HAUFE.
Ausgabe 02/2025

ACCESS
IM UNTERNEHMEN

LASTSCHRIFTEINZUG PER VBA

Automatisieren Sie Ihre Geldeingänge mit VBA und der günstigen DDBAC-Lizenz (ab Seite 28).

In diesem Heft:

SQL SERVER-TABELLEN EINFACH VERKNÜPFEN

Verwenden Sie unseren Assistenten, um SQL Server-Tabellen blitzschnell einzubinden.

SEITE 58

BILDER IM SQL SERVER SPEICHERN

Wie speichern wir Daten, die in Access im Anlagefeld liegen, im SQL Server?

SEITE 22

NAVIGATIONSBEREICH PROGRAMMIEREN

Erfahren Sie, wie Sie mehr aus dem Navigationsbereich herausholen.

SEITE 2

Lassen wir das Geld einfach kommen!

Viele unserer Leser dürften selbständige Softwareentwickler sein. Wie sieht es aus – stellen Sie noch Rechnungen überlassen dem Kunden die Überweisung – auf die Gefahr hin, dass sich dies auch einmal verzögert? Oder arbeiten Sie bereits mit alternativen Möglichkeiten wie beispielsweise der Lastschrift? Das Anfordern von Einzugsermächtigungen bietet sich natürlich besonders an, wenn wir regelmäßige Zahlungen vom Kunden erwarten. Das ist zum Beispiel bei Wartungsverträgen der Fall oder bei Abonnements. Hier sind Lastschriften eine gute Alternative. Wir zeigen in dieser Ausgabe, wie Sie Lastschriften per VBA und mit der DDBAC-Bibliothek einrichten können.

Wer ein Magazin wie dieses vertreibt, kann Rechnungen versenden und darauf warten, dass diese bezahlt werden. Erfolgt die Zahlung nicht rechtzeitig, sind Erinnerungsschreiben und Mahnungen fällig – bis hin zum Inkasso. Wenn man sich vom Kunden eine Einzugsermächtigung holt, ist man zwar nicht automatisch vor Zahlungsausfällen gefeit. Allerdings kann man zumindest selbst festlegen, zu welchem Zeitpunkt der gewünschte Betrag abgebucht wird. Hier können dann noch folgende Probleme auftreten: Das Konto des Kunden ist nicht gedeckt oder der Kunde widerspricht der Lastschrift. Die Anzahl solcher Vorfälle dürfte sich jedoch in Grenzen halten.

Zum Einrichten einer Lastschrift brauchen wir nur ein Lastschriftmandat sowie einige weitere Vorbereitungen – zum Beispiel die Installation der DDBAC-Bibliothek. Eine Lizenz dieser Bibliothek kostet nur wenige Euro im Monat, und damit können wir neben dem Einrichten von Lastschriften auch noch andere Vorgänge wie das Einlesen von Kontoständen und Umsätzen oder Überweisungen durchführen. Wie Sie selbst Lastschriften mit VBA einrichten können, lesen Sie im Beitrag **Lastschriften mit DDBAC und VBA** ab Seite 36.

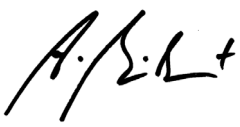
Außerdem zeigen wir in dieser Ausgabe, wie wir in unserer Access-Datenbank sehr einfach Verknüpfungen zu SQL Server-Tabellen herstellen können. Dazu bauen wir ein eigenes Access-Add-In, das wir von jeder Datenbankanwendung aufrufen und zum Einbinden von SQL-Server-Tabellen nutzen können. Damit gelingt diese Aufgabe viel schneller als mit den eingebauten Tools von Microsoft für diesen Zweck. Mehr dazu unter dem Titel **Tabellenverknüpfungsassistent** ab Seite 58.

Den Navigationsbereich nutzen wir alle mehr oder wenige. Aber nicht viele verwenden tatsächlich alle Möglichkeiten, die dieser bietet. Daher beschreiben wir im Beitrag **Benutzerdefinierter Navigationsbereich im Griff** ab Seite 2, wie wir diesen beispielsweise per VBA programmieren können.

Bei der Migration einer Access-Datenbank zum SQL Server stellt sich die Frage, wie man die in Anlagefeldern enthaltenen Bilder im SQL Server speichern soll. Eine Methode zeigen wir im Beitrag **Bilder im SQL Server mit varbinary(max)** ab Seite 22.

Und schließlich gehen wir in **Fehler bei verknüpften ODBC-Tabellen in Recordsets** (ab Seite 17) darauf ein, welche Fehler wir nach der Migration zum SQL Server im DAO-Code noch beheben müssen, zeigen unter **VBA-InputBox richtig nutzen in Access** ab Seite 20, welche Feinheiten in der **InputBox**-Funktion stecken und liefern einen schnellen Tipp für den **Dateidownload per API** (ab Seite 74).

Und nun viel Spaß beim Lesen!



Ihr André Minhorst

Benutzerdefinierter Navigationsbereich im Griff

Den Navigationsbereich von Access kennt jeder Access-Entwickler. Damit öffnen und verwalten wir die Access-Objekte beim Entwickeln einer Datenbankanwendung. Aber der Navigationsbereich bietet noch mehr Möglichkeiten, als einfach nur die Tabellen, Abfragen, Formulare, Berichte, Makros und Module sortiert nach Kategorien anzuzeigen und eine einfache Suchfunktion dafür bereitzustellen. Wir können damit auch noch benutzerdefinierte Kategorien und Gruppen erstellen, in denen wir die Objekte individuell strukturieren können. Während die Verwendung dieser Funktionen eher trivial ist, weshalb wir nur einen kurzen Blick darauf werfen, interessiert uns aus Entwicklersicht, wo diese Kategorien und Gruppen sowie die enthaltenen Elemente gespeichert werden. In diesem Beitrag werden die notwendigen Schritte beschrieben, um diese Einstellungen und Elemente bei der Übertragung der Anwendung in eine neue Datenbankdatei zu berücksichtigen.

Benutzerdefinierte Anpassungen im Navigationsbereich

Wenn wir den Navigationsbereich nutzen wollen, um die Datenbankobjekte anders zu strukturieren, können wir die von Access bereitgestellten Methoden dazu nutzen.

Dazu klicken wir mit der rechten Maustaste auf die Titelleiste des Navigationsbereichs und wählen aus dem Kontextmenü den Befehl **Navigationsoptionen...** aus.

Dies öffnet den Dialog aus Bild 1.

Hier sehen wir zwei Listenfelder, von denen das linke die Kategorien anzeigt und das rechte die in der aktuell ausgewählten Kategorie enthaltenen Gruppen.

Die Standardkategorie heißt **Objekttyp** und sie zeigt die Access-Objekte gruppiert nach dem Objekttyp an, also nach Tabellen, Abfragen, Formularen, Berichten, Makros und Modulen. Die zweite eingebaute Kategorie heißt **Tabellen und damit verbundene Sichten**. Sie sorgt dafür,

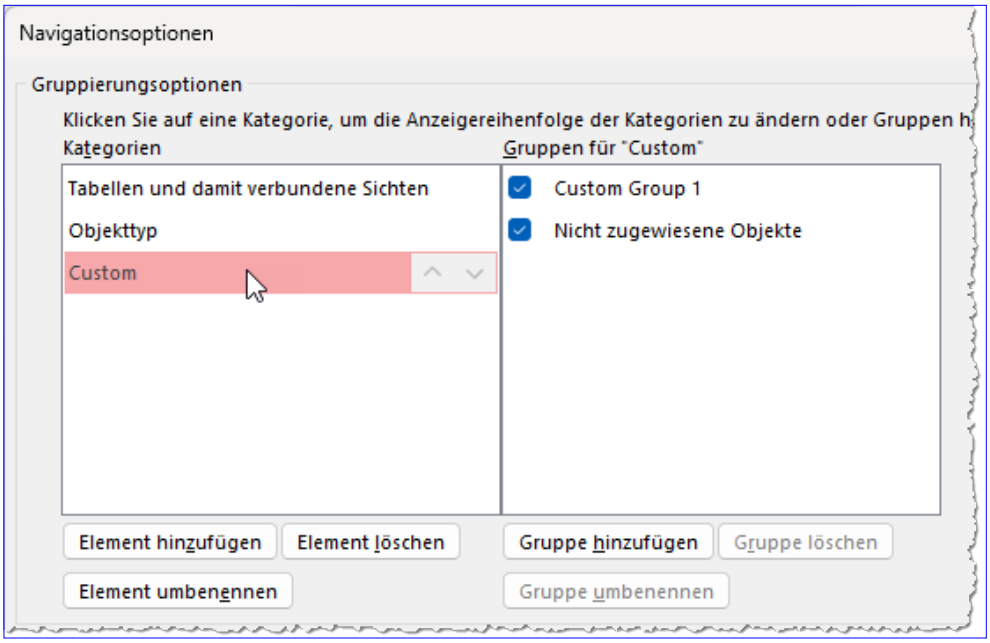


Bild 1: Einstellungen für den benutzerdefinierten Navigationsbereich

dass die Tabellen als Gruppenköpfe angezeigt werden und darunter alle Objekte, die in irgendeiner Weise auf die jeweilige Tabelle zugreifen. Dies sieht im Navigationsbereich wie in Bild 2 aus. Um zu dieser Kategorie zu wechseln, klicken wir auf die Titelleiste des Navigationsbereichs und wählen den Eintrag **Tabellen und damit verbundene Sichten** aus.

Uns interessiert jedoch viel mehr die dritte Kategorie namens **Custom**, die standardmäßig in jeder neuen Access-Datenbank angelegt wird und die wir auch oben in der Abbildung bereits gesehen haben. Für diese Kategorie sehen wir auf der rechten Seite zwei Gruppen namens **Custom Group 1** und **Nicht zugewiesene Objekte**.

Diese aktivieren wir ebenfalls durch einen Klick auf die Titelleiste des Navigationsbereichs und anschließende Auswahl von **Custom**.

Danach erhalten wir die Ansicht aus Bild 3 im Navigationsbereich. Hier sehen wir keine Elemente in der Gruppe **Custom Group 1** und alle Elemente in der Gruppe **Nicht zugewiesene Objekte**.

Anpassen von Kategorien und Gruppen

Im Dialog **Navigationsoptionen** können wir nun zum Erstellen einer eigenen Kategorie entweder die vorhandene Kategorie **Custom** umbenennen oder diese löschen und neu erstellen. Dazu verwenden wir die Schaltflächen unterhalb des linken Listenfeldes. Wir benennen die Kategorie in diesem Fall in **Objekte nach Bereich** um, weil wir verschiedene Gruppen anlegen wollen, in denen wir die Datenbankobjekte nach verschiedenen Kriterien gruppieren wollen.

Gruppe ausblenden

In der rechten Liste sehen wir nach wie vor die beiden Gruppen **Custom Group 1** und **Nicht zugewiesene Objekte**. Erstere können wir löschen oder umbenennen, die Gruppe **Nicht zugewiesene Objekte** muss jedoch erhalten bleiben. Wenn wir diese Gruppe später nicht mehr

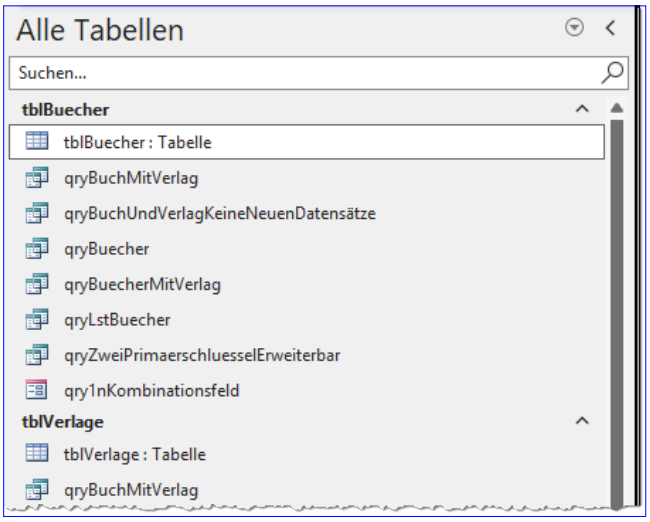


Bild 2: Kategorisierung nach Tabellen und ihren abhängigen Objekten

sehen wollen, können wir diese allerdings auch ausblenden. Dazu klicken wir mit der rechten Maustaste auf den Gruppenkopf und wählen den Kontextmenübefehl **Ausblenden** aus (siehe Bild 4).

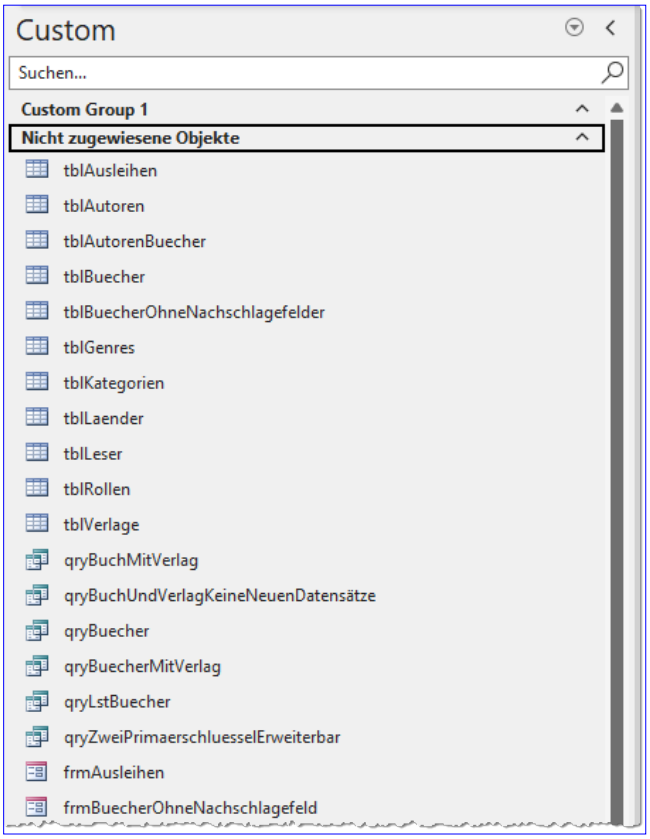


Bild 3: Auswahl der benutzerdefinierten Kategorie

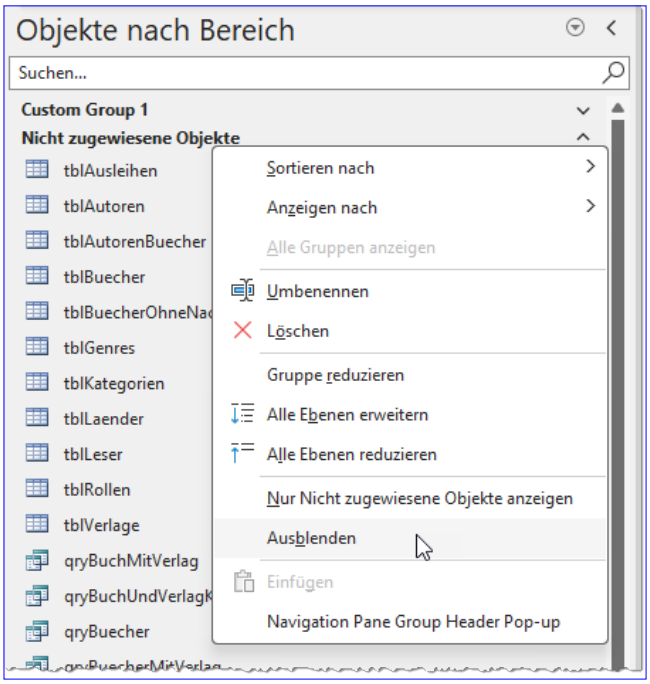


Bild 4: Ausblenden einer Gruppe

Ausgeblendete Gruppe wieder einblenden

Nun sehen wir die ausgeblendete Gruppe nicht mehr, was problematisch wird, wenn wie einmal ein neues Datenbankobjekt hinzufügen und dieses einer der Gruppen zuordnen wollen. Also müssen wir die Gruppe **Nicht zugewiesene Objekte** dazu kurzzeitig wieder einblenden.

Dazu öffnen wir die Navigationsoptionen wie oben beschrieben und aktivieren die Option **Ausgeblendete Objekte** anzeigen (siehe Bild 5).

Danach sehen wir die aktuell ausgeblendeten Gruppen und Elemente ausgegraut wieder im Navigationsbereich. Wir können die gewünschten Elemente nun aus diesem Bereich in die entsprechende Gruppe verschieben und den Bereich wieder ausblenden.

Objekte zu den Gruppen zuweisen

Hier stellt sich noch die Frage, wie wir überhaupt Datenbankobjekte zu einer der Gruppen zuordnen wollen. Im Dialog **Navigationsoptionen** ist dies jedenfalls nicht möglich. Die Lösung ist aber einfach: Wir legen zunächst wie in Bild 6 die benötigten Gruppen für unsere benutzerdefinierte Kategorie an.

Schließen wir den Dialog **Navigationsoptionen** dann wieder und wählen im Navigationsbereich die benutzerdefinierte Kategorie namens **Objekte nach Bereich** aus, sehen wir die Ansicht aus Bild 7. Hier können wir nun die benötigten Objekte in die einzelnen Kategorien hineinziehen.

