HAUFE.
Ausgabe 06/2024

ACCESS
IM UNTERNEHMEN

ÜBERSICHTSFORMULARE PER MAUSKLICK

Erstellen Sie mit wenigen Mausklicks vollständige Formulare zur Übersicht über die Daten einer Tabelle oder Abfrage (ab Seite 46).

In diesem Heft:

FRONTEND AUTOMATISCH AKTUALISIEREN

Lernen Sie eine Technik kennen, um neue Frontendversionen automatisch auf die Benutzerrechner zu laden.

SEITE 2

BERICHTE ALS PDF ERSTELLEN

Nutzen Sie die Methoden der Benutzeroberfläche und von VBA zum Erstellen von PDF-Berichten.

SEITE 16, 34

VBA-PROJEKTE SIGNIEREN

Machen Sie Ihre Anwendungen sicherer mit einer digitalen Signatur.

SEITE 22

Übersichtsformulare schnell erstellen

Es gibt nichts Langweiligeres beim Programmieren mit Microsoft Access, als immer wieder die gleichen Handgriffe zu tätigen. Und wenn man viele Anwendungen entwickelt, gibt es eine Menge davon. Zum Beispiel das Erstellen von Detailformularen, Übersichtsformularen oder auch von m:n-Formularen. Um die Detailformulare haben wir uns bereits in Ausgabe 2/2024 gekümmert. Nun kommen die Übersichtsformulare an die Reihe: Formulare, welche die Daten einer Tabelle oder Abfrage im Unterformular anzeigen und außerdem noch Steuerelemente enthalten, mit denen die angezeigten Daten durchsucht, ergänzt, bearbeitet oder gelöscht werden können.

Wie das mit wenigen Mausklicks gelingt, erfahren Sie im Beitrag **Übersichtsformular per Mausklick** ab Seite 46. Zum Erstellen eines vollständigen Übersichtsformulars benötigen wir nur wenige Daten, die wir mit einem Konfigurationsformular eingeben können. Anschließend reicht ein Klick auf die »Erstellen«-Schaltfläche, um Access den Rest der Arbeit erledigen zu lassen. Das Beste ist: Die Lösung aus diesem Beitrag brauchen Sie einfach nur zu installieren und können es dann in jeder beliebigen Access-Anwendung über die Benutzeroberfläche einsetzen.

In einem weiteren Beitrag namens **Add-In für Übersichtsformulare im Einsatz** schauen wir uns ab Seite 69 an, wie wir das Optimum aus diesem Add-In herausholen können. Dabei betrachten wir auch, wie der Code aufgebaut sein muss, den wir vom Add-In automatisch zum Formular hinzufügen lassen.

Ein oft gefragtes Thema ist die Aktualisierung der Datenbank-Frontends von Anwendungen, die auf mehreren Arbeitsplätzen verteilt sind. Wie lässt sich das technisch realisieren, ohne dass man jeden Benutzer persönlich besuchen oder sich zumindest auf seinen Rechner aufschalten muss?

Das gelingt mit den Techniken, die wir im Beitrag **Access-Frontend automatisch aktualisieren** ab Seite 2 vorstellen. Hier schauen wir uns an, wie wir prüfen können, ob es eine neue Version gibt und diese dann automatisch über die vorhandene Version installieren.

Immer wieder interessant ist auch das Thema **PDF erstellen mit Access im Detail**, in dieser Ausgabe ab Seite 16 behandelt. In diesem Fall schauen wir uns auch einmal genauer an, ob und wie wir mit Access Berichte im Format PDF/A-3 erzeugen können. Dieses Format wird für die Erstellung von ZUGFeRD-Rechnungen benötigt und tatsächlich können wir dieses mit aktuelleren Access-Versionen erstellen.

Natürlich schauen wir uns nicht nur den Weg über die Benutzeroberfläche an, sondern auch den per VBA – und zwar ab Seite 34 im Beitrag **PDF per VBA erstellen im Detail**. Per VBA ist das Erstellen von PDF/A-3-Dokumenten so einfach wie über die Benutzeroberfläche und wir benötigen hier eine spezielle Technik. Dazu gehört der Einsatz eines gespeicherten Exports. Was das ist und wie wir mit gespeicherten Importen und Exporten arbeiten können, zeigen wir unter dem Titel **Gespeicherte Importe und Exporte verwalten** ab Seite 10.

Schließlich wollen wir noch einen Blick auf die wiederbelebte Funktion zum digitalen Signieren von VBA-Projekten werfen. Alles zu diesem Thema finden Sie im Beitrag **VBA-Projekte mit digitalem Zertifikat signieren** ab Seite 22.

Viel Spaß beim Lesen!

Ihr André Minhorst

Access-Frontend automatisch aktualisieren

Wenn man allein mit einer Access-Anwendung arbeitet, die man währenddessen weiterentwickelt, ist man immer auf dem aktuellsten Stand. Wer eine Anwendung auf die Arbeitsplätze seiner eigenen Mitarbeiter oder sogar die der Mitarbeiter eines Kunden installiert hat, muss schon ein wenig Aufwand betreiben, wenn jeder immer die aktuellste Version der Anwendung vorfinden soll. Die alte Version muss entfernt und die neue installiert werden. Hier kann man die Benutzer informieren, sodass diese die notwendigen Schritte manuell durchführen oder man erledigt dies selbst. Praktischer wäre es jedoch, wenn die Anwendung selbst erkennen würde, wenn es eine neue Version gibt, und dann selbstständig ein Update durchführt. Das geht natürlich nicht, weil sie sich nicht selbst löschen und durch eine neue Version ersetzen kann, während sie läuft. Also benötigen wir einen kleinen Workaround: Wir stellen dem Öffnen der eigentlichen Anwendung eine weitere Access-Anwendung voran, die auf Updates prüft und diese durchführt und erst dann die eigentliche Anwendung aufruft. Alle notwendigen Schritte erläutern wir in diesem Beitrag.

Varianten von Frontend-Updatern

Eine solche vorgeschaltete Anwendung nennt man auch Frontend-Updater. Dabei muss es sich nicht zwingend um eine Access-Anwendung handeln – wenn über die notwendigen Programmierkenntnisse und Tools verfügt, kann man dies auch in Form einer Exe-Datei auf Basis VB6, VB.NET oder twinBASIC erledigen.

Die Aufgabe bleibt jedoch gleich: Der Frontend-Updater soll vor der eigentlichen Access-Anwendung geöffnet werden und prüfen, ob die derzeit auf dem Rechner befindliche Frontend-Datenbank die aktuellste verfügbare Version ist. Ist diese Bedingung erfüllt, kann die Frontend-Datenbank geöffnet werden. Falls nicht, soll die vorhandene Frontend-Datenbank gelöscht (oder zumindest umbenannt) werden und die neue Version der Frontend-Datenbank soll stattdessen in das entsprechende Verzeichnis kopiert werden. Danach wird entweder die vorhandene oder die neu hinzugefügte Frontend-Datenbank gestartet.

Hier ergeben sich einige technische Herausforderungen:

- Wie finden wir heraus, ob sich die aktuellste Version auf dem aktuellen Rechner befindet?
- Wenn nicht – wie kopieren wir die aktuellere Version auf den Rechner und löschen die vorhandene Version oder benennen diese um?
- Wie können wir von einer Access-Datenbank aus eine andere Access-Datenbank starten?
- Welche Voraussetzungen sind für ein solches Vorgehen überhaupt erforderlich?

Voraussetzungen für ein automatisches Update des Frontends

Ein Update kann man nicht in jedem Fall ohne Weiteres über die vorhandene Version kopieren und dann einfach weiterarbeiten.

Die erste Voraussetzung ist, dass die Daten, die von den Frontends aus bearbeitet werden, sich in einer Backend-Datenbank befinden. Das sollte aber im Mehrbenutzerbe-

trieb ohnehin der Fall sein. Gegebenenfalls befinden sich die Tabellen mit den Geschäftsdaten sogar in einer SQL Server-Datenbank.

Die zweite Voraussetzung ist, dass sich auch keine benutzerdefinierten Daten in der Frontend-Datenbank befinden. Es kommt beispielsweise oft vor, dass der Benutzer Optionen für die Anwendung selbst einstellen kann. So findet er beim Öffnen der Anwendung auf seinem Desktop-Rechner immer die Konfiguration vor, die er selbst eingestellt hat. Solche Daten speichert man üblicherweise in einer oder mehreren Optionen-Tabellen in der Frontend-Datenbank.

Um dies zu umgehen, gibt es beispielsweise die folgenden Alternativen:

- Die Einstellungen für den jeweiligen Benutzer werden in einer Optionentabelle im Backend auf dem Server gespeichert, wobei dort noch ein Feld hinzuzufügen ist, dass kenntlich macht, zu welchem Benutzer die Einstellungen gehören.

- Die Einstellungen werden in einem weiteren Backend gespeichert, das sich im gleichen Verzeichnis wie das Frontend des Benutzers befindet. Auf diese Weise können wir das Frontend austauschen, ohne dass die Einstellungen des Benutzers verlorengehen. Wir müssen nur sicherstellen, dass sich eine neue Version des Frontends automatisch wieder mit dem Backend mit den Optionentabellen verknüpft.

- Oder wir speichern Optionen gar nicht in einer Tabelle, sondern in der Registry in dem für den aktuellen Benutzer für VBA-Anwendungen vorgesehenen Bereich. Wie das gelingt, zeigen wir beispielsweise im Beitrag **Optionen einfach in der Registry speichern** (**www.access-im-unternehmen.de/1512**).

Information über die aktuelle Version speichern

Wenn wir vergleichen wollen, ob die auf dem Server liegende Version des Datenbank-Frontends aktueller ist, als die vorliegende Version, müssen wir irgendwo in den Datenbankdateien die Versionsnummer speichern.

Zunächst einmal schlagen wir vor, hier einfach eine durchlaufende Nummer zu verwenden. Auf diese Weise können wir am einfachsten vergleichen, welche Version die aktuellere ist.

Dann benötigen wir einen geeigneten Ort für diese Information. Dazu gibt es mindestens die folgenden beiden Vorschläge:

- Speichern der Versionsnummer in einer Optionentabelle der Frontends

- Speichern der Versionsnummer in einer Eigenschaft der Access-Datenbank

Beide Informationen können wir einfach per VBA abfragen und manuell oder per Code setzen. Später beschreiben wir die technische Umsetzung.

Prüfen, ob die vorliegende Version die aktuellste Version ist

Wenn sich sowohl im vorliegenden Frontend als auch in der Version auf dem Server die Information über die Version entweder in einer Tabelle oder in einer Eigenschaft befindet, können wir diese einfach einlesen und vergleichen.

In beiden Fällen müssen wir von der Frontend-Updater-Anwendung aus per VBA auf die jeweilige Version zugreifen. Wenn wir, wie oben vorgeschlagen, eine einfache Versionsnummer wählen, können wir diese auch ganz einfach vergleichen und entscheiden, ob die Version vom Server die aktuelle Version ersetzen soll.

Umbenennen/Löschen der alten Version und Kopieren der neuen Version

Wenn wir herausgefunden haben, dass sich auf dem Server eine aktuellere Version der Frontend-Datenbank befindet, sind die folgenden Schritte durchzuführen:

- Löschen oder Umbenennen der vorliegenden Version, wobei wir das Umbenennen bevorzugen würden. Auf diese Weise könnten wir die vorliegende Version wiederherstellen, wenn das Update aus irgendeinem Grund nicht funktioniert. Beim Umbenennen könnten wir einfach die aktuelle Version hinten an den Dateinamen anhängen. Das Umbenennen oder auch das Löschen lässt sich per VBA realisieren.

- Kopieren der neueren Version vom Server auf den aktuellen Rechner. Dies gelingt ebenfalls mit einem einfachen VBA-Befehl.

Starten der eigentlichen Datenbank von der Updater-Anwendung aus

Schließlich fehlt noch der Schritt, der bei jedem Start durchgeführt werden muss: Die Frontend-Updater-Datenbank muss die eigentliche Datenbank starten, unabhängig davon, ob diese zuvor aktualisiert wurde oder nicht.

Testszenario

Zum Testen verwenden wir zwei gleiche Datenbanken, die lediglich die Tabelle mit der Versionsnummer enthalten. Die zu prüfende und gegebenenfalls zu ersetzende Datenbank liegt in einem Verzeichnis mit der Frontend-Updater-Datenbank.

Statt eines Servers verwenden wir ein Verzeichnis namens **Server**, das sich im gleichen Verzeichnis wie die beiden zuvor genannten Datenbankdateien befindet.

Die Struktur sieht also wie folgt aus:

```
Frontend.accdb
FrontendUpdater.accdb
Server
  Frontend.accdb (neuere Version)
```

Die **Frontent.accdb**-Datenbanken enthalten eine Tabelle namens **tblVersion** mit einem einzigen Feld mit dem Datentyp **Zahl** und dem Namen **Version**.

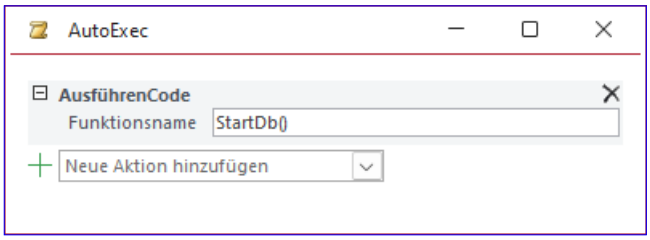


Bild 1: **AutoExec**-Makro, das beim Start automatisch ausgeführt wird

Nun bearbeiten wir die Datenbank **FrontendUpdater.accdb**, damit diese die notwendige Prüfung durchführt, gegebenenfalls das Frontend ersetzt und dieses dann startet.

Code beim Starten automatisch ausführen

Die Voraussetzung für eine automatische Prüfung beim Start der Datenbank **FrontendUpdater.accdb** ist, dass diese beim Starten den entsprechenden Code ausführt. Das können wir am einfachsten mit einem **AutoExec**-Makro realisieren, der wir den Namen der zu startenden VBA-Funktion für die Makroaktion **AusführenCode** übergeben (siehe Bild 1).

Die Funktion **StartDb** können wir testhalber zunächst wie folgt formulieren:

```
Public Function StartDb()
    MsgBox "Start"
End Function
```

Starten wir die Datenbank nun neu, sollte das Meldungsfenster erscheinen.

Version des vorliegenden Frontends einlesen

Mit der Funktion **GetVersion** aus Listing 1, der wir den Pfad zu der zu untersuchenden Datenbankdatei übergeben, erstellen wir ein **Database**-Objekt auf Basis der mit **OpenDatabase** geöffneten Datenbankdatei. Für diese öffnen wir ein Recordset auf Basis der Tabelle **tblVersion**. Enthält diese zumindest einen Datensatz, lesen wir daraus die Version aus und geben diese als Funktionswert zurück. Anderenfalls geben wir eine entsprechende Meldung aus.

Die zweite Version von GetVersion ist noch ein wenig kürzer (siehe Listing 2). Hier integrieren wir den Namen der zu untersuchenden Datenbank in die **OpenRecordset**-Methode. Dadurch sparen wir das Initialisieren des **Database**-Objekts für die Frontend-Datenbank.

```
Public Function GetVersion(strFrontend As String) As Long
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = OpenDatabase(strFrontend)
    Set rst = db.OpenRecordset("SELECT Version FROM tblVersion")
    If Not rst.EOF Then
        GetVersion = rst!Version
    Else
        MsgBox "Keine Versionsnummer in '" & strFrontend & "' gefunden."
    End If
End Function
```

Listing 1: Einlesen der Version, längere Variante

Egal, welche Version wir wählen – wir können damit sowohl die Version des vorliegenden Frontends als auch die der Version vom Server ermitteln und dann in per **If...Then**-Bedingung ermitteln, welche Aktion durchzuführen ist. Dazu nutzen wir die neue Version der Prozedur **StartDb** aus Listing 3.

```
Public Function GetVersion1(strFrontend As String)
    Dim rst As DAO.Recordset
    Set rst = CurrentDb.OpenRecordset("SELECT Version FROM tblVersion IN '" & strFrontend & "'")
    If Not rst.EOF Then
        GetVersion1 = rst!Version
    Else
        MsgBox "Keine Versionsnummer in '" & strFrontend & "' gefunden."
    End If
End Function
```

Listing 2: Einlesen der Version, kürzere Variante

```
Public Function StartDb()
    Dim strCurrent As String
    Dim strServer As String
    Dim lngVersionCurrent As Long
    Dim lngVersionServer As Long
    strCurrent = CurrentProject.Path & "\Frontend.accdb"
    strServer = CurrentProject.Path & "\Server\Frontend.accdb"
    lngVersionCurrent = GetVersion(strCurrent)
    lngVersionServer = GetVersion1(strServer)
    If lngVersionCurrent < lngVersionServer Then
        MsgBox "Muss von Version '" & lngVersionCurrent & "' auf Version '" & lngVersionServer & "' aktualisiert werden."
    Else
        MsgBox "Keine Aktualisierung nötig."
    End If
End Function
```

Listing 3: Vergleichen der Version und Ausgabe der zu treffenden Maßnahme

Gespeicherte Importe und Exporte verwalten

Microsoft Access bietet die Möglichkeit, nach Abschluss eines Imports oder Exports die Importschritte oder Exportschritte zu speichern. Dazu erscheint ein Dialog, in dem man wenige Parameter eingestellt werden können. Diese gespeicherten Importe und Exporte kann man anschließend wiederholen, indem man diese über einen entsprechenden Dialog auswählt. Wir wollen uns in diesem Beitrag einmal ansehen, was hier überhaupt geschieht, wo die Daten gespeichert werden und wie wir diese gegebenenfalls anpassen können, ohne dass wir den Import oder Export erneut durchführen müssen, um die geänderte Konfiguration zu erhalten. Und wie sich zeigen wird, gibt es sogar eine Erweiterung des Objektmodells von VBA zu diesem Zweck, das wir uns genau ansehen werden.

