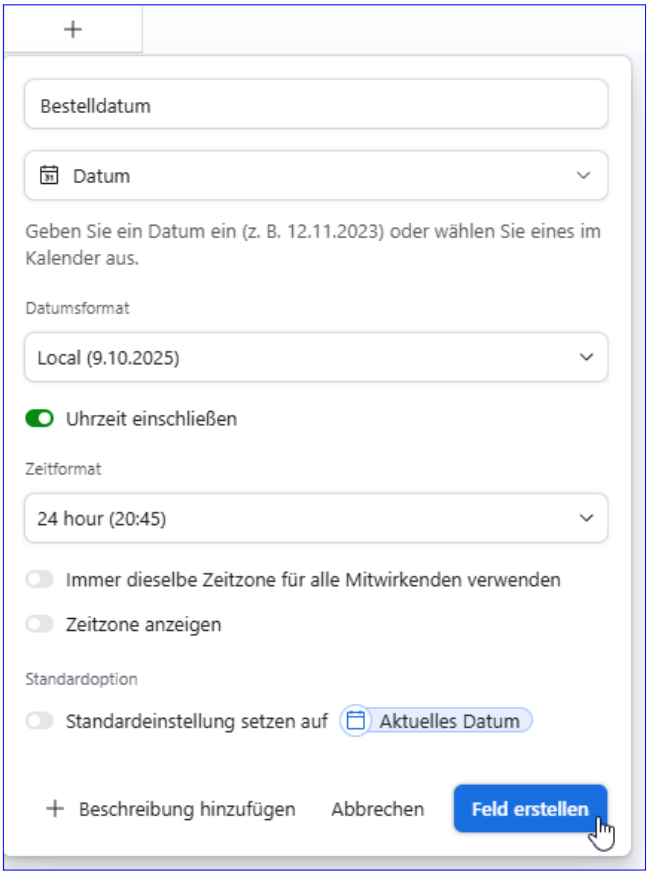

Bild 6: Einstellen der Eigenschaften des Datumsfeldes

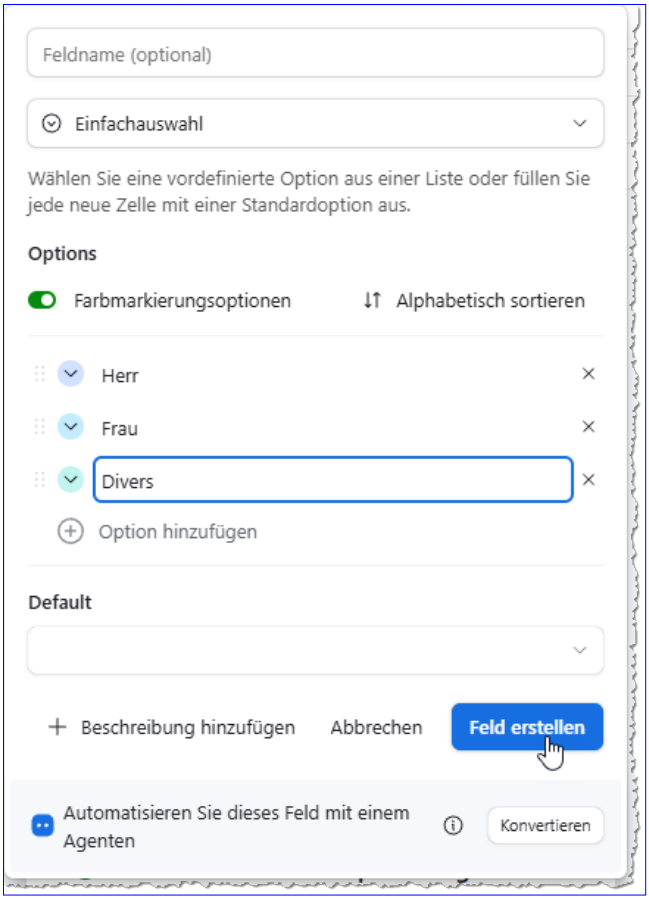

Bild 7: Anlegen eines Felds mit dem Typ **Einfachauswahl**

Airtable per Rest-API synchronisieren

Das Datenbanksystem Airtable bietet eine Menge Funktionen an, aber offeriert leider keine direkte Schnittstelle, mit der man direkt etwa von Access auf die enthaltenen Daten zugreifen kann. Es gibt zwar kostenpflichtige ODBC-Schnittstellen von Drittanbietern, aber wir wollen den Zugriff selbst programmieren. Wie für moderne SaaS-Tools üblich, bietet auch Airtable eine Rest-API als Schnittstelle für den Zugriff auf die Daten an. Diese wollen wir im vorliegenden Artikel untersuchen und zeigen, wie wir auf die enthaltenen Daten zugreifen und Informationen aus einer lokalen Datenbank in eine Airtable-Datenbank schreiben können.

Der erste Schritt für den Zugriff auf eine Rest-API ist immer die Ermittlung des dazu notwendigen API-Schlüssels. Diesen holen wir uns nach dem Anmelden auf **airtable.com**, indem wir zunächst oben rechts auf den Button für unser Konto klicken und aus dem nun erscheinenden Menü den Eintrag **Builder Hub** auswählen (siehe Bild 1).

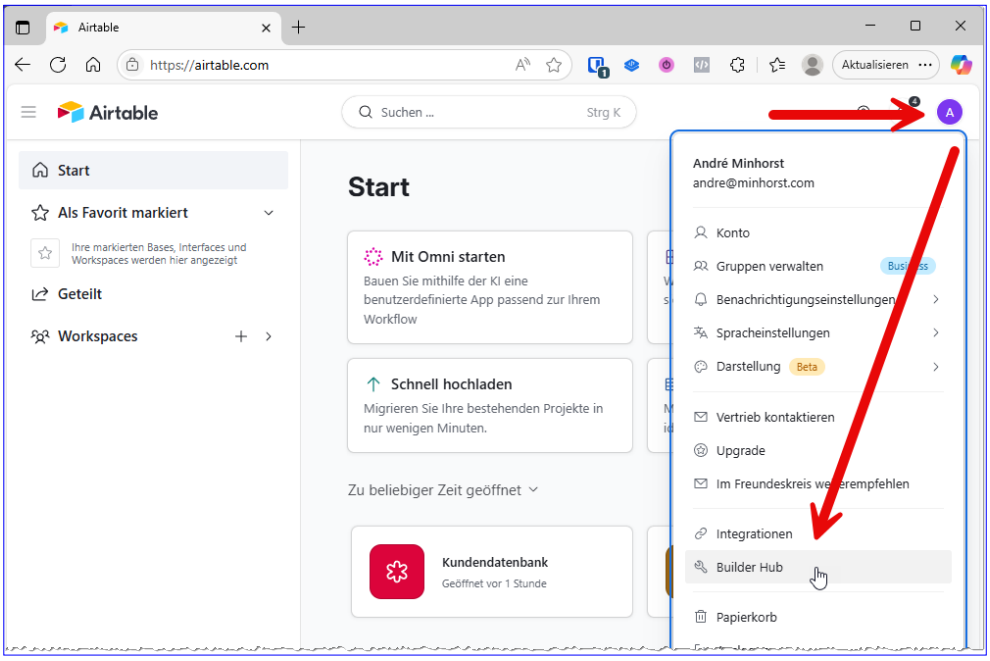

Bild 1: Öffnen des Builder Hubs

Danach landen wir im **Builder Hub**, wo wir über

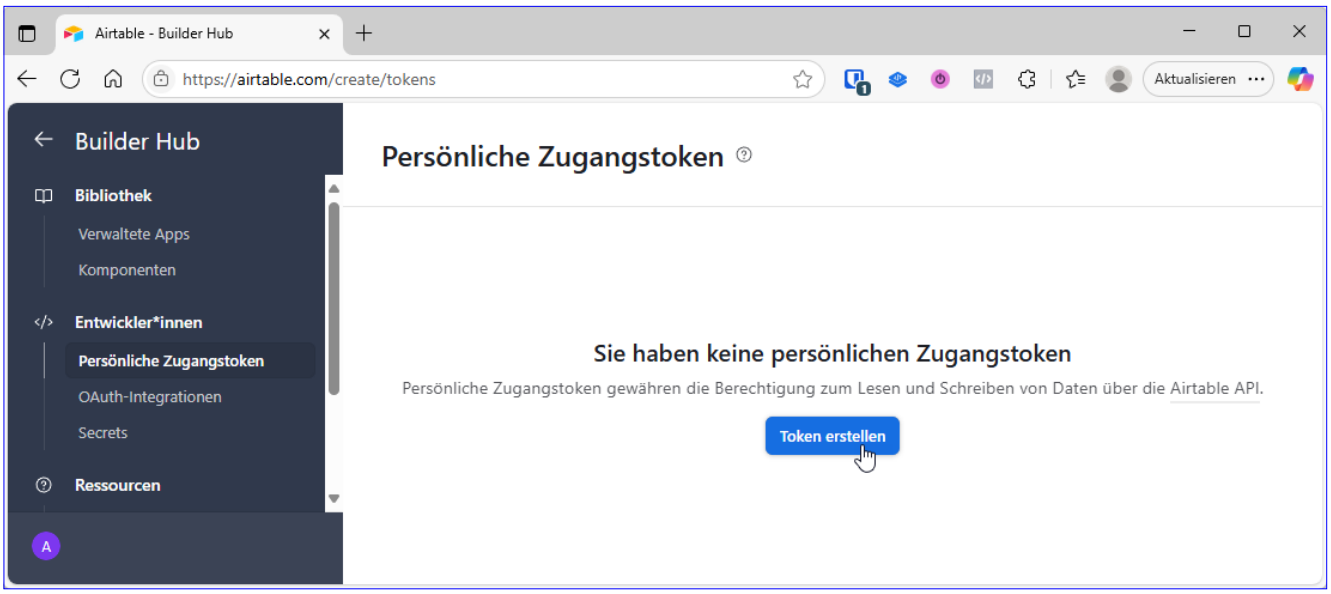

Bild 2: Anlegen eines Tokens

den Menübefehl **Persönliche Zugangstoken** eine Schaltfläche namens **Token erstellen** einblenden können (siehe Bild 2).

Persönlicher Zugangstoken

Neben dem persönlichen Zugangstoken gibt es noch OAuth-Token. Für das Synchronisieren einer Access-Datenbank mit einer Airtable-Datenbank ist das persönliche Zugangstoken die beste Lösung.

Das OAuth-Token benötigen wir nur, wenn wir eine App entwickeln, mit der wir im Namen andere Benutzer auf deren Airtable-Datenbanken zugreifen wollen.