Das Ergebnis sehen wir schließlich in Bild 8. Hier haben wir zunächst einmal einfach alle Elemente nach dem Namen der jeweiligen Gruppe gefiltert und dann alle Objekte,

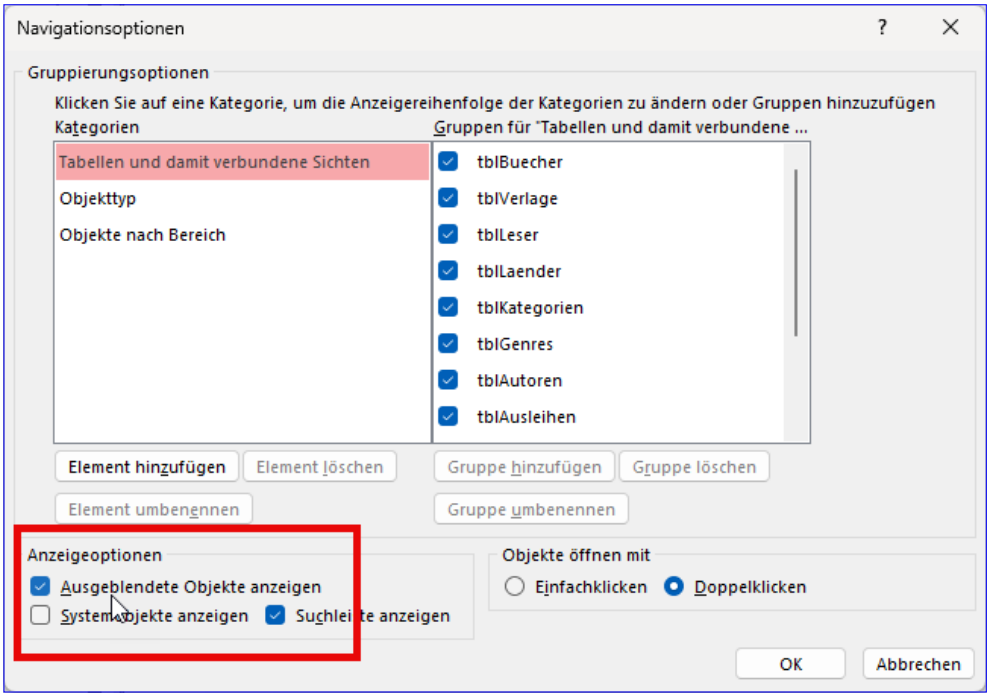


Bild 5: Einblenden ausgeblendeter Elemente

die zu der Gruppe passen, in diese hineingezogen. Das haben wir für alle Gruppen wiederholt.

Bei manchen Objekten tauchen im Name aber gleich mehrere Gruppenbezeichnungen auf. Auch das ist kein Problem, denn innerhalb der benutzerdefinierten Gruppen werden, wie die Icons bereits zeigen, ohnehin nur Verknüpfungen zu den jeweiligen Datenbankobjekten gespeichert.

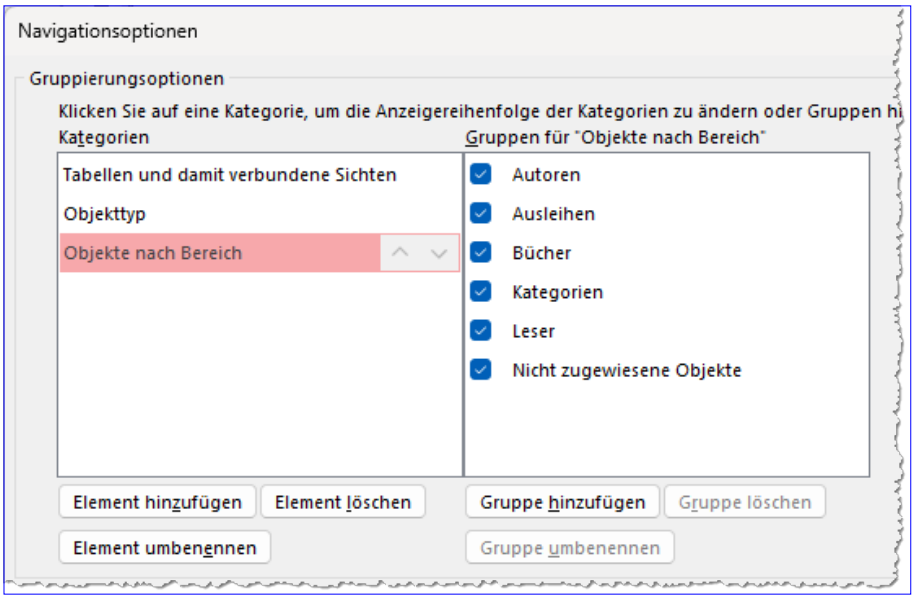


Bild 6: Einige benutzerdefinierte Gruppen

Das heißt, dass wir beispielsweise die Verknüpfung auf die Tabelle **tblAutorenBuecher** nicht nur in der Kategorie **Autoren** unterbringen, sondern auch in der Kategorie **Bücher**.

Damit haben wir uns eine übersichtlichere Darstellung der zu programmierenden Objekte erschaffen. Nun wollen wir uns darum kümmern, wo sich die Informationen zu diesen Kategorien, Gruppen und den zugewiesenen Elementen befinden und wie wir diese auslesen, manipulieren und gegebenenfalls mit den ganzen Objekten in eine andere

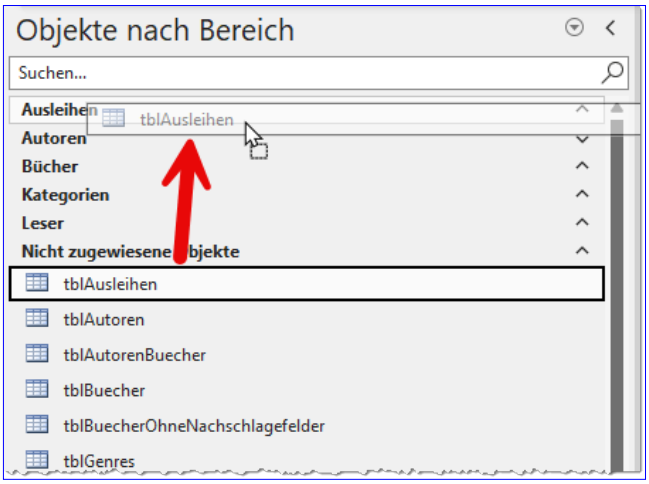


Bild 7: Zuweisen von Datenbankobjekten zu den benutzerdefinierten Gruppen

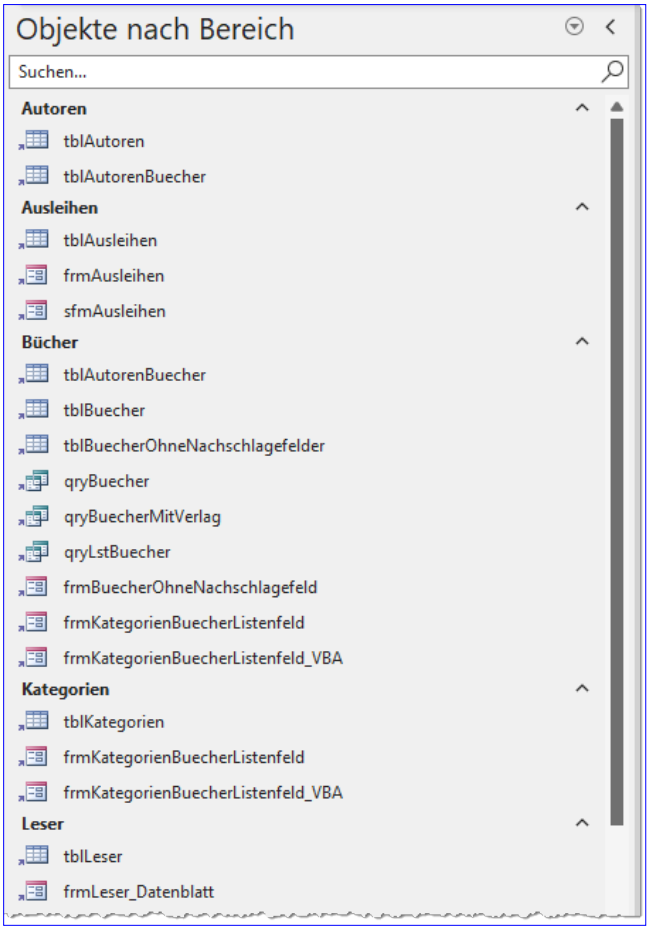


Bild 8: Die benutzerdefinierten Gruppen mit den zugewiesenen Datenbankobjekten

Datenbank übertragen können. Letzteres kann beispielsweise einmal nötig sein, wenn die Datenbank korrupt ist.

Informationen zum Navigationsbereich in den Systemtabellen

Bei den Daten rund um die Kategorien, Gruppen und die zugeordneten Datenbankobjekte handelt es sich um strukturierte Informationen, die in Access üblicherweise in Tabellen gespeichert werden. Nur sehen wir aktuelle keine Tabelle, die dafür in Frage kommt.

Das ändert sich allerdings, wenn wir die Navigationsoptionen erneut öffnen und die Optionen **Ausgeblendete Objekte anzeigen** und **Systemobjekte anzeigen** beide aktivieren.

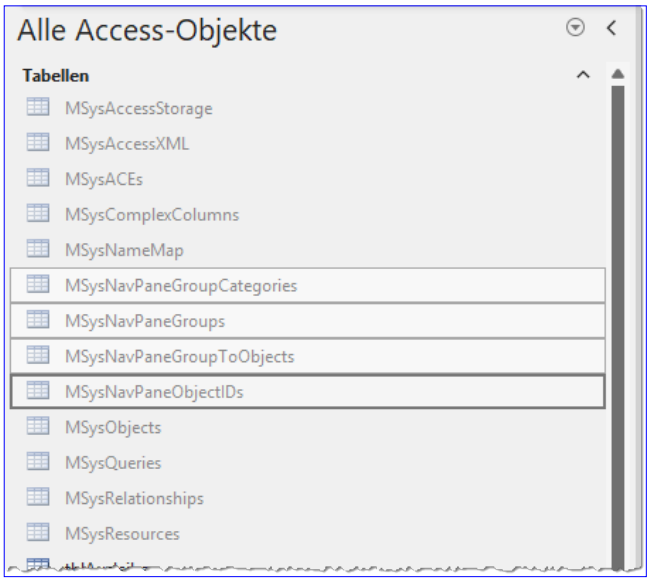


Bild 9: Die Tabellen zum Verwalten der Daten zum Navigationsbereich

Im Navigationsbereich erscheinen nun automatisch die Systemtabellen und die ausgeblendeten Tabellen – teilweise sind die Systemtabellen auch noch als ausgeblendet markiert, sodass das Aktivieren beider Optionen zum Einblenden durchaus angebracht war (siehe Bild 9).

Hier sehen wir auch vier Tabellen, deren Namen zu unserer Suche passen:

- **MSysNavPaneGroupCategories**
- **MSysNavPaneGroups**
- **MSysNavPaneGroupToObjects**
- **MSysNavPaneObjectIDs**

Tabelle zum Speichern der Objekte

In der Tabelle **MSysNavPaneObjectIDs** werden die Namen aller Objekte der Datenbank gespeichert – samt einem Zahlenwert für den Objekttyp, wie er auch in der Tabelle **MSysObjects** zu finden ist, und einem eindeutigen Wert im Feld **Id** (siehe Bild 10).

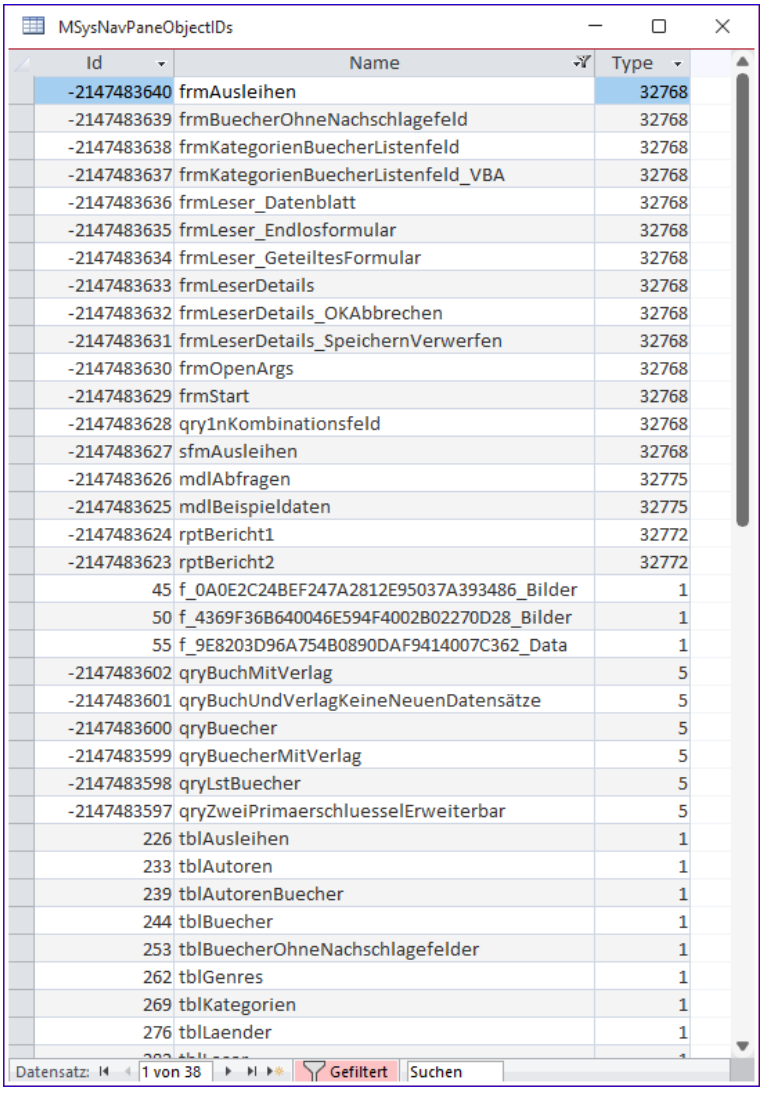


Bild 10: Die Tabelle der **NavPane...**-Objekte

Fehler bei verknüpften ODBC-Tabellen in Recordsets

Wer eine Migration seiner Access-Datenbank zum SQL Server durchgeführt hat und dabei beispielsweise den SQL Server Migration Assistant genutzt hat, hofft vielleicht, damit schon am Ziel zu sein. Tatsächlich kann das in ganz wenigen Fällen so sein. Das ist aber nicht der Fall, wenn man nicht nur mit Abfragen, Formularen oder Berichten auf den nun statt der ursprünglichen Tabellen verwendeten Tabellenverknüpfungen arbeitet, sondern auch mit VBA - insbesondere mit DAO-Recordsets - auf diese Daten zugreift. Dabei treten gelegentlich Fehler auf. Warum diese nur gelegentlich auftreten und wie wir diese Fehler endgültig verhindern, zeigen wir in diesem Beitrag.

Voraussetzungen

Um die nachfolgenden Techniken verwenden zu können, benötigen wir die folgenden Voraussetzungen:

- Wir verwenden eine oder mehrere Tabellen einer SQL Server-Datenbank als Quelle für Tabellenverknüpfungen in unserer Access-Datenbank.
- Wir greifen mit DAO-Anweisungen wie **db.OpenRecordset** oder **db.Execute** auf diese Tabellenverknüpfungen zu, wobei **db** ein Verweis auf das **Database**-Objekt der aktuellen Access-Datenbank ist.

Fehler 3622 bei Verwendung von OpenRecordset oder Execute

Greifen wir mit der **OpenRecordset**-Methode auf die Daten einer Tabellenverknüpfung zu oder ändern Datensätze mit der **Execute**-Methode auf Basis einer Tabellenverknüpfung, tritt häufig der Fehler **3622** auf (siehe Bild 1):

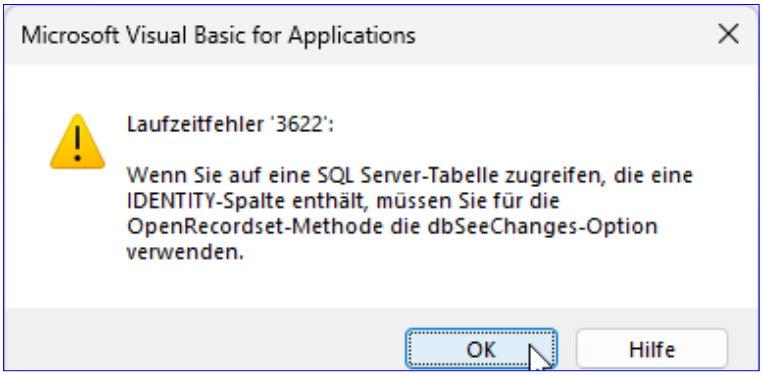


Bild 1: Fehlermeldung bei Verwendung einer **Recordset**-Anweisung

```
Wenn Sie auf eine SQL Server-Tabelle zugreifen, die eine
IDENTITY-Spalte enthält, müssen Sie für die OpenRecordset-
Methode die dbSeeChanges-Option verwenden.
```

Was bedeutet das im Detail?

Schauen wir uns die Meldung genau an:

- wir greifen auf eine SQL Server-Tabelle zu,
- diese hat eine **IDENTITY**-Spalte und
- wir müssen dafür offensichtlich eine Option namens **dbSeeChanges** verwenden.