Importe und Exporte können wir in Access auf verschiedene Arten starten. Der offensichtlichste Web ist der über die Befehle im Ribbon, die wir unter **Externe Daten** finden (siehe Bild 1).

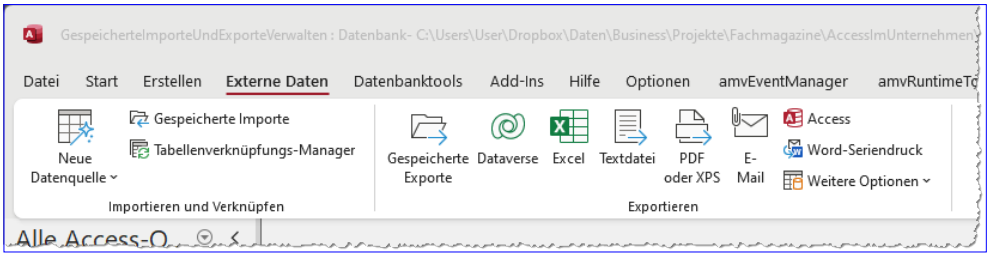


Bild 1: Starten von Import-, Verknüpfungs- und Exportvorgängen über das Ribbon

Am Ende eines jeden dieser Vorgänge erscheint die Meldung aus Bild 2.

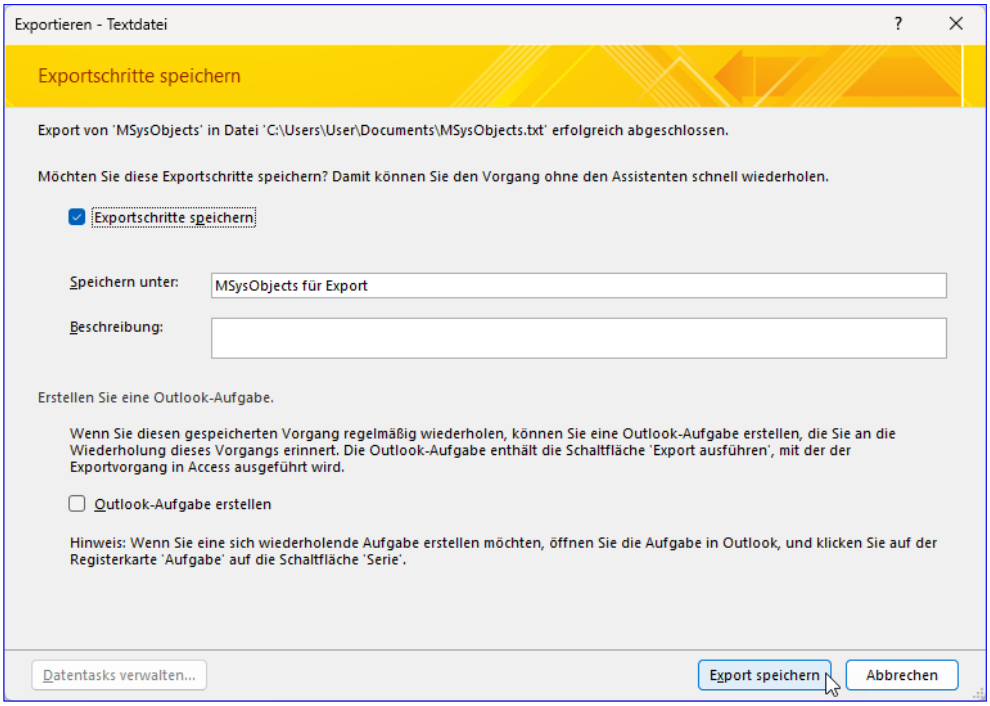


Bild 2: Dialog zum Speichern der Exportschritte

Hier klicken wir zunächst auf **Exportschritte speichern**, um die übrigen Optionen einzublenden. Diese sind recht begrenzt: Wir geben einfach den Namen ein, unter dem wir den Import oder Export später wieder aufrufen können. Außerdem können wir noch eine Beschreibung hinzufügen.

Weiter unten können wir noch die Option Outlook-

Aufgabe erstellen aktivieren. Diese sorgt dafür, dass eine neue Outlook-Aufgabe erstellt wird, die jedoch lediglich einen Text mit einer Anleitung enthält, wie der Import oder Export in Access gestartet werden kann (siehe Bild 3).

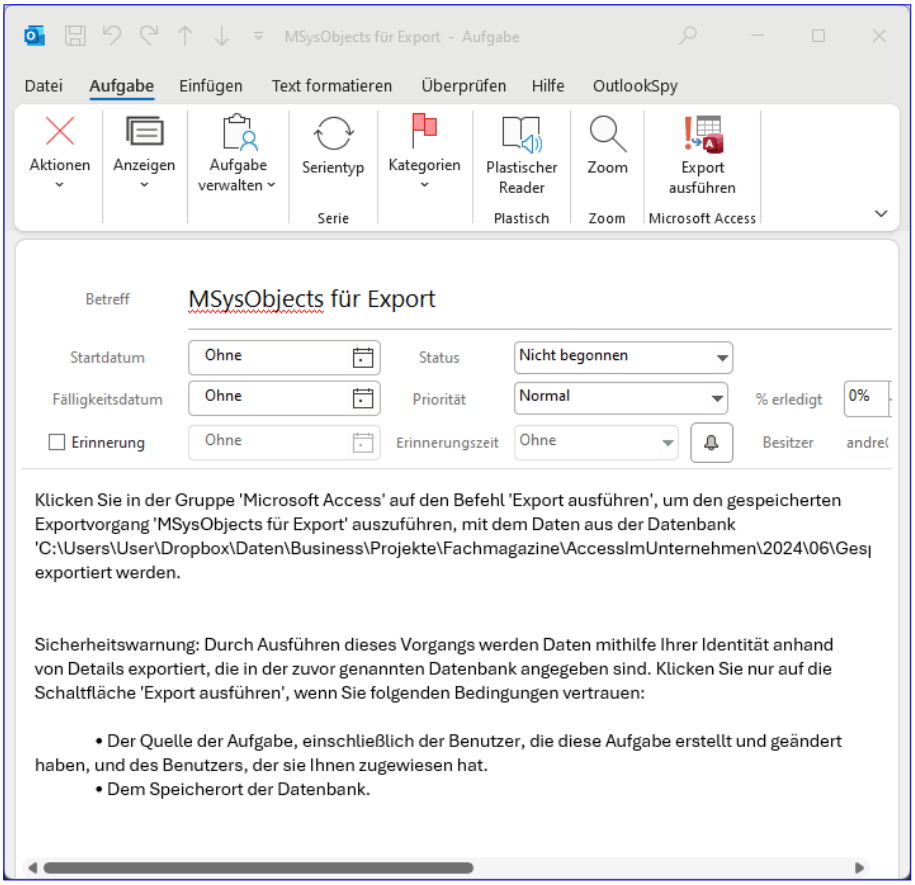


Bild 3: Outlook-Aufgabe mit einer Anleitung für den Import oder Export

Import oder Export erneut starten

Wenn wir diesen gespeicherten Import oder Export nun erneut ausführen wollen, betätigen wir im Ribbon unter **Externe Daten** eine der Schaltflächen **Gespeicherte Importe** oder **Gespeicherte Exporte** (siehe Bild 4).

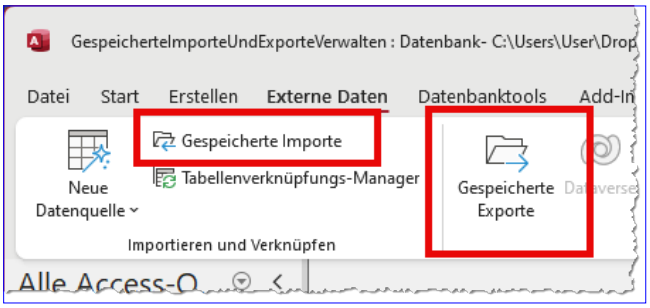


Bild 4: Anzeigen der gespeicherten Importe und Exporte

Dies öffnet in beiden Fällen den gleichen Dialog, der jeweils eine Registerkarte für die Importe und die Exporte enthält (siehe Bild 5).

Klicken wir doppelt auf einen der Einträge, wird der Import oder Export erneut ausgeführt. Dazu können wir den gewünschten Eintrag auch zuvor markieren und dann die Schaltfläche **Ausführen** betätigen. Wir können auch hier wieder eine Outlook-Aufgabe auf Basis des Imports oder Exports erstellen oder den Import oder Export löschen.

Bild 5: Anzeigen der gespeicherten Importe und Exporte per Ribbonbutton

Gespeicherte Importe und Exporte per VBA

Spannend wird es, wenn wir uns ansehen, wie wir die gespeicherten Importe oder Exporte per VBA nutzen und verwalten können. Wenn wir im VBA-Editor im Objektkatalog nach Elementen suchen, die Import oder Export im Namen tragen, finden wir die folgenden interessanten Einträge:

- **RunSavedImportExport**: Methode der **DoCmd**-Klasse, mit der wir einen gespeicherten Import oder Export starten können
- **acCmdSavedExports**: **RunCommand**-Parameter. Öffnet den Dialog zur Anzeige von gespeicherten Importen und Exporten und aktiviert die Registerseite mit den Exporten.
- **acCmdSavedImports**: **RunCommand**-Parameter. Öffnet den Dialog zur Anzeige von gespeicherten Importen und Exporten und aktiviert die Registerseite mit den Importen.
- **ImportExportSpeficications**: Auflistung der **CurrentProject**- und der **CodeProject**-Klasse. Erlaubt den Zugriff auf die gespeicherten Importe und Exporte.
- **ImportExportSpecification**: Klasse, über die wir auf die Eigenschaften von gespeicherten Importen oder Exporten zugreifen können.

Schauen wir uns an, was wir mit den verschiedenen VBA-Befehlen anstellen können.

Anzeigen des Dialogs zur Anzeige von gespeicherten Importen

Mit dem folgenden Befehl zeigen wir per VBA den Dialog zur Anzeige von gespeicherten Importen an:

```
RunCommand acCmdSavedImports
```

Anzeigen des Dialogs zur Anzeige von gespeicherten Exporten

Mit dem folgenden Befehl zeigen wir per VBA den Dialog zur Anzeige von gespeicherten Exporten an:

```
RunCommand acCmdSavedExports
```

Aufrufen eines gespeicherten Imports oder Exports per VBA

Gegebenenfalls soll der Benutzer selbst Importe oder Exporte durchführen können, die wir zuvor angelegt haben. Da in professionellen Access-Anwendungen die eingebauten Ribbons nicht angezeigt werden, fügen wir entsprechende Steuerelemente im Ribbon oder in Formularen ein.

Wenn wir einen Import oder Export mit einem bestimmten Namen per VBA ausführen wollen, verwenden wir beispielsweise den folgenden Befehl:

```
DoCmd.RunSavedImportExport "Name des Exports oder Imports"
```

Arbeiten mit den gespeicherten Importen oder Exporten per VBA

Auf die einzelnen gespeicherten Import- und Exportspezifikationen können wir über die Auflistung **ImportExportSpecifications** zugreifen. Die enthaltenen Elemente haben den Typ **ImportExportSpecification**. Um die Namen der verfügbaren Import- und Exportspezifikationen auszugeben, können wir beispielsweise die folgende Prozedur verwenden:

```
Public Sub ImporteUndExporte()
    Dim objSpecification As ImportExportSpecification
    For Each objSpecification In _
            CurrentProject.ImportExportSpecifications
        Debug.Print objSpecification.Name
    Next objSpecification
End Sub
```

PDF erstellen mit Access im Detail

Das Erstellen von PDF-Dateien mit Access ist eine wichtige Technik, denn viele Daten sollen einmal über einen Bericht in eine PDF-Datei exportiert werden. Das Paradebeispiel dafür sind Angebote, Aufträge, Rechnungen et cetera, also alles, was mit der Verarbeitung und Abrechnung eines Kaufs oder eines Projekts zusammenhängt. Aber es gibt auch andere Anwendungszwecke. In diesem Beitrag schauen wir uns die verschieden nativen Möglichkeiten von Access zum Erstellen von PDF-Dokumenten an und gehen dabei auch auf Spezialitäten wie das Format PDF/A ein und wie wir Dokumente in diesem Format erstellen können. Es ist nicht schon immer selbstverständlich, dass wir dies mit Access erledigen können – lange waren dafür spezielle PDF-Druckertreiber von Drittherstellern oder andere Tricks nötig. Seit einiger Zeit jedoch lässt sich dies mit dem integrierten Druckertreiber erledigen.

Früher waren einige Verrenkungen nötig, um einen Bericht in ein PDF-Dokument zu speichern. Heute ist dies alles kein Problem mehr. Um dies zu demonstrieren, haben wir einen sehr einfachen Bericht erzeugt, der in der Entwurfsansicht wie in Bild 1 aussieht.

PDF-Dokument über die Benutzeroberfläche erstellen

Diesen können wir nun auf verschiedene Arten in Form eines PDF-Dokuments speichern. Die einfachste Methode ist scheinbar der Aufruf des Befehls **Drucken** aus dem Kontextmenü des Eintrags für den Bericht im Navigationsbereich (siehe Bild 2). Dies druckt den Bericht jedoch mit dem Standarddrucker aus. Ist dieser nicht auf den PDF-Druckertreiber eingestellt, landet der Bericht gegebenenfalls direkt auf einem Blatt Papier.

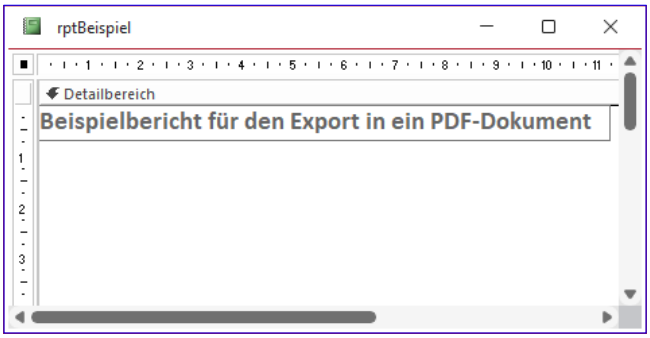


Bild 1: Entwurf eines Beispielberichts

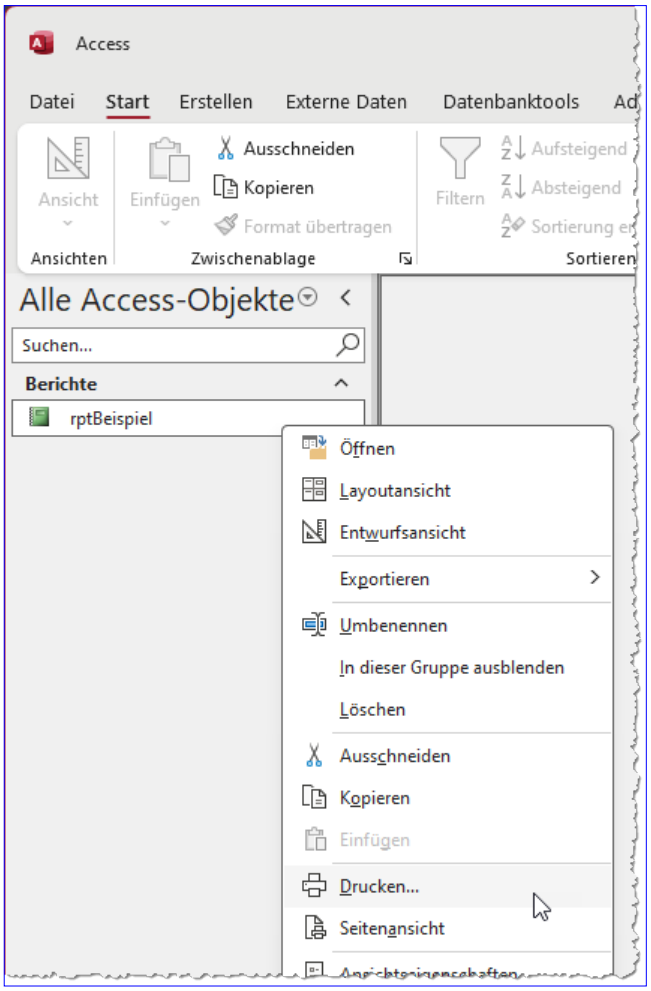


Bild 2: Drucken des Berichts

Der nächstbessere Ansatz ist das Öffnen des Berichts in der Seitenansicht, was sich ebenfalls über dieses Kontextmenü realisieren lässt – diesmal über den Befehl **Seitenansicht**.

Danach können wir über den Eintrag **Drucken...** des Kontextmenüs des Berichts erneut den Druckvorgang starten (siehe Bild 3). Wieso sollten wir aber diesmal ein anderes Ergebnis erhalten als beim vorherigen Anlauf – also warum sollte nun ein **Drucken**-Dialog erscheinen und nicht wieder direkt der Druckvorgang starten? Dies erkennen wir an einem kleinen, aber feinen Unterschied im Text des Kontextmenü-Eintrags: **Drucken** ohne drei Punkte am Ende startet direkt den Druckvorgang, **Drucken...** hingegen öffnet den Dialog (dieses Verhalten war zumindest unter Access von Office 365 zu beobachten).