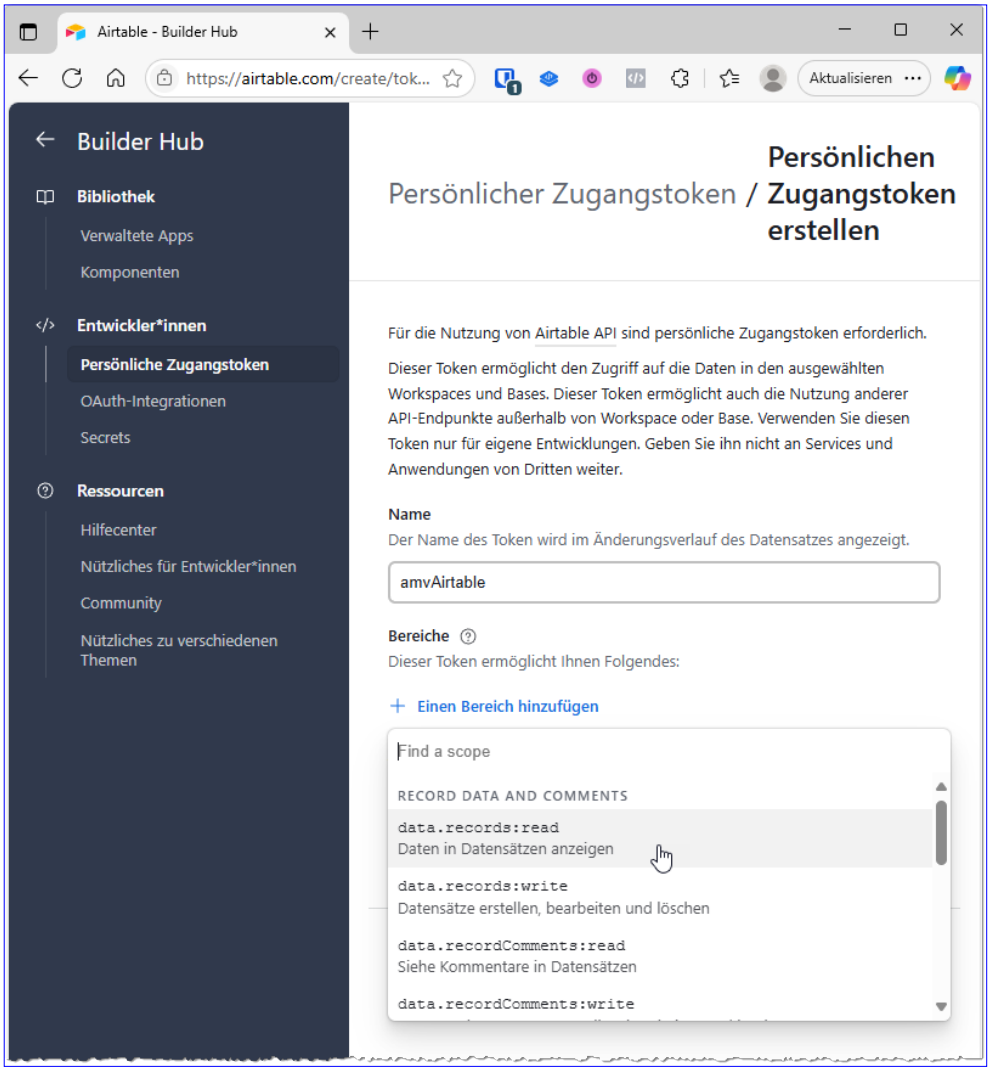

Bild 3: Hinzufügen von Bereichen

Bereiche für das Token festlegen

Nach dem Anlegen des persönlichen Tokens legen wir fest, für welche Bereiche dieses Token gelten soll. Diese wählen wir aus, indem wir die Liste der verfügbaren Bereiche mit einem Klick auf **Einen Bereich hinzufügen** öffnen und dort die gewünschten Bereiche hinzufügen (siehe Bild 3).

Wir wollen ausgiebig experimentieren, also fügen wir die Bereiche aus Bild 4 hinzu.

Anschließend wollen wir noch festlegen, auf welche Datenbanken in unserer Airtable-Instanz sich die hinzugefügten Berechtigungen auswirken sollen. Das er-

Bild 4: Alle notwendigen Bereiche

ledigen wir im gleichen Bereich weiter unten, wo wir mit einem Klick auf **Alle Ressourcen hinzufügen** direkt alle Datenbanken gleichzeitig freischalten oder mit **Eine Base hinzufügen** gezielt eine Datenbank für den Zugriff auswählen (siehe Bild 5).

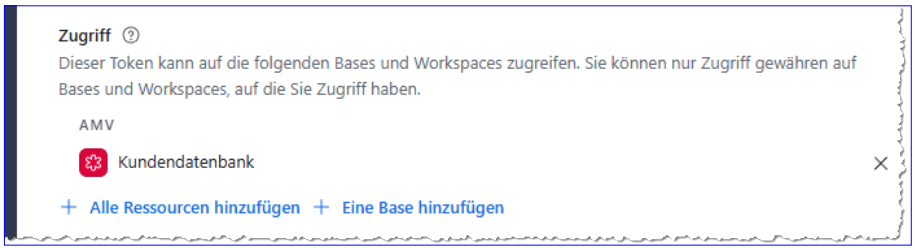

Bild 5: Hinzufügen der Datenbank, auf die wir zugreifen wollen

Nun wird das Token erstellt und wir können es aus dem nun erscheinenden Dialog in die Zwischenablage kopieren (siehe Bild 6).

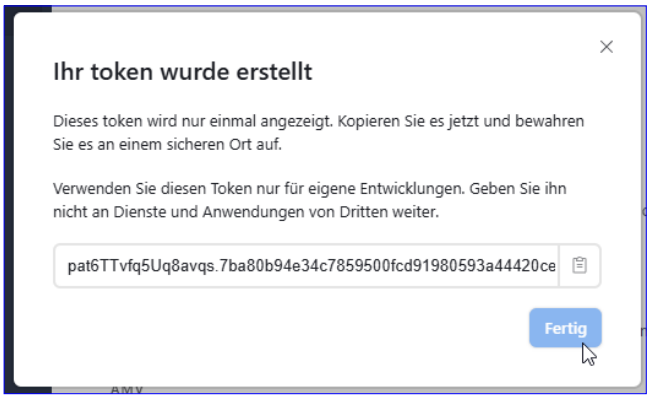

Bild 6: Das fertige Token

Achtung: Das Token können wir nur einmalig abrufen. Also fügen wir es direkt als Wert einer Konstanten namens **cStrAPIToken** in ein neues VBA-Modul im VBA-Projekt der Datenbank ein, von der aus wir auf die Airtable-Datenbank zugreifen wollen. Im gleichen Zuge legen wir auch direkt noch eine weitere Konstante für die Basis-URL für die Rest-API-Zugriffe fest:

```
Private Const cStrBaseURL As String = "https://api.airtable.com/v0/"
Private Const cStrAPIToken As String = "pat6TTvfq5Uq8avqs.7baxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxc7553910b58"
```

ID der Airtable-Datenbank ermitteln

Außerdem benötigen wir die ID der Airtable-Datenbank, mit der wir arbeiten wollen.

Dazu öffnen wir die folgende URL unter der wir auch noch andere wichtige Informationen finden:

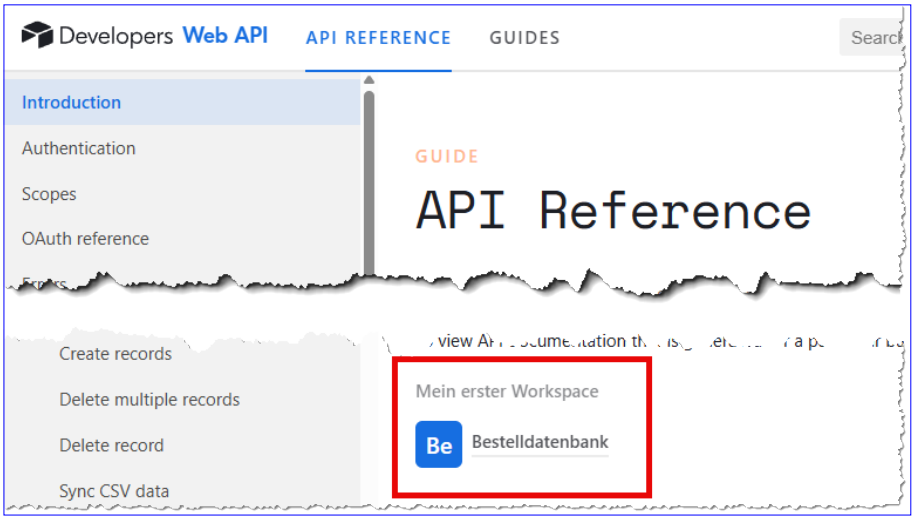

Bild 7: Übersicht der Airtable-Datenbanken

```
https://airtable.com/developers/web/api/introduction
```

Hier sehen wir alle Datenbanken des aktuellen Kontos (siehe Bild 7). Also klicken wir die gewünschte Datenbank an.

Anschließend erscheint eine Ansicht, die unscheinbar in einem Absatz die ID der zu verwendenden Datenbank enthält (siehe Bild 8).

Auch diese kopieren wir zunächst in die Zwischenablage und fügen sie als Wert einer weiteren Konstanten namens **cStrBaseID** ein:

```
Private Const cStrBaseID As String =
"appxxxxxxxxxxxxxx" 'Kundenverwaltung
```

Achtung: Wir benötigen die ID ohne den abschließenden Punkt.

Beispiele für die verschiedenen Datenbankoperationen

Danach schauen wir uns diese Seite genauer an. Links in der Übersichtsleiste sehen wir nämlich die Tabellen unserer Datenbank (siehe Bild 9). Öffnen wir einen dieser Einträge, sehen wir Untereinträge, die nach dem Anklicken weitere Informationen liefern – zum