Der Fehler tritt auf, wenn wir Folgendes durchführen:

- Wir rufen Daten mit **OpenRecordset** ab.
- Wir löschen einen Datensatz mit **Execute "DELETE FROM ..."**
- Wir ändern einen Datensatz mit **Execute "UPDATE ..."**

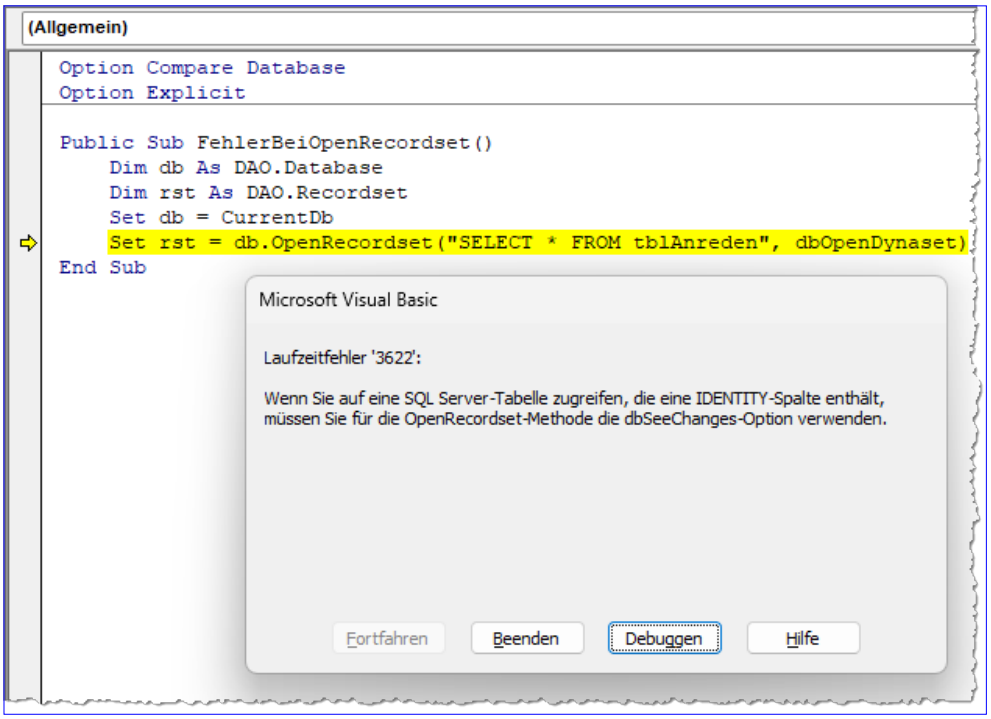


Bild 2: Die für den Fehler verantwortliche Anweisung

Schauen wir uns Beispiele im Detail an. All diese Anweisungen lösen den Fehler aus:

```
CurrentDb.OpenRecordset("SELECT * FROM tblAnreden ⤵
                        WHERE AnredeID = 1").Fields(0)
CurrentDb.Execute "DELETE FROM tblAnreden ⤵
                   WHERE AnredeID = 15", dbFailOnError
CurrentDb.Execute "UPDATE tblAnreden SET Anrede = 'Test' ⤵
                   WHERE AnredeID = 15", dbFailOnError
```

Bei dieser Anweisung hier tritt der Fehler nicht auf:

```
CurrentDb.Execute "INSERT INTO tblAnreden(Anrede) VA-
LUES('Etwas')", dbFailOnError
```

Der Grund ist, dass wir die **Autowert**-Funktion von Access zum SQL Server in Form einer **IDENTITY**-Eigenschaft migriert haben.

Damit Access über Änderungen an den Datensätzen im SQL Server-Backend informiert ist, müssen wir hier eine zusätzliche Option namens **dbSeeChanges** setzen.

Das ist sowohl bei **OpenRecordset** als auch bei **Execute** nötig.

Bei **Execute** jedoch nur bei **UPDATE**- und **DELETE**-Anweisungen, beim **INSERT INTO** existiert ja noch kein Datensatz und somit auch kein **IDENTITY**-Wert, den man hier abgleichen müsste.

Lösung

Die Lösung besteht im Hinzufügen des bereits erwähnten Parameters.

Wir schauen uns nochmal den Fehler und ein Beispiel für die fehlerhafte Zeile an (siehe Bild 2).

Bei **OpenRecordset** müssen wir nun noch den Wert **dbSeeChanges** für den dritten Parameter **Options** einsetzen, also zum Beispiel wie folgt:

```
Set rst = db.OpenRecordset("SELECT * FROM tblAnreden",
dbOpenDynaset, dbSeeChanges)
```

Schauen wir uns nun das folgende Beispiel mit mehreren **Execute**-Anweisungen an (siehe Listing 1). Hier fügen wir zuerst einen neuen Datensatz zur Tabelle **tblAnreden** hinzu. Dies gelingt ohne Probleme. Dann ermitteln wir den Primärschlüsselwert des neu hinzugefügten Datensatzes.

Anschließend ändern wir die Anrede des Datensatzes mit der Anrede **Neue Anrede** auf **Andere Anrede**. Auch dies gelingt problemlos.

VBA-InputBox richtig nutzen in Access

Es gibt zwei Elemente, über die man von VBA aus mit Bordmitteln mit dem Benutzer interagieren kann. Die MsgBox-Anweisung liefert ein Meldungsfenster, das man mit verschiedenen Texten, Icons und Schaltflächen ausstatten kann. Die InputBox-Anweisung bietet ein Eingabefenster für einfache Texte an, der auch vorgeben werden kann. Während die InputBox eigentlich recht einfach zu programmieren ist, gibt es dennoch einen Trick, auf den wir in diesem Beitrag eingehen werden. Zusätzlich liefern wir alle Infos, damit Sie die InputBox in Ihren Access-Anwendungen effizient einsetzen können und dem Benutzer die Eingabe notwendiger Daten möglichst einfach gestalten.

In Microsoft Access nutzt man eigentlich für jeglichen Austausch von Informationen mit dem Benutzer Formulare. Diese kann man flexibel mit allen notwendigen Steuerelementen ausstatten. Mit der **MsgBox**- und der **InputBox**-Funktion gibt es jedoch zwei Tools, mit denen wir in einigen Fällen um die Programmierung eines Formulars herumkommen können. In diesem Beitrag liegt unser Hauptaugenmerk auf der **InputBox**-Funktion.

Diese lässt sich eigentlich sehr einfach programmieren. Sie besteht aus dem Befehl **InputBox** sowie einige Parametern, mit denen wir das Aussehen und das Verhalten der **InputBox** festlegen können. Die Parameter lauten:

- **Prompt**: Meldung, die in der **InputBox** angezeigt werden soll
- **Title**: Text in der Titelleiste der **InputBox** (optional)
- **Default**: Standardwert, der im Textfeld der **InputBox** angezeigt wird (optional)
- **XPos**: Abstand der **InputBox** vom linken Rand des Bildschirms in Twips (optional)
- **YPos**: Abstand der **InputBox** vom oberen Rand des Bildschirms in Twips (optional)
- **Helpfile** und **Context**: Informationen zur Verwendung einer Hilfedatei, auf die wir hier nicht eingehen werden (optional)

Die einfachste Form einer **InputBox** erwartet lediglich die Angabe der anzuzeigenden Meldung (siehe Bild 1).

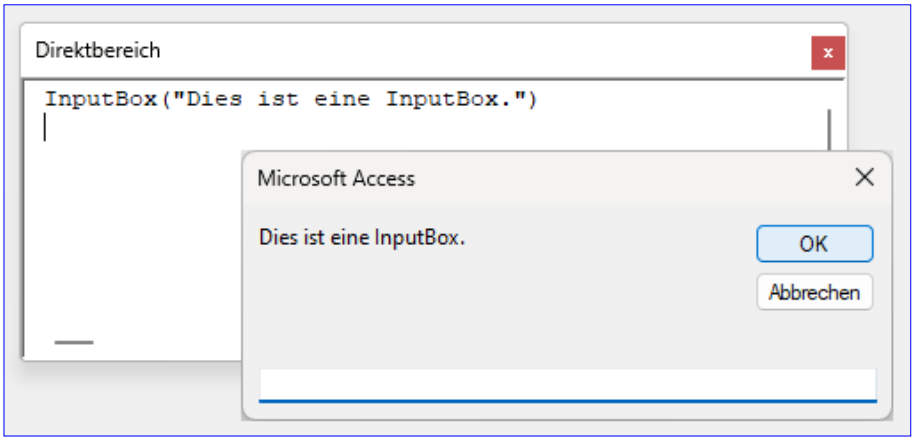


Bild 1: Beispiel für eine einfache InputBox

Jegliche Eingabe landet jedoch im Nirwana, da wir keine Variable verwenden, der wir das Ergebnis der **InputBox** zuweisen.

Ergebnis der InputBox auswerten

Wenn wir das Ergebnis der Funktion **InputBox** auswerten wollen, müssen wir diese wie eine

Bilder im SQL Server mit varbinary(max)

Wie man Bilder in Access speichert, ist seit der Version 2010 eindeutig vorgegeben – nämlich in Anlagefeldern. Die Inhalte dieser Felder können auch leicht im Bild-Steuerelement angezeigt werden, dazu braucht man nur eine einfache Bindung an das entsprechende Feld. Allerdings bietet der SQL Server keinen solchen Datentyp, sondern nur verschiedene Alternativen. Dabei handelt es sich beispielsweise um den Datentyp varbinary(max) oder um FILESTREAM. Die erstgenannte Alternative schauen wir uns im vorliegenden Beitrag an.

Beispielmaterial

Wir starten mit einer einfachen Beispieldatenbank, der wir eine Tabelle namens **tblBilder** hinzugefügt haben. Diese enthält die Felder wie in Bild 1.

Dazu haben wir unter anderem ein Anlagefeld eingefügt, dem wir nicht nur Bilder, sondern auch noch andere Dateien zuweisen können.

Hier können wir in der Datenblattansicht durch einen Doppelklick auf das im Anlagefeld angezeigt Büroklammer-Symbol einen Dialog namens **Anlagen** öffnen, mit dem wir die enthaltenen Anlagen verwalten können (siehe Bild 2).

Access-Bilder im Formular

Ein so eingefügtes Bild möchten wir auch im Formular anzeigen können. Dazu erstellen wir ein neues, leeres Formular, fügen diesem unsere Tabelle als Datensatzquelle hinzu und ziehen alle gewünschten Felder in den Formularentwurf.

Das Anlagefeld wird hier als Steuerelement des Typs **Anlage** hinzugefügt, das an das Anlagefeld der Tabelle gebunden ist.

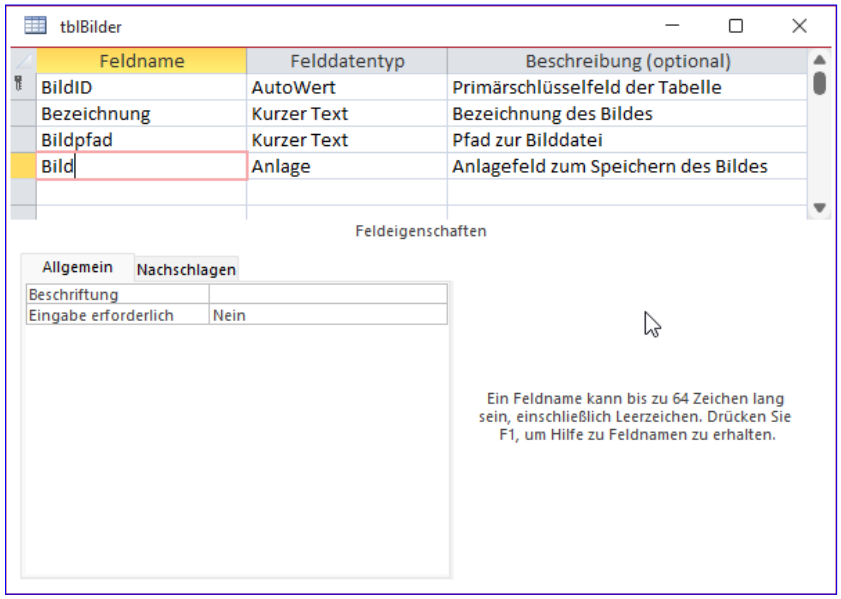


Bild 1: Tabelle zum Speichern von Bildern

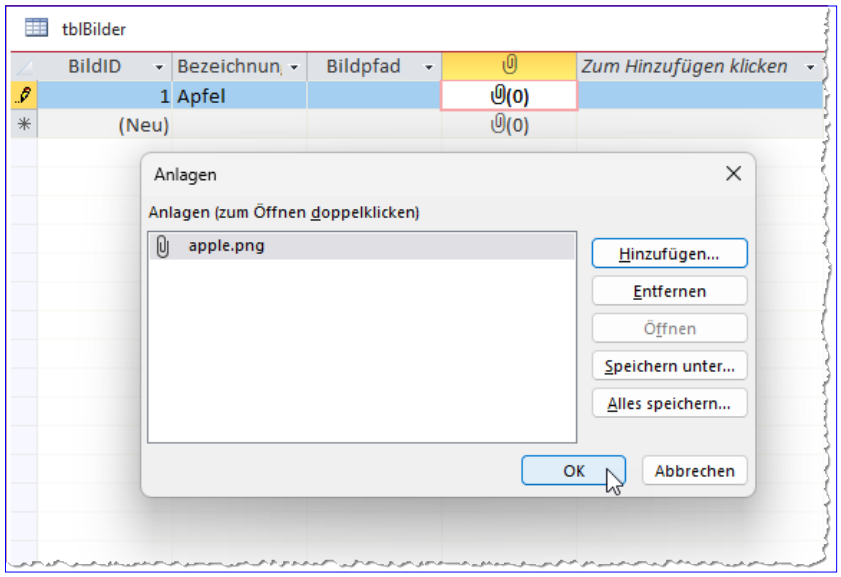


Bild 2: Einfügen eines Bildes

Alternativ können wir auch ein Steuerelement des Typs **Bild** einfügen.

Der Unterschied zwischen diesen beiden Steuerelementen ist im Wesentlichen, dass wir mit dem Anlagefeld auch die enthaltenen Dateien verwalten können.

Dazu brauchen wir nur das Steuerelement anzuklicken und sehen dann das Menü wie in Bild 3.

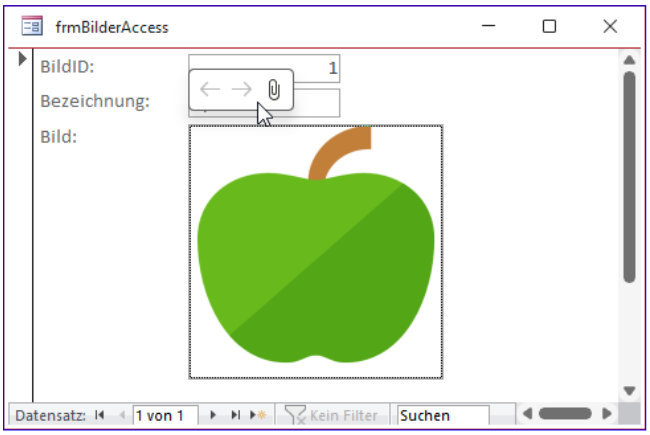


Bild 3: Ein Bild im Anlage-Steuerelement

Hier können wir zwischen den Bildern wechseln, wenn das Anlagefeld mehrere enthalten sollte, und mit einem Klick auf die Büroklammer den Dialog **Anlagen** zum Verwalten der Anlagen aufrufen.

Das Bild-Steuerelement zeigt einfach nur das erste Bild im Anlagefeld an.

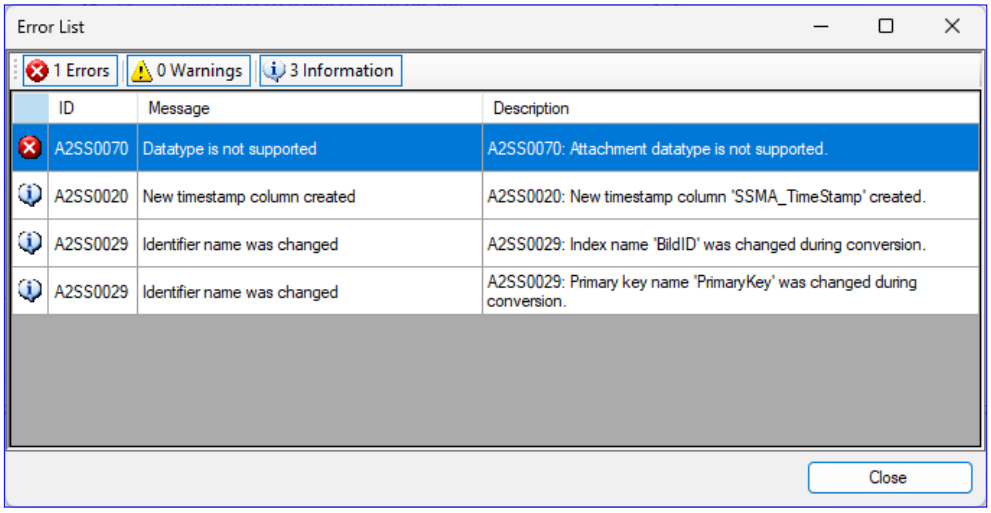


Bild 4: Meldung beim Versuch, ein Anlagefeld zum SQL Server zu migrieren

Anlagefeld im SQL Server?

Es wäre praktisch, wenn es im SQL Server einen entsprechenden Datentyp gäbe. Den gibt es allerdings nicht in dieser Form. Das erfahren wir auf die harte Tour, wenn wir versuchen, unsere Beispieldatenbank mit dem SQL Server Migration Assistant zu migrieren.