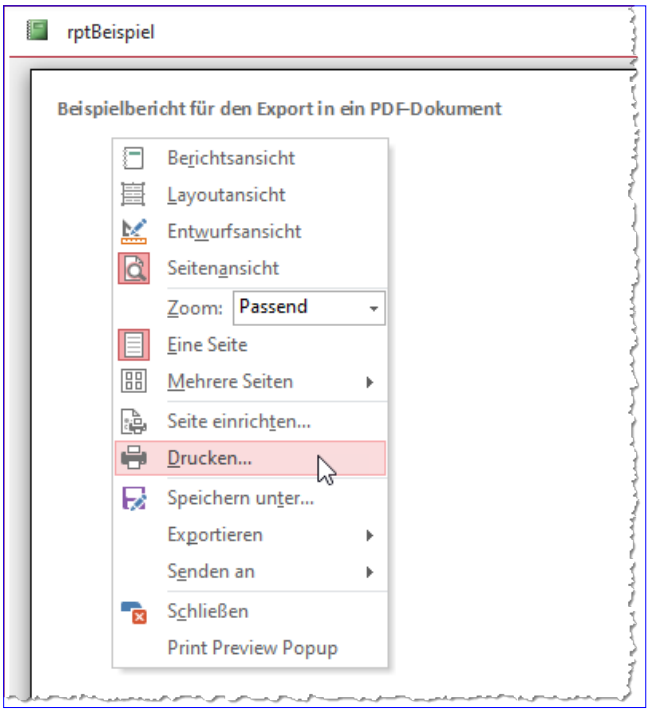


Bild 3: Kontextmenü des Berichts in der Seitenansicht

Der Drucken-Dialog

Der nun erscheinende Dialog **Drucken** aus Bild 4 bietet die Auswahl des Druckers an. Hier können wir unter **Drucker** für **Name** den Eintrag **Microsoft Print To PDF** auswählen.

Das anschließende Anklicken der **OK**-Schaltfläche blendet den Dialog **Druckausgabe speichern unter** ein, mit dem wir die zu erzeugende oder zu überschreibende PDF-Datei auswählen können (siehe Bild 5).

Dies speichert den Bericht in die angegebene PDF-Datei.

Wenn wir uns den Dateidialog genauer anschauen, finden wir hier keine weiteren Optionen – wir können lediglich das Zielverzeichnis, den Dateinamen und den Dateityp einstellen, und für letzteren steht lediglich der Eintrag **PDF-Dokument (*.pdf)** zur Verfügung.

Dies steht im Gegensatz zu den Optionen, die wir in der nachfolgend vorgestellten Variante vorfinden werden. Vorerst haben wir jedoch eine einfache Methode gefunden, um einen Bericht als PDF-Dokument zu speichern.

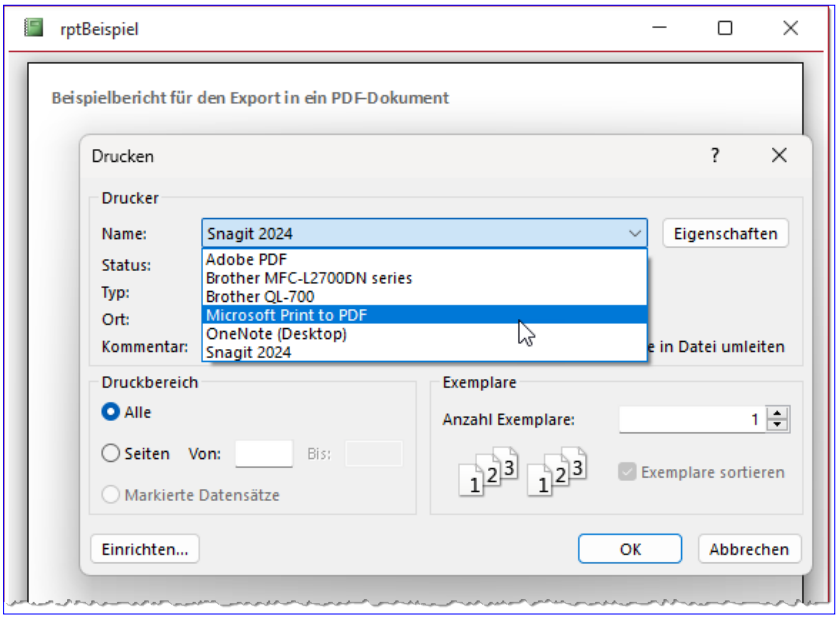


Bild 4: Drucker für das Erstellen eines PDF-Dokuments auswählen

Mehr Optionen beim Drucken über das Datei-Menü

Die nächste Methode, den Bericht als PDF-Dokument zu speichern, finden wir im Datei-Menü. Bevor wir dieses anzeigen, müssen wir den als PDF-Dokument zu speichernden Bericht markieren.

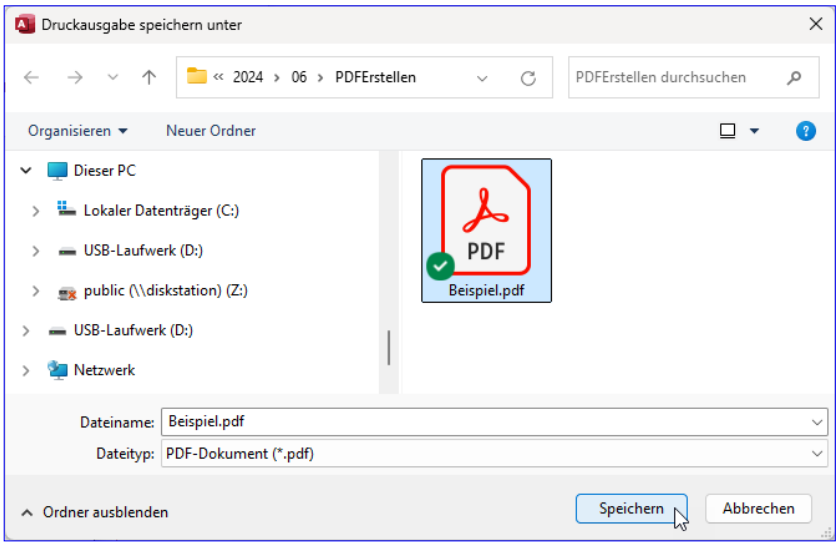


Bild 5: Speichern des Berichts im PDF-Format

Dann wechseln wir zum Ribbon-Tab **Datei** und wählen im linken Bereich den Befehl **Speichern unter** aus. In Zusammenhang mit dem Erstellen einer PDF-Datei hört sich dies schon einmal logischer an als der Befehl **Drucken**. Diesen Befehl gibt es zwar auch im **Datei**-Bereich, aber dieser hat die gleiche Funktion wie die zuvor vorgestellten Varianten.

Aktivieren wir den Bereich **Speichern unter**, finden wir zunächst die verschiedenen Möglichkeiten zum Speichern der aktuellen Datenbank vor. Diese benötigen wir nicht, wir wechseln mit einem Klick auf **Objekt speichern als** auf den alternativen Bereich.

Hier finden wir unter **Aktuelles Datenbankobjekt speichern** die Option **PDF oder XPS** vor, die wir entweder auswählen und mit der Schaltfläche **Speichern unter** ausführen oder direkt per Doppelklick auf **PDF oder XPS** starten (siehe Bild 6).

Im nun erscheinenden Dialog **Als PDF oder XPS veröffentlichen** finden wir ebenfalls die Möglichkeit, Ver-

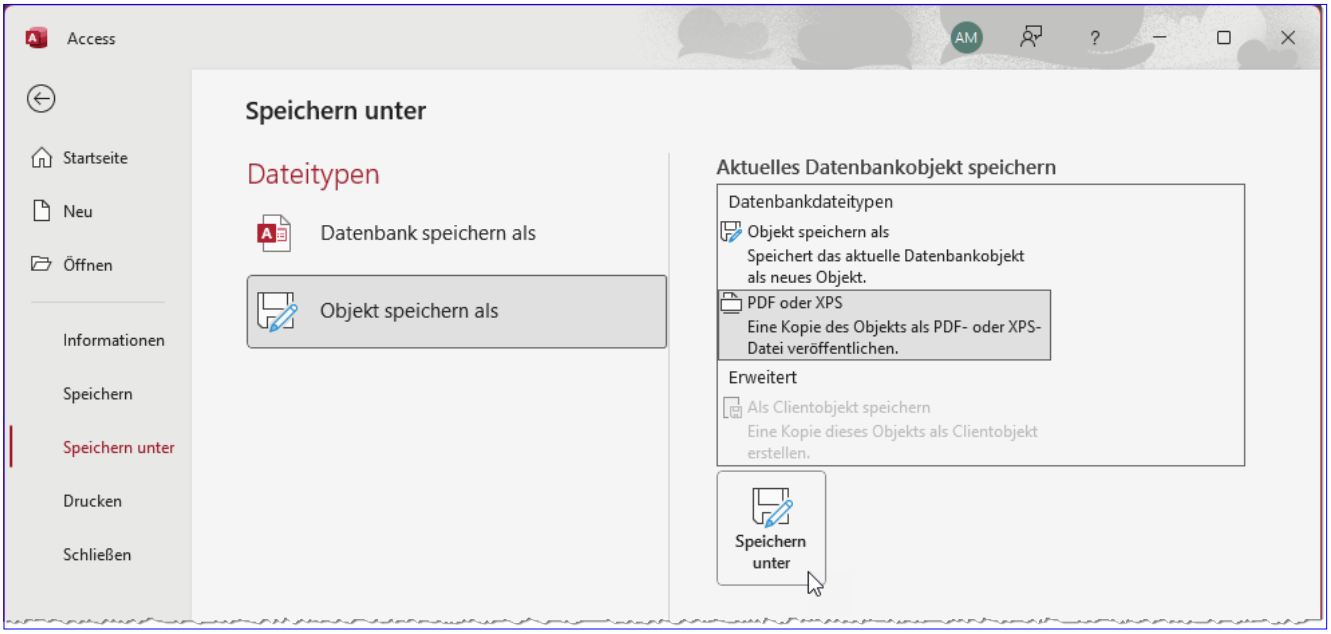


Bild 6: Speichern als PDF oder XPS

VBA-Projekte mit digitalem Zertifikat signieren

Das Signieren von VBA-Projekten ist ein 2024 erneut eingeführter Sicherheitsprozess, bei dem wir ein VBA-Projekt mit einem digitalen Zertifikat versehen. Dieses Zertifikat bestätigt, dass der Code von uns stammt und seit der Signierung nicht verändert wurde. Das Signieren ist besonders wichtig, wenn VBA-Makros in sensiblen oder gemeinsam genutzten Umgebungen ausgeführt werden, zum Beispiel in Unternehmensnetzwerken. In diesem Beitrag schauen wir uns an, welche Arten von Zertifikaten es gibt, wie diese funktionieren und wie wir ein VBA-Projekt mit einer digitalen Zertifikat versehen können. Außerdem betrachten wir, ob das signieren mit einem Zertifikat überhaupt sinnvoll ist.

In allen Umgebungen sollte eigentlich sichergestellt sein, dass keine Schadsoftware unerwünscht ausgeführt werden. Insbesondere mit VBA-Code können nahezu beliebige Aktionen wie das Löschen von Dateien oder der Zugriff auf die Registry durchgeführt werden.

Bei VBA-Projekten, die mit verschiedenen Dokumenten wie Access-Datenbanken, Excel-Arbeitsmappen, Word-Dokumenten oder auch PowerPoint-Präsentationen auf dem System landen können, besteht grundsätzlich die Möglichkeit, dass diese unbemerkt vom Benutzer ausgeführt werden können, wenn das System dies nicht unterbindet. Also sollte man gerade in Umgebungen, die sensible Daten beinhalten, Vorsicht walten lassen.

Wie können wir die Ausführung von VBA-Code unterbinden?

Die Office-Anwendungen bieten verschiedene Einstellungen, mit denen wir festlegen können, ob VBA-Code ausgeführt werden kann. Diese finden wir in den Optionen der jeweiligen Anwendung. Im Optionen-Dialog klicken wir dazu auf den Bereich **Trust Center** und dann auf **Einstellungen für das Trust Center...** (siehe Bild 1).

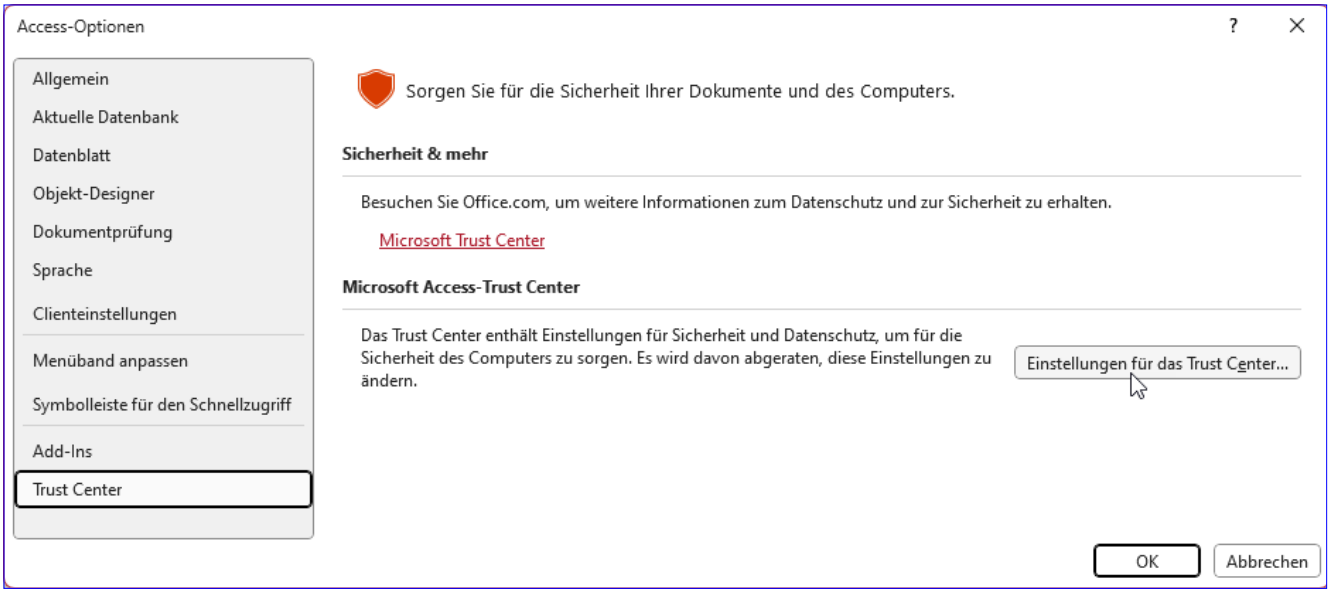


Bild 1: Öffnen der Einstellungen für das Trust Center

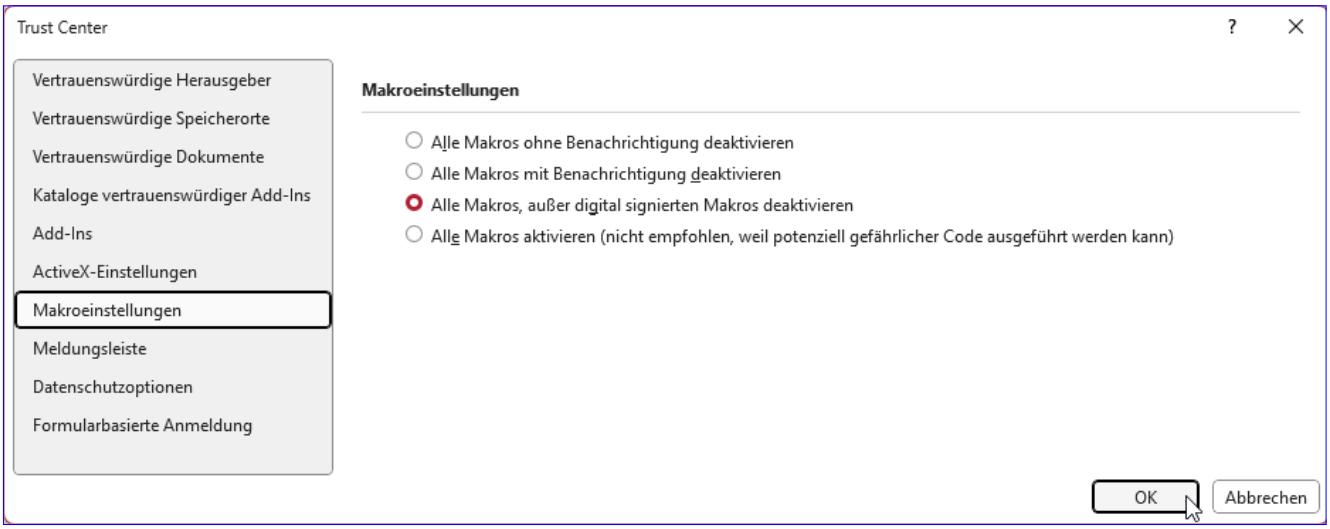


Bild 2: Einstellungen für die Aktivierung oder Deaktivierung von Makros

Im Bereich **Makroeinstellungen** sehen wir die gewünschten Optionen (siehe Bild 2):

Diese lauten wie folgt:

- **Alle Makros ohne Benachrichtigung deaktivieren**: Dies führt dazu, dass gar kein VBA-Code ausgeführt werden kann. Versuchen wir hier, Code aufzurufen, erhal-