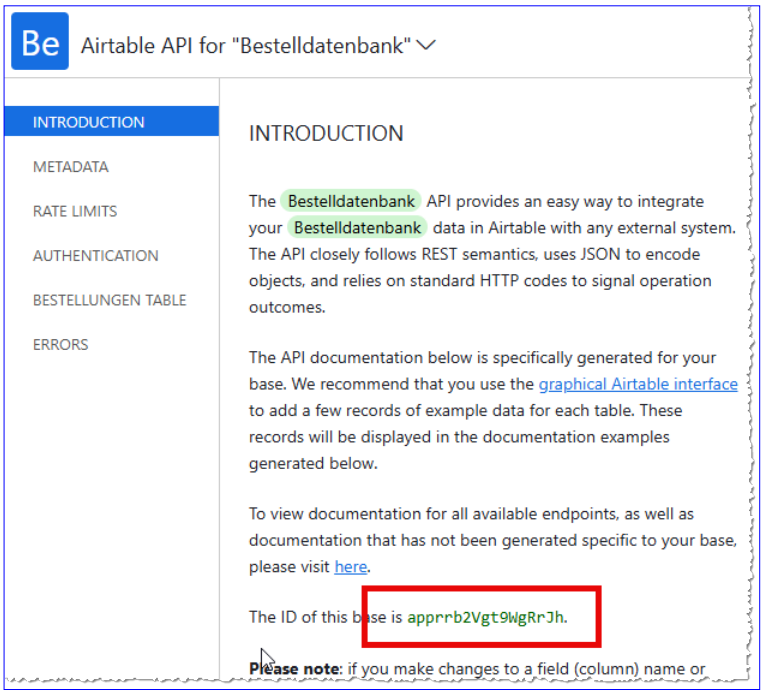

Bild 8: Ermitteln der ID der Datenbank

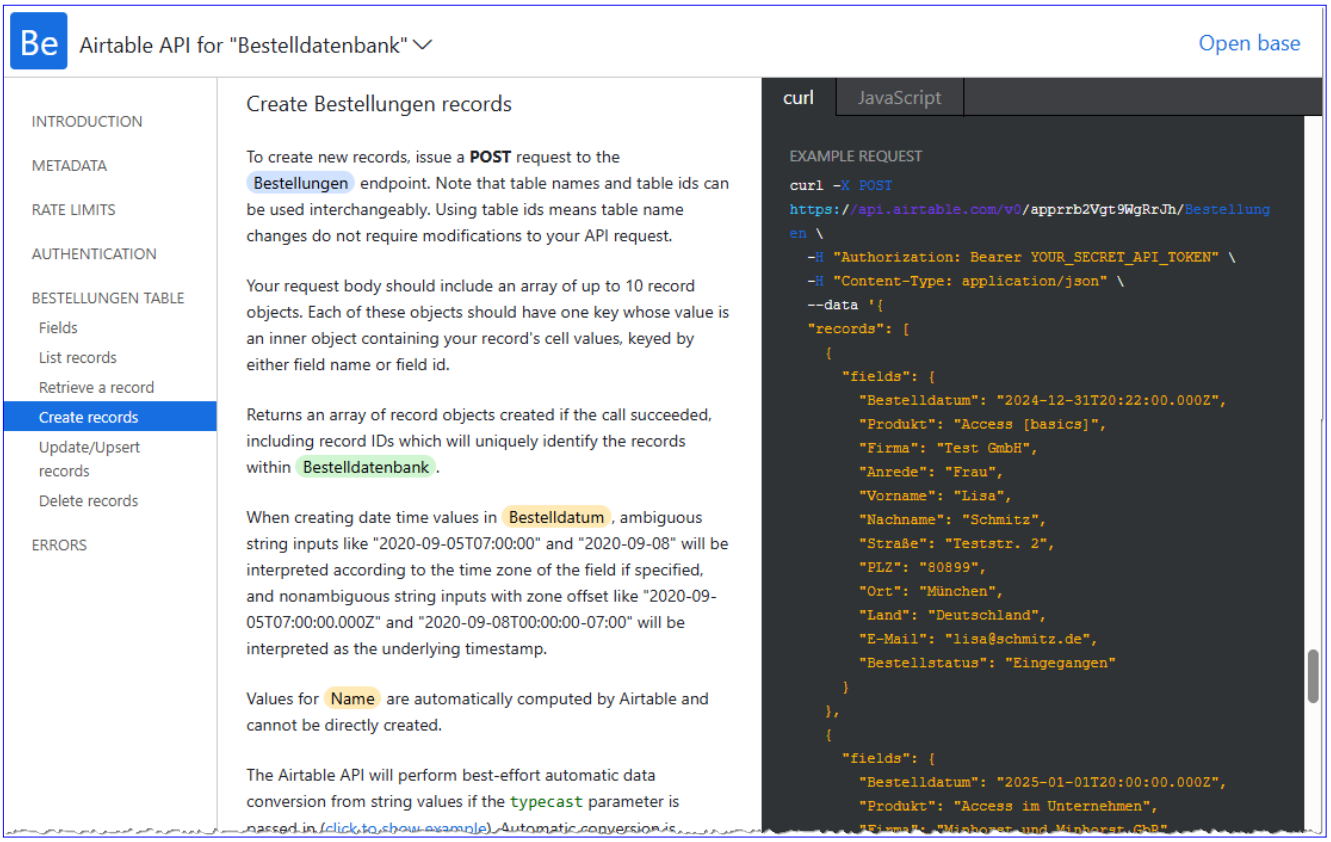

Bild 9: Beispiele für die verschiedenen Datenbank-Operationen

Beispiel, wie man Datensätze der jeweiligen Tabelle ausgibt, einzelne Datensätze ermittelt, diese erstellt, aktualisiert oder löscht. Auf der rechten Seite sehen wir eine Beschreibung und den notwendigen JSON-Code, den wir mit dem jeweiligen Aufruf der Rest-API übergeben müssen.

Liste der Tabellen ermitteln

Mit der Rest-API können wir verschiedene Informationen unserer Airtable-Datenbank abrufen. Bevor wir in das Abfragen, Bearbeiten oder Löschen von Daten aus Airtable-Tabellen einsteigen, ermitteln wir erst einmal die grundlegenden Informationen der in der Datenbank enthaltenen Tabellen.

Dazu verwenden wir die Prozedur **GetAirtableTables** aus Listing 1. Hier erstellen wir ein Objekt des Typs **XMLHTTP60**, mit dem wir auf die Rest-API zugreifen. Wir stellen in **strURL** die Adresse für den Zugriff auf unsere Datenbank zusammen, wobei wir die ID der Datenbank aus der oben definierten Konstanten **cStrBaseID** integrieren. Die URL sieht nun beispielsweise wie folgt aus:

```
https://api.airtable.com/v0//meta/bases/app...RrJh/tables
```

Das API-Token aus **cStrAPIToken** übergeben wie mit dem **RequestHeader** namens **Authorization**. Dann starten wir den Aufruf und erhalten das Ergebnis mit **objXMLHTTP.Response** im JSON-Format. Dieses verarbeiten wir mit der Funktion **ParseJson** aus dem Modul **mdlJSON** in ein Objekt aus **Dictionary**- und **Collection**-Elementen, auf das wir dann zugreifen können. Damit wir wissen, wie wir auf die enthaltenen Elemente zugreifen können, geben wir diese mit der **GetJSONDOM**-Funktion im Direktbereich aus.

```
Sub GetAirtableTables()
    Dim objXMLHTTP As MSXML2.XMLHTTP60
    Dim objJSON As Object
    Dim strURL As String
    Dim i As Integer

    Set objXMLHTTP = New MSXML2.XMLHTTP60

    strURL = cStrBaseURL & "meta/bases/" & cStrBaseID & "/tables"

    objXMLHTTP.Open "GET", strURL, False
    objXMLHTTP.setRequestHeader "Authorization", "Bearer " & cStrAPIToken
    objXMLHTTP.send

    Debug.Print GetJSONDOM(objXMLHTTP.responseText, True)

    Set objJSON = ParseJson(objXMLHTTP.responseText)

    Debug.Print "TabelleID:" & vbTab & vbTab & vbTab & vbTab & vbTab & "Tabellenname:"

    For i = 1 To objJSON.Item("tables").Count
        Debug.Print objJSON.Item("tables").Item(i).Item("id"), objJSON.Item("tables").Item(i).Item("name")
    Next i
End Sub
```

Listing 1: Prozedur zum Auslesen aller Tabellen der Airtable-Datenbank

Dadurch sehen wir, dass wir auf die Tabellendaten beispielsweise wie folgt zugreifen können:

```
objJson.Item("tables").Item(1).Item("id"): tblMUktvNEoOFPRYb
objJson.Item("tables").Item(1).Item("name"): Bestellungen
```

Daraus können wir eine **For...Next**-Schleife ableiten, um die Werte für die Felder **id** und **name** auszulesen und einzeln im Direktbereich auszugeben. Das Ergebnis sieht wie folgt aus:

```
TabelleID:              Tabellenname:
tblMUktvNEoOFPRYb       Bestellungen
```

Damit haben wir nun auch die ID der Tabelle, mit der wir in den folgenden Abschnitten auf die Tabelle **Bestellungen** zugreifen und diese auslesen, bearbeiten oder um neue Datensätze ergänzen können.

Grundlegender Aufbau eines Aufrufs der Rest-API für den Tabellenzugriff

Airtable stellt eine Basis-URL für den Zugriff auf die Tabellendaten einer Airtable-Datenbank zur Verfügung. Wir erstellen eine Basisfunktion mit verschiedenen Parametern, um alle Operationen abzudecken. Den ersten Teil dieser Funktion finden wir in Listing 2. Hier sehen wir zunächst die Parameter der Funktion:

- **strTable**: Nimmt den Namen der zu bearbeitenden oder zu lesenden Tabelle entgegen.
- **strMethod**: Nimmt die Methode für die auszuführende Operation entgegen. Die zu verwendenden Werte inklusive weiterer benötigter Informationen beschreiben wir weiter unten.
- **strRequest**: JSON-Dokument mit dem Inhalt der gewünschten Änderung, also zum Beispiel der Daten für den anzulegenden Datensatz oder eines zu ändernden Datensatzes
- **strId**: ID des zu bearbeitenden oder zu löschenden Datensatzes
- **strResponse**: Antwort der Rest-API, falls eine geliefert wird
- **intErrorNumber**: Fehlernummer beziehungsweise Statuswert. Im Erfolgsfall **200**, die übrigen Werte sind im Code beschrieben.
- **strErrorDescription**: Beschreibung des Fehlers, falls der Statuswert nicht **200** lautet.
- **varParameters**: Paramarray, das die Name-Wert-Paare für eventuelle URL-Parameter enthält.

Die Funktion stellt im ersten Schritt die URL für den Zugriff auf die Rest-API zusammen. Diese besteht aus der Basis-URL (**cStrBaseURL**), der ID für die zu verwendende Datenbank (**cStrBaseID**), einem Slash und dem Namen der Tabelle oder der ID der Tabelle – hier verwenden wir der Einfachheit halber den Namen. Die Verwendung der ID hätte den Vorteil, dass wir in der Datenbank auch einmal den Namen der Tabelle ändern könnten und der Code immer noch funktioniert. Die IDs haben wir zuvor mit der Prozedur **GetAirtableTables** ermittelt. Die URL könnte nun wie folgt aussehen:

```
https://api.airtable.com/v0/app...RrJh/Bestellungen
```

Wenn wir einen vorhandenen Datensatz bearbeiten oder löschen wollen, können wir den Parameter **strID** verwenden. Damit übergeben wir die ID des zu bearbeitenden Datensatzes. Diese bekommen wir, indem wir den Datensatz zunächst abfragen. Auch dazu sehen wir uns später ein Beispiel an.

Die ID aus **strID** hängen wir, soweit vorhanden, im nächsten Schritt an die URL an. Die URL zum Bearbeiten oder Löschen eines Datensatzes würde wie folgt aussehen:

`https://api.airtable.com/v0/app...RrJh/Bestellungen/recElpIemkgW9bBQC`

Danach folgt die Verarbeitung eventueller Parameter. Wie wir einen solchen Parameter zusammenstellen, beschreiben wir weiter unten.

Da wir auch mehrere Parameter übergeben können, verwenden wir dazu ein **ParamArray**. Wir prüfen im nächsten Schritt mit **IsMissung**, ob **varParameters** Parameter enthält. Ist das der Fall, parsen wir diese in der Funktion **ParseParameters**, die wir im Anschluss beschreiben.

Nun erstellen wir ein Objekt des Typs **MSXML2.XMLHTTP**. Seiner **Open**-Funktion übergeben wir die Methode aus **strMethod** (also **GET**, **POST**, **PUT** oder **DELETE**), die vorher zusammengestellte URL und den Wert **False** für den Parameter **varAsync**, damit der Aufruf asynchron ausgeführt und der Code erst nach dem Beenden des Aufrufs weiterläuft. Für das **XMLHTTP60**-Objekt legen wir nun mit **setRequestHeader** das Authentifizierungstoken fest sowie den Inhaltstyp, hier **application/json**.

Schließlich senden wir den Aufruf mit der **send**-Methode ab. Diese erhält den Wert des Parameters **strRe-**

```
Public Function Airtable_CRUD(strTable As String, strMethod As String, strRequest As String, _
        Optional strId As String, Optional strResponse As String, Optional intErrorNumber As Integer, _
        Optional strErrorDescription As String , ParamArray varParameters() As Variant) As Boolean
    Dim strURL As String
    Dim objRequest As MSXML2.XMLHTTP60

    strURL = "https://api.airtable.com/v0/" & cStrBaseID & "/" & strTable

    If Not Len(strId) = 0 Then
        strURL = strURL & "/" & strId
    End If

    If Not IsMissing(varParameters) Then
        strURL = strURL & "?" & ParseParameters(varParameters)
    End If

    Set objRequest = New MSXML2.XMLHTTP60

    With objRequest
        .Open strMethod, strURL, False
        .setRequestHeader "Authorization", "Bearer " & cStrAPIToken
        .setRequestHeader "Content-Type", "application/json"
        .send strRequest
    End With

    strResponse = objRequest.responseText
    intErrorNumber = objRequest.status

    ...
```