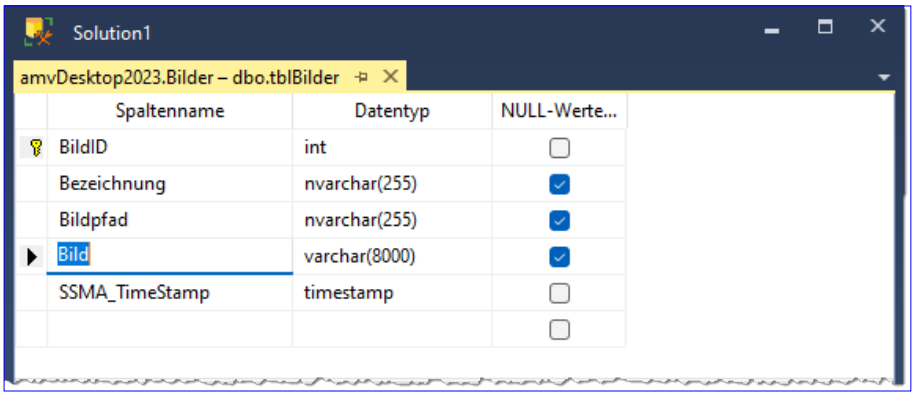


Bild 5: Das Anlagefeld wird in ein Feld mit dem Datentyp **varchar(8000)** migriert.

Die Fehlerliste aus Bild 4 zeigt, dass der Datentyp **Attachment** nicht unterstützt wird.

Es wird auch keine Alternative aufgezeigt, sondern das Feld wird einfach in eines mit dem Datentyp **varchar(8000)** migriert – das entnehmen wir dem Entwurf der neu erzeugten Tabelle im SQL Server Management Studio (siehe Bild 5).

Allerdings ist die Größe von 8.000 Zeichen etwas übertrieben – tatsächlich speichert die Tabelle nach der Migration nur noch den Namen der Datei in diesem Feld (siehe Bild 6).

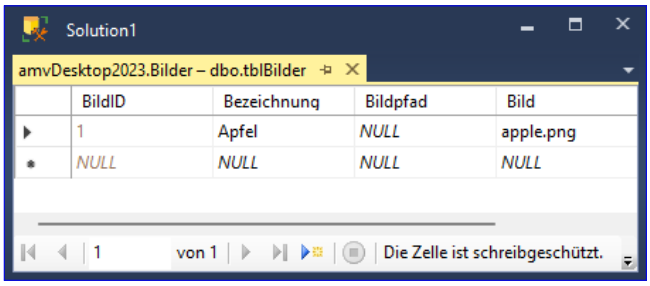


Bild 6: Das Feld **Bild** enthält nun nur noch den Dateinamen.

Das heißt, dass der Ansatz über den SQL Server Migration Assistant hier nicht brauchbar ist und wir nacharbeiten müssen.

Wo und wie Bilder im SQL Server speichern?

Navigieren wir im SQL Server Management Studio durch die Datentypen, die uns die Entwurfsansicht für Tabellen anbietet, finden wir schnell den Eintrag **image**. Ist das nicht der passende Datentyp?

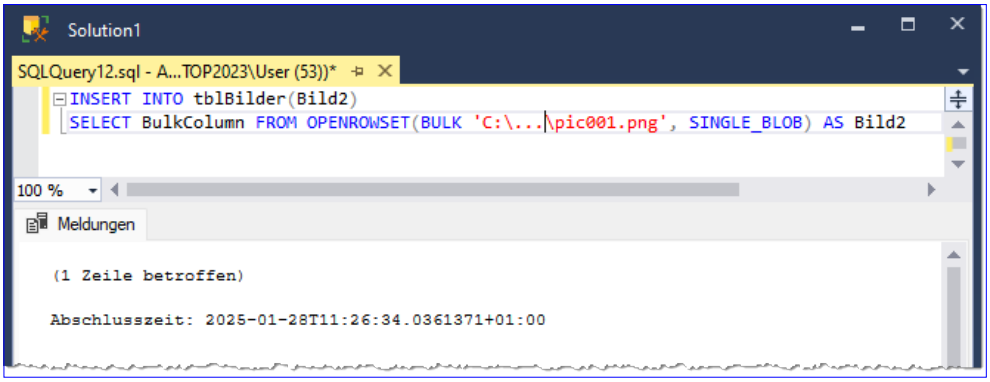


Bild 7: Hinzufügen eines Bildes per **INSERT INTO**-Anweisung

Schauen wir uns das einmal an. Zunächst einmal betrachten wir den Weg, dort ein Bild zu speichern. Dieser ist relativ einfach umzusetzen.

Wir nutzen eine einfache **INSERT INTO**-Anweisung, um dem **image**-Feld namens **Bild2** ein Bild hinzuzufügen. Diese sieht wie folgt aus:

```
INSERT INTO tblBilder(Bild2)
SELECT BulkColumn
FROM OPENROWSET(BULK 'C:\...\pic001.png', SINGLE_BLOB)
  AS Bild2
```

Die **OPENROWSET**-Funktion liest die angegebene Datei als einzelnen Datenblock (**SINGLE_BLOB**) und schreibt das Ergebnis in das Feld **Bild2**. Das Ergebnis sehen wir in Bild 7.

Zum Hinzufügen eines Bildes zu einem bestehenden Datensatz nutzen wir folgende Syntax:

```
UPDATE tblBilder
SET Bild2 = (
        SELECT bulkcolumn
        FROM OPENROWSET(
                BULK 'C:\...\pic002.png',
                SINGLE_BLOB) AS Bild2)
WHERE BildID = 1
```

Bild von image-Feld in Access anzeigen

Damit kehren wir zur Access-Datenbank zurück. Hier finden wir, da wir weiter oben die Tabelle **tblBilder** zum SQL Server migriert haben, nunmehr die Originaltabelle mit dem Namen **SSMA$tblBilder$local** vor sowie die verknüpfte Tabelle namens **tblBilder**.

Damit das Feld **Bild2**, das wir soeben zur SQL Server-Tabelle **tblBilder** hinzugefügt haben, auch in Access sichtbar wird, müssen wir die Verknüpfung aktualisieren. Das gelingt für eine einzelne Tabelle am einfachsten, indem wir für den Eintrag im Navigationsbereich den Kontextmenü-Befehl **Link aktualisieren** aufrufen (siehe Bild 8).

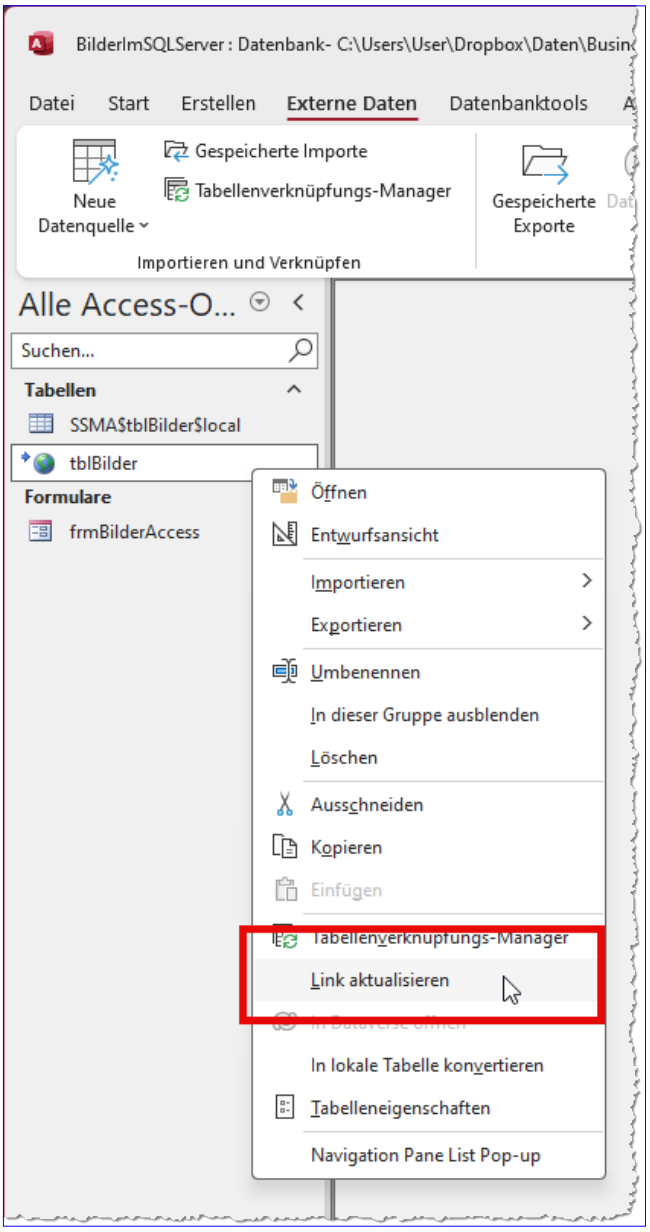


Bild 8: Aktualisieren einer verknüpften Tabelle

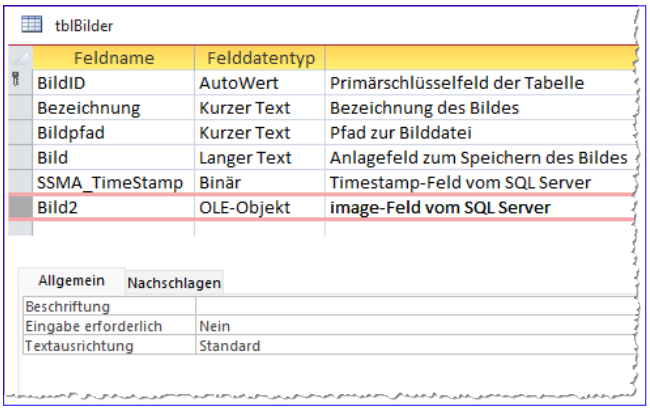


Bild 9: Das **image**-Feld erscheint in Access als **OLE-Objekt**

he Bild 10). Hier war nicht viel mehr zu erwarten, denn Tabellen können eben nur Texte und gegebenenfalls noch ein Kontrollkästchen anzeigen.

Wie aber sieht es im Formular aus? Hier konnten wir die über den SQL Server eingefügten Bilddateien in keinem der verfügbaren Steuerelemente sichtbar machen.

image-Datentyp ist veraltet

Neben der Tatsache, dass es keinen einfachen Weg zu geben scheint, den Inhalt unseres **image**-Feldes in einem Access-Formular sichtbar zu machen, ist der **image**-Datentyp auch noch als veraltet gekennzeichnet.

Das heißt, er wird nur noch aus Gründen der Kompatibilität beibehalten. Der Datentyp kann aber auch in einer der folgenden SQL Server-Versionen schlicht entfallen.

Also schauen wir uns direkt nach Alternativen um.

Wenn wir uns nun die aktualisierte Tabellenverknüpfung ansehen, finden wir das Feld **Bild2** mit dem Datentyp **OLE-Objekt** vor (siehe Bild 9).

Wechseln wir nun zur Datenblattansicht, um uns den Inhalt anzusehen, finden wir lediglich den Text OLE-Objekt vor (sie-

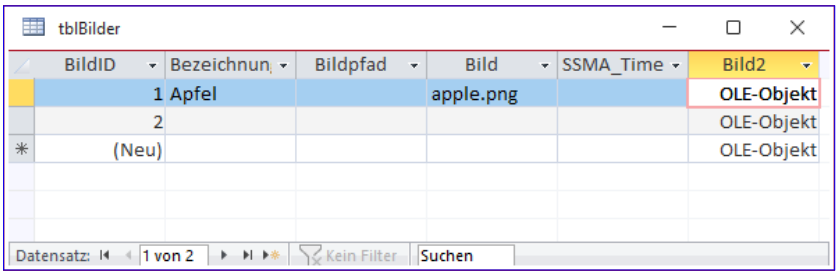


Bild 10: Der Inhalt wird ebenfalls als OLE-Objekt angezeigt.

Der Datentyp varbinary(max)

Eine weitere Möglichkeit, Bilder oder Dateien in einem Feld einer SQL Server-Datenbank zu speichern, ist der Datentyp **varbinary(max)**.

Dieser erlaubt es ebenfalls, Bilder mit der folgenden Anweisung hineinzuschreiben:

```
INSERT INTO tblBilder(BildVarBinary)
SELECT BulkColumn
FROM OPENROWSET(BULK 'C:\...\pic001.png', SINGLE_BLOB)
  AS BildVarBinary
```

Im Gegensatz zum Datentyp **image** ist der Datentyp **varbinary(max)** nicht veraltet.

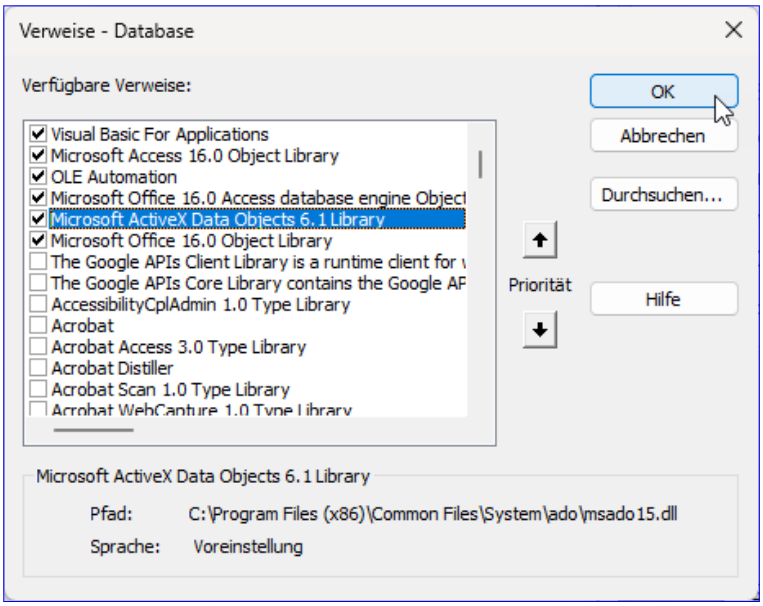


Bild 11: Verweis auf die **ADODB**-Bibliothek

Das Anzeigen von Bildern aus **varbinary(max)**-Feldern ist, soviel können wir bereits vorwegnehmen, auf verschiedene Arten möglich.

Damit allein ist es jedoch nicht getan, denn wir müssen uns um folgende Aufgaben kümmern:

- Bilder zu einem Datensatz anlegen: Wir müssen eine Schaltfläche hinterlegen, mit der wir einen Dateiauswahldialog öffnen und anschließend die gewählte Bilddatei in der SQL Server-Tabelle speichern.

- Bilder auslesen und im Bild-Steuerelement anzeigen

Der erste Punkt ist etwas komplizierter, als er aussieht. Wir können hier nicht einfach die Methoden nutzen, die das Anlegefeld uns bietet, sondern müssen direkt auf den SQL Server zugreifen. Wie das gelingt, schauen wir uns nun an. Danach schauen wir uns zwei Methoden an, um Bilder im Bild-Steuerelement anzuzeigen.

Verweis auf die ADODB-Bibliothek hinzufügen

Wir werden in diesem Beitrag auf die Methoden der ADODB-Bibliothek zugreifen. Diese bietet für unseren Anwendungsfall deutlich einfachere Methoden als die DAO-Bibliothek.

Dazu öffnen wir im VBA-Editor den **Verweise**-Dialog, in dem wir den Eintrag **Microsoft ActiveX Data Objects 6.1 Library** auswählen (siehe Bild 11).

In dem Zuge legen wir auch gleich noch einen Verweis auf die Bibliothek **Microsoft Office 16.0 Object Library** an, den wir später für die Verwendung von Dateidialogen benötigen.

Zwei Methoden: Umweg über Datei oder direkter Weg

In den folgenden Abschnitten schauen wir uns zwei Varianten an, um ein Bild aus einem **varbinary(max)**-Feld einer SQL Server-Tabelle in einem Bild-Steuerelement anzuzeigen.

Die erste speichert das Bild im Dateisystem und referenziert dieses dann mit dem Bild-Steuerelement.

Die zweite liest das Bild aus dem **varbinary(max)**-Feld ein und weist es direkt dem Bild-Steuerelement zu.

Formular anlegen

Wir beginnen mit der Variante, mit der wir das Bild zunächst in der aktuellen Datenbank speichern. Dazu legen wir erst einmal das Formular **frmBildVarBinary_Temp** an. Diesem weisen wir die per ODBC verknüpfte Tabelle **tblBilder** als Datensatzquelle zu.

Wir fügen die beiden Felder **BildID** und **Bezeichnung** aus dieser Tabelle ein und legen zusätzlich eine Schaltfläche und ein Bild-Steuerelement an (siehe Bild 12).

Für das **Bild**-Steuerelement stellen wir die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** jeweils auf **Beide** ein. So wird das Steuerelement vergrößert, wenn wir das Formular vergrößern.

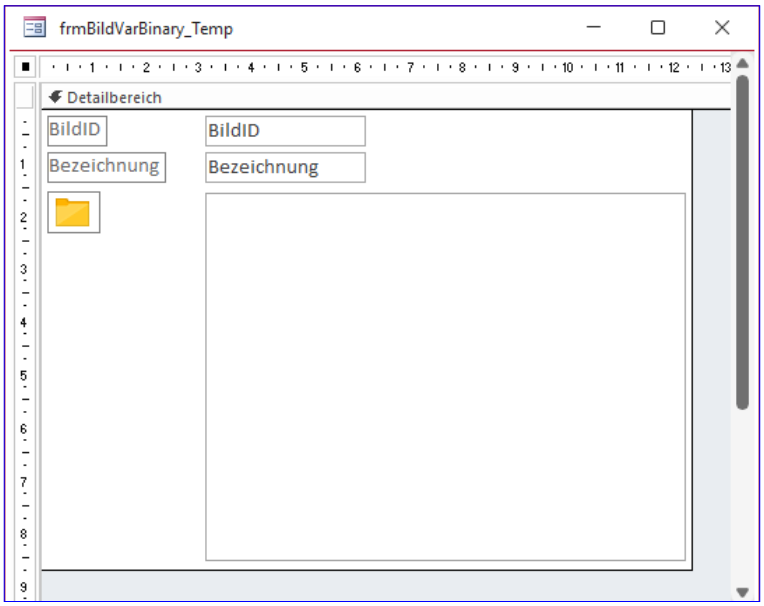


Bild 12: Entwurf des Formulars zur Anzeige von Dateien aus dem varbinary(max)-Feld

Hinzufügen eines Bildes

Zum Hinzufügen eines Bildes programmieren wir die Schaltfläche **cmdBildAuswaehlen** wie in Listing 1.