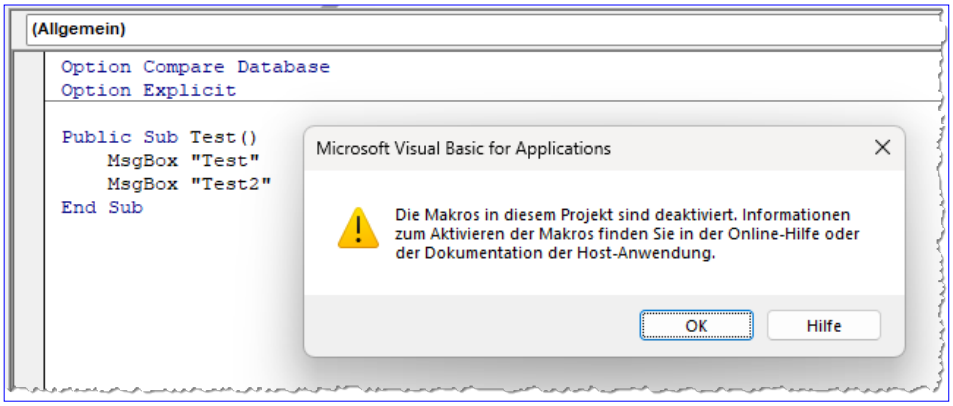


Bild 3: Meldung, dass die Makros deaktiviert sind

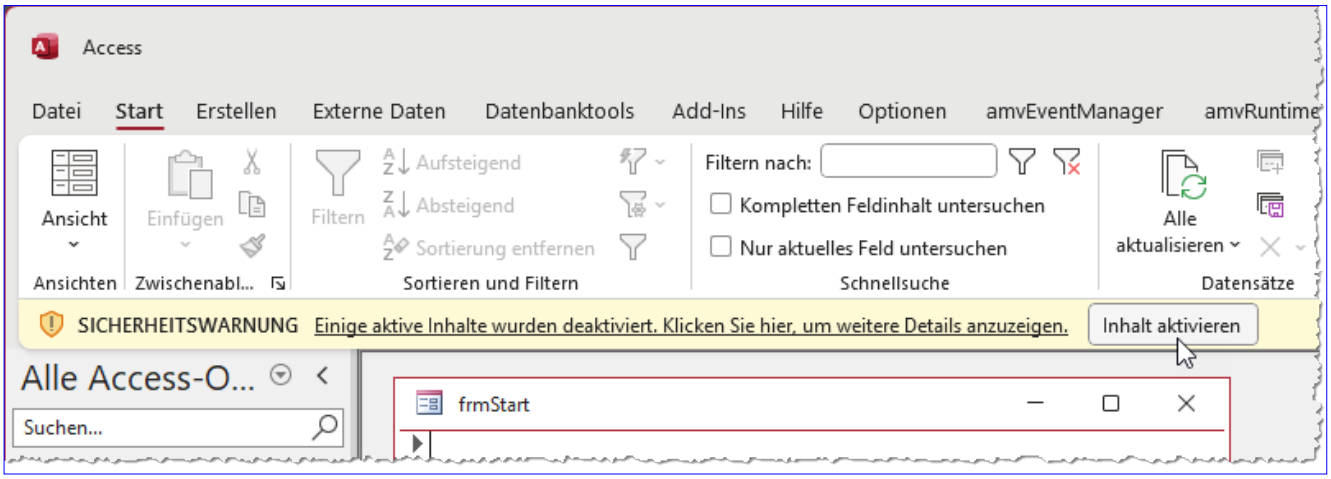


Bild 4: Hinweis, dass der Inhalt noch nicht aktiviert ist

ten wir eine Meldung wie in Bild 3. Code, der beispielsweise durch Formularereignisse ausgelöst werden soll, wird ohne Meldung ignoriert.

- **Alle Makros mit Benachrichtigung deaktivieren**: In diesem Fall erscheint ein gelber Balken mit einem Hinweis, dass einige Inhalte deaktiviert wurden (siehe Bild 4). Wir können hier auf **Inhalt aktivieren** klicken und so das Dokument zu den vertrauenswürdigen Dokumenten hinzufügen. Gegebenenfalls erscheint hier keine solche Meldung und der Code wird anstandslos ausgeführt. In diesem Fall wurde das Dokument offensichtlich bereits zu den vertrauenswürdigen Dokumenten hinzugefügt. Es kann jedoch auch sein, dass das Dokument sich durch eine digitale Signatur als vertrauenswürdig erkennbar gemacht hat.

- **Alle Makros, außer digital signierten Makros deaktivieren**: Mit dieser Option werden auch die VBA-Projekte, die über ihr Dokument zu den vertrauenswürdigen Dokumenten hinzugefügt wurden, nicht mehr ausgeführt, sondern nur noch solche, die mit einem digitalen Zertifikat signiert wurden.

- **Alle Makros aktivieren (nicht empfohlen, weil potenziell gefährlicher Code ausgeführt werden kann)**: Bei Auswahl dieser nicht empfohlenen Option wird der Code dieser Datenbank ohne weitere Nachfrage ausgeführt.

Hinzufügen zu vertrauenswürdigen Dokumenten

Wenn wir die oben erwähnte Meldung in dem gelben Balken erhalten, die besagt, dass Inhalte deaktiviert wurden, können wir den enthaltenen Code durch einen Klick auf die Schaltfläche **Inhalt aktivieren** scharf schalten.

Was geschieht dann genau? Unser Dokument wird zu den vertrauenswürdigen Dokumenten für dieses Benutzerkonto hinzugefügt. Wenn man dauerhaft diese Einstellung

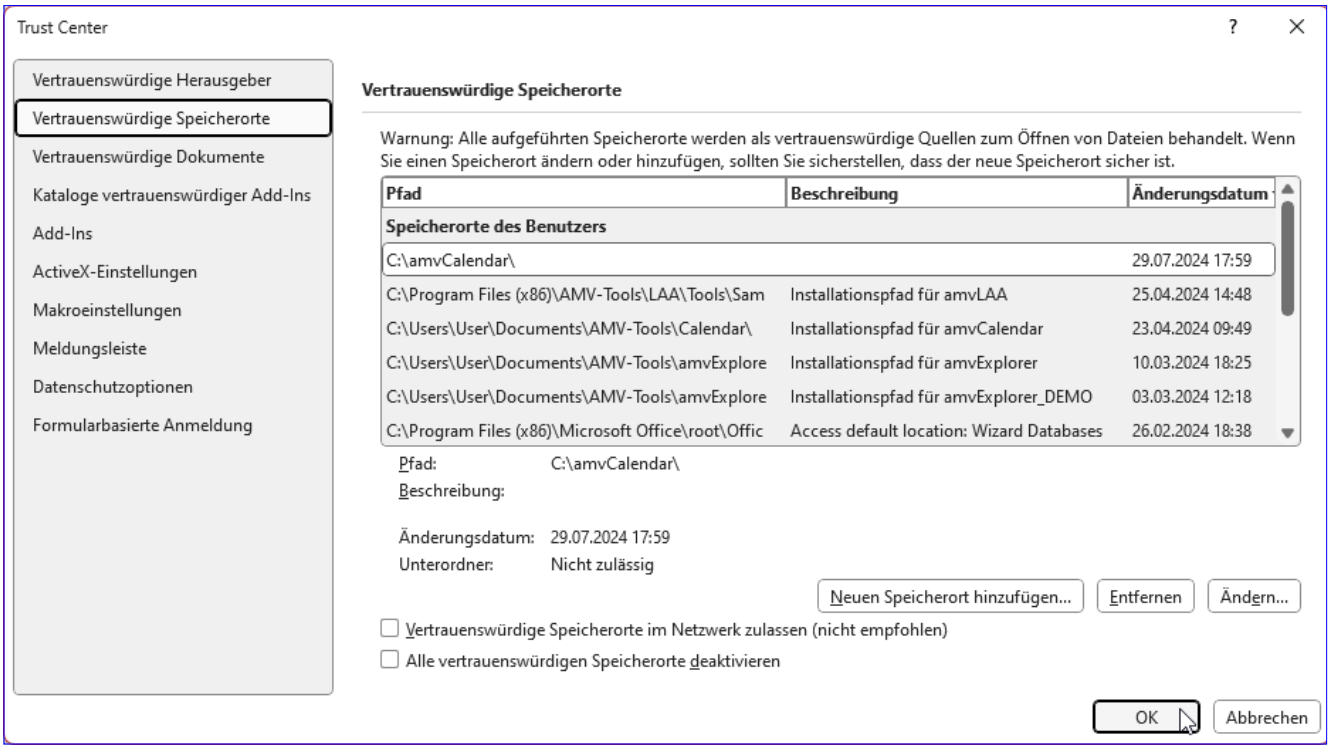


Bild 5: Vertrauenswürdige Speicherorte

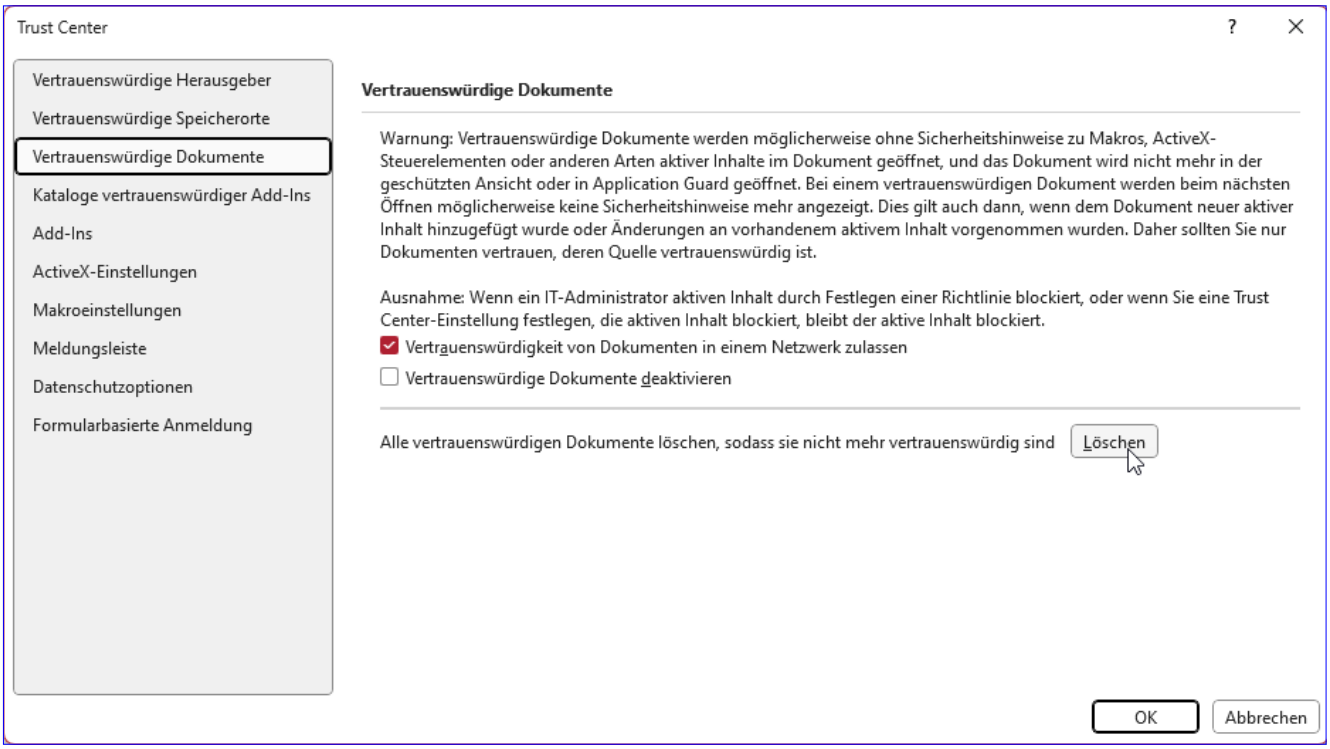


Bild 6: Vertrauenswürdige Dokumente

verwendet, werden sich im Laufe der Zeit einige Datenbankdateien in der Sammlung der vertrauenswürdigen Dokumente finden. Leider kann man diese nicht so einfach einsehen (siehe Bild 6).

Es gibt allerdings eine Möglichkeit, Dokumente in einem oder mehreren Verzeichnissen automatisch als vertrauenswürdig einzustufen. Diese verwalten wir ebenfalls im Dialog **Trust Center**, und zwar im Bereich **Vertrauenswürdige Speicherorte** (siehe Bild 5).

Hier können wir Ordner hinzufügen und ihre Eigenschaften einstellen, aber auch vertrauenswürdige Ordner wieder entfernen.

Wie gelingt dies, wenn wir ein einzelnes Dokument aus den vertrauenswürdigen Dokumenten entfernen möchten?

Wenn wir zum Bereich **Vertrauenswürdige Dokumente** wechseln, sehen wir nur die Möglichkeit, alle Dokumente zu löschen.

Es muss also einen anderen Ort geben, an dem wir die Zuordnung zu den vertrauenswürdigen Dokumenten finden. Diese finden wir schließlich in der Registry, in unserem Fall im folgenden Zweig:

```
Computer\HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\
Access\Security\Trusted Documents\TrustRecords
```

Hier können wir die Einträge einzeln löschen (siehe Bild 7).

Vertrauenswürdiger Speicherort per Setup

Wenn wir unsere Anwendung mit einem Setup verteilen, können wir dafür sorgen, dass das Installationsverzeichnis als vertrauenswürdiger Speicherort gekennzeichnet wird und die Anwendung direkt ausgeführt werden kann. Das ist hilfreich, aber wenn wir unsere Anwendung mit offenem Quellcode ausliefern, was manchmal eine Kundenanforderung ist, dann kann der Benutzer den Code der Datenbank ändern, ohne dass die Anwendung die Vertrauenswürdigkeit verliert. Was aber, wenn wir sicherstellen wollen, dass der Benutzer keine Funktionen einbaut, die

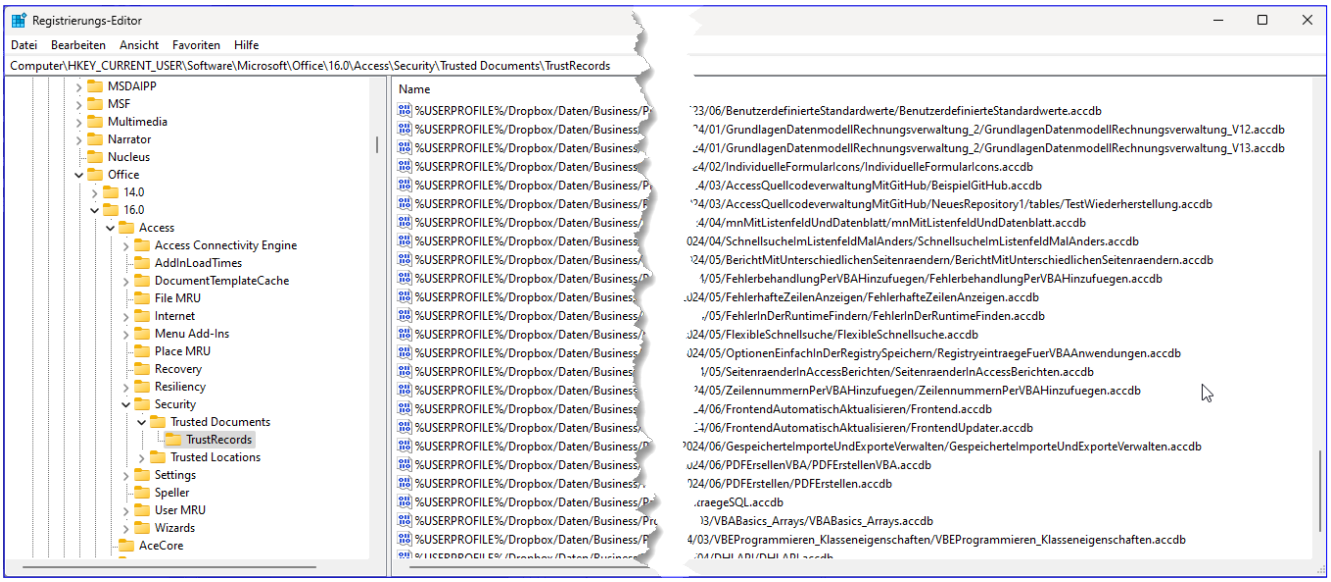


Bild 7: Vertrauenswürdige Dokumente in der Registry

gegebenenfalls Schaden verursachen? Dann kommen wir um die Anwendung eines Zertifikats nicht herum, denn nur dieses garantiert, dass das VBA-Projekt im offenen Zustand nicht geändert wurde.

Eine andere Möglichkeit, zusätzlich sicherzustellen, dass der Benutzer den Code nicht anpasst, ist die Erstellung einer **.accde**-Version der Anwendung. Diese enthält dann allerdings nur noch den kompilierten Quellcode. Aber auch diese kann mit einem digitalen Zertifikat signiert werden.

VBA-Projekt zertifizieren

Um ein VBA-Projekt mit einem digitalen Zertifikat zu signieren, benötigen wir zunächst einmal ein solches Zertifikat. Hier gibt es zum Beispiel die folgenden Möglichkeiten:

- Wir erstellen ein eigenes Zertifikat.
- Wir kaufen ein Zertifikat.

Eigenes Zertifikat erstellen

Als Erstes schauen wir uns an, was wir mit einem selbst erstellten Zertifikat erledigen können.