Listing 2: Funktion zum Erstellen, Aktualisieren, Lesen und Löschen von Datensätzen in Tabellen (Teil 1)

```
....
    Select Case objRequest.status
        Case 200
            Airtable_CRUD = True
        Case Else
            Select Case objRequest.status
                Case 400
                    strErrorDescription = "Bad Request. The request encoding is invalid; the request can't be " _
                        & "parsed as a valid JSON."
                Case 401
                    strErrorDescription = "Unauthorized. Accessing a protected resource without authorization " _
                        & "or with invalid credentials."
                Case 402
                    strErrorDescription = "Payment Required. The account associated with the API key making " _
                        & "requests hits a quota that can be increased by upgrading the Airtable account plan."
                Case 403
                    strErrorDescription = "Forbidden. Accessing a protected resource with API credentials that " _
                        & "don't have access to that resource."
                Case 404
                    strErrorDescription = "Not Found. Route or resource is not found. This error is returned " _
                        & "when the request hits an undefined route, or if the resource doesn't exist (e.g. has " _
                        & "been deleted)."
                Case 413
                    strErrorDescription = "Request Entity Too LargeThe request exceeded the maximum allowed " _
                        & "payload size. You shouldn't encounter this under normal use."
                Case 422
                    strErrorDescription = "Invalid Request. The request data is invalid. This includes most of " _
                        & "the base-specific validations. You will receive a detailed error message and code " _
                        & "pointing to the exact issue."
                Case 429
                    strErrorDescription = "Too Many Requests. The API is limited to 5 requests per second per " _
                        & "base. You will receive a 429 status code and a message 'Rate limit exceeded. Please " _
                        & "try again later' and will need to wait 30 seconds before subsequent requests will " _
                        & "succeed. To learn more about rate limits, please visit our Rate Limits guide."
                Case 500
                    strErrorDescription = "Internal Server ErrorThe server encountered an unexpected condition."
                Case 502
                    strErrorDescription = "Bad GatewayAirtable 's servers are restarting or an unexpected outage is " _
                        & "in progress. You should generally not receive this error, and requests are safe to retry."
                Case 503
                    strErrorDescription = "Service UnavailableThe server could not process your request in time. " _
                        & "The server could be temporarily unavailable, or it could have timed out processing " _
                        & "your request. You should retry the request with backoffs."
            End Select
            Airtable_CRUD = False
    End Select
    Set objRequest = Nothing
End Function
```

Listing 3: Funktion zum Erstellen, Aktualisieren, Lesen und Löschen von Datensätzen in Tabellen (Teil 2)

quest. Diesen benötigen wir nur, wenn wir beispielsweise Änderungen an einem Datensatz durchführen oder einen Datensatz hinzufügen wollen. **strRequest** füllen wir dann vor dem Aufruf mit einem JSON-Dokument, das die Werte für den zu ändernden oder anzulegenden Datensatz enthält – mehr dazu in den Beispielen weiter unten. Die Antwort erhalten wir mit dem **responseText**-Parameter von **objXMLHTTP60**.

Außerdem liefert **status** eine Information darüber, ob der Aufruf erfolgreich war (**200**) oder ob ein Fehler ausgelöst wurde – beispielsweise durch ein ungültiges JSON-Dokument in **strRequest**, eine ungültige URL oder einen anderen Fehler.

Den Wert der Eigenschaft **status** werten wir im zweiten Teil der Funktion aus Listing 3 in einer ersten **Select Case**-Bedingung aus, die zunächst prüft, ob **status** den Wert **200** (erfolgreicher Aufruf, Rückgabewert **True**) oder einen anderen Wert enthält (Rückgabewert **False**).

In diesem Fall folgt eine zweite **Select Case**-Bedingung, in der wir den Fehlercode auswerten und in den Rückgabeparameter **strErrorDescription** eintragen.

Parameter für das Auflisten von Datensätzen

Wenn wir eine Liste von Datensätzen ermitteln wollen, übergeben wir für den Parameter **strMethod** den Wert **GET**. Außerdem können wir über **varParameters** mit Name-Wert-Paaren weitere Einstellungen für das Ermitteln der gewünschten Datensätze festlegen. Diese lauten folgendermaßen:

- **fields**: Hiermit können wir angegeben, welche Feldinhalte zurückgegeben werden sollen. Beispiel: **fields=Name** liefert nur die Daten des Feldes **Name** zurück. Wenn wir zwei Felder erhalten wollen, geben wir diese als zwei Parameter an: **"fields=Name", "fields=Produkt"**.

- **filterByFormula**: Hier geben wir einen Filterausdruck an. Ein Beispiel lautet: **{Name} = 'Visual Basic Entwickler'**. Dieser muss URL-kodiert werden, sodass zusammen mit dem Parameternamen anschließend ein Ausdruck wie dieser herauskommt: **filterByFormula=%7BName%7D%20%3D%20%27Visual%20Basic%20Entwickler%27**

- **maxRecords**: Hiermit geben wir die maximale Anzahl zurückzuliefernder Datensätze an, zum Beispiel **maxRecords=5**.

- **pageSize**: Dies gibt die Anzahl der Einträge pro zurückgelieferter Seite an, zum Beispiel **pageSize=10**. Der Standardwert lautet **100**. Dies ist auch der maximale Wert.

- **sort**: Gibt an, in welcher Reihenfolge die Einträge sortiert werden. Der Wert für **sort** lautet beispielsweise **[{field: "Name", direction: "desc"}]** für eine absteigende Sortierung nach dem Inhalt des Feldes **Name**. Auch dies muss wieder URL-kodiert werden, sodass sich insgesamt dieser Ausdruck ergeben würde: **sort=sort%5B0%5D%5Bfield%5D=Name** und **sort%5B0%5D%5Bdirection%5D=desc**.

Wenn wir beispielsweise nur den Wert eines Feldes zurückgeben wollen, verwenden wir die folgenden Parameter:

```
"sort[0][field]=Firma", "sort[0][direction]=asc"
```

Diese müssen wir noch parsen, was wir in der Funktion **ParseParameters** erledigen (siehe Listing 4). Diese nimmt das ParamArray aus **varParameters** als einfachen **Variant**-Parameter entgegen.

Sie durchläuft jedes Element aus **varParameters** in einer **For Each**-Schleife. Darin ermitteln wir die Position des Gleichheitszeichens und speichern sie in **lngPosEqual**. Mit der **Left**-Funktion lesen wir den

Dateimanagement mit dem FileSystemObject

VBA bietet bereits einige Befehle, mit denen wir Dateioperationen ausführen können. Wir können mit MkDir neue Verzeichnisse erstellen, mit Dir prüfen, ob Dateien oder Verzeichnisse vorhanden sind oder mit Kill Dateien löschen. Diese Befehle sind aber recht kompliziert in der Handhabung. Daher schauen wir uns in diesem Artikel einmal die Klasse »FileSystemObject« an, mit der wir deutlich komfortabler mit Dateien und Verzeichnissen arbeiten können. Damit lässt sich alles erledigen, was mit dem Anlegen, Kopieren, Verschieben und Löschen zusammenhängt – und vieles mehr.

Definition

Im Rahmen dieses Artikels werden wir folgende Bezeichnungen verwenden:

Verzeichnis: Laufwerksbuchstabe plus Unterverzeichnisse, zum Beispiel **c:\Temp**

Dateiname: Nur der Dateiname ohne Verzeichnisse, also beispielsweise **Test.txt**

Pfad: Verzeichnis plus Dateiname, hier also **c:\Temp\Test.txt**

Die Klasse FileSystemObject

Die Klasse **FileSystemObject** ist Teil der Bibliothek **Microsoft Scripting Runtime**, die wir als Erstes referenzieren müssen.

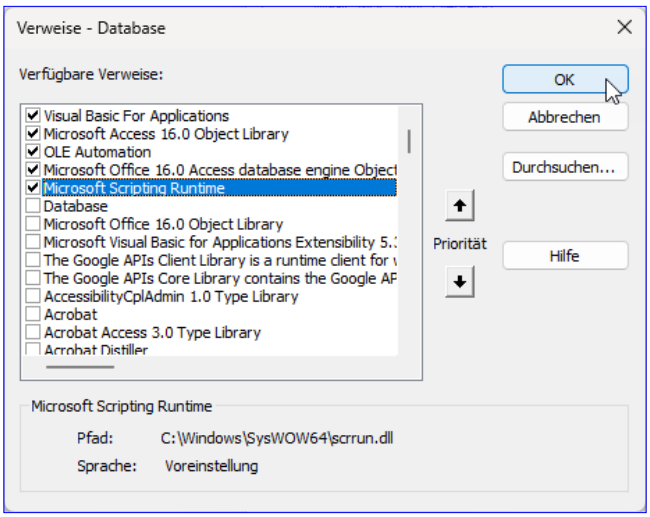

Bild 1: Verweis auf die Bibliothek **Microsoft Scripting Runtime** hinzufügen

Dazu öffnen wir im VBA-Editor den **Verweise**-Dialog und fügen den entsprechenden Eintrag hinzu (siehe Bild 1).

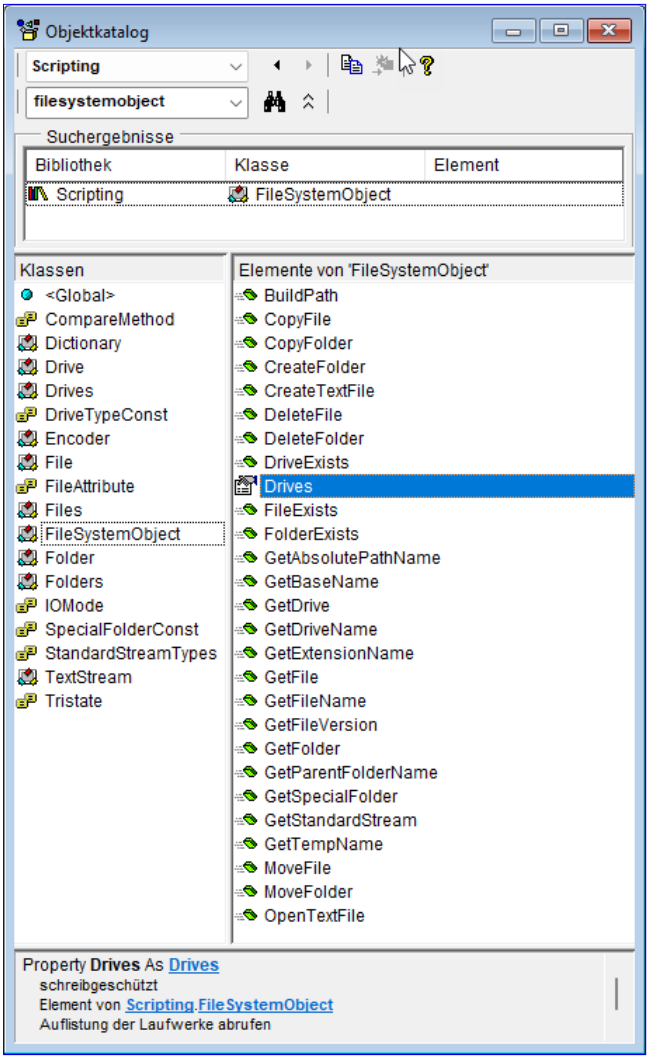

Bild 2: Die Elemente der Klasse **FileSystemObject** in der Übersicht im Objektkatalog

Danach können wir den Objektkatalog öffnen und finden alle Elemente der Klasse **FileSystemObject** vor (siehe Bild 2).