Diese Prozedur ruft zunächst die Funktion **ChooseAttachment** auf, die wie in Listing 2 aussieht und einen Dateidialog zur Auswahl der gewünschten Bilddatei öffnet.

Das Ergebnis speichert die Prozedur **cmdBildAuswaehlen** in der Variablen **strPfad**. Dann prüft sie als Erstes, ob **strPfad** überhaupt eine Datei enthält. Dazu nutzen wir die **Dir**-Funktion und untersuchen damit **strPfad**.

Dir liefert in diesem Fall den Namen der Datei aus dem Pfad zurück und wenn dieser eine Länge größer als **0** hat, ist die Datei vorhanden.

```
Private Sub cmdBildAuswaehlen_Click()
    Dim strPfad As String
    Dim strDateiname As String
    strPfad = ChooseAttachment
    If Not Len(Dir(strPfad)) = 0 Then
        If IsNull(Me.BildID) Then
            strDateiname = Mid(strPfad, InStrRev(strPfad, "\") + 1)
            Me.Bezeichnung = strDateiname
            Me.Dirty = False
        End If
        SpeichereBildInDatenbank strPfad, Me.BildID
        BildInBildSteuerelement Me.BildID
    End If
End Sub
```

Listing 1: Start der Auswahl und der Anzeige einer Bilddatei

Lastschriften mit DDBAC und VBA

Das Anlegen von Lastschriften ist eine wichtige Funktion, wenn man Kunden neben Zahlungsarten wie Rechnung, PayPal, Kreditkarte et cetera auch noch die Möglichkeit geben möchte, die Bankverbindung anzugeben, um den Betrag zum Fälligkeitsdatum einfach abbuchen zu lassen. Das ist gerade für wiederkehrende Leistungen wie Abonnements praktisch. Dazu sind jedoch erstens einige Voraussetzungen zu erfüllen wie zum Beispiel das Aktivieren der Möglichkeiten zum Einziehen von Lastschriften und das Beantragen einer sogenannten Gläubiger-Identifikationsnummer. Schließlich benötigen wir noch den passenden VBA-Code. Allein damit kommen wir aber nicht aus: Wenn wir gemütlich von unserer Datenbankanwendung aus Lastschriften anlegen wollen, benötigen wir eine zusätzliche Bibliothek. Dabei handelt es sich um die in vielen Anwendungen verwendete DDBAC-Bibliothek. Wie wir die einzelnen Schritte erledigen, um unsere Anwendung zum Anlegen von Lastschriften verwenden zu können, zeigen wir in diesem Beitrag.

Hinweis

Zum Umsetzen der Beispiele aus diesem Beitrag ist eine kostenpflichtige Lizenz der DDBAC-Komponenten Voraussetzung. Sie erhalten diese auf der folgenden Webseite:

`https://www.andreminhorst.de/ddbac`

Benötigte Informationen für eine Lastschrift

Für eine SEPA-Lastschrift benötigen wir diese Informationen:

- Daten des Zahlungspflichtigen (Zahlungsempfängers)
- Name und Adresse des Kontoinhabers
- IBAN des Kontos
- BIC (bei internationalen Lastschriften außerhalb des SEPA-Raums)
- Gläubiger-Identifikationsnummer (CID): Wird von der Bundesbank oder der jeweiligen nationalen Zentralbank vergeben und kennzeichnet den Zahlungsempfänger eindeutig.
- Mandatsreferenz: Eindeutige Nummer, die das Mandat identifiziert. Wird vom Zahlungsempfänger vergeben und muss einmalig sein.
- Mandatserteilung: Schriftliches oder elektronisches SEPA-Lastschriftmandat. Enthält Erlaubnis zur Abbuchung vom Konto des Zahlungspflichtigen und muss vom Zahlungspflichtigen unterschrieben werden.

Außerdem benötigen wir folgende Angaben:

- Betrag der Abbuchung
- Fälligkeitsdatum (Einzugstermin)
- Verwendungszweck (z. B. Rechnungsnummer)
- Wiederkehrende oder einmalige Lastschrift

Zusätzliche Anforderungen:

- Pre-Notification: Der Zahlungspflichtige muss mindestens 14 Tage vor Abbuchung über den Einzug informiert

werden (es sei denn, eine kürzere Frist wurde vereinbart)

Schritt für Schritt

Wie bereits eingangs erwähnt, sind einige Schritte nötig, die wir uns in der folgenden Reihenfolge ansehen:

- Aktivieren der Lastschrift-Funktionen für das Bankkonto
- Beantragen einer Gläubiger-Identifikationsnummer
- Beantragen einer Registrierungsnummer für Onlinebanking
- Holen der DDBAC-Komponenten
- Holen des Lastschrift-Mandats
- Herstellen einer Verbindung per Onlinebanking
- Programmieren der Lastschrift-Funktionen

Aktivieren der Lastschrift-Funktionen für das Bankkonto

Ein Bankkonto ist normalerweise nicht für den Einsatz von Lastschriften vorgesehen. Dies müssen wir zuerst aktivieren. In meinem Fall reichte es aus, meinen Ansprechpartner bei der Bank anzurufen und ihn zu bitten, dies einzurichten. Man muss dazu ein, zwei Formulare ausfüllen. Der Hintergrund ist, dass es bei einer Lastschrift nicht sichergestellt ist, dass das Geld auch auf dem Konto verbleibt. Der Kunde hat immer die Möglichkeit, eine Lastschrift zurückzuholen. Die Aktivierung von Lastschriften ging in unserem Fall recht schnell – es hängt aber von der jeweiligen Bank ab.

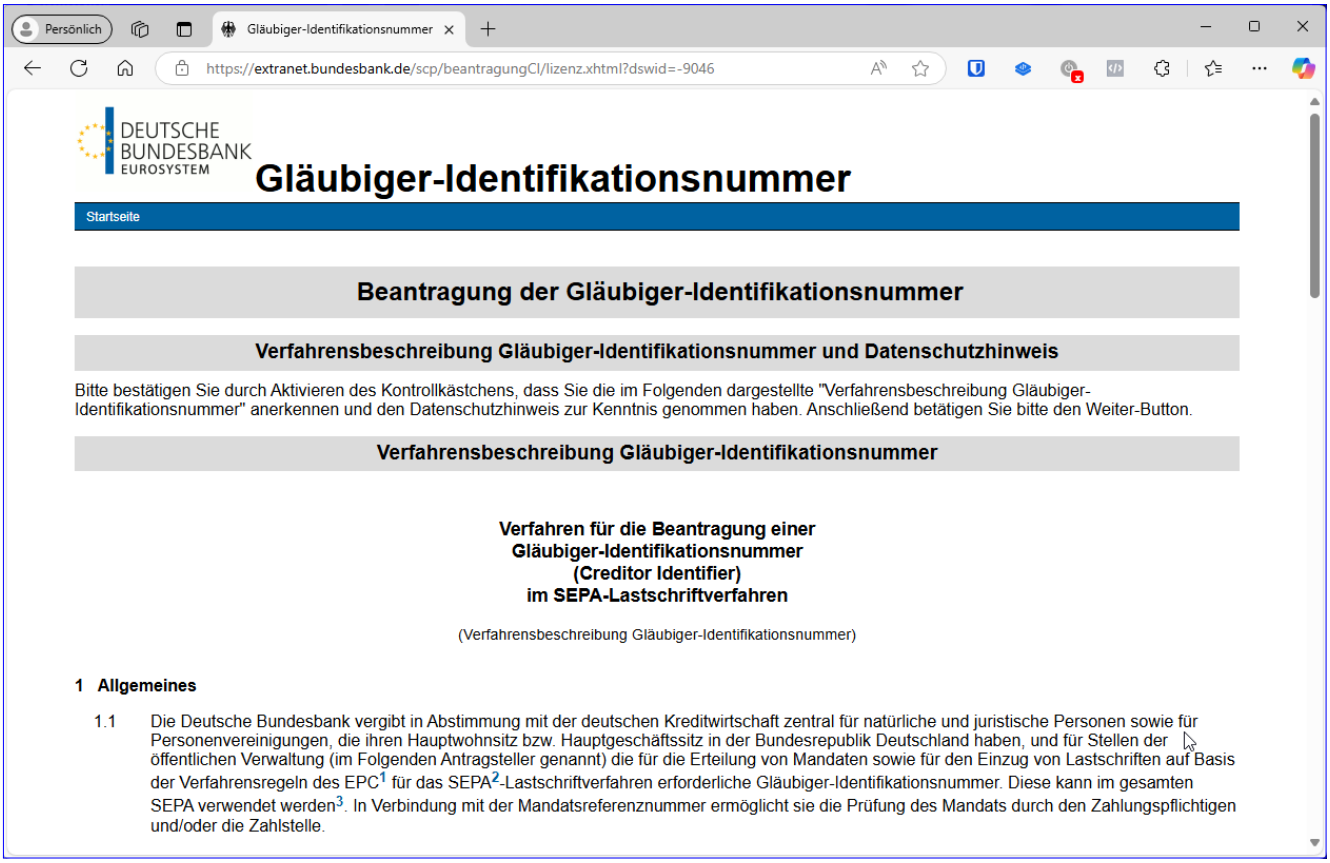


Bild 1: Start der Beantragung einer Gläubiger-Identifikationsnummer

Beantragen einer Gläubiger-Identifikationsnummer

Wie oben beschreiben, benötigen wir auch eine Gläubiger-Identifikationsnummer. Diese können wir bei der Deutschen Bundesbank beantragen. Die Startseite dort sieht beispielsweise wie in Bild 1 aus. Der Vorgang ist hier gut dokumentiert und es sind keine weiteren Voraussetzungen zu erfüllen, um diese ID zu erhalten. Der Erhalt sollte in der Regel innerhalb eines Werktages zu bewerkstelligen sein.

Wem dies zu lange dauert, der kann auch eine Gläubiger-Identifikationsnummer zum Testen holen. Diese finden wir auf der Seite mit den häufig gestellten Fragen zu der Gläubiger-Identifikationsnummer.

Die Test-ID lautet: **DE98ZZZ09999999999**

Beantragen einer Registrierungsnummer für Onlinebanking

Wer mit Komponenten wie den DDBAC-Komponenten Onlinebanking betreiben möchte, benötigt dazu eine eigene Registrierungsnummer.

Diese kann hier beantragt werden (siehe Bild 2):

`https://www.fints.org/de/hersteller/produktregistrierung`

Hier finden wir unter dem Link **Registrierungsprozess** ein Formular, in das wir die notwendigen Informationen eintragen können.

Die Vergabe dieser Nummer dauerte bei uns einige Tage, aber wir wissen nicht, wie schnell der Prozess aktuell durchgeführt wird.

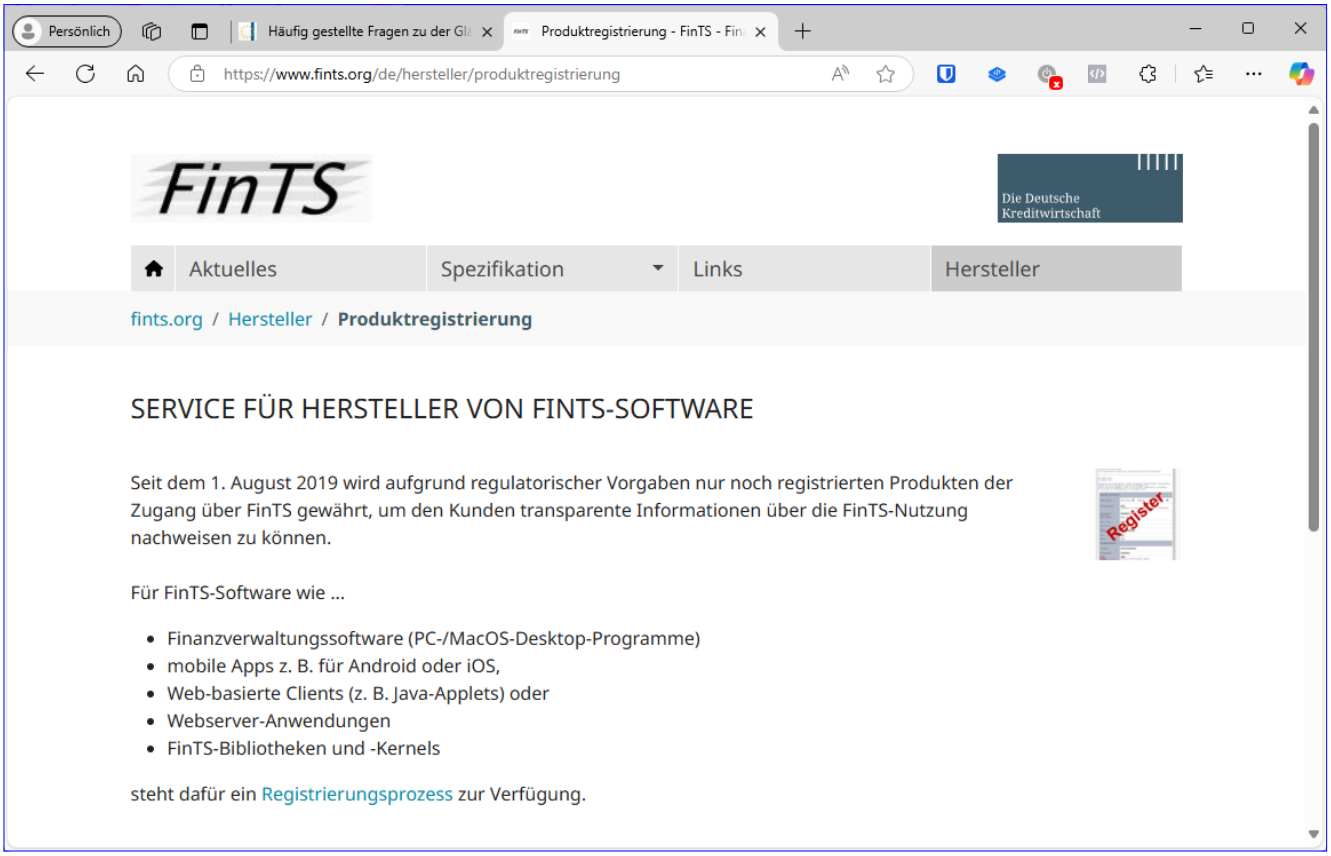


Bild 2: Seite zum Registrieren einer FinTS-Nummer

Diese Nummer machen wir im VBA-Projekt beispielsweise über eine wie folgt angelegte Konstante verfügbar:

```
Public Const cstrFints As String = "XXXXXXXXXXXXXXXXXXXX"
```

Holen der DDBAC-Komponenten

Schließlich benötigen wir die kostenpflichtigen DDBAC-Komponenten. Diese finden wir auf der folgenden Webseite:

```
https://www.andreminhorst.de/ddbac
```

Installieren der DDBAC-Komponenten

Im Download der DDBAC-Komponenten befinden sich zwei Zip-Dateien. Wir installieren die aus der Zipdatei **DDBACNetWrapper-Version-x-x-x-x.zip**.

Hier finden wir wiederum drei Verzeichnisse, von denen wir das Verzeichnis **DDBACWrapper** auswählen. Hier installieren wir abhängig davon, ob das installierte Windows 32-Bit oder 64-Bit verwendet, die Datei **DDBACSetup.msi** oder **DDBACSetupx64.msi**.

Die meisten Windows-Versionen haben derzeit 64-Bit.

Holen des Lastschrift-Mandats

Ein sehr wichtiger Faktor ist, dass wir von dem Kunden, von dem wir Geld per Lastschrift einziehen wollen, ein Lastschriftmandat erhalten müssen. Zu beschreiben, wie dieser Vorgang abläuft, würde hier den Rahmen sprengen. Wichtig ist jedoch, dass die Lastschrift mit Zustimmung des Kunden geschieht.

Herstellen einer Verbindung per Onlinebanking

Wenn die Onlinebanking-Komponenten installiert sind, benötigen wir vor dem eigentlichen Programmieren erst einmal eine Verbindung zum Bankserver.

Dazu starten wir eine Anwendung, die wir in der Systemsteuerung unter dem Namen **Homebanking Administrator (32-Bit)** finden (siehe Bild 3).