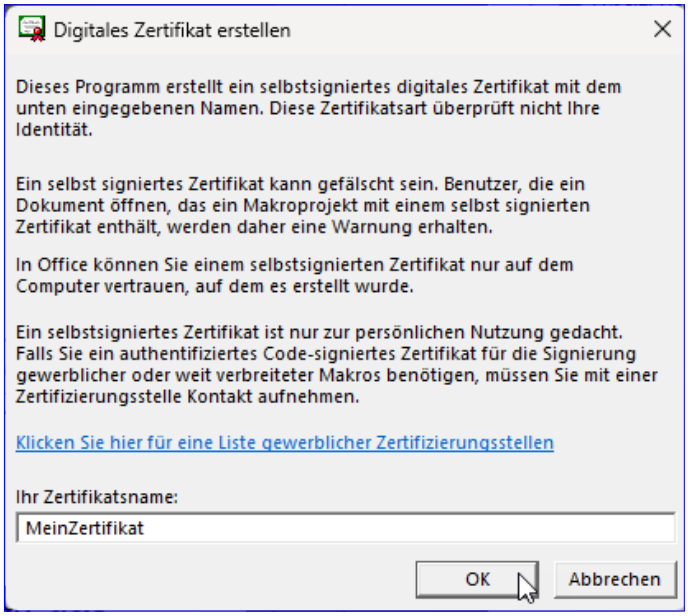


Bild 8: Erstellen eines eigenen Zertifikats

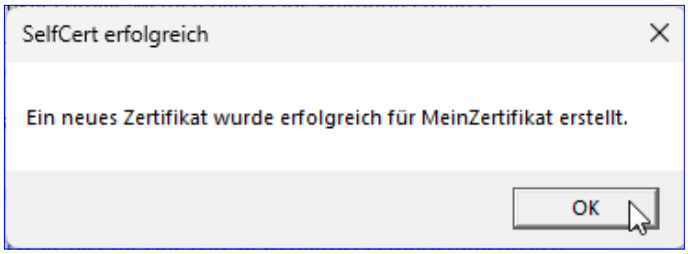


Bild 9: Erfolgsmeldung über die Erstellung des Zertifikats

PDF per VBA erstellen im Detail

Im Beitrag »PDF erstellen mit Access im Detail« (www.access-im-unternehmen.de/1523) haben wir uns bereits angesehen, wie wir PDF-Dokumente über die Benutzeroberfläche erstellen können. Typischerweise möchte man einem Benutzer allerdings nicht die Aufgabe überlassen, einen Bericht zu öffnen und dann die entsprechenden Schaltflächen im Ribbon aufzurufen, um den Bericht als PDF-Dokument zu speichern. Dazu stellen wir einen eigenen Button zur Verfügung, mit dem wir den Bericht direkt als PDF speichern. Es fehlt also nur noch der passende VBA-Befehl. In diesem Beitrag schauen wir uns an, wie dieser lautet und welche Alternativen es gibt. Außerdem werfen wir einen Blick darauf, wie wir Dokumente im PDF/A-Format erzeugen. Während wir dazu über die Benutzerfläche lediglich eine Option im Dateiauswahl-Dialog aktivieren mussten, ist der Aufwand unter VBA bereits deutlich höher. Aber immerhin ist dieser nur einmalig durchzuführen, weshalb wir gern darauf eingehen.

Bericht als PDF exportieren per DoCmd.OutputTo

Die erste und offensichtliche Möglichkeit zum Exportieren eines Berichts ist die **OutputTo**-Methode des **DoCmd**-Objekts.

Diese Methode erwartet die folgenden Parameter:

- **ObjectType**: Typ des zu exportierenden Objekts, in diesem Fall **acOutputReport**
- **ObjectName**: Name des Berichts
- **OutputFormat**: Ausgabeformat, hier **acFormatPDF**
- **OutputFile**: Pfad der zu erstellenden Datei
- **Autostart**: Gibt an, ob die exportierte Datei nach dem Export automatisch angezeigt werden soll.
- **TemplateFile**: Relevant für Exporte in andere Dateiformate wie Word oder Excel.
- **Encoding**: Nur relevant für den Export in das Textformat.
- **OutputQuality**: Gibt die Qualität der zu erstellenden Datei an. Wir können die beiden Werte **acExportQualityPrint** und **acExportQualityScreen** angeben.

Ein Aufruf sieht beispielsweise wie folgt aus:

```
DoCmd.OutputTo acOutputReport, "rptBeispiel", acFormatPDF, CurrentProject.Path & "\rptBeispiel_Screen.pdf", True, , , acExportQualityScreen
```

Bericht mit Kriterium als PDF exportieren

Natürlich kommt es immer mal vor, dass man einen Bericht genau so als PDF-Dokument speichern möchte, wie man diesen auch per Doppelklick auf seinen Namen im Navigationsbereich anzeigen würde – also ohne die Angabe von Kriterien et cetera.

Allerdings gibt es auch Berichte, die vor der Ausgabe im PDF-Format gefiltert werden sollen. Zu Beispielzwecken haben wir einen weiteren Bericht namens **rptArtikel** angelegt, der in der Entwurfsansicht wie in Bild 1 erscheint.

Wenn wir den Bericht nun gefiltert öffnen, sodass er beispielsweise nur die Artikel mit den Werten bis **10** im Feld

ArtikelID ausgibt, öffnen wir diesen wie folgt:

```
DoCmd.OpenReport "rptArtikel", acViewPreview, , "ArtikelID <= 10"
```

Das Ergebnis sehen wir in Bild 2. Hier gibt der Bericht wie erwartet nur die ersten zehn Datensätze aus.

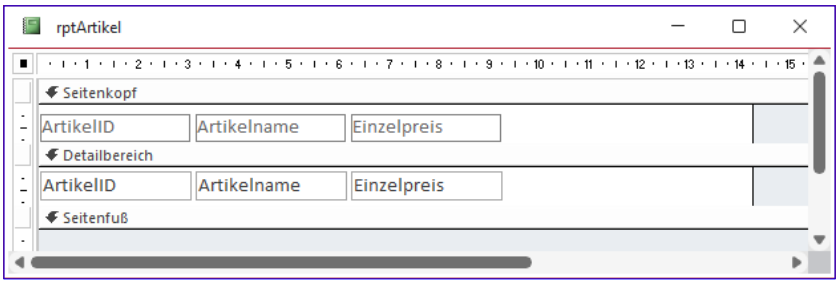


Bild 1: Entwurf eines Beispielberichts mit einer Artikelliste

Nun schließen wir diesen Bericht und exportieren ihn im PDF-Format:

```
DoCmd.OutputTo acOutputReport, "rptArtikel", acFormatPDF, CurrentProject.Path & "\rptArtikel.pdf", True, , , acExportQualityScreen
```

Dies liefert logischerweise den Bericht mit allen Datensätzen (siehe Bild 3). Nun wollen wir diesen allerdings in gefilterter Form in einem PDF-Dokument speichern.

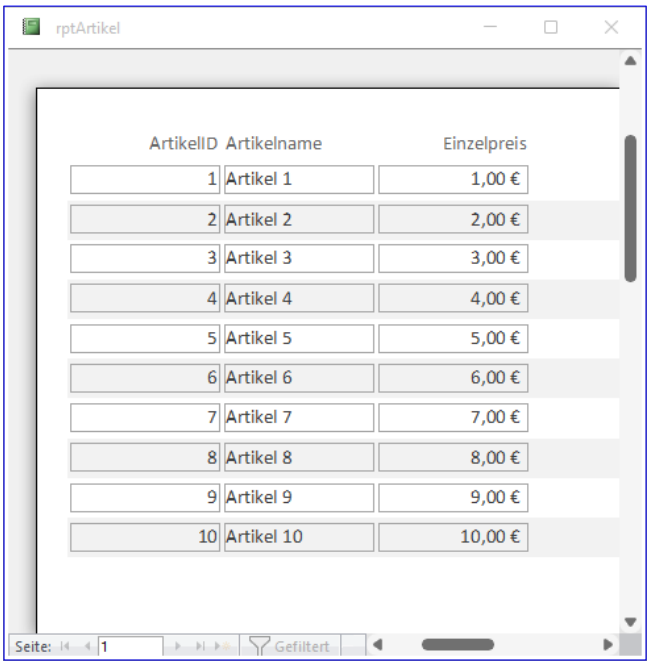


Bild 2: Bericht mit **Where**-Bedingung

Dummerweise liefert uns die **DoCmd.OutputTo**-Methode keine Möglichkeit, einen Parameter namens **WhereCondition** für den Bericht anzugeben.

Also müssen wir einen alternativen Weg finden, um den Bericht gefiltert auszugeben.

Dazu gehen wir zunächst den folgenden Weg:

- Wir öffnen den Bericht unter Angabe der **WhereCondition** in der Seitenansicht.
- Dann exportieren wir den Bericht.

Das läuft nun ohne Probleme und liefert das gewünschte Ergebnis. Wir öffnen den Bericht, dessen Namen wir zuvor in der Variablen **strReport** gespeichert haben, mit **DoCmd.OpenReport**, geben diesen dann mit

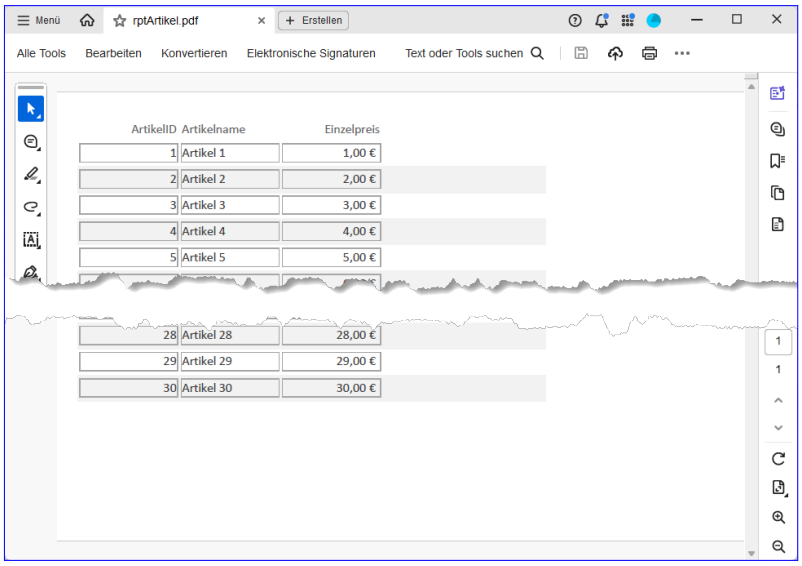


Bild 3: Artikel-Bericht als PDF

DoCmd.OutputTo aus und schließen ihn mit **DoCmd.Close** wieder.

Die verwendete Prozedur sieht wie in Listing 1 aus.

```
Public Sub ArtikellisteGefiltert()
    Dim strReport As String
    Dim strPfad As String
    strReport = "rptArtikel"
    strPfad = CurrentProject.Path & "\" & strReport & ".pdf"
    DoCmd.OpenReport strReport, acViewPreview, , "ArtikelID <= 10"
    DoCmd.OutputTo acOutputReport, strReport, acFormatPDF, strPfad
    DoCmd.Close acReport, strReport
End Sub
```

Listing 1: Exportieren der zuvor gefilterten Artikelliste

Achtung, Fehler 2501!

Es kann beim Aufruf der Prozedur **DoCmd.OutputTo** passieren, dass wir den Fehler **2501** erhalten. Wir haben einige Minuten gebraucht, um herauszufinden, weshalb dieser aufgetreten ist. Der Grund war einfach: Wir hatten die zuvor bereits einmal erzeugte PDF-Datei namens **rptArtikel.pdf** bereits geöffnet. Eine geöffnete PDF-Datei kann jedoch nicht einfach überschrieben werden. Dabei erscheint die hier wenig aussagekräftige Meldung **Die Aktion OutputTo wurde abgebrochen**.

Bericht mit dem Standard PDF/A-3 speichern

Im Beitrag **PDF erstellen im Detail** (**www.access-im-unternehmen.de/1523**) haben wir gezeigt, wie wir zumindest mit neueren Access-Versionen Berichte so als PDF-Dokumente exportieren konnten, dass diese dem PDF/A-3-Standard entsprechen. Dazu mussten wir im **Speichern**-Dialog, den wir mit dem Ribbonbefehl **Seitenansicht|Daten|PDF oder XPS** geöffnet haben, die Optionen einblenden und dort die Option **PDF/A-kompatibel** aktivieren (siehe Bild 4). Dies führte je nach Access-Version dazu, dass die PDF-Datei im Format PDF/A-1 oder PDF/A-3 erzeugt wurde.

Wenn wir uns den **Speichern**-Dialog genauer ansehen, finden wir hier mit **Datei nach dem Veröffentlichen öffnen** und **Optimieren für** auch noch zwei Optionen, die wir bereits von der **OutputTo**-Methode des **DoCmd**-Objekts kennen. Da stellt sich die Frage, ob wir vielleicht auch noch die PDF/A-3-Kompatibilität per VBA sicherstellen können. Allerdings liefert selbst die Aktivierung der Anzeige versteckter Elemente im Objektkatalog keinerlei Hinweise auf weitere Möglichkeiten.

Wenn wir den Export-Vorgang über diesen Dialog abschließen, taucht jedoch noch der Schritt **Exportschritte speichern** auf (siehe Bild 5). Vielleicht liefert dieser ja noch hilfreiche Informationen?

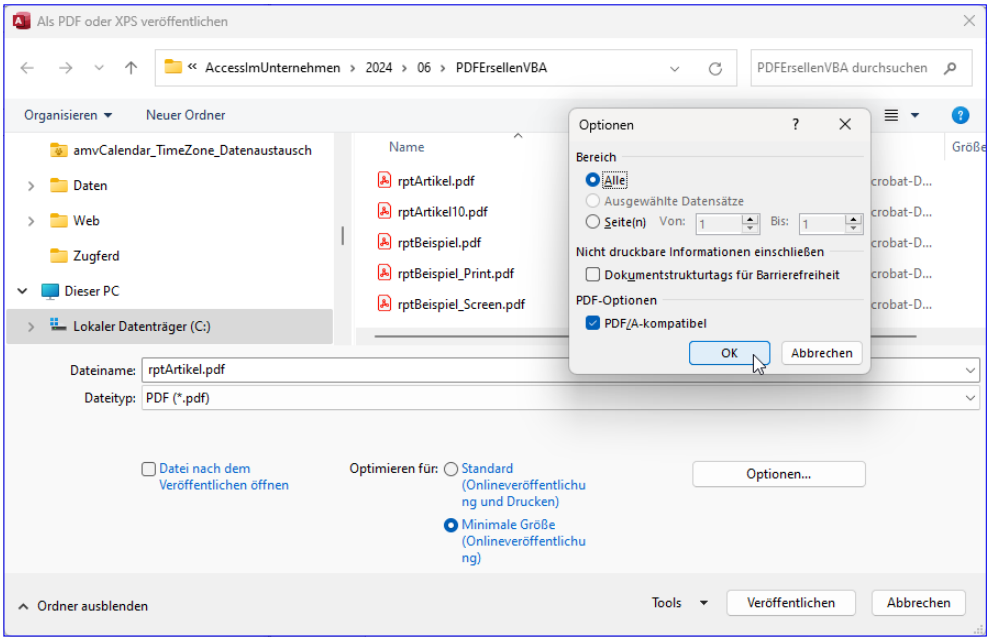


Bild 4: Exportieren im PDF/A-3-Format über die Benutzeroberfläche

Und was geschieht eigentlich, wenn wir das Speichern der Exportschritte wie im Screenshot aktivieren? Also probieren wir das einmal aus und nennen den Export **Exportschritte_rptArtikel**.

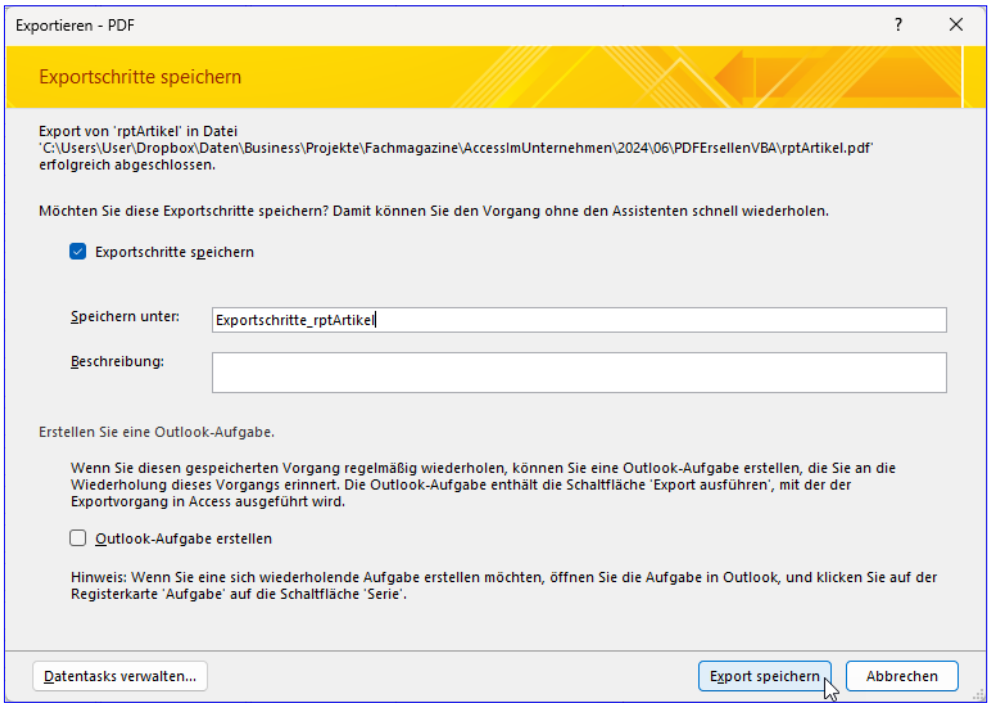


Bild 5: Speichern der Exportschritte

Danach stellt sich allerdings die Frage, wo diese Exportschritte überhaupt gespeichert werden und wie wir diese nutzen können. Wenn wir erneut versuchen, den Bericht zu exportieren, finden wir jedenfalls keine Gelegenheit, diese auszuwählen.

Suchen wir ein wenig weiter, finden wir allerdings im Ribbon unter **Externe Daten** einen Befehl namens **Gespeicherte Exporte** (siehe Bild 6).

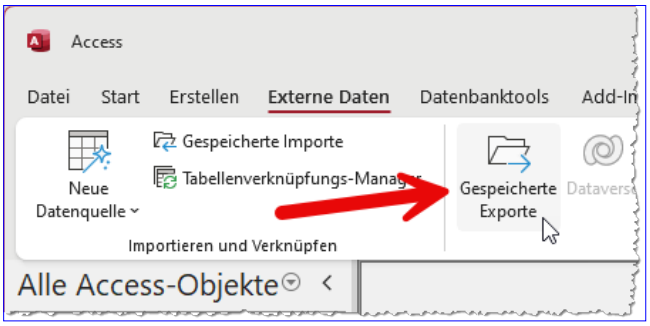


Bild 6: Im Ribbon befindet sich ein Button namens **Gespeicherte Exporte**.