Hier steigen wir nun ein und schauen uns die verschiedenen Elemente erst einmal in der Übersicht an – später gehen wir detailliert auf jedes einzelne ein:

- **BuildPath**: Stellt aus Verzeichnis und Dateinamen einen Pfad zusammen und ergänzt gegebenenfalls das fehlende Backslash-Zeichen zwischen den beiden Elementen.
- **CopyFile**: Kopiert eine Datei.
- **CopyFolder**: Kopiert ein Verzeichnis.
- **CreateFolder**: Erstellt ein Verzeichnis.
- **CreateTextFile**: Erstellt eine Textdatei.
- **DeleteFile**: Löscht eine Datei.
- **DeleteFolder**: Löscht ein Verzeichnis.
- **DriveExists**: Prüft, ob ein Laufwerk vorhanden ist.
- **Drives**: Liefert eine Auflistung von **Drive**-Objekten, die Informationen zu den einzelnen Laufwerken liefern.
- **FileExists**: Prüft, ob eine Datei vorhanden ist.
- **FolderExists**: Prüft, ob ein Verzeichnis vorhanden ist.
- **GetAbsolutePathname**: Holt einen absoluten Pfadnamen.
- **GetBaseName**: Holt den Basisdateinamen ohne Dateiendung.
- **GetDrive**: Ermittelt ein **Drive**-Objekte auf Basis eines Laufwerksbuchstabens, des Laufwerksverzeichnisses oder des Rootverzeichnisses.
- **GetDriveName**: Ermittelt das Verzeichnis eines Laufwerkes auf Basis eines Verzeichnisses, das auch Unterordner enthalten kann.
- **GetExtensionName**: Holt die Dateinamenerweiterung.
- **GetFile**: Holt ein **File**-Objekt.
- **GetFileName**: Ermittelt einen Dateinamen.
- **GetFileVersion**: Ermittelt die Version einer Datei.
- **GetFolder**: Holt ein **Folder**-Objekt.
- **GetParentFolderName**: Ermittelt das übergeordnete Verzeichnis.
- **GetSpecialFolder**: Ermittelt ein **Folder**-Objekt zu einem Spezialverzeichnis.
- **GetStandardStream**: Liefert ein Stream-Objekt auf Basis einer Datei.
- **GetTempName**: Liefert einen zufälligen temporären Dateinamen.
- **MoveFile**: Verschiebt eine Datei.
- **MoveFolder**: Verschiebt ein Verzeichnis.
- **OpenTextFile**: Öffnet eine Textdatei.

Die Klasse FileSystemObject nutzen

Um die Elemente der Klasse **FileSystemObjekt** zu nutzen, müssen wir zunächst ein Objekt auf Basis dieser Klasse erstellen.

Das erledigen wir bei Early Binding, also mit vorhandenem Verweis auf die Bibliothek **Microsoft Scripting Runtime**, wie folgt:

```
Dim fso As Scripting.FileSystemObject
Set fso = New Scripting.FileSystemObject
```

Wenn wir Late Binding nutzen, also ohne Verweis arbeiten wollen, nutzen wir:

```
Dim fso As Object
Set fso = CreateObject("Scripting.FileSystemObject")
```

Der Vorteil von Early Binding liegt darin, dass wir während der Entwicklung die Funktionen von IntelliSense nutzen können.

Wir nutzen hier eine Objektvariable mit der Bezeichnung **fso**. Man könnte auch beispielsweise **objFileSystemObject** nutzen, damit man direkt weiß, welche Klasse damit referenziert wird.

Wir verwenden hier aus Platzgründen die kürzere Variante (eine Alternative ist **objFSO**).

Laufwerksinformationen ermitteln mit Drive

Die **Drives**-Auflistung liefert uns Informationen über die verschiedenen Laufwerke. Wir können folgende Informationen erhalten:

- **AvailableSpace**: Liefert den freien Speicherplatz in Bytes.
- **DriveLetter**: Liefert nur den Laufwerksbuchstaben, also beispielsweise **c**.
- **DriveType**: Liefert den Typ des Laufwerks als Zahlenwert mit den möglichen Werten **0** (**Unknown Type**), **1** (**Removable**), **2** (**Fixed**), **3** (**Remote**), **4** (**CDRom**) oder **5** (**RamDisk**).
- **FileSystem**: Liefert beispielsweise den Wert **FAT32** oder **NTFS**.
- **FreeSpace**: Liefert den freien Speicherplatz für den aktuellen Benutzer. Dieser kann identisch sein mit dem Wert von **AvailableSpace**.
- **IsReady**: Gibt an, ob das Laufwerk verfügbar ist.
- **Path**: Gibt den Laufwerkbuchstaben gefolgt von einem Doppelpunkt an, zum Beispiel **c:**.
- **RootFolder**: Gibt den Laufwerkbuchstaben gefolgt von einem Doppelpunkt und einem Backslash an, zum Beispiel **c:**.
- **SerialNumber**: Gibt die Seriennummer des Laufwerks an.
- **ShareName**: Liefert den Namen der Windows-Freigabe, wenn das Laufwerk ein Netzlaufwerk ist. Anderenfalls liefert es eine leere Zeichenkette.
- **TotalSize**: Gibt den gesamten Speicherplatz des Laufwerks in Bytes zurück.
- **VolumeName**: Beschriftung des Laufwerks, wie er im Windows Explorer angezeigt wird.

Mit der Prozedur aus Listing 1 geben wir alle Informationen aller Laufwerke aus, indem wir diese in einer **For Each**-Schleife über die **Drives**-Auflistung durchlaufen. Dabei weisen wir die einzelnen Elemente einer Objektvariablen namens **objDrive** mit dem Typ **FileSystemObject.Drive** zu.

Für das Laufwerk **C:** erhalten wir beispielsweise folgende Ausgabe im Direktbereich:

```
AvailbableSpace: 207972556800
DriveLetter:C
```

```
DriveType:2
FileSystem:NTFS
FreeSpace:207972556800
IsReady:Wahr
Path:C:
RootFolder:C:\
SerialNumber:-966946150
ShareName:
TotalSize:1999516987392
VolumeName:Hauptlaufwerk
```

Laufwerk referenzieren mit GetDrive

Mit der Funktion **GetDrive** können wir ein **Drive**-Objekt zu einem Ausdruck wie **c**, **c:** oder **c:** ermitteln.

Im folgenden Beispiel deklarieren wir eine **Drive**-Variable namens **objDrive** und weisen ihr das Ergebnis der **GetDrive**-Funktion für **c:** zu.

```
Public Sub LaufwerkReferenzieren()
    Dim fso As Scripting.FileSystemObject
    Dim objDrive As Scripting.Drive

    Set fso = New Scripting.FileSystemObject
    Set objDrive = fso.GetDrive("c:\")
    Debug.Print objDrive.AvailableSpace
End Sub
```

Danach geben wir den verfügbaren Speicherplatz dieses Laufwerks aus:

Die Standardeigenschaft der **Drive**-Klasse ist die Eigenschaft **Path**.

Wir können **c:** also auch einfach so ausgeben lassen:

```
Debug.Print objDrive
```

```
Public Sub LaufwerksbuchstabenErmitteln()
    Dim fso As Scripting.FileSystemObject
    Dim objDrive As Scripting.Drive

    Set fso = New Scripting.FileSystemObject

    For Each objDrive In fso.Drives
        Debug.Print "AvailbableSpace: " & objDrive.AvailableSpace
        Debug.Print "DriveLetter:" & objDrive.DriveLetter
        Debug.Print "DriveType:" & objDrive.DriveType
        Debug.Print "FileSystem:" & objDrive.FileSystem
        Debug.Print "FreeSpace:" & objDrive.FreeSpace
        Debug.Print "IsReady:" & objDrive.IsReady
        Debug.Print "Path:" & objDrive.Path
        Debug.Print "RootFolder:" & objDrive.RootFolder
        Debug.Print "SerialNumber:" & objDrive.SerialNumber
        Debug.Print "ShareName:" & objDrive.ShareName
        Debug.Print "TotalSize:" & objDrive.TotalSize
        Debug.Print "VolumeName:" & objDrive.VolumeName
    Next objDrive
End Sub
```

Listing 1: Prozedur zur Ausgabe von Laufwerksinformationen im Direktbereich

Name eines Laufwerks ermitteln mit GetDriveName

Mit der Funktion **GetDriveName** können wir den Namen eines Laufwerks für einen beliebigen Pfad ermitteln.

Wenn wir also etwas das Laufwerk benötigen, auf dem sich die aktuelle Datenbankdatei befindet, nutzen wir die folgenden Anweisungen (siehe Funktion **LaufwerkNameErmitteln**), der wir **CurrentProject.Path** als Parameter übergeben:

```
Dim fso As Scripting.FileSystemObject
Dim objDrive As Scripting.Drive
Set fso = New Scripting.FileSystemObject
Debug.Print fso.GetDriveName(CurrentProject.Path)
```

Mit DriveExists prüfen, ob ein Laufwerk existiert

Die Funktion **DriveExists** prüft, ob ein Laufwerk existiert. Dazu können wir den Laufwerksbuchstaben (**c**),

das Verzeichnis (**c:**) oder das Rootverzeichnis (**c:**) übergeben. Für das Laufwerk **d:** liefert die folgende Prozedur beispielsweise den Wert **False**, wenn kein entsprechendes Laufwerk vorhanden ist:

```
Dim fso As Scripting.FileSystemObject
Dim objDrive As Scripting.Drive
Set fso = New Scripting.FileSystemObject
Debug.Print fso.DriveExists("d:\")
```

Mit Verzeichnissen arbeiten

Nachdem wir die Funktionen für die Laufwerke erledigt haben, kommen wir zu den Funktionen für den Umgang mit Verzeichnissen. Hier können wir Verzeichnisse erstellen, prüfen, ob ein Verzeichnis existiert, Verzeichnisse löschen, kopieren oder verschieben, Eigenschaften von Verzeichnissen auslesen und übergeordnete Verzeichnisse oder Spezialverzeichnisse ermitteln.

Verzeichnisse erstellen mit CreateFolder

Mit der Methode **CreateFolder** können wir ein Verzeichnis erstellen. Dazu geben wir einfach den kompletten Pfad zum Verzeichnis an, zum Beispiel **c:\Temp\Ordner 1**:

```
fso.CreateFolder "c:\Temp\Ordner 1"
```

Wenn wir versuchen, einen Ordner zu erstellen, der bereits vorhanden ist, erhalten wir Fehler **58**, **Datei existiert bereits**. Um dies zu verhindern, können wir mit **FolderExists** vorab prüfen, ob das Verzeichnis bereits vorhanden ist (siehe weiter unten).

Wir können mit **CreateFolder** immer nur ein Verzeichnis gleichzeitig erstellen. Um **c:\Temp\Ordner 1\Ordner 2** zu erstellen, benötigen wir zwei Anweisungen:

```
fso.CreateFolder "c:\Temp\Ordner 1"
fso.CreateFolder "c:\Temp\Ordner 1\Ordner 2"
```

Wir können auch das Ergebnis von **CreateFolder** mit einer **Folder**-Variablen referenzieren, um dann die Operationen der **Folder**-Klasse auszuführen – mehr dazu weiter unten (dort finden wir eine weitere Methode zum Anlegen von Verzeichnissen):

```
Dim objFolder As Scripting.Folder
Set objFolder = fso.CreateFolder("c:\Temp\Ordner 1")
```

Mit FolderExists prüfen, ob ein Verzeichnis existiert

Mit der Funktion **FolderExists** können wir prüfen, ob ein Verzeichnis bereits existiert. Dieser Funktion übergeben wir den vollständigen Pfad bis zu dem zu untersuchenden Verzeichnis:

```
Debug.Print fso.FolderExists("c:\Temp\Ordner 1")
```

Vor den verschiedenen Operationen auf Basis eines Verzeichnisses sollten wir in einer **If...Then**-Bedingung prüfen, ob das Verzeichnis überhaupt vorhanden ist:

```
If fso.FolderExists("c:\Temp\Ordner 1") Then
    '... Operationen mit diesem Verzeichnis
End If
```

Verzeichnisse löschen mit DeleteFolder

Mit der Methode **DeleteFolder** können wir Verzeichnisse löschen.

```
fso.DeleteFolder "c:\Temp\Ordner 1"
```

Wenn sich im Verzeichnis beispielsweise eine aktuell geöffneten Access-Datenbank befindet, liefert **DeleteFolder** Fehler **70**, **Zugriff verweigert**. Das Gleiche ist der Fall, wenn der Ordner schreibgeschützt ist.