Hier klicken wir auf die Schaltfläche **Neu...** und finden einen weiteren Dialog namens **HBCI/FinTS-Kontakt** vor. Hier geben wir beispielsweise die Bankleitzahl oder den

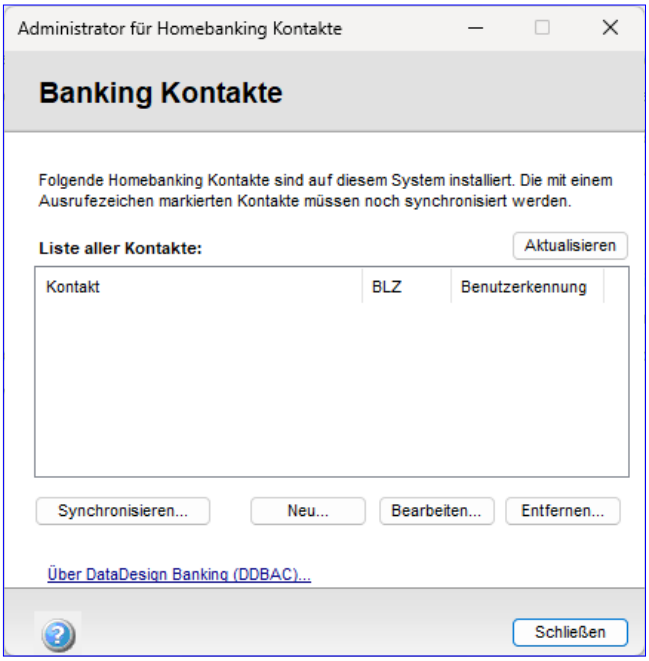


Bild 3: Anlegen eines neuen Banking-Kontakts

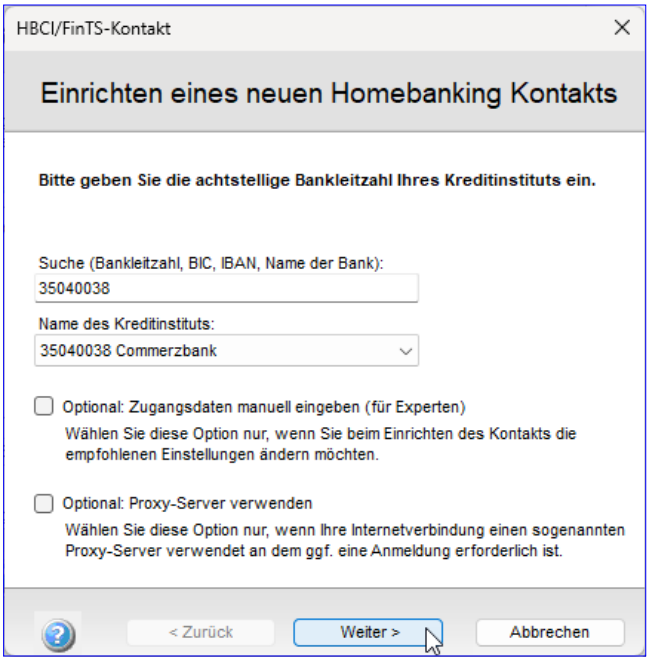


Bild 4: Auswahl der Bank

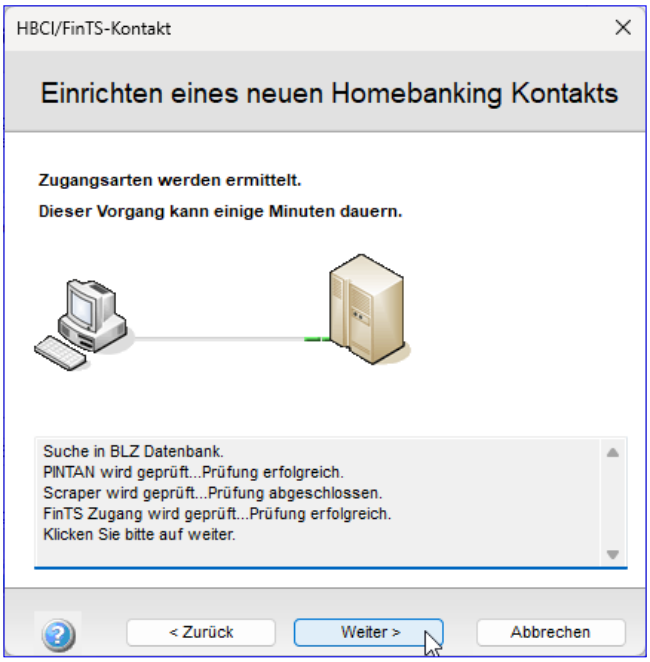


Bild 5: Ermitteln der Zugangsdaten

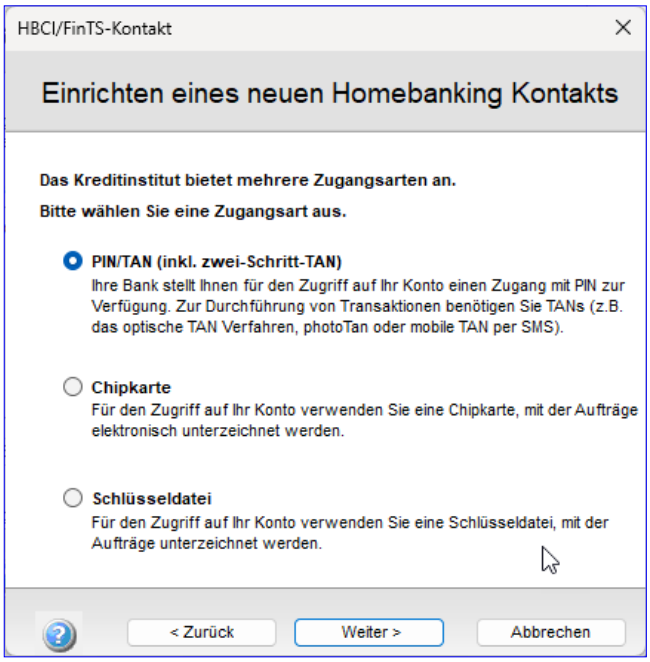


Bild 6: Auswahl der Zugangsart

Banknamen ein, um die Bank zu finden. Nachdem wir die Bank ausgewählt haben, klicken wir auf **Weiter** (siehe Bild 4).

Das Programm ermittelt dann die verfügbaren Zugangsdaten (siehe Bild 5).

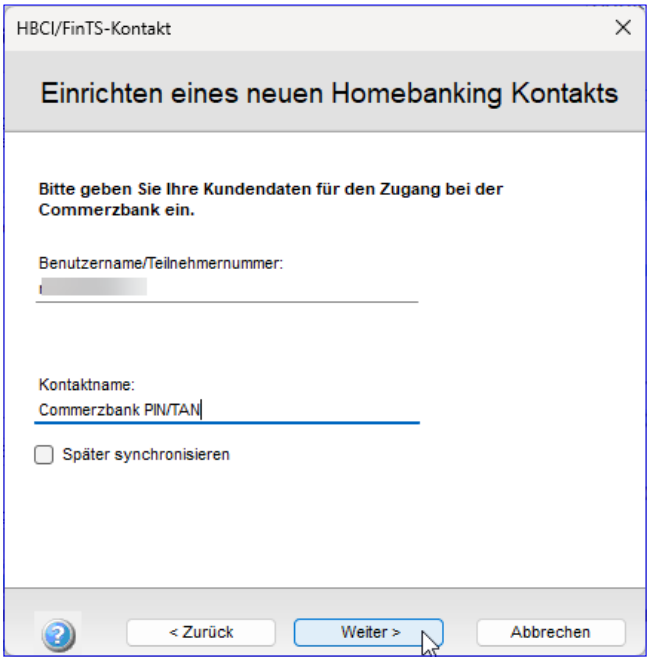


Bild 7: Angabe der Teilnehmernummer

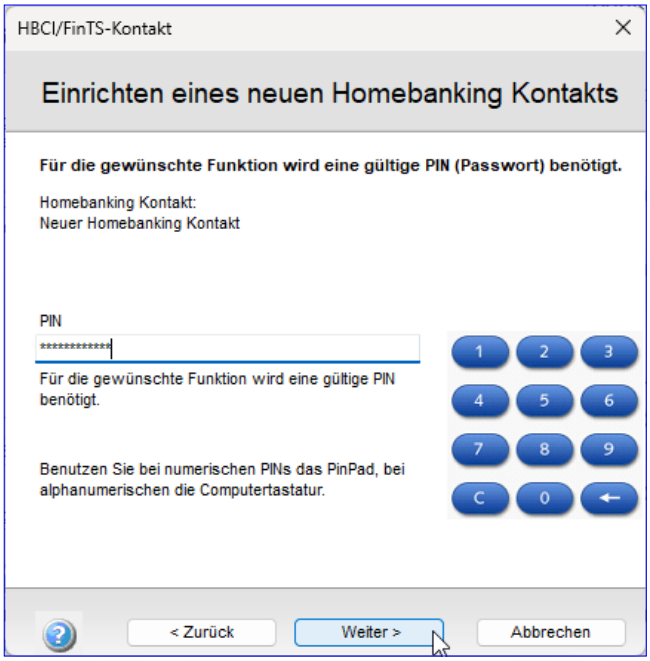


Bild 8: Eingabe des PIN

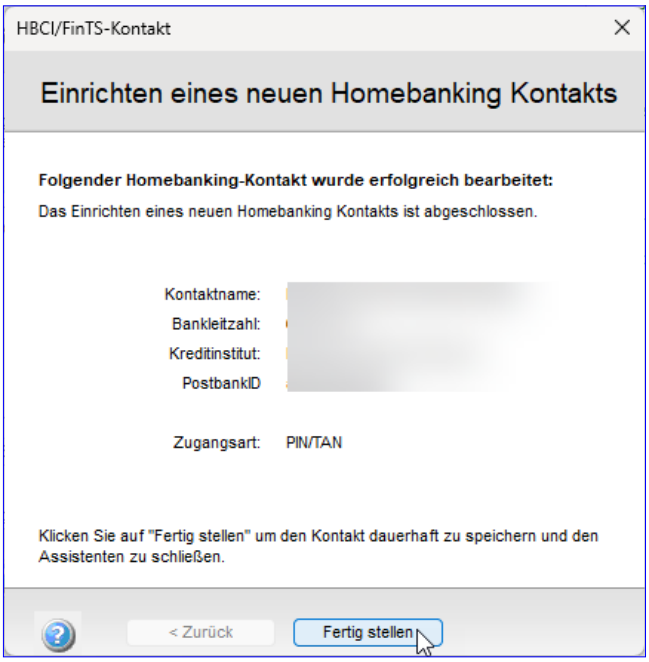


Bild 9: Erfolgsmeldung

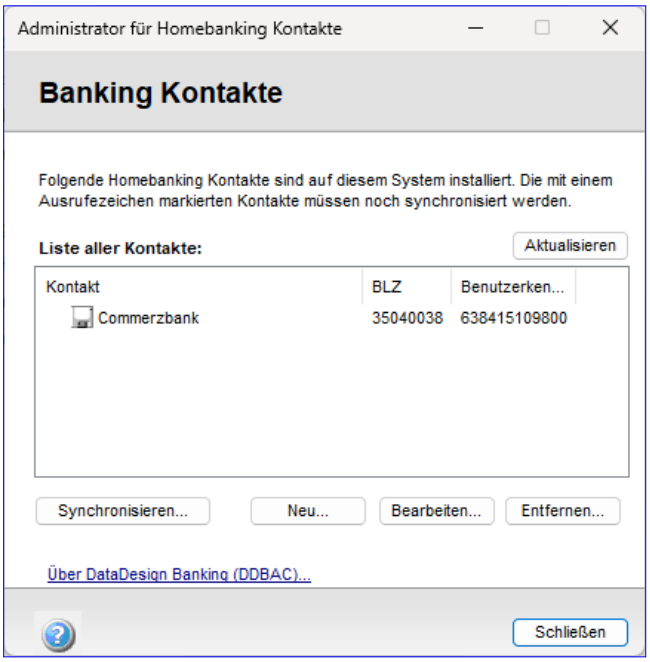


Bild 10: Der neue Banking Kontakt in der Liste der Kontakte

Die verfügbaren Zugangsarten erscheinen dann im folgenden Dialog zur Auswahl. Hier wählt man in der Regel **PIN/TAN**, was üblicherweise auch alle aktuellen Zugangsarten wie **photoTan** abdeckt (siehe Bild 6).

Anschließend müssen wir den Benutzernamen beziehungsweise die Teilnehmernummer eingeben. Das ist die gleiche Nummer, unter der Sie sich vermutlich auch beim Onlinebanking anmelden (siehe Bild 7).

Schließlich benötigen wir zum Synchronisieren mit diesem Bankkontakt noch die Eingabe der PIN (siehe Bild 8).

Danach folgen je nach Bank und Zugangsart noch weitere Abfragen beispielsweise zum Sicherheitsverfahren. Abschließend erhalten wir die Meldung über die erfolgreiche Synchronisierung (siehe Bild 9).

Damit liegen die grundlegenden Daten nun auf dem aktuellen Rechner und wir können von Access aus per VBA auf die Informationen nicht nur des Bank Kontakts, sondern auch auf die darin enthaltenen Bankverbindungen zugreifen.

Bank Kontakt und Konten einsehen

Bleiben wir noch kurz im **Administrator für Homebanking Kontakte** und schauen uns an, wie wir die Konten zu diesem Kontakt einsehen können. Dazu klicken wir auf

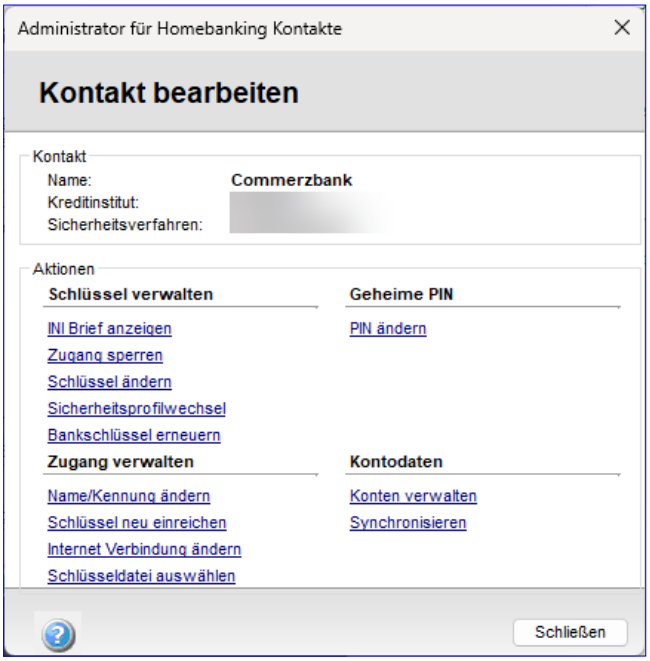


Bild 11: Bearbeiten eines Banking Kontakts

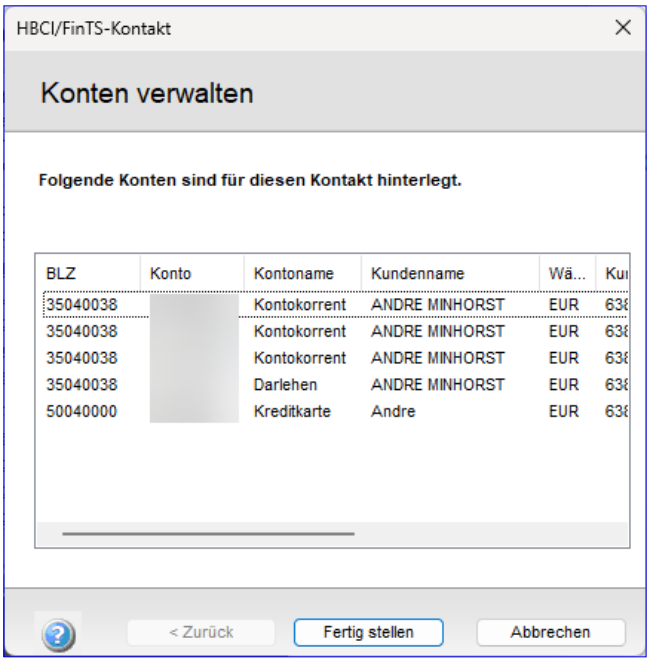


Bild 12: Dialog zum Verwalten der Konten

den Banking Kontakt und dann auf die Schaltfläche **Bearbeiten** (siehe Bild 10).

Dies öffnet einen weiteren Dialog (siehe Bild 11). Hier finden wir neben einigen anderen Möglichkeiten, unseren Banking Kontakt zu bearbeiten, auch den Link **Konten verwalten**.

Wenn wir diesen anklicken, landen wir auf einer weiteren Seite, die uns alle Konten anzeigt, die mit diesem Banking Kontakt verknüpft sind (siehe Bild 12).

Programmieren der Lastschrift-Funktionen

Damit kommen wir zum eigentlichen Thema dieses Beitrags, nämlich dem Programmieren der Lastschrift.

Dazu wechseln wir in unserer Beispieldatenbank als Erstes zum VBA-Editor und fügen einen Verweis auf die benötigte Bibliothek der DDBAC-Komponenten hinzu.

Dabei handelt es sich um den Eintrag **DataDesign DDBAC FinTS 4.0** (siehe Bild 13).

Unterschiede zwischen SEPA-Basislastschrift und SEPA-Firmenlastschrift

Für SEPA-Lastschriften gibt es zwei Verfahren: die SEPA-Basislastschrift und die SEPA-Firmenlastschrift.

- Die SEPA-Basislastschrift (**CORE**) kann sowohl von Verbrauchern als auch Unternehmen genutzt werden. Nach einer Belastung hat der Zahler die Möglichkeit, die Lastschrift innerhalb von acht Wochen ohne Angabe von Gründen zurückzugeben. Die Buchung wird dabei rückgängig gemacht.

- Die SEPA-Firmenlastschrift (**B2B**) ist ausschließlich für Geschäftskunden (Nicht-Verbraucher) vorgesehen und dient als zusätzliches Verfahren zur Vereinfachung der geschäftlichen Zahlungsabwicklung. Im Gegensatz zur Basislastschrift besteht bei der Firmenlastschrift keine Möglichkeit zur Rückgabe der Lastschrift.