Klicken wir diesen an, öffnet sich ein Dialog, der unseren gespeicherten Export anzeigt (siehe Bild 7).

Hier haben wir jedoch auch nur die Möglichkeit, den Export nochmals zu starten.

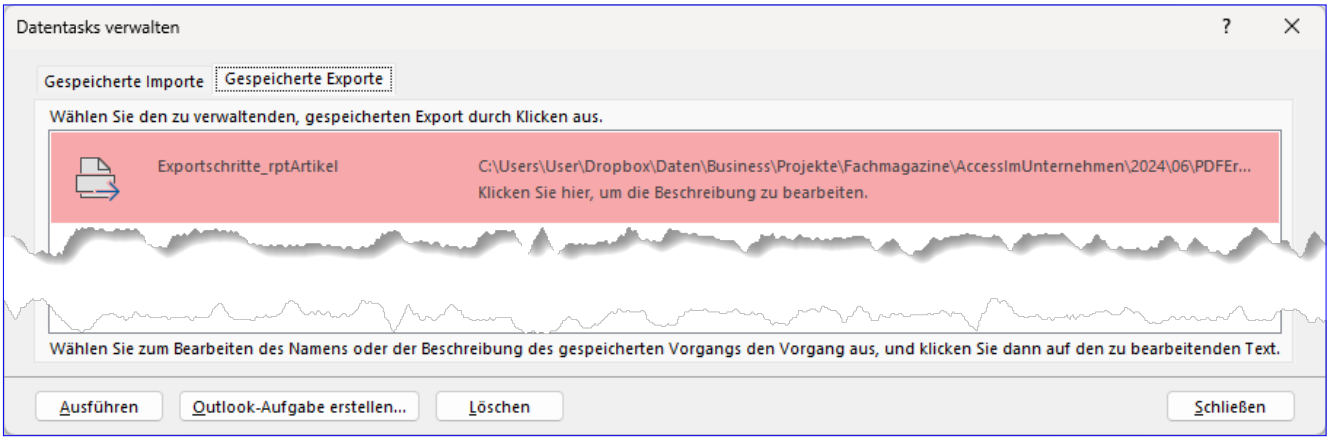


Bild 7: Unser gespeicherter Export

Gespeicherten Export per VBA aufrufen

Logischerweise gibt es auch noch einen VBA-Befehl, mit dem wir einen gespeicherten Export starten können.

Dieser ist ebenfalls eine Methode des **DoCmd**-Objekts und heißt **RunSavedImportExport** und wir rufen diese wie folgt auf:

```
DoCmd.RunSavedImportExport
"Exportschritte_rptArtikel"
```

Mit dieser Methode können wir den Bericht exportieren. Die Frage ist noch, ob wir damit eine PDF/A-3-kompatible Datei erstellt haben. Dazu wollen wir nun einen Validierer hinzuziehen.

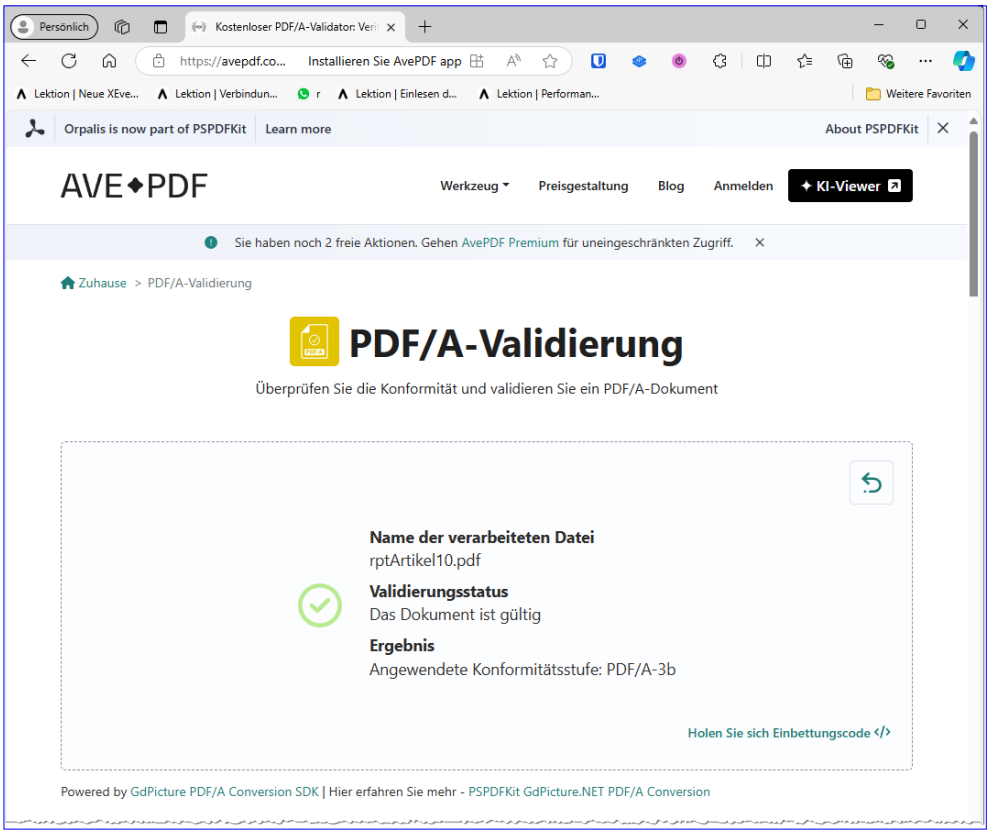


Bild 8: Validieren von PDF/A-3-Dateien

Validieren, ob ein PDF-Dokument dem Standard PDF/A-3 entspricht

Unter dem folgenden Link haben wir unsere mit Access erzeugten Dokumente auf PDF/A-3 validiert:

```
https://avepdf.com/de/pdfa-validation
```

Hier kann man sein Dokument angeben und dieses prüfen lassen. In unserem Beispiel kam das Ergebnis aus Bild 8 heraus. Auch in der kostenpflichtigen Version von Adobe Acrobat können wir erkennen, ob es sich um eine PDF/A-3-Datei handelt. Wenn wir das PDF-Dokument damit öffnen, erscheint erstens oben die Meldung, dass die Datei schreibgeschützt geöffnet wurde, weil sie Konformität mit dem PDF/A-Standard verlangt (siehe Bild 9).

Außerdem sehen wir rechts ein Icon, mit dem wir den Bereich **Standards** öffnen können (der übrigens bei Dateien ohne PDF/A-Standard gar nicht erscheint). In diesem Bereich erhalten wir die Information, dass dieses Dokument dem Standard PDF/A-3B entspricht.

PDF-Dateien, die nicht mit neueren Access-Versionen erstellt wurden, weisen teilweise nur den Standard **PDF/A-1B** auf. Wir konnten nicht genau herausfinden, ab welchen Versionen PDF/A-3B erzeugt wird, daher sollte man dies immer selbst prüfen, wenn PDF-Dokumente mit diesem Standard benötigt werden – beispielsweise als Basis für ZUGFeRD-Rechnungen.

Damit haben wir grundsätzlich eine Lösung, mit der wir unseren Bericht im PDF/A-3-Format exportieren können. Wir benötigen allerdings nun noch eine Möglichkeit, die auszugebenden Daten wie im vorherigen Beispiel durch

Übersichtsformular per Mausklick

Manche Aufgaben im Alltag eines Access-Entwicklers wiederholen sich immer wieder und unterscheiden sich nur durch Kleinigkeiten. Zum Beispiel sehen Detailformulare, Übersichtsformulare oder auch Formulare zum Verwalten von Daten in Haupt- und Unterformular immer gleich aus – wenn auch für andere Daten. Im Beitrag »Tabellendaten mit Übersicht und Details anzeigen« (www.access-im-unternehmen.de/1488) haben wir grundsätzlich gezeigt, wie wir ein Hauptformular mit einer Übersicht in einem Unterformular anlegen und programmieren können. Der Beitrag »Detailformular per Mausklick erstellen« (www.access-im-unternehmen.de/1490) wiederum liefert ein Beispiel dafür, wie wir ein einfaches Detailformular schnell definieren und per Mausklick erstellen können. Im vorliegenden Beitrag wollen wir beides kombinieren – also ein Übersichtsformular mit Haupt- und Unterformular in einem Konfigurationsformular einrichten und dann automatisch erstellen.

Im oben genannten Beitrag **Tabellendaten mit Übersicht und Details anzeigen** (**www.access-im-unternehmen.de/1488**) haben wir gezeigt, wie ein einfaches Formular zum Anzeigen der Übersicht über die Daten eines Formulars erstellt werden kann.

Dieses soll folgende Elemente beinhalten und sieht in der Formularansicht wie in Bild 1 aus:

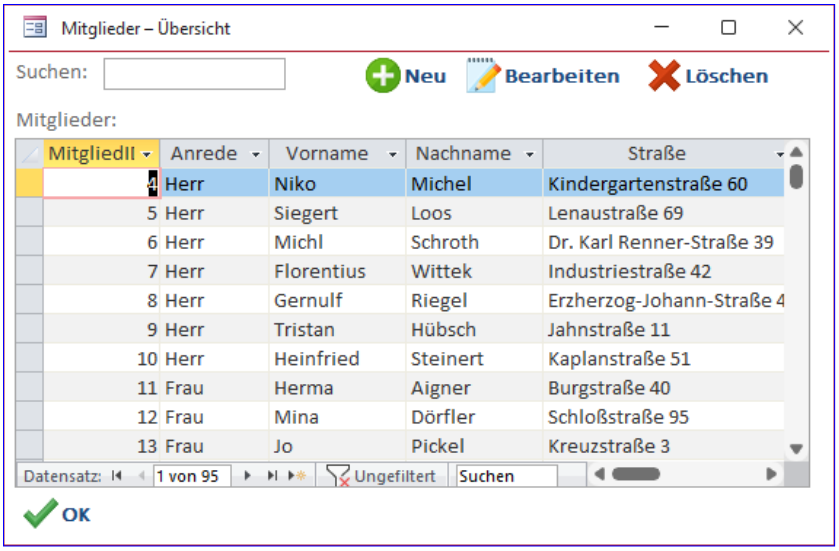


Bild 1: Beispiel für ein Übersichtsformular

- Unterformular mit den anzuzeigenden Daten
- Schaltfläche zum Anlegen neuer Datensätze
- Schaltfläche zum Bearbeiten des aktuell markierten Datensatzes
- Schaltfläche zum Löschen des aktuell markierten Datensatzes
- Schaltfläche zum Schließen des Formulars
- Textfeld zum Eingeben von Suchbegriffen zum Suchen nach Datensätzen im Unterformular

Außerdem soll sich das Unterformular an die Größe des Hauptformulars anpassen, wenn der Benutzer dieses vergrößert. Zum Erstellen eines solchen Formulars wollen wir in diesem Beitrag ein Konfigurationsformular erstellen sowie den Code, der nötig ist, um das Formular auf Knopfdruck anhand der vorgenommenen Einstellungen zu erstellen.

Dabei wollen wir auch direkt die Möglichkeit schaffen, das im Beitrag **Detailformular per Mausklick erstellen** (**www.access-im-unternehmen.de/1490**) erstellte Formular zum Erstellen neuer oder zum Bearbeiten vorhandener Datensätze aufzurufen beziehungsweise den Namen dieses Formulars zu diesem Zweck auszuwählen.

Wir beginnen, indem wir das Konfigurationsformular Schritt für Schritt zusammenstellen und dabei gleichzeitig den Code entwickeln, der zum Erstellen des Formulars dienen soll.

Im gleichen Zuge stellen wir auch die Tabellen zusammen, welche die Konfigurationsdaten erfassen. Wir wollen diese speichern und jederzeit erneut abrufen können.

Die dazu verwendeten Tabellen heißen **tblOverviewforms** und **tblOverviewFields**. Die Tabellen sind ähnlich aufgebaut wie in der Lösung zum Erstellen von Detailformularen. Die Tabelle **tblOverviewForm** sehen wir in der Entwurfsansicht in Bild 2.

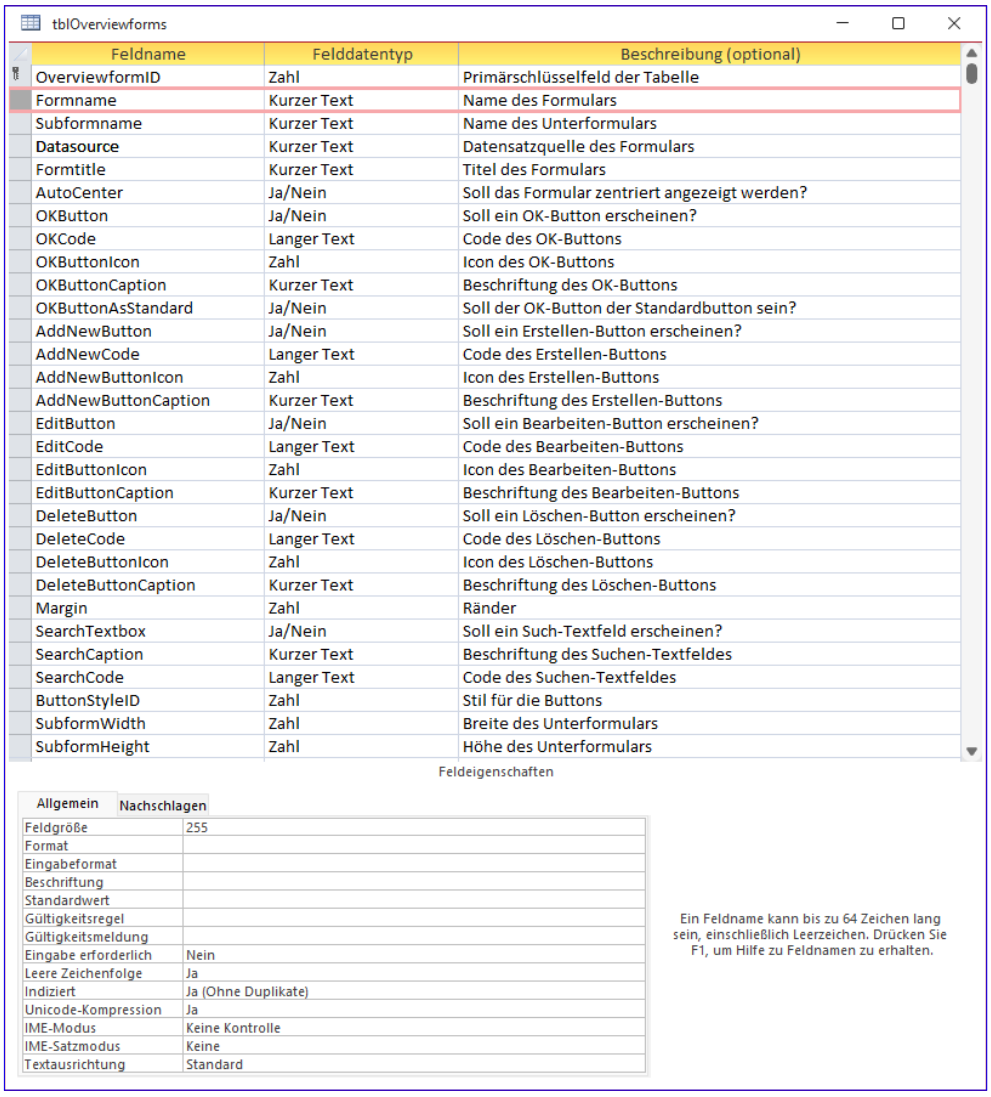


Bild 2: Tabelle für die Daten zum Erstellen von Haupt- und Unterformular

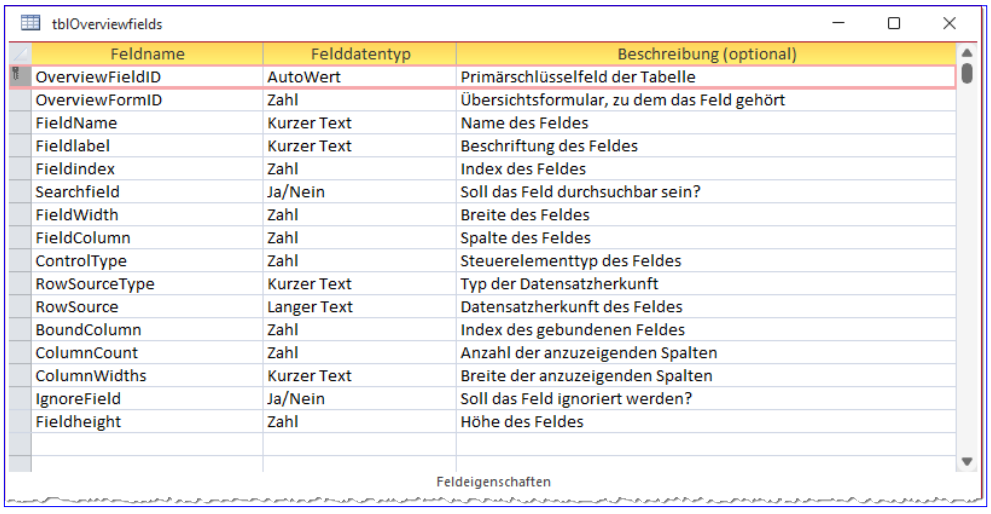


Bild 3: Tabelle für die Daten der Felder im Unterformular

Sie enthält die wesentlichen Informationen zum Erstellen des Hauptformulars und der Steuerelemente im Hauptformular.

Die Tabelle **tblOverviewFields** enthält die Daten der Felder, die für die Übersicht im Unterformular angezeigt werden sollen (siehe Bild 3).