Wenn wir ein solches Verzeichnis dennoch löschen wollen, können wir **DeleteFolder** für den zweiten Parameter **Force** den Wert **True** übergeben:

UstIdNr, IBAN und Co. per Kontextmenü

Eine Sache, die mich seit Jahren nervt, ist das ständige Heraussuchen von Informationen, die ich mir (nicht mehr) merken kann oder will: Umsatzsteuer-Identifikationsnummern, IBAN, Kreditkartennummern, Anschriften et cetera. Ich habe verschiedene Orte ausprobiert: Kurznotizen von Windows, Textdateien oder Notizen in Outlook. Ich fand es dennoch immer zu aufwendig, erst den entsprechenden Ort zu öffnen, die gewünschte Information zu kopieren und dann am gewünschten Ort einzufügen. Schließlich kam ich auf die Idee, wie es schneller und besser geht: Ich wollte das Kontextmenü von Windows so erweitern, dass die entsprechenden Informationen per Mausklick auf den gewünschten Eintrag in der Zwischenablage landen, von wo aus ich diese schnell per Strg + V an der entsprechenden Stelle einfügen kann. Diese Lösung stelle ich in diesem Artikel vor. Dabei schauen wir uns zuerst an, wie das grundsätzlich funktioniert. Anschließend bauen wir eine kleine Access-Datenbank, in der wir diese Einträge verwalten und diese schnell im Kontextmenü verfügbar machen können.

Das Kontextmenü von Windows bietet die Möglichkeit der Erweiterung um eigene Einträge, sogar in einer Struktur von Untereinträgen. Das ist wichtig, weil sich einige Informationen ansammeln, die man schnell verfügbar haben möchte – und so können wir die Einträge strukturieren und übersichtlich halten.

Kontextmenü von Windows erweitern

Der erste Schritt ist das Erweitern des Kontextmenüs. Hier führt der Weg über die Registry von Windows. Es gibt einen Bereich, wo wir die notwendigen Informationen speichern können.

Dazu öffnen wir erst einmal die Registry, indem wir in der Windows-Suche **Regedit** eingeben und den nun erscheinenden Eintrag **Registrierungs-Editor** anklicken (Profis haben dafür einen eigenen Eintrag in der Taskleiste). Der Pfad zu dem Bereich lautet:

```
Computer\HKEY_CLASSES_ROOT\Directory\Background\shell
```

Einen Eintrag direkt zum Kontextmenü hinzufügen

Um einen Eintrag direkt auf der obersten Ebene des Kontextmenüs zu platzieren, legen wir unter dem Element **shell** einen neuen Schlüssel an. Dazu klicken wir mit der rechten Maustaste auf **shell** und wählen den Befehl **Neu|Schlüssel** aus. Diesen Eintrag nennen wir so, wie er im Kontextmenü erscheinen soll, also beispielsweise **Umsatzsteuer-Identifikationsnummer**.

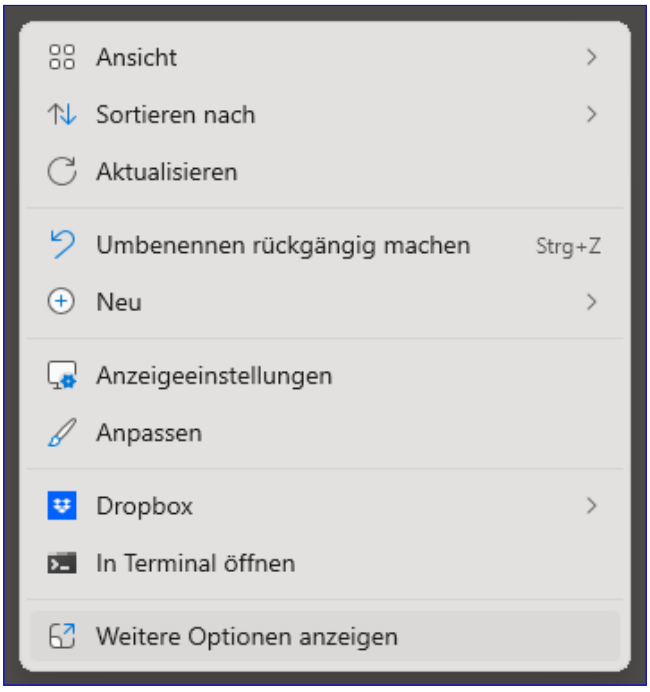

Bild 1: Kontextmenü mit dem Befehl **Weitere Optionen anzeigen**

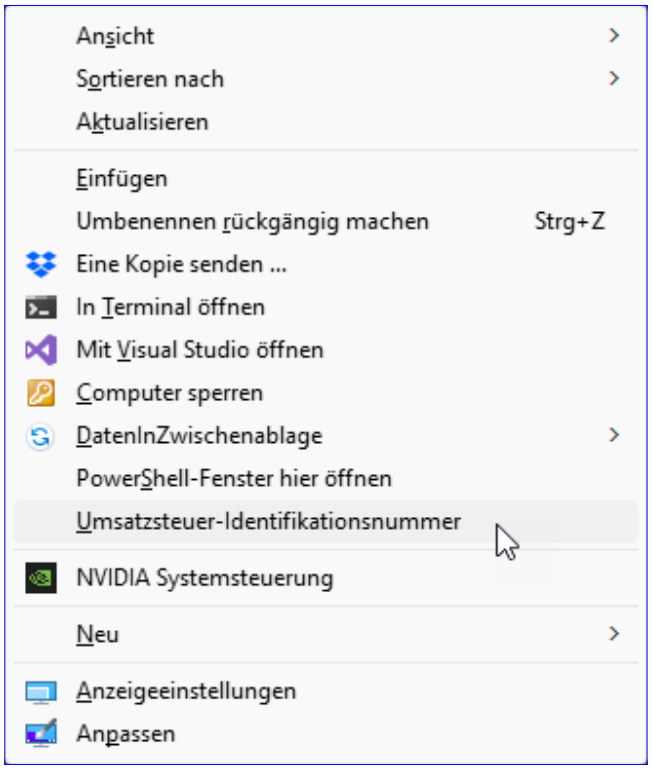

Bild 2: Altes Kontextmenü mit dem gesuchten Eintrag

Allein dadurch zeigen wir bereits einen neuen Eintrag im Kontextmenü an.

Sollte dieser nicht direkt erscheinen, liegt das daran, dass Windows die neuen Kontextmenüs anzeigt, die nicht direkt manipuliert werden können.

Man muss dann noch auf **Weitere Optionen anzeigen** klicken (siehe Bild 1).

Erst dann erscheint ein weiteres Kontextmenü mit dem gesuchten Befehl (siehe Bild 2).

Wir können das alte Kontextmenü allerdings auch direkt anzeigen, indem wir beim Rechtsklick die Umschalttaste gedrückt halten.

Unterhalb des neuen Eintrags in der Registry legen wir einen weiteren Schlüssel namens **command** an. Diesem fügen wir als Wert für das Element **(Standard)** den Befehl hinzu, der beim Betätigen ausgeführt werden soll.

Auszuführende Aktion festlegen

Hier können wir diverse Aktionen festlegen. In unserem Fall ist die einfachste Variante, eine **.bat**-Datei aufzurufen, die den gewünschten Inhalt in die Zwischenablage kopiert.

Der Befehl, den wir in dieser Datei speichern, lautet:

```
set /p="DE295638726" <nul | clip
```

Die Datei (siehe Bild 3) speichern wir wie in unter einem aussagekräftigen Namen in einem Verzeichnis, indem wir auch alle weiteren derartigen Dateien speichern.

Der Befehl **set /p="DE295638726" <nul | clip** kopiert den angegebenen Text ohne Zeilenumbruch direkt in die Windows-Zwischenablage, ohne dass ein Konsolenfenster auf eine Eingabe warten muss. Dadurch blinkt die Eingabeaufforderung nur kurz auf – ein Umstand, mit dem ich leben kann.

Nachdem wir diese Datei gespeichert haben, holen wir uns den Pfad. Das gelingt am schnellsten, indem wir die Datei markieren und die Tastenkombination **Strg + Umschalt + C** betätigen. Allein diese Tasten-

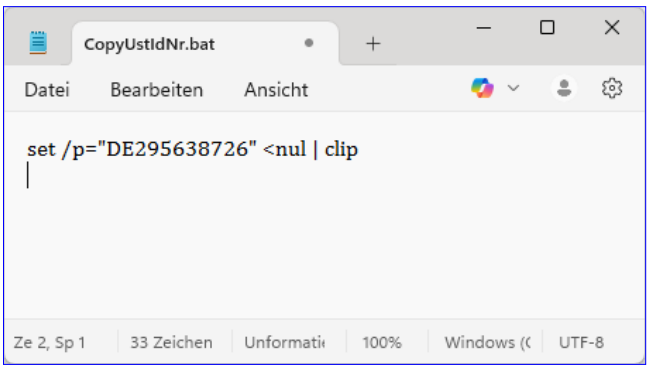

Bild 3: **.bat**-Datei mit dem Befehl zum Kopieren eines Textes in die Zwischenablage

Windows-Kontextmenü mit TreeView verwalten

Im Artikel »UstIdNr, IBAN und Co. per Kontextmenü« (www.vbentwickler.de/480) haben wir gezeigt, wie wir die Windows-Kontextmenüs erweitern können. In diesem Fall haben wir Daten, die man gegebenenfalls nicht alle im Kopf hat, aber regelmäßig benötigt, einfach über ein Kontextmenü in die Zwischenablage einfügen kann, um diese dann an der gewünschten Stelle beispielsweise in einem Bestellformular einträgt. Um das Hinzufügen und Aktualisieren dieser Einträge weiter zu vereinfachen und die zahlreichen Handgriffe zu ersparen, zeigen wir im vorliegenden Artikel eine Access-Lösung, mit der wir die Kontextmenü-Befehle mit einem einfachen TreeView verwalten und erweitern können.

Dazu wollen wir die Struktur der benutzerdefinierten Kontextmenü-Einträge im Windows-Menü in einem **TreeView**-Steuerelement abbilden.

Darin können wir Ordner, Unterordner und die enthaltenen Befehle verwalten. Nachdem wir diese Elemente angelegt haben, wollen wir diese per Mausklick direkt in die Registry schreiben können.

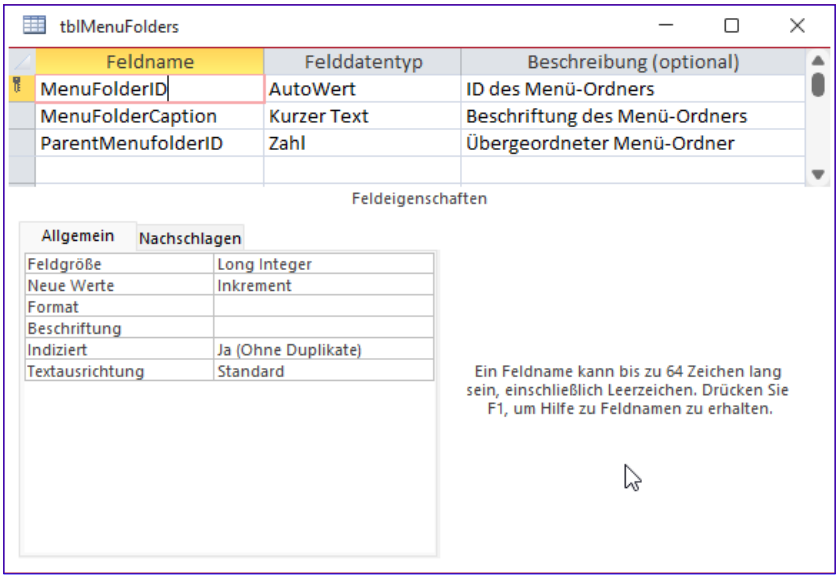

Bild 1: Tabelle zum Speichern der Menü-Ordner

Datenmodell für die Anwendung

Wir wollen die Daten, wir wie sie zum Windows-Kontextmenü hinzufügen wollen, strukturiert in Tabellen ablegen, damit wir sie dynamisch ergänzen und bearbeiten können. Dazu benötigen wir zwei Tabellen:

- **tblMenuFolders**: Enthält die Ordner und die Unterordner.
- **tblMenuEntries**: Enthält die Einträge zu den jeweiligen Ordnern und Unterordnern.