Unabhängig vom Verfahren kann eine nicht autorisierte Lastschrift – also ein Lastschrifteinzug ohne gültiges Man-

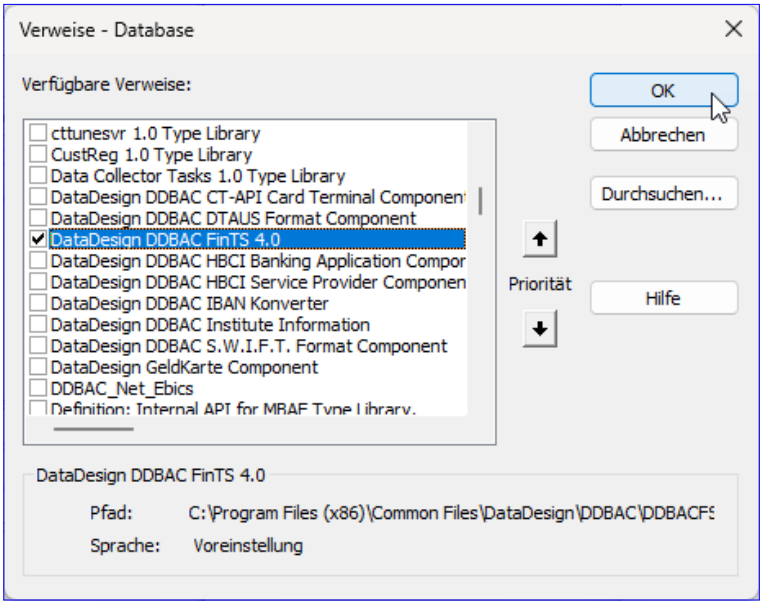


Bild 13: Verweis auf die DDBAC-Komponente

dat – innerhalb von 13 Monaten nach der Kontobelastung vom Zahler zurückgegeben werden.

Banking Kontakte und Accounts in Access einlesen

Damit wir von Access aus per VBA Lastschriften einreichen können, müssen wir auf die entsprechenden Banking Kontakte und die darin enthaltenen Konten zugreifen können.

Dabei handelt es sich um die Informationen, die wir soeben bereits über die Anwendung **Homebanking Administrator** in der Systemsteuerung hinterlegt haben.

Die Banking Kontakte wollen wir in einem Formular in ein Kombinationsfeld einlesen. Der Benutzer kann dann einen der Einträge auswählen.

Nachdem dies geschehen ist, sollen die Accounts, also die Konten, dieses Banking Kontakts in einem zweiten Auswahlfeld bereitgestellt werden. Hat der Benutzer auch hier einen Wert selektiert, haben wir schon einmal die Informationen zu dem Konto, zu dessen Gunsten die Lastschrift ausfallen soll.

Formular anlegen

Dazu legen wir als Erstes ein neues, leeres Formular an. Diesem fügen wir ein Kombinationsfeld namens **cboBankingkontakte** für die Banking Kontakte hinzu und ein weiteres Kombinationsfeld namens **cboAccounts** für die Konten.

Für beide Kombinationsfelder stellen wir die Eigenschaft **Datensatzherkunft** auf den Eintrag **Wertliste** ein (siehe Bild 14).

Banking Kontakte füllen

Beim Füllen der Banking Kontakte wollen wir direkt berücksichtigen, ob diese überhaupt das Hinzufügen von Lastschriften erlauben.

Dies können wir ebenfalls aus den Informationen abfragen, die für unsere Banking Kontakte gespeichert wurden.

Dazu nutzen wir die Funktion **getContacts** aus Listing 1. Für die Liste der Bankkontakte stellen wir eine Variable namens **m_Contacts** bereit, in der wir die Kontakte temporär speichern. Diese deklarieren wir wie folgt:

```
Private m_Contacts As BACContacts
```

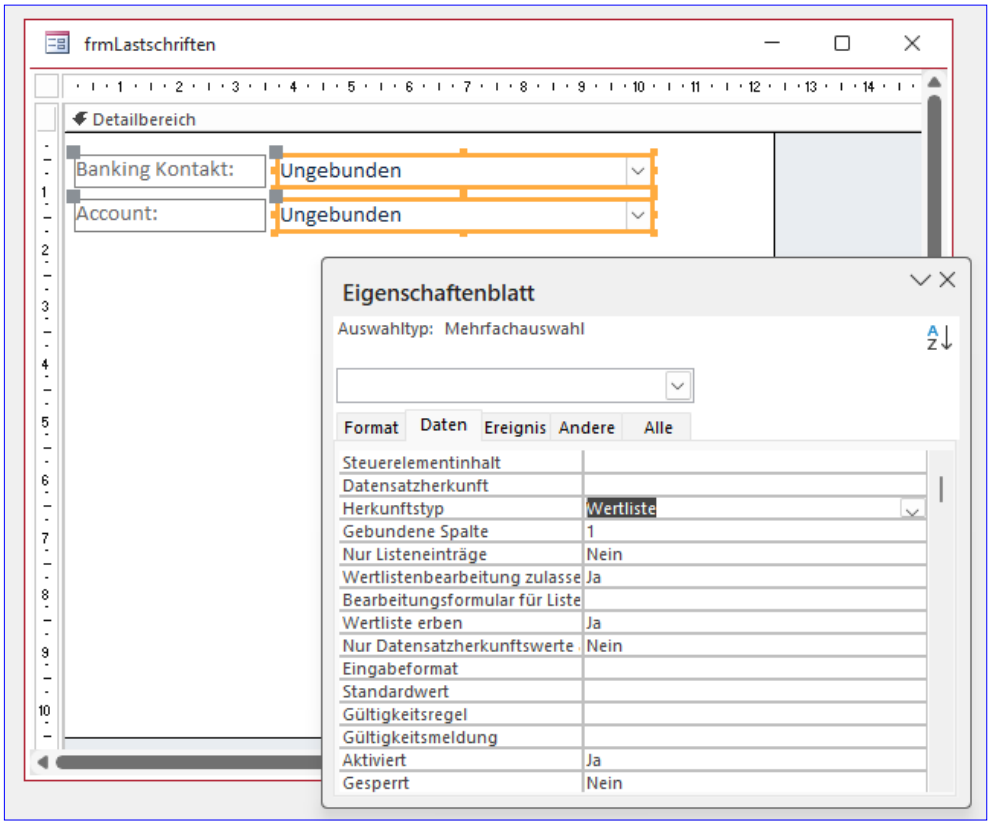


Bild 14: Formular zur Auswahl der Banking Kontakte und Accounts

SQL Server-Tabellenverknüpfungsassistent

Die Bordmittel von Access zum Herstellen oder Aktualisieren von Tabellenverknüpfungen zum SQL Server sind teilweise etwas umständlich zu bedienen und erfordern zwangsläufig den Einsatz von Data Source Names (DSN). Diese möchte man unter Umständen aber gar nicht nutzen, sondern einfach alle Informationen in die Verbindungszeichenfolge für die Tabelle schreiben. Natürlich kann man sich ein Set von Tabellen und Formularen zusammenstellen, mit denen man die Verbindungszeichenfolgen und Tabellenverknüpfungen verwaltet. Diese werden dann bei Bedarf der Datenbank hinzugefügt, deren Tabellenverknüpfungen man pflegen möchte. Aber manchmal möchte man einfach nur schnell mal etwas ausprobieren und dazu ist der Aufwand, die Datenbank um diese Tools zu erweitern, zu aufwendig. Wie wäre es also, wenn wir diese Tools einfach in ein Access-Add-In auslagern, mit dem wir unsere Verbindungszeichenfolgen und Verknüpfungen verwalten können? In diesem Beitrag schauen wir uns an, wie dies aussieht.

Anlass zur Programmierung dieses Access-Add-Ins war der Umstand, dass ich immer wieder mal Testdatenbanken vom Kunden bekomme oder selbst Beispiele programmiere, deren Daten im SQL Server liegen und für die ich schnell eine Tabellenverknüpfung erstellen möchte. Das ist mir allerdings über den Assistenten zum Importieren oder Verknüpfen von Tabellen aus ODBC-Datenbanken etwas zu aufwendig (siehe Bild 1).

Ich möchte einmal die Verbindungszeichenfolge festlegen und dann so schnell wie möglich Tabellenverknüpfungen erstellen oder aktualisieren.

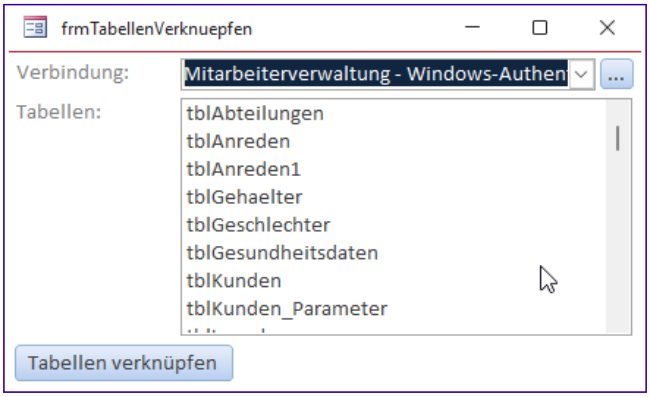


Bild 2: Hauptformular unseres Access-Add-Ins

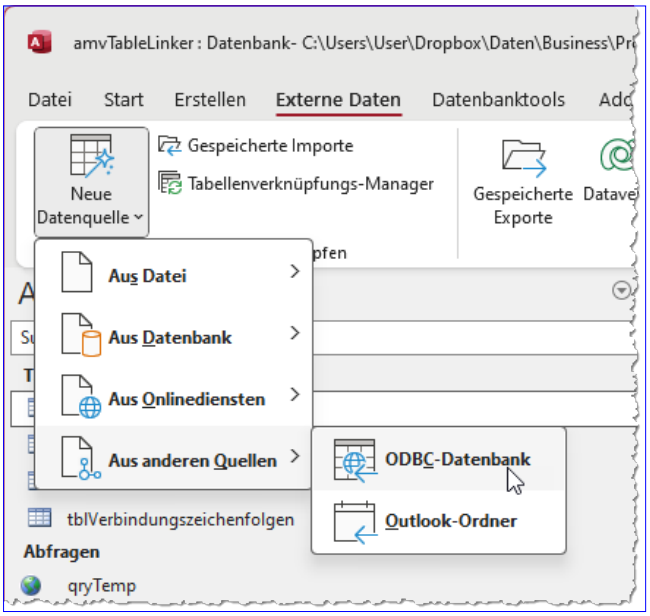


Bild 1: Herstellen einer Tabellenverknüpfung mit Bordmitteln

Tabellen per Add-In verknüpfen

Das gelingt mit dem Access-Add-In, das wir in diesem Beitrag vorstellen, ganz einfach.

Dazu brauchen wir nur über den Ribbon-Befehl **Datenbanktools|Add-Ins|Add-Ins|amvTableLinker** das Haupt-

formular unseres Add-Ins zu starten, das direkt nach dem Aufruf wie in Bild 2 aussieht. Hier finden wir nur wenige Elemente vor:

- Das Kombinationsfeld oben zeigt alle Datenbankverbindungen an, die wir bereits im Add-In hinterlegt haben und dient zur Auswahl der zu verwendenden Datenbankverbindung.

- Das Listenfeld zeigt alle Tabellen und Views an, die wir in der referenzierten Datenbank vorfinden. Das Listenfeld erlaubt die Auswahl mehrerer Tabellen gleichzeitig. Dies geschieht bei gedrückter **Alt**-Taste, um einzelne Einträge aus- oder abzuwählen und bei gedrückter **Umschalttaste** lassen sich mehrere zusammenhängende Tabellen markieren.

- Die Schaltfläche **Tabellen verknüpfen** verknüpft schließlich alle aktuell markierten Tabellen.

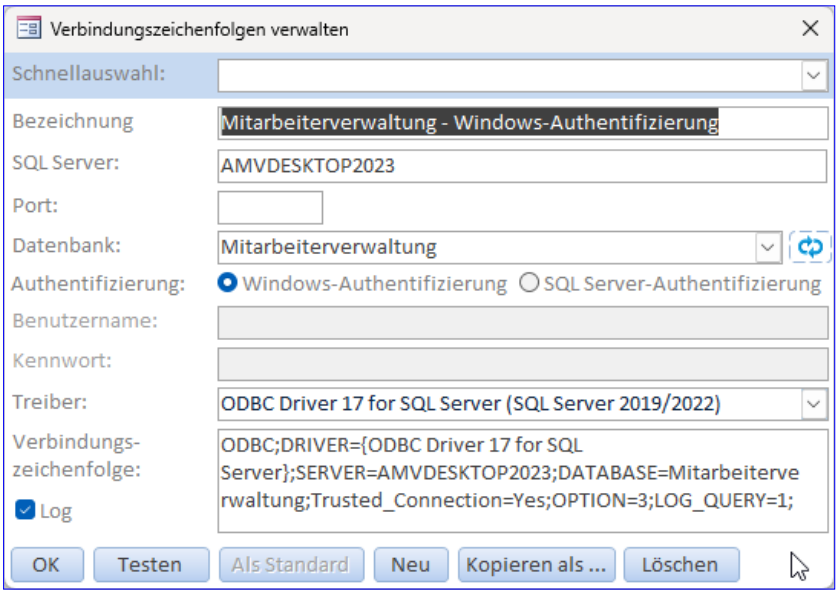


Bild 3: Verwalten von Verbindungszeichenfolgen

Nach Auswahl einer Verbindung bzw. beim Öffnen des Formulars werden in dem Listenfeld zudem alle Namen der Tabellen markiert, deren Namen den Tabellenverknüpfungen in der Access-Datenbank entsprechen. Um alle aktuell vorhandenen Tabellenverknüpfungen zu aktualisieren, braucht man also nur die richtige Verbindungszeichenfolge auszuwählen und die Schaltfläche **Tabellen verknüpfen** zu betätigen.

Verbindungszeichenfolgen verwalten

Schließlich finden wir oben rechts noch eine Schaltfläche mit drei Punkten, mit der wir den Dialog zum Verwalten der Verbindungszeichenfolgen öffnen können.

Dieser sieht wie in Bild 3 aus und erlaubt das Eintragen der wesentlichen Merkmale einer SQL Server-Verbindung. Hier finden wir folgende Eingabemöglichkeiten:

- **Bezeichnung**: Selbst vergebene Bezeichnung für die Verbindungszeichenfolge

- **SQL Server**: Adresse des SQL Servers, zum Beispiel Servername oder IP, gegebenenfalls unter Angabe des Instanznamens

- **Port**: Port des SQL Servers, wenn abweichend vom Standardport **1433**

- **Datenbank**: Name der Datenbank. Dieser kann nach Festlegung des SQL Servers auch aus den vorhandenen Datenbanken ausgewählt werden.

- **Authentifizierung**: Hier wählt man zwischen **Windows-Authentifizierung** und **SQL Server-Authentifizierung**.

- **Benutzername** und **Kennwort**: Angabe der Zugangsdaten im Falle der SQL Server-Authentifizierung

- **Treiber**: Auswahl des Treibers für die Verbindungszeichenfolge. Diese können wiederum über eine Tabelle im Add-In verwaltet werden.

- **Verbindungszeichenfolge**: Stellt aus den gegebenen Informationen die Vorschau der Verbindungszeichenfolge zusammen.

Die Schaltflächen erlauben die folgenden Funktionen:

- **OK**: Schließen des Formulars
- **Testen**: Testaufruf der Verbindungszeichenfolge

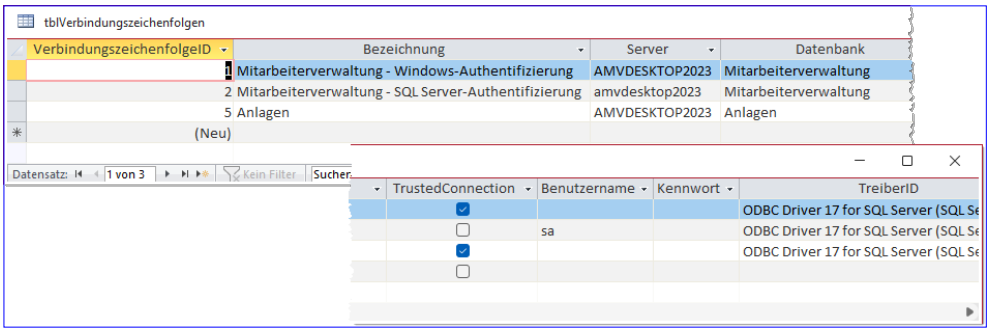


Bild 4: Die Tabelle **tblVerbindungszeichenfolgen**

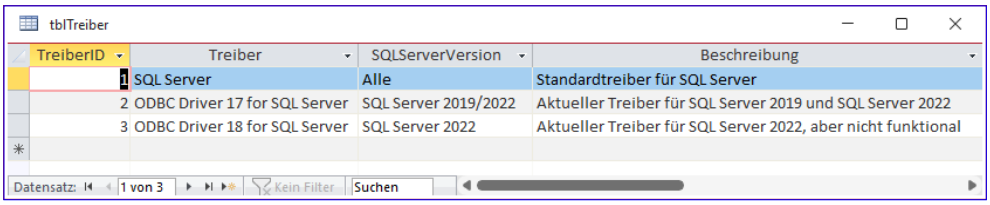


Bild 5: Die Tabelle **tblTreiber**

- **Als Standard**: Setzt die aktuelle Verbindungszeichenfolge als Standard
- **Neu**: Legt eine neue, leere Verbindungszeichenfolge an
- **Kopieren als...**: Kopiert die aktuelle Verbindungszeichenfolge als neue Zeichenfolge.
- **Löschen**: Löscht die aktuelle Zeichenfolge.

Oben finden wir noch das Kombinationsfeld **Schnellauswahl**. Hier werden alle verfügbaren Verbindungszeichenfolgen zur Auswahl angeboten.