Von der einfachen Access-Anwendung zum Add-In

Wir arbeiten wieder in zwei Schritten: Wir werden zunächst das Konfigurationsformular in der gleichen Anwendung entwickeln, in der wir auch testweise die Formulare anlegen und aus der die Tabellen oder Abfragen stammen, die als Grundlage für das Übersichtsformular dienen. Erst wenn dies erledigt ist, wandeln wir die Datenbankdatei in eine Add-In-Datenbank um.

Das Konfigurationsformular sieht in der Formularansicht wie in Bild 4 aus.

Es enthält oben links die Basisinformationen des zu erstellenden Formulars:

- **Datensatzquelle**: Tabelle oder Abfrage mit den in der Übersicht anzuzeigenden Daten
- **Formularname**: Name des zu erstellenden Formulars
- **Formulartitel**: Text für die Titelleiste des Formulars
- **Rand/Abstand**: Gibt die Abstände der Steuerelemente zu den Rändern und zu den anderen Steuerelementen an.
- **Automatisch zentrieren**: Legt den Wert für die Eigenschaft **Automatisch zentrieren** fest.
- Bereich **Suchen-Textfeld**: Hier finden wir Informationen darüber, ob ein Suchfeld angezeigt werden soll, wie die Beschriftung des Suchfeldes heißt und welcher Code ausgelöst werden soll, wenn sich der Wert des Suchen-Feldes ändert.
- Bereich **OK-Button**: Enthält Informationen darüber, ob eine OK-Schaltfläche angezeigt werden soll, welchen Text sie enthalten soll, welchen Code sie ausführen und welches Symbol sie enthalten soll.
- Bereich **Anlegen-Button**: siehe **OK-Button**
- Bereich **Bearbeiten-Button**: siehe **OK-Button**
- Bereich **Löschen-Button**: siehe **OK-Button**

Im unteren Bereich sehen wir schließlich noch ein Unterformular, das alle Felder der für die Eigenschaft **Datensatzquelle** ausgewählten Tabelle oder Abfrage aufnimmt. Hier können wir individuelle Einstellungen für diese Felder vornehmen, zum Beispiel, ob dieses Feld angezeigt oder ignoriert werden soll. Außerdem können wir die Beschriftung für die Spaltenköpfe hier abweichend von den Feldnamen und den im Tabellenentwurf gespeicherten Beschriftungen vornehmen. Mit **Fieldindex** geben wir die Reihenfolge an und mit **FieldWidth** die Breite des Feldes. Außerdem finden wir hier noch die typischen Eigenschaften für Felder, die als Kombinationsfelder ausgelegt sind und die wir hier auch nochmal individuell anpassen können.

Tabelle oder Abfrage mit den anzuzeigenden Daten auswählen

Der erste Schritt ist die Auswahl der Tabelle oder Abfrage, deren Daten im Unterformular angezeigt werden soll. Dazu benötigen wir ein Kombinationsfeld, mit dem wir alle Tabellen und Abfragen der aktuellen Datenbank auswählen können.

Dieses Kombinationsfeld wollen wir mit allen Tabellen und Abfragen füllen, die in der Tabelle **MSysObjects** vorhanden sind. Dabei müssen wir die Elemente erfassen, die im Feld **Type** einen der Werte **1**, **4**, **5** oder **6** enthalten und deren Wert im Feld **Name** nicht mit **F_** oder **MSys** beginnt.

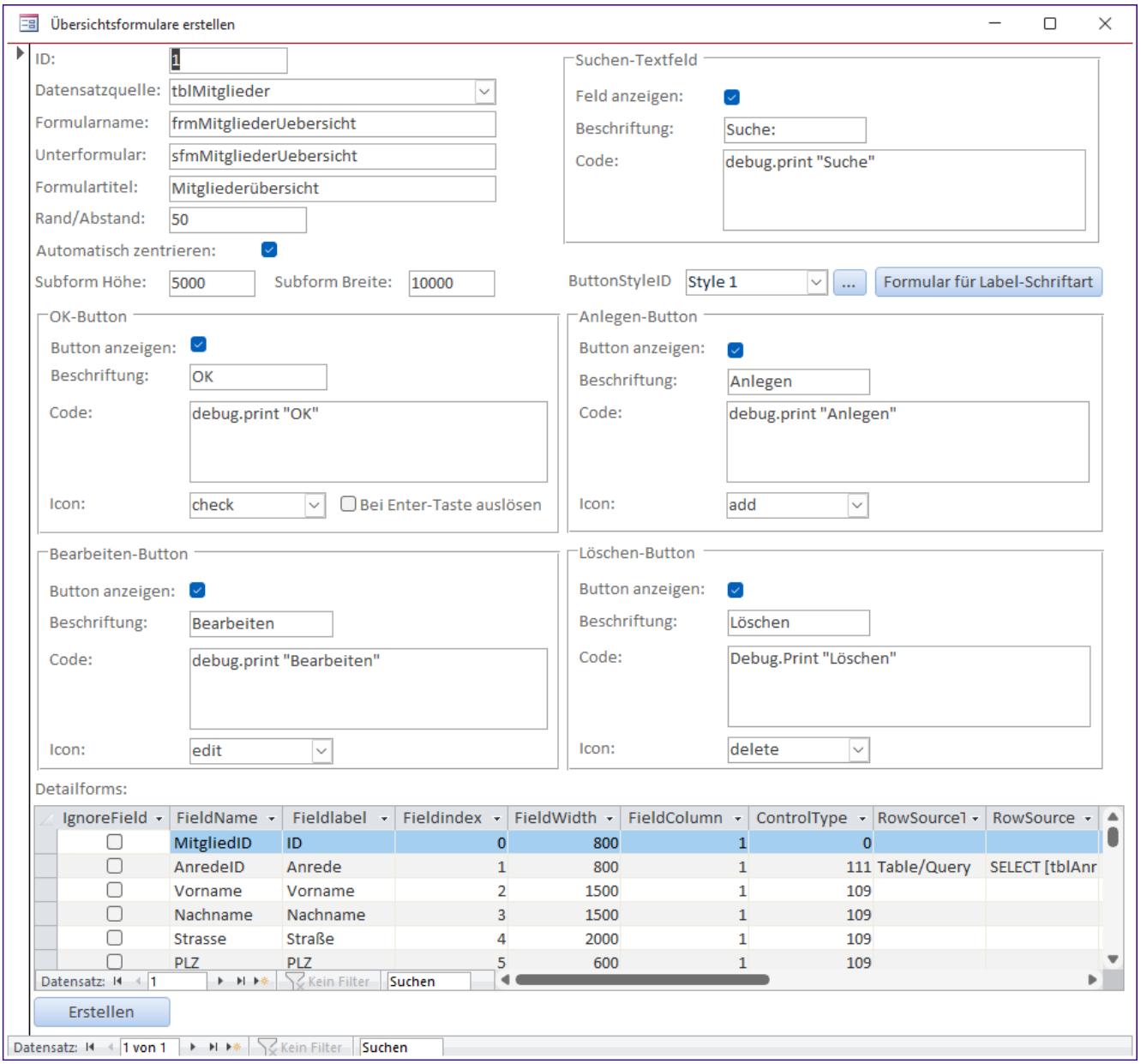


Bild 4: Formulare mit den Daten zum Erstellen des Übersichtsformulars

Allerdings können wir dazu später in der Add-In-Datenbank nicht einfach eine Abfrage zuweisen, die auf der Tabelle **MSysObjects** basiert. Damit würden wir nämlich auf die Tabelle **MSysObjects** der Add-In-Datenbank zugreifen und nicht auf die der Host-Datenbank, der wir das Formular hinzufügen wollen. Also müssen wir ein Recordset erstellen und dieses mit den entsprechenden Daten der Tabelle **MSysObjects** der Host-Datenbank füllen.

Das erledigen wir in der Prozedur, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird (siehe Listing 1).

Die Prozedur definiert zwei verschiedene **Database**-Objekte, von denen das erste mit **CurrentDb** das **Database**-Objekt der Host-Datenbank referenziert und das zweite mit **CodeDb** das **Database**-Objekt der Add-In-Datenbank.

```
Private Sub Form_Load()
    Dim db As DAO.Database
    Dim dbc As DAO.Database
    Dim rst As DAO.Recordset
    Dim rstIcons As DAO.Recordset
    Set db = CurrentDb
    Set dbc = CodeDb
    Set rst = db.OpenRecordset("SELECT Name FROM MSysObjects WHERE Type IN (1, 4, 5, 6) AND Name NOT LIKE 'F_*' " _
        & "AND Name NOT LIKE 'MSys*' AND Name NOT LIKE 'USys*' AND Name NOT LIKE '~*'", dbOpenDynaset)
    Set Me!cboDatasource.Recordset = rst
    Set rstIcons = dbc.OpenRecordset("SELECT id, [Name] FROM MSysResources WHERE Extension = 'png' ORDER BY [Name]", _
        dbOpenDynaset)
    Set Me.cboOKIcon.Recordset = rstIcons
    Me!chkOKButton.DefaultValue = True
    Me!txtOKCaption.DefaultValue = """OK"""
    Me!txtOKCode.DefaultValue = """Debug.Print """"OK""""""""
    Me!cboOKIcon.DefaultValue = 13
    Set Me!cboAddNewIcon.Recordset = rstIcons
    Me!chkAddNewButton.DefaultValue = True
    Me!txtAddNewCaption.DefaultValue = """Anlegen"""
    Me!txtAddNewCode.DefaultValue = """Debug.Print """"Anlegen""""""""
    Me!cboAddNewIcon.DefaultValue = 10
    Set Me!cboDeleteIcon.Recordset = rstIcons
    Me!chkDeleteButton.DefaultValue = True
    Me!txtDeleteCaption.DefaultValue = """Löschen"""
    Me!txtDeleteCode.DefaultValue = """Debug.Print """"Löschen""""""""
    Me!cboDeleteIcon.DefaultValue = 12
    Set Me!cboEditIcon.Recordset = rstIcons
    Me!chkEditButton.DefaultValue = True
    Me!txtEditCaption.DefaultValue = """Bearbeiten"""
    Me!txtEditCode.DefaultValue = """Debug.Print """"Bearbeiten""""""""
    Me!cboEditIcon.DefaultValue = 11
    Me!chkAutoCenter.DefaultValue = True
    Me!cboButtonStyleID.DefaultValue = dbc.OpenRecordset("SELECT ButtonStyleID FROM tblButtonStyles", _
        dbOpenDynaset).Fields(0)
    Me!txtSubformHeight.DefaultValue = 5000
    Me!txtSubformWidth.DefaultValue = 10000
    Me!txtMargin.DefaultValue = 50
    Me!chkSearchTextbox.DefaultValue = True
    Me!txtSearchCaption.DefaultValue = """Suchen nach"""
    Me!txtSearchCode.DefaultValue = """Debug.Print """"Suchen""""""""
    If Not Len(Nz(Me.OpenArgs, "")) = 0 Then
        Me!cboDatasource = Me.OpenArgs
        cboDatasource_AfterUpdate
    End If
End Sub
```

Listing 1: Aktionen beim Starten des Formulars **frmOverviewForms**

```
Private Sub cboDatasource_AfterUpdate()
    If ReadOverviewFieldsFromTable(Me!cboDatasource) = True Then
        Me!sfmOverviewforms.Form.Requery
    End If
    If Len(Nz(Me!txtFormname, "")) = 0 Then
        Me!txtFormname = Replace(Replace(Me!cboDatasource, "tbl", "frm"), "qry", "frm")
    End If
    If Len(Nz(Me!txtUnterformularname, "")) = 0 Then
        Me!txtUnterformularname = Replace(Me!txtFormname, "frm", "sfm")
    End If
    If Len(Nz(Me!txtFormTitle, "")) = 0 Then
        Me!txtFormTitle = Replace(Replace(Me!cboDatasource, "tbl", ""), "qry", "")
    End If
End Sub
```

Listing 2: Aktualisieren des Kombinationsfeldes **cboDatasource**

Solange wir noch mit einer Datenbank entwickeln, beziehen sich beide Objektvariablen auf das gleiche **Database**-Objekt. Mit der Variablen **rst** referenzieren wir die Datensätze der Tabelle **MSysObjects** der Host-Datenbank für das Kombinationsfeld der Datensatzquellen. Dieses weisen wir der **Recordset**-Eigenschaft des Kombinationsfeldes **cboDatasource** zu.

Das Recordset **rstIcons** stellen wir auf die Tabelle **MSysResources** der Add-In-Datenbank ein. Diese Tabelle enthält die Icons, die wir den Schaltflächen der zu erstellenden Formulare zuweisen wollen.

Das Recordset weisen wir den Kombinationsfeldern **cboOKIcon**, **cboAddNewIcon**, **cboDeleteIcon** und **cboEditIcon** zu.

Außerdem stellt die Prozedur beim Laden Standardwerte für die Kontrollkästchen und Textfelder der Schaltflächen ein.

Auswählen einer Datensatzquelle für das zu erstellende Übersichtsformular

Der erste Schritt beim Erstellen ist die Auswahl einer Datensatzquelle, auf deren Basis das Übersichtsformular erstellt werden soll. Das Auswählen eines der Einträge des Kombinationsfeldes **cboDatasource** löst das Ereignis **Nach Aktualisierung** aus, für das wir die Prozedur aus Listing 2 hinterlegen.

Die Prozedur ruft eine weitere Funktion namens **ReadOverviewFieldsFromTable** auf, der wir die gewählte Tabelle oder Abfrage übergeben und die wir gleich beschreiben. Hat diese Funktion die Felder erfolgreich ausgelesen und in die Tabelle **tblOverviewFields** geschrieben, wird das Unterformular **sfmOverviewforms** aktualisiert.

Wenn **txtFormname** noch nicht gefüllt ist, trägt die Prozedur automatisch den Namen der Datenquelle ein, wobei vorher Präfixe wie **tbl** oder **qry** durch **frm** ersetzt werden. Das Unterformular erhält den gleichen Namen, allerdings wird das Präfix **frm** durch **sfm** ersetzt.

Für den Titel wird der gleiche Ausdruck eingetragen, jedoch werden die Präfixe **tbl** und **qry** ersatzlos gestrichen.

Eintragen der Feldnamen für die Übersicht im Unterformular

Die im Unterformular anzuzeigenden Felder werden erstmalig ausgelesen, sobald der Benutzer die Datensatzquelle selektiert hat. Dies löst die Prozedur **cboDatasource_AfterUpdate** und in der Folge die Funktion **ReadOverviewFieldsFromTable** aus. Diese Funktion nimmt den Namen der Tabelle oder Abfrage entgegen, die

```
Public Function ReadOverviewFieldsFromTable(strDataSource As String) As Boolean
    Dim db As DAO.Database, dbc As DAO.Database
    Dim rst As DAO.Recordset, rstSource As DAO.Recordset
    Dim bolReadFields As Boolean, lngControlType As AcControlType
    Dim varRowSourceType As Variant, varRowSource As Variant, varBoundColumn As Variant
    Dim varCaption As Variant, varColumnCount As Variant, varColumnWidths As Variant
    Dim fld As DAO.Field, i As Long
    Set db = CurrentDb
    Set dbc = CodeDb
    Set rst = dbc.OpenRecordset("SELECT * FROM tblOverviewfields WHERE OverviewformID = " & Me!OverviewformID, _
        dbOpenDynaset)
    If Not rst.EOF Then
        rst.MoveLast
    End If
    bolReadFields = True
    Me.Dirty = False
    If Not rst.RecordCount = 0 Then
        bolReadFields = MsgBox("Sollen die Felder neu eingelesen werden?", vbYesNo + vbExclamation, _
            "Felder neu einlesen?") = vbYes
    End If
    If bolReadFields = True Then
        dbc.Execute "DELETE FROM tblOverviewfields WHERE OverviewformID = " & Me!OverviewformID, dbFailOnError
        Set rstSource = db.OpenRecordset(strDataSource, dbOpenDynaset)
        For Each fld In rstSource.Fields
            Call GetControlProperties(fld, varCaption, intControlType, varBoundColumn, varRowSourceType, _
                varRowSource, varColumnCount, varColumnWidths)
            rst.AddNew
            rst!OverviewformID = Me!txtOverviewFormID
            rst!FieldName = fld.Name
            rst!Fieldindex = i
            rst!FieldWidth = 2000
            rst!FieldHeight = 300
            rst!FieldColumn = 1
            rst!Fieldlabel = Nz(varCaption, fld.Name)
            rst!ControlType = lngControlType
            rst!RowSourceType = varRowSourceType
            rst!RowSource = varRowSource
            rst!BoundColumn = varBoundColumn
            rst!ColumnCount = varColumnCount
            rst!ColumnWidths = varColumnWidths
            rst.Update
            i = i + 1
        Next fld
        ReadOverviewFieldsFromTable = True
        Me!sfmOverviewforms.Form.Requery
    Else
        ReadOverviewFieldsFromTable = False
    End If
End Function
```

Listing 3: Aktualisieren der Felder für das Übersichtsformular

als Datensatzquelle gewählt wurde (siehe Listing 3). Dann referenziert sie die Host- und die Add-In-**Database**-Objekte mit den Variablen **db** und **dbc**.

Für die Add-In-Datenbank referenziert sie mit der Variablen **rst** die Datensätze der Tabelle **tblOverviewFields**, die über das Fremdschlüsselfeld **OverviewFormID** mit dem aktuellen Primärschlüsselwert des Hauptformulars verbunden sind. Findet die Funktion hier bereits Datensätze, verschiebt sie den Datensatzzeiger auf den letzten Datensatz, um mit **rst.RecordCount** die Anzahl der Datensätze zu ermitteln. Dann stellt sie die Variable **bolReadFields** auf **True** und speichert alle Änderungen im Formular.