Die Tabelle **tblMenuFolders** gestalten wir wie in Bild 1. Das Feld **ParentMenuFolderID** verknüpfen wir mit dem Feld **MenuFolderID** der gleichen Tabelle, damit wir die Zuordnung von Unterordnern zu Ordnern definieren können.

Die zweite Tabelle **tblMenuEntries** nimmt die eigentlichen Einträge auf (siehe Bild 2). Sie nimmt die ID, die Beschriftung des Menü-Eintrags und den Text auf, der beim Betätigen des Menü-Eintrags in die Zwischenablage kopiert werden soll.

Außerdem wird über das Feld **ParentMenuFolderID** die Zuordnung zu einem Ordner angegeben.

Im Datenmodell sehen wir die anzulegenden Beziehungen (siehe Bild 3). Die Tabelle **tblMenuEntries** ist über das Feld **ParentMenuFolderID** mit der Tabelle **tblMenuFolders** verknüpft. Die Tabelle **tblMenuFolders** ist außerdem über das Feld **ParentMenuFolderID** mit dem Feld **MenuFolderID** der gleichen Tabelle verknüpft. Für beide Beziehungen haben wir referenzielle Integrität mit Löschweitergabe definiert, damit beim Löschen eines übergeordneten Menüeintrags auch alle untergeordneten Elemente gelöscht werden.

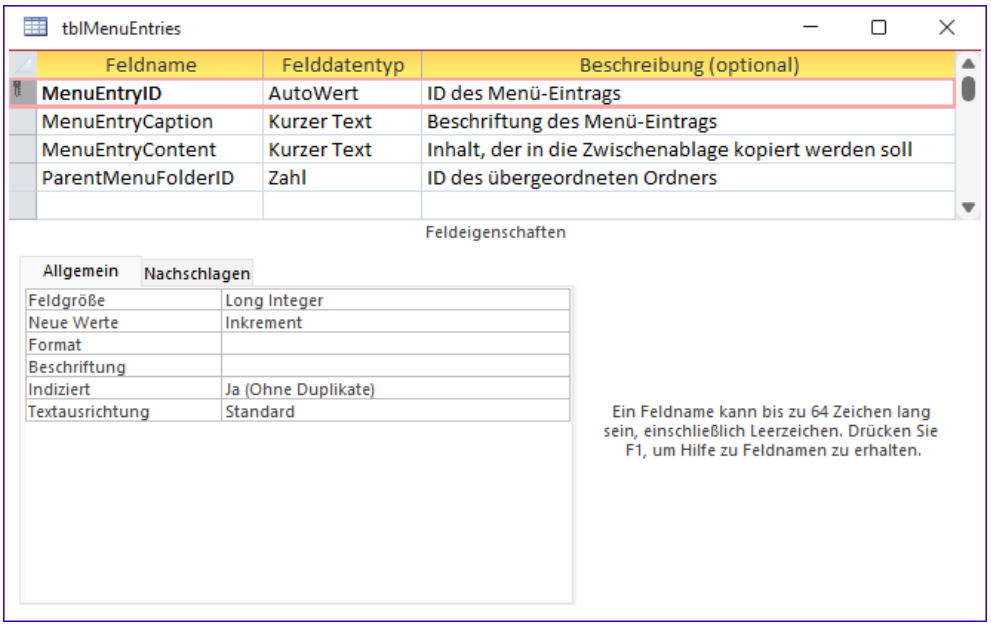

Bild 2: Tabelle zum Speichern der Menü-Einträge

Aufbau des TreeView-Steuerelements

Wir fügen einem neuen Formular namens **frmWindowsKontextmenue** ein **TreeView**-Steuerelement und ein **ImageList**-Steuerelement hinzu. Diese nennen wir **ctlTreeView** und **ctlImageList** (siehe Bild 4).

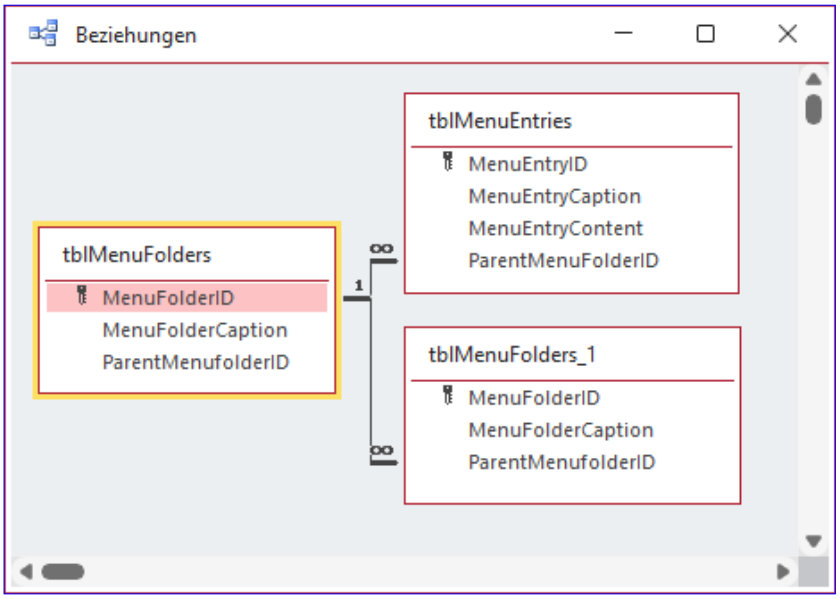

Bild 3: Datenmodell der Anwendung

Damit können wir nun den Code zusammenstellen, der zum Anzeigen der Einträge aus den Tabellen **tblMenuFolders** und **tblMenuEntries** verwendet wird.

```
Private objTreeView As MSComctlLib.TreeView
```

Die Prozedur soll beim Laden des Formulars ausgelöst werden und sieht wie in Listing 1 aus. Hier deklarieren wie zunächst eine modulweit erreichbare Variable für das **TreeView**-Steuerelement mit dem Schlüsselwort **WithEvents**:

Die Prozedur **Form_Load** füllt zunächst das **ImageList**-Steuerelement mit allen Bildern, die wir in der Tabelle **MSysResources** finden. Die benötigten Icons haben wir zuvor hinzugefügt, in dem wir ein Formular in der Entwurfsansicht geöffnet und die gewünschten

Icons mit dem Befehl **Bild einfügen** über das Ribbon hinzugefügt haben. Diese Bilder werden automatisch in der Tabelle **MSysResources** gespeichert (siehe Bild 5).

Diese Bilder müssen wir nun zum **ImageList**-Steuerelement hinzufügen, damit wir sie für die **TreeView**-Einträge verwenden können. Das erledigen die ersten Anweisungen der Prozedur **Form_Load**. Hier referenzieren wir die Eigenschaft **Object** des **ImageList**-Steuerelements mit der Variablen **objImageList**. Dann stellen wir Höhe und Breite der Bilder jeweils auf **16** ein.

Schließlich öffnen wir ein Recordset auf Basis der Tabelle **MSysResources** für alle Einträge, deren Typ **img** lautet. Diese durchlaufen wir ein einer **Do While**-Schleife und rufen jeweils die Prozedur **amvAddIconToImageList** auf. Diese fügt die einzelnen Bilder aus der Tabelle in das **ImageList**-Steuerelement ein. Die Prozedur **amvAddIconToImageList** und ihre Hilfsfunktionen wollen wir aus Platzgründen an dieser Stelle nicht erläutern.

Indem wir die Bilder dynamisch einlesen, können wir diese auch einmal ersetzen, ohne dass wir umständlich am Entwurf des Steuerelements operieren müssen.

Nun referenzieren wir das **TreeView**-Steuerelement mit der Variablen **objTreeView**, die wir bereits weiter oben deklariert haben.

Wir leeren eventuell noch vorhandene Nodes mit der **Clear**-Methode und weisen diesem Steuerelement das **ImageList**-Steuerelement als Quelle für die Icons zu.

Dann erstellen wir ein neues Recordset auf Basis der Einträge der Tabelle **tblMenuFolders**, aber nur für die Einträge, die keinen übergeordneten Eintrag haben. Diese durchlaufen wir in einer weiteren **Do While**-Schleife.

In dieser Schleife stellen wir einen eindeutigen Schlüssel für jedes **Node**-Element im **TreeView**-Steuerelement zusammen, das für Ordner aus dem Buchstaben **f** und der ID des jeweiligen Datensatzes besteht (zum Beispiel **f1**).

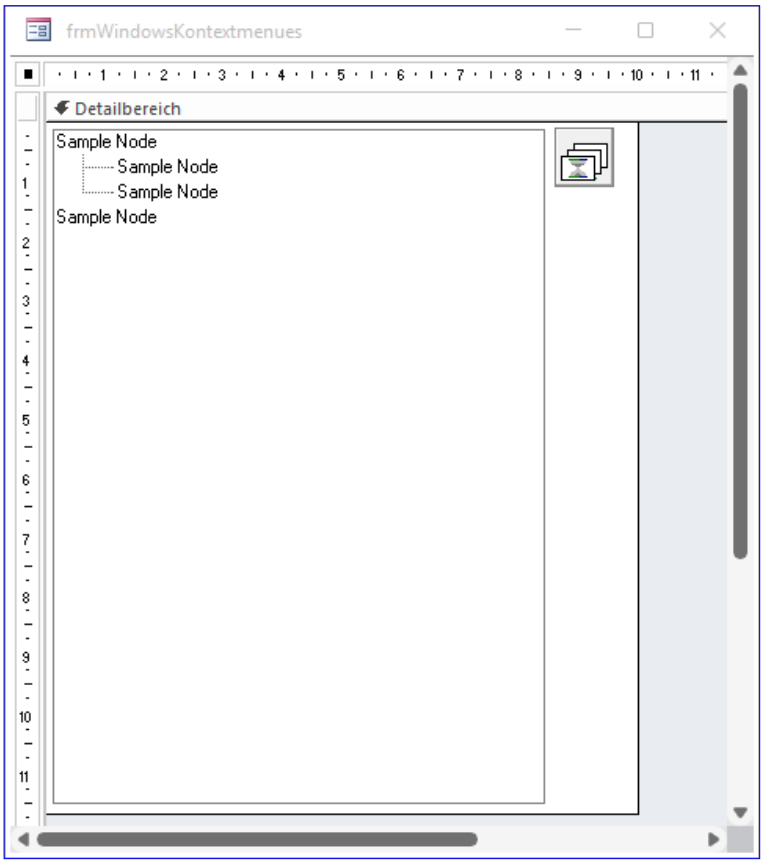

Bild 4: Formular mit **TreeView**- und **ImageList**-Steuerelement

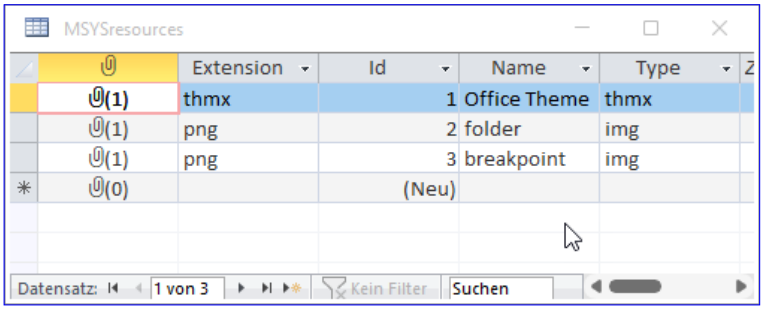

Bild 5: Bilder für die **TreeView**-Einträge aus der Tabelle **MSysResources**

```
Private objTreeView As MSComctlLib.TreeView

Private Sub Form_Load()
    Dim db As dao.Database
    Dim rstImages As dao.Recordset
    Dim rstFolders As dao.Recordset
    Dim rstEntries As dao.Recordset
    Dim strKey As String
    Dim objNode As MSComctlLib.Node
    Dim objImageList As MSComctlLib.ImageList

    Set db = CurrentDb

    Set objImageList = Me.ctlImageList.Object
    objImageList.ImageHeight = 16
    objImageList.ImageWidth = 16
    Set rstImages = dbc.OpenRecordset("SELECT * FROM MSysResources WHERE type = 'img'", dbOpenDynaset)
    Do While Not rstImages.EOF
        amvAddIconToImageListFromResources objImageList, rstImages!ID, rstImages!Name
        rstImages.MoveNext
    Loop


    Set objTreeView = Me.ctlTreeview.Object
    objTreeView.Nodes.Clear
    Set objTreeView.ImageList = objImageList

    Set rstFolders = db.OpenRecordset("SELECT * FROM tblMenuFolders WHERE ParentMenuFolderID IS NULL", dbOpenDynaset)
    Do While Not rstFolders.EOF
        strKey = "f" & rstFolders!MenuFolderID
        Set objNode = objTreeView.Nodes.Add(, , strKey, rstFolders!MenuFolderCaption, "folder")
        objNode.Expanded = True
        Call AddSubfolders(db, rstFolders!MenuFolderID, strKey)
        Call AddEntries(db, rstFolders!MenuFolderID, strKey)
        rstFolders.MoveNext
    Loop


    Set rstEntries = db.OpenRecordset("SELECT * FROM tblMenuEntries WHERE ParentMenuFolderID IS NULL", dbOpenDynaset)
    Do While Not rstEntries.EOF
        strKey = "e" & rstEntries!MenuEntryID
        objTreeView.Nodes.Add , , strKey, rstEntries!MenuEntryCaption, "breakpoint"
        rstEntries.MoveNext
    Loop
End Sub
```