Das Formular ist an die Tabelle **tblVerbindungszeichenfolgen** gebunden (siehe Bild 4).

Die Treiber, die über das Feld **TreiberID** ausgewählt werden können, stammen aus der Tabelle **tblTreiber** (siehe Bild 5).

Automatische Übernahme der Verbindungszeichenfolge

Wenn wir den Dialog zum Verwalten der Verbindungszeichenfolge vom Formular zum Verknüpfen der Tabellen aus öffnen, wird dort direkt die Verbindungszeichenfolge angezeigt, die wir im aufrufenden Formular ausgewählt haben. Legen wir hier eine neue Verbindungszeichenfolge an oder wählen eine andere aus, wird diese nach dem Schließen des Dialogs automatisch im aufrufenden Formular selektiert.

Datenbank merkt sich die Verbindungszeichenfolge

Außerdem wollten wir noch eine Funktion hinzufügen, die dafür sorgt, dass beim Öffnen von **amvTableLinker** auf irgendeine Weise gespeichert wird, welche Verbindungszeichenfolge zuletzt zum Verknüpfen von Tabellen verwendet wurde. Damit wollen wir die Voraussetzung schaffen, dass beim nächsten Öffnen des **amvTableLinkers** in der gleichen Datenbank automatisch die richtige Verbindungszeichenfolge ausgewählt wird.

Dies haben wir realisiert, indem wir die Bezeichnung der Verbindungszeichenfolge in einer benutzerdefinierten Eigenschaft der Datenbank speichern.

Beim Öffnen von **amvTableLinker** prüfen wir, ob es diese Eigenschaft gibt und falls ja, wählen wir die entsprechende Verbindungszeichenfolge aus.

```
Private Sub Form_Load()
    Dim db As DAO.Database
    Dim dbc As DAO.Database
    Dim prp As DAO.Property
    Dim rst As DAO.Recordset
    Dim strVerbindung As String
    Set db = CurrentDb
    Set dbc = CodeDb
    On Error Resume Next
    Set prp = db.Properties("Verbindung")
    On Error GoTo 0
    If Not prp Is Nothing Then
        strVerbindung = prp.Value
        Set rst = dbc.OpenRecordset("SELECT * FROM tblVerbindungszeichenfolgen WHERE Bezeichnung = '" _
            & strVerbindung & "'", dbOpenDynaset)
        If Not rst.EOF Then
            Me.cboVerbindung = rst!VerbindungszeichenfolgeID
            Call cboVerbindung_AfterUpdate
        End If
    End If
End Sub
```

Listing 1: Diese Prozedur wird beim Laden des Formulars **frmTabellenVerknuepfen** ausgelöst.

Programmieren der Formulare für amvTableLinker

Nun schauen wir uns an, wie wir die Formulare des Access-Add-Ins programmiert haben.

Wir beginnen mit dem Formular **frmTabellenVerknuepfen**, das in der Entwurfsansicht wie in Bild 6 aussieht.

Beim Laden des Formulars wird die Prozedur aus Listing 1 ausgelöst. Diese deklariert gleich zwei Variablen des Typs **DAO.Database**. Die erste namens **db** soll das **Database**-Objekt der Datenbank referenzieren, die das Add-In aufruft. Die zweite heißt **dbc** und soll das **Database**-Objekt der Add-In-Datenbank aufnehmen. Außerdem deklarieren wir Variablen für ein **Property**- und ein **Recordset**-Objekt.

Die beiden verschiedenen **Database**-Objekte füllen wir mit den Funktionen **CurrentDb** (für die aufrufende Datenbank) und **CodeDb** (für die Add-In-Datenbank).

Danach versuchen wir, die Eigenschaft **Verbindung** der aufrufenden Datenbank zu referenzieren.

In dieser haben wir gegebenenfalls zuvor die Bezeichnung der Verbindungszeichenfolge gespeichert, mit der wir die Tabellenverknüpfungen hergestellt haben. Dies erledigen wir bei deaktivierter Fehlerbehandlung, weil es auch sein

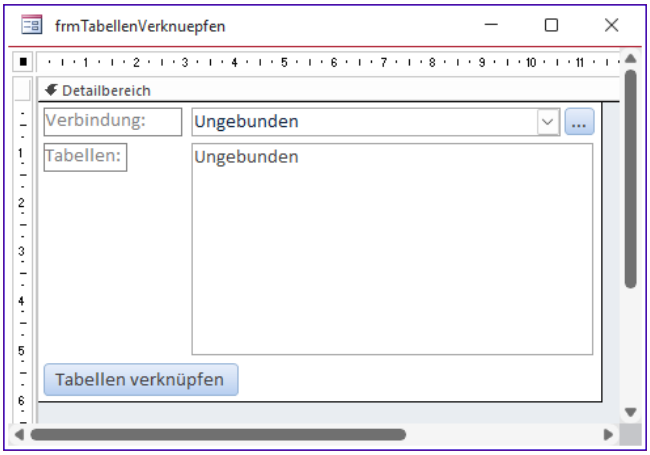


Bild 6: Entwurf des Formulars **frmTabellenVerknuepfen**

kann, dass diese Eigenschaft noch gar nicht vorliegt – und das würde wiederum einen Fehler auslösen, den wir somit umgehen.

Ob das Zuweisen an die **prp**-Variable erfolgreich war, prüfen wir anschließend, indem es mit **Is Nothing** vergleichen. In diesem Fall ist **prp** vorhanden und wir lesen den

```
Private Sub cboVerbindung_AfterUpdate()
    Dim db As DAO.Database
    Dim dbc As DAO.Database
    Dim qdf As DAO.QueryDef
    Dim rstTables As DAO.Recordset
    Dim strVerbindungszeichenfolge As String
    Dim lngErrorNumber As Long
    Dim strErrorDescription As String
    Dim var As Variant
    Set dbc = CodeDb
    Set db = CurrentDb
    If Len(strBenutzername) = 0 Then
        strBenutzername = Nz(dbc.OpenRecordset("SELECT Benutzername FROM tblVerbindungszeichenfolgen " _
        & "WHERE VerbindungszeichenfolgeID = " & Me!cboVerbindung).Fields(0), "")
    End If
    If Len(strKennwort) = 0 Then
        strKennwort = Nz(dbc.OpenRecordset("SELECT Kennwort FROM tblVerbindungszeichenfolgen " _
           & "WHERE VerbindungszeichenfolgeID = " & Me!cboVerbindung).Fields(0), "")
    End If
    If VerbindungTesten(Me!cboVerbindung, strVerbindungszeichenfolge, lngErrorNumber, strErrorDescription) = True Then
        Set qdf = db.CreateQueryDef("")
        With qdf
            .SQL = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE IN ('BASE TABLE', 'VIEW') " _
               & "ORDER BY TABLE_TYPE, TABLE_NAME"
            .Connect = strVerbindungszeichenfolge
            .ReturnsRecords = True
        End With
        Set Me!lstTabellen.Recordset = qdf.OpenRecordset
        Set rstTables = db.OpenRecordset("SELECT Name FROM MSysObjects WHERE NOT Connect IS NULL", dbOpenDynaset)
        Do While Not rstTables.EOF
            For var = 0 To Me.lstTabellen.ListCount - 1
                If Me!lstTabellen.ItemData(var) = rstTables!Name Then
                    Me.lstTabellen.Selected(var) = True
                End If
            Next var
            rstTables.MoveNext
        Loop
    Else
        MsgBox "Keine gültige Verbindung." & vbCrLf & vbCrLf & strErrorDescription
    End If
End Sub
```

Listing 2: Anzeigen aller Tabellen der gewählten Verbindungszeichenfolge

Wert über die Eigenschaft **Value** in die Variable **strVerbindung** ein.

Damit können wir nun das Recordset **rst** füllen, und zwar mit einen Recordset auf Basis der Tabelle **tblVerbindungszeichenfolgen**. Für dieses legen wir fest, dass das Feld **Bezeichnung** mit dem Wert aus **strVerbindung** übereinstimmen soll. Das Recordset öffnen wir dabei mit der **OpenRecordset**-Methode der **CodeDb**-Datenbank aus **dbc**.

Ist das Recordset anschließend nicht leer, stellen wir das Kombinationsfeld **cboVerbindung** auf den Primärschlüsselwert des Recordsets ein und rufen die Prozedur **cboVerbindung_AfterUpdate** auf, die auch nach der manuellen Auswahl einer Verbindungszeichenfolge ausgelöst wird.

Anzeigen der Tabellen einer Verbindungszeichenfolge

Die oben erwähnte Prozedur **cboVerbindung_AfterUpdate** soll nach dem Selektieren eines Eintrags dieses Kombinationsfeldes alle Tabellen der Datenbank anzeigen, die in dieser Verbindungszeichenfolge referenziert wird.

Die Prozedur sehen wir in Listing 2. Sie verwendet ebenfalls zwei **Database**-Objektvariablen. Außerdem deklariert sie eine **QueryDef**-Variable und eine **Recordset**-Variable. Daneben benötigen wir eine **String**-Variable für die Verbindungszeichenfolge sowie eine **Variant**-Variable zum Durchlaufen der Listenfeld-Einträge.

Nach dem Füllen der **Database**-Variablen ermitteln wir die eventuell festgelegten Daten für die Anmeldung an die Datenbank und speichern diese in den Variablen **strBenutzername** und **strKennwort**, die als öffentliche Variablen im Modul **mdlSQLServer** deklariert sind. Hier würden wir normalerweise der Einfachheit halber **DLookup**-Funktionen nutzen. Aber damit würden wir immer auf die Datenbank zugreifen, die das Add-In aufgerufen hat. Daher müssen wir hier die **OpenRecordset**-Methode für das mit der Variablen **dbc** referenzierte **Database**-Objekt der Add-In-Datenbank verwenden.

Anschließend ruft die Prozedur die Funktion **VerbindungTesten** auf und übergibt den Primärschlüsselwert der zu untersuchenden Verbindungszeichenfolge sowie die beiden Variablen **lngErrorNumber** und **strErrorDescription** für eventuelle Rückmeldungen über Fehler und erwartet für den Parameter **strVerbindungszeichenfolge** die entsprechende Zeichenfolge zurück. Diese Funktion beschreiben wir weiter unten.

War der Aufruf erfolgreich, erstellt die Prozedur mit der **CreateQueryDef**-Methode ein neues, temporäres **QueryDef**-Objekt. Das erreichen wir, indem wir für den Parameter **Name** eine leere Zeichenkette übergeben.

Dieser weisen wir nun über die Eigenschaft **SQL** eine SQL-Anweisung zu, die alle Tabellennamen aus der Systemtabelle **Information_Schema.Table** holt, deren Typ **BASE TABLE** oder **VIEW** lautet. Als Verbindungszeichenfolge geben wir für die Eigenschaft **Connect** den Wert aus **strVerbindungszeichenfolge** an. Schließlich stellen wir die Eigenschaft **ReturnsRecords** auf **True** ein, damit die Abfrage Datensätze zurückliefert (im Gegensatz zu einer Aktionsabfrage).

Das mit der **OpenRecordset**-Methode auf Basis des **QueryDef**-Objekts aus der Variablen **qdf** geöffnete Recordset weisen wir direkt der gleichnamigen Eigenschaft des Listenfeldes **lstTabellen** zu.

Außerdem erstellen wir ein Recordset namens **rstTables**, das wir mit Datensätzen der Systemtabelle **MSysObjects** der aufrufenden Access-Datenbank füllen. Diese beschränken wir auf alle Datensätze, deren Eigenschaft **Connect** nicht leer ist – was in der Regel die per ODBC eingebundenen Tabellen eindeutig identifiziert.

Dieses Recordset durchlaufen wir in einer **Do While**-Schleife. Innerhalb der Schleife durchlaufen wir in

einer **For...Each**-Schleife alle Elemente des Listenfeldes **lstTabellen**. Dabei vergleichen wir jeweils den Eintrag des Listenfeldes mit dem Namen der aktuellen Tabelle aus dem Recordset.

Sind beide gleich, bedeutet dies, dass die Tabelle aus dem SQL Server bereits in Form einer Tabellenverknüpfung in der aufrufenden Datenbank hinterlegt ist. In diesem Fall soll dieser Eintrag im Listenfeld selektiert werden, was wir durch Einstellen der Eigenschaft **Selected** für das Element **var** auf den Wert **True** erledigen.

Auf diese Weise füllen wir das Listenfeld **lstTabellen** und markieren gleichzeitig alle bereits verknüpften Tabellen. Dies gelingt übrigens nur, wenn der Name der Tabelle im SQL Server mit dem Namen der Verknüpfung übereinstimmt.

Testen der Verbindung

Damit wir Tabellenverknüpfungen herstellen können, benötigen wir eine funktionsfähige Verbindungszeichenfolge. Diese prüfen wir zuvor mit der Funktion **VerbindungTesten**, die wir in Listing 3 finden. Sie nimmt den Primärschlüsselwert des Eintrags der Tabelle **tblVerbin-**

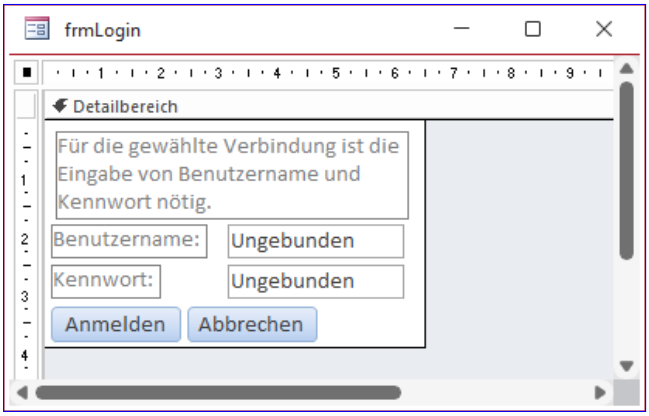


Bild 7: Entwurf des Formulars **frmLogin**

```
Public Function VerbindungTesten(lngVerbindungszeichenfolgeID As Long, _
        Optional strVerbindungszeichenfolge As String, Optional lngErrorNumber As Long, _
        Optional strErrorDescription As String) As Boolean
    Dim dbc As dao.Database
    Dim bolTrustedConnection As Boolean
    Dim bolVerbindungHergestellt As Boolean
    Set dbc = CodeDb
    bolTrustedConnection = dbc.OpenRecordset("SELECT TrustedConnection FROM tblVerbindungszeichenfolgen " _
        & "WHERE VerbindungszeichenfolgeID = " & lngVerbindungszeichenfolgeID).Fields(0)
    If (Len(strBenutzername) * Len(strKennwort) = 0) And Not bolTrustedConnection Then
        If LogindatenErmitteln(lngVerbindungszeichenfolgeID) = False Then
            Exit Function
        End If
    End If
    strVerbindungszeichenfolge = VerbindungszeichenfolgeNachID(lngVerbindungszeichenfolgeID)
    On Error Resume Next
    bolVerbindungHergestellt = VerbindungHerstellen(strVerbindungszeichenfolge, lngErrorNumber, strErrorDescription)
    If bolVerbindungHergestellt = False Then
        VerbindungTesten = False
        Exit Function
    End If
    VerbindungTesten = True
End Function
```

Listing 3: Testen der Verbindung für die Verbindungszeichenfolge

Dateidownload per API

Manchmal kommt es vor, dass man einen oder mehrere Downloads durchführen muss. Normalerweise führt man diesen Download durch einen Klick auf den entsprechenden Link durch. Die Datei landet danach üblicherweise im Download-Ordner, von wo wir ihn dann zum gewünschten Ort verschieben. Wenn wir zuvor die URL der herunterzuladenden Datei kennen und auch wissen, in welchen Ordner und unter welchem Namen die Datei im System landen soll, können wir dies auch per VBA realisieren. Dazu benötigen wir nur eine einfache API-Funktion und ihren Aufruf.

Um einen einfachen Download zu realisieren, können wir die API-Funktion **URLDownloadToFile** nutzen. Diese deklarieren wir wie in Listing 1. Außerdem finden wir dort eine Funktion zur Vereinfachung des Aufrufs. Dieser brauchen wir nur noch die URL zu der herunterzuladenden Datei zu übermitteln sowie den Pfad der Zieldatei.

Ein solcher Aufruf könnte wie folgt aussehen:

```
DownloadFile "https://access-im-unternehmen.de/aiu_1533.zip", CurrentProject.Path & "\aiu_1533.zip"
```

Wenn der Download erfolgreich war, landet die Datei genau im angegebenen Pfad.

Auf diese Weise können wir leicht immer wieder auftretende Downloads automatisieren.

```
#If VBA7 Then
    Private Declare PtrSafe Function URLDownloadToFile Lib "urlmon" Alias "URLDownloadToFileA" ( _
        ByVal pCaller As LongPtr, ByVal szURL As String, ByVal szFileName As String, _
        ByVal dwReserved As Long, ByVal lpfnCB As LongPtr) As Long
#Else
    Private Declare Function URLDownloadToFile Lib "urlmon" Alias "URLDownloadToFileA" ( _
        ByVal pCaller As Long, ByVal szURL As String, ByVal szFileName As String, _
        ByVal dwReserved As Long, ByVal lpfnCB As Long) As Long
#End If

Public Sub DownloadFile(strURL As String, strPath As String)
    Dim lngResult As Long

    lngResult = URLDownloadToFile(0, strURL, strPath, 0, 0)

    If lngResult = 0 Then
        MsgBox "Download erfolgreich!", vbInformation
    Else
        MsgBox "Download fehlgeschlagen!", vbCritical
    End If
End Sub
```

Listing 1: Die Deklaration der API-Funktion und eine Wrapper-Funktion