Listing 1: Prozedur zum Initialisieren des **TreeView**- und des **ImageList**-Steuerelements

```
Private Sub AddSubfolders(db As dao.Database, lngParentFolderID As Long, strKey As String)
    Dim rstSubfolders As dao.Recordset
    Dim strSubkey As String
    Dim objNode As MSComctlLib.Node

    Set rstSubfolders = db.OpenRecordset("SELECT * FROM tblMenuFolders WHERE ParentMenuFolderID = " _
        & lngParentFolderID, dbOpenDynaset)
    Do While Not rstSubfolders.EOF
        strSubkey = "f" & rstSubfolders!MenuFolderID
        Set objNode = objTreeView.Nodes.Add(strKey, tvwChild, strSubkey, rstSubfolders!MenuFolderCaption, "folder")
        objNode.Expanded = True
        Call AddSubfolders(db, rstSubfolders!MenuFolderID, strSubkey)
        Call AddEntries(db, rstSubfolders!MenuFolderID, strSubkey)
        rstSubfolders.MoveNext
    Loop
End Sub
```

Listing 2: Prozedur zum Einfügen der untergeordneten Ordner

Danach fügen wir das neue Element mit der **Add**-Methode der **Nodes**-Auflistung des **TreeView**-Steuerelements hinzu.

Dabei geben wir den Key an, den Namen aus dem Feld **MenuFolderCaption** und das zu verwendende Bild (**folder**). Außerdem stellen wir die Eigenschaft **Expanded** für das neue Element auf **True** ein, damit untergeordnete Einträge direkt angezeigt werden.

Schließlich rufen wir zwei weitere Prozeduren auf. **AddSubfolders** fügt die untergeordneten Ordner hinzu und **AddEntries** die eigentlichen Einträge.

Beiden übergeben wir einen Verweis auf das **Database**-Objekt, die ID des übergeordneten Ordners und den Key, den wir für das übergeordnete Element verwendet haben.

Da wir auch Elemente in der obersten Ebene anzeigen wollen, fügen wir noch Code hinzu, der die Einträge der Tabelle **tblMenuEntries** durchläuft, die im Feld **ParentMenuFolderID** den Wert **NULL** enthalten. Für diese legen wir als Key den Buchstaben **e** (für Entry) sowie den anzuzeigenden Text fest.

Unterordner zum TreeView hinzufügen

Die Prozedur **AddSubfolders** (siehe Listing 2) soll die Unterordner hinzufügen, die dem mit dem Parameter **lngParentFolderID** entsprechenden Datensatz der Tabelle **tblMenuFolders** untergeordnete sind.

Dazu öffnen wir ein Recordset, das genau diese Datensätze aus der Tabelle **tblMenuFolders** liefert. In der folgenden **Do While**-Schleife durchlaufen wir diese Datensätze und stellen zunächst in der Variablen **strSubkey** den Key für die zu erstellenden Untereinträge zusammen. Diese bestehen wiederum aus dem Buchstaben f und dem Primärschlüsselwert des neuen Elements, also beispielsweise **f2**.

Dann erstellen wir ein neues **Node**-Objekt unterhalb des übergeordneten Objekts. Das übergeordnete Element referenzieren wir mit dem Key dieses Elements und geben mit dem zweiten Parameter **tvwChild** an, dass das neue Element unterhalb dieses Elements angeordnet werden soll. Der dritte Parameter nimmt den neuen Key auf und der vierte den anzuzeigenden Text. Mit dem fünften Parameter übergeben wir wieder das Icon **folder**. Auch für dieses Element stellen wir wieder die Eigenschaft **Expanded** auf **True** ein.

```
Private Sub AddEntries(db As dao.Database, lngParentFolderID As Long, strKey As String)
    Dim rstEntries As dao.Recordset
    Dim strSubkey As String
    Dim objNode As MSComctlLib.Node

    Set rstEntries = db.OpenRecordset("SELECT * FROM tblMenuEntries WHERE ParentMenuFolderID = " _
        & lngParentFolderID, dbOpenDynaset)
    Do While Not rstEntries.EOF
        strSubkey = "e" & rstEntries!MenuEntryID
        Set objNode = objTreeView.Nodes.Add(strKey, tvwChild, strSubkey, rstEntries!MenuEntryCaption, "breakpoint")
        rstEntries.MoveNext
    Loop
End Sub
```

Listing 3: Prozedur zum Einfügen der untergeordneten Einträge

Danach rufen wir erneut die Prozedur **AddSubfolders** auf, falls noch weitere Unterordner vorhanden sind.

Außerdem verwenden wir die Prozedur **AddEntries**, um die enthaltenen Einträge hinzuzufügen.

Elemente zum TreeView hinzufügen

Die Prozedur **AddEntries** (siehe Listing 3) fügt die eigentlichen Einträge hinzu. Sie erwartet die gleichen Einträge wie **AddSubfolders**. Hier definieren wir ein Recordset, dass alle Datensätze der Tabelle **tblMenuEntries** enthält, die dem mit **lngParentFolderID** übergebenen Ordner zugeordnet sind.

Für jeden Datensatz dieses Recordsets fügen wir wieder ein Element zum **TreeView**-Steuerelement hinzu, wobei wir ähnlich wie in der zuvor beschriebenen Prozedur vorgehen.

Testen des TreeView-Steuerelements

Nachdem wir manuell einige Einträge zu den Tabellen hinzufügt haben, werden diese beim Öffnen des Formulars **frmWindowsKontextmenue** wie in Bild 6 dargestellt.

Kontextmenüs zum Hinzufügen und Löschen von Einträgen anlegen

Wir wollen die Tabelleneinträge nicht immer direkt in die Tabellen eingeben, sondern diese über ein Kontextmenü hinzufügen. Der Hauptknoten **Daten** sollte beispielsweise die Einträge **Neuer Ordner** und **Neues Element** anzeigen. Für die Unterordner sollen die Einträge **Neuer Ordner**, **Neues Element** und Ordner löschen erscheinen. Schließlich sollen die Elemente im Kontextmenü den Befehl **Element löschen** anbieten.

Die Kontextmenüs zeigen wir an, wenn das Ereignis **MouseDown** des **TreeView**-Steuerelements ausgelöst wird (siehe Listing 4). Hier prüfen wir, ob die rechte Maustaste zum Anklicken verwendet wurde.

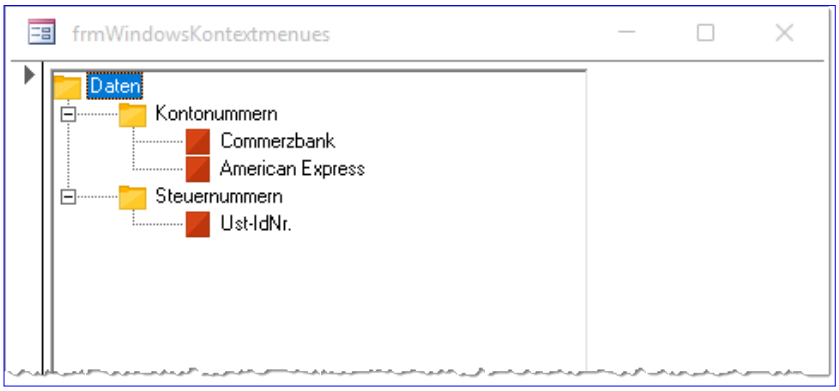

Bild 6: **TreeView**-Steuerelement mit einigen Einträgen

```
Private Sub ctlTreeview_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Long, ByVal y As Long)
    Dim objNode As MSComctlLib.Node, lngID As Long, strChar As String, cbr As CommandBar
    Dim cbbNewFolder As CommandBarButton, cbbNewEntry As CommandBarButton, cbbDeleteFolder As CommandBarButton
    Dim cbbDeleteEntry As CommandBarButton

    If Button = 2 Then
        Set objNode = objTreeView.HitTest(x, y)
        If Not objNode Is Nothing Then
            strChar = Left(objNode.Key, 1)
            lngID = Mid(objNode.Key, 2)
            Select Case strChar
                Case "f"
                    On Error Resume Next
                    CommandBars("Folders").Delete
                    On Error GoTo 0
                    Set cbr = CommandBars.Add("Folders", msoBarPopup, , True)
                    Set cbbNewFolder = cbr.Controls.Add(msoControlButton, , , , True)
                    cbbNewFolder.Caption = "Ordner hinzufügen"
                    cbbNewFolder.OnAction = "=AddFolder(" & lngID & ")"
                    Set cbbDeleteFolder = cbr.Controls.Add(msoControlButton, , , , True)
                    cbbDeleteFolder.Caption = "Ordner löschen"
                    cbbDeleteFolder.OnAction = "=DeleteFolder(" & lngID & ")"
                    Set cbbNewEntry = cbr.Controls.Add(msoControlButton, , , , True)
                    cbbNewEntry.Caption = "Element hinzufügen"
                    cbbNewEntry.OnAction = "=AddEntry(" & lngID & ")"
                    cbr.ShowPopup
                Case "e"
                    On Error Resume Next
                    CommandBars("Entries").Delete
                    On Error GoTo 0
                    Set cbr = CommandBars.Add("Entries", msoBarPopup, , True)
                    Set cbbDeleteEntry = cbr.Controls.Add(msoControlButton, , , , True)
                    cbbDeleteEntry.Caption = "Element löschen"
                    cbbDeleteEntry.OnAction = "=DeleteElement(" & lngID & ")"
                    cbr.ShowPopup
            End Select
        Else
            On Error Resume Next
            CommandBars("Folders").Delete
            On Error GoTo 0
            Set cbr = CommandBars.Add("Folders", msoBarPopup, , True)
            Set cbbNewFolder = cbr.Controls.Add(msoControlButton, , , , True)
            cbbNewFolder.Caption = "Ordner hinzufügen"
            cbbNewFolder.OnAction = "=AddFolder(0)"
            Set cbbNewEntry = cbr.Controls.Add(msoControlButton, , , , True)
            cbbNewEntry.Caption = "Element hinzufügen"
            cbbNewEntry.OnAction = "=AddEntry(0)"
            cbr.ShowPopup
        End If
    End If
End Sub
```

Listing 4: Prozedur zum Anzeigen der Kontextmenüs