Sind nun bereits Datensätze vorhanden, fragt die Funktion den Benutzer, ob diese neu eingelesen werden sollen. Antwortet der Benutzer mit Ja, behält **bolReadFields** den Wert **True** und die Felder werden nachfolgend neu ausgelesen. Anderenfalls werden die Felder nicht neu eingelesen.

Im Falle des Neueinlesens löscht die Funktion zunächst alle Einträge der Tabelle **tblOverviewfields**, die dem aktuell im Formular angezeigten Datensatz der Tabelle **tblOverviewForms** zugeordnet sind.

Dann stellt sie das Recordset **rstSource** auf die mit **strDataSource** übergebene Tabelle oder Abfrage ein. Anschließend durchläuft sie alle Felder dieses Recordsets. Dabei ruft sie für jedes Feld die Funktion **GetControlProperties** auf und übergibt dieser mit **fld** einen Verweis auf das Feld sowie die Variablen **varCaption**, **intControlType**, **varBoundColumn**, **varRowSourceType**, **varRowSource**, **varColumnCount** und **varColumnWidths**. Diese Funktion, die wir weiter unten beschreiben, liest die Werte für die **Variant**-Parameter aus den Eigenschaften des **Field**-Objekts aus **fld** aus.

Die folgenden Anweisungen tragen einige Informationen in das Recordset auf Basis der Tabelle **tblOverviewfields** ein. Dazu gehört der Primärschlüsselwert des aktuell im Hauptformular angezeigten Datensatzes der Tabelle **tblOverviewForms**, damit die Daten dem jeweils zu erstellenden Formular zugeordnet werden können.

Außerdem trägt sie den Feldnamen, den Index, den Wert 2000 als Feldbreite, die Beschriftung, den Steuerelementinhalt und einige Eigenschaften bezüglich eventuell in der Tabelle definierter Nachschlagefelder ein. Dazu gehören **ControlType**, **RowSourceType**, **RowSource**, **BoundColumn**, **ColumnCount** und **ColumnWidths**. Damit wird für jedes Feld der Datensatzquelle ein Datensatz in der Tabelle **tblOverviewfields** gespeichert und das Unterformular zur Anzeige dieser Felder aktualisiert.

Einlesen der Feldeigenschaften

Die soeben angesprochene Funktion **GetControlProperties** liest die verschiedenen Eigenschaften des mit dem Parameter **fld** übergebenen Feldes ein und liefert diese in Rückgabeparametern zurück (siehe Listing 4).

Einige Eigenschaften von Feldern kann man einfach so einlesen, aber bei einigen ist dies nicht so einfach möglich: Es werden nämlich nicht alle Eigenschaften für jedes Feld angelegt. Beispielweise wird die Eigenschaft **Caption** nur für ein Feld hinterlegt, wenn der Benutzer die Eigenschaft **Beschriftung** über den Tabellenentwurf eingestellt hat. Anderenfalls ist die **Property** namens **Caption** schlicht nicht vorhanden und der Versuch, diese auszulesen, löst einen Fehler aus.

Daher lesen wir diese Werte bei deaktivierter Fehlerbehandlung ein. Das ist der Fall bei der Eigenschaft **Caption**, aber auch bei **DisplayControl**. Letztere legt fest, ob ein Feld im Datenblatt als Textfeld, Kontrollkästchen oder Kombinationsfeld angezeigt wird. Im Falle eines Kombinationsfeldes stehen noch einige weitere Eigenschaften zur Verfügung. Um also herauszufinden, ob wir diese auch einlesen müssen, lesen wir zuerst den Wert der Eigenschaft **DisplayControl** ein. Lautet dieser **acComboBox**, lesen wir auch noch die nun verfügbaren Eigenschaften **BoundColumn**, **RowSourceType**, **RowSource**,

```
Public Function GetControlProperties(fld As DAO.Field, varCaption As Variant, intControlType As AcControlType, _
        varBoundColumn As Variant, varRowSourceType As Variant, varRowSource As Variant, varColumnCount As Variant, _
        varColumnWidths As Variant)
    intControlType = 0
    varCaption = Null
    varBoundColumn = Null
    varRowSourceType = Null
    varRowSource = Null
    varColumnCount = Null
    varColumnWidths = Null
    On Error Resume Next
    varCaption = fld.Properties("Caption")
    intControlType = fld.Properties("DisplayControl")
    Select Case intControlType
        Case acTextBox
            Debug.Print fld.Name, "acTextBox"
        Case acComboBox
            varBoundColumn = fld.Properties("BoundColumn")
            varRowSourceType = fld.Properties("RowSourceType")
            varRowSource = fld.Properties("RowSource")
            varColumnCount = fld.Properties("ColumnCount")
            varColumnWidths = fld.Properties("ColumnWidths")
        Case acCheckBox
            Debug.Print fld.Name, "acCheckBox"
        Case Else
            Debug.Print "anderer Controltype", intControlType
    End Select
    On Error GoTo 0
End Function
```

Listing 4: Einlesen der Feldeigenschaften

ColumnCount und **ColumnWidths** in die entsprechenden Rückgabeparameter ein. Im Falle von **acTextBox** oder **acCheckBox** müssen keine weiteren Parameter eingelesen werden.

Übersichtsformulare per Code erstellen

Damit haben wir alle Informationen erfasst, die wir zum Erstellen des Übersichtsformulars benötigen und können nun den Code zusammenstellen. Dabei beginnen wir mit dem Unterformular, dass die Daten der Tabelle in der Datenblattansicht anzeigen soll. Wenn wir dieses erstellt haben, fügen wir den Code zum Erstellen des Hauptformulars hinzu. Dieses wird schließlich auch ein Unterformularsteuerelement enthalten, das wir mit dem Unterformular füllen.

Funktion zum Erstellen eines neuen Formulars

Wir verwenden eine einfache Funktion zum Erstellen der beiden Formulare. Diese heißt **CreateNewForm** und erwartet den Namen des zu erstellenden Formulars als einzigen Parameter. Wie daran gut zu erkennen ist, erledigt die Funktion nichts anderes, als ein neues Formular zu erstellen (siehe Listing 5). Sie prüft mit der Hilfsfunktion ExistsForm, ob bereits ein Formular mit dem angegebenen Namen vorhanden ist:

```
Public Function ExistsForm(strForm As String) As Boolean
    Dim objForm As AccessObject
    For Each objForm In CurrentProject.AllForms
        If objForm.Name = strForm Then
```

Add-In für Übersichtsformulare im Einsatz

Im Beitrag »Übersichtsformular per Mausklick« (www.access-im-unternehmen.de/1528) haben wir gezeigt, wie Sie ein Access-Add-In programmieren können, mit dem wir Übersichtsformulare auf Basis von Tabellen oder Abfragen schnell erstellen können. In diesem Beitrag schauen wir uns nun an, wie wir dieses optimal nutzen können, um schnell die gewünschten Übersichtsformulare zu erstellen. Dabei werfen wir auch einen Blick auf die zu erstellenden VBA-Prozeduren für die Schaltflächen und das Suchfeld, das wir optional zu diesem Formular hinzufügen können.

Das Access-Add-In, das wir in dem oben genannten Beitrag vorgestellt haben, bietet hilfreiche Unterstützung, wenn Sie schnell einfache Übersichtsformulare auf Basis vorhandener Tabellen in Ihre Anwendung integrieren wollen. Diese Übersichtsformulare gehen über die Formulare hinaus, die wir mit den entsprechenden Assistenten von Access anlegen können.

Sie erstellen zusätzlich noch ein Suchfeld sowie Schaltflächen zum Erstellen neuer Datensätze und zum Bearbeiten oder Löschen vorhandener Datensätze.

Dies geschieht über ein Konfigurationsformular, das zuerst die Auswahl der Tabelle oder Abfrage erlaubt, deren Daten wir im Übersichtsformular anzeigen wollen. Das Formular öffnen wir über den Ribbon-Befehl **amvOverviewformBuilder** aus Bild 1. Es gibt noch eine zweite Möglichkeit, diesen Assistenten zu öffnen – in diesem Fall markieren wir zuerst die Tabelle oder Abfrage, die

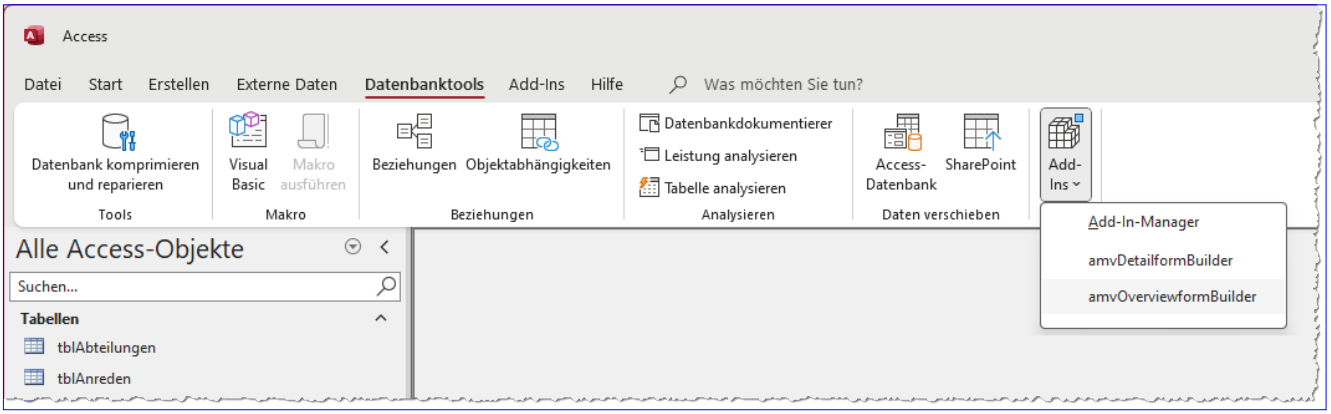


Bild 1: Aufruf des Access-Add-Ins zum Erstellen von Übersichtsformularen

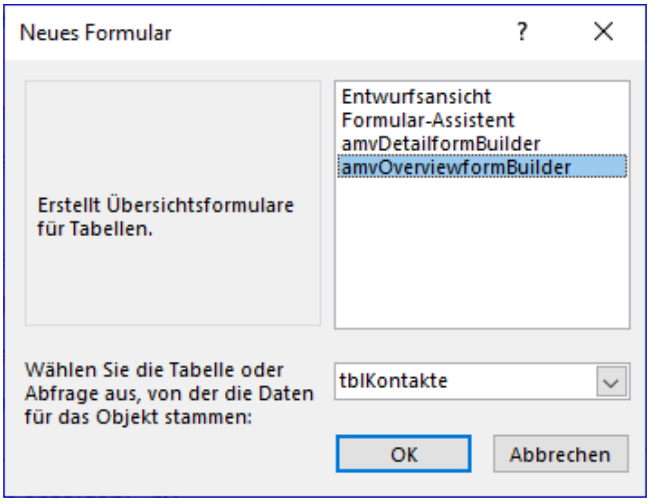


Bild 2: Aufruf des Add-Ins als Assistent zum Erstellen eines Formulars

wir im Übersichtsformular abbilden wollen, und betätigen dann den Befehl **ErstellenlFormularelFormular-Assistent**. Dies öffnet den Dialog aus Bild 2, mit dem wir einen der Assistenten auswählen können.

Leider wird der Eintrag, der dafür sorgt, dass unser Add-In hier erscheint, nicht von allen Access-Versionen korrekt ausgeführt, sodass Sie gegebenenfalls den Weg über das Add-In-Menü gehen müssen.

In beiden Fällen gelangen wir zum Formular Übersichtsformulare erstellen, wo entweder bereits die entsprechende Datensatzquelle ausgewählt ist oder wo wir dies noch nachholen müssen wir in Bild 3.

Nach der Auswahl der Tabelle, hier **tblMitarbeiter**, geschehen bereits einige Schritte automatisch. Zunächst macht der Assistent Vorschläge für den Namen der zu erstellenden Formulare, also für das Haupt- und das Unterformular. Und auch der Formulartitel wird aus dem Namen der gewählten Datensatzquelle abgeleitet (siehe Bild 4).

Wir können diese Informationen jedoch anschließend nach den eigenen Wünschen anpassen – gegebenenfalls gibt es ja bereits Formulare mit den vorgeschlagenen Namen.

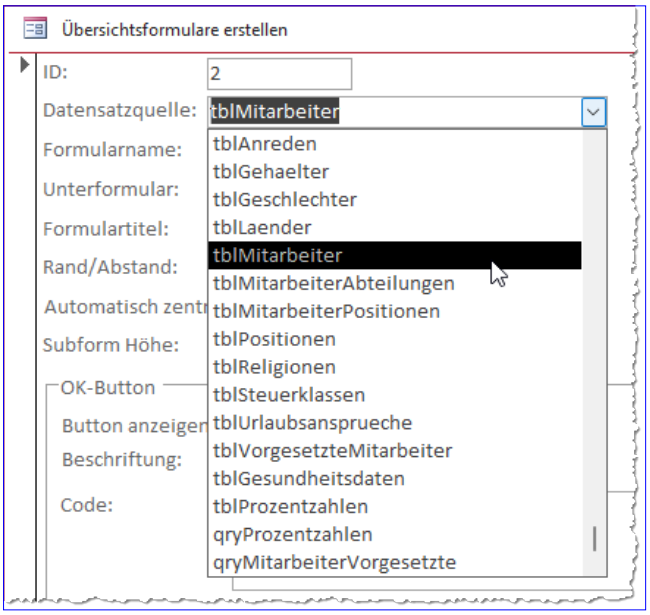


Bild 3: Auswahl der Tabelle oder Abfrage

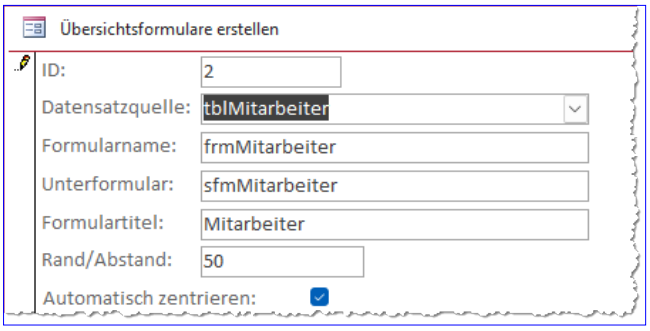


Bild 4: Automatisch generierte Objektnamen und Beschriftungen

Vorbereitung der zu konfigurierenden Felder

Wesentlich wichtiger ist jedoch, dass nach der Auswahl der Tabelle oder Abfrage alle Felder dieser Datensatzquelle angezeigt werden, damit wir diese konfigurieren können. Dies geschieht ebenfalls automatisch und sieht zunächst wie in Bild 5 aus.

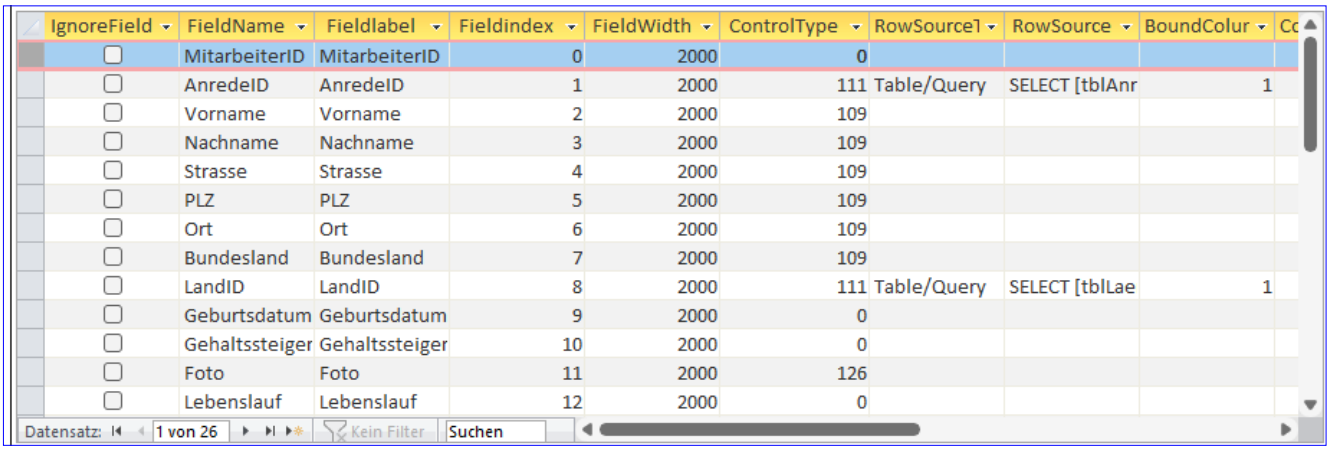
Hier finden wir einige Einstellungsmöglichkeiten für die Felder. Die erste Spalte erlaubt es uns, die Felder auszuwählen, die nicht angezeigt werden sollen.

In der zweiten finden wir den eigentlichen Feldnamen, in der dritten einen Vorschlag für die Spaltenüberschrift für dieses Feld. Hier wird üblicherweise zunächst der Feldname eingetragen und falls vorhanden der Wert der Eigenschaft **Beschriftung** aus dem Tabellenentwurf.

Die Spalte **Fieldindex** legt fest, in welcher Reihenfolge die Felder angezeigt werden sollen. Die Spalte **FieldWidth** enthält einen Wert für die Spaltenbreite in Twips.

Damit können wir festlegen, wie breit die Spalten direkt beim Öffnen sind.

Unter **ControlType** sehen wir, welchen Typ das jeweilige Steuerelement hat. **0** und **109** stehen für Textfelder, **111** für Nachschlagefelder und **106** für **Ja/Nein**-Felder.



IgnoreField	FieldName	Fieldlabel	Fieldindex	FieldWidth	ControlType	RowSourceT	RowSource	BoundColur	Cc
☐	MitarbeiterID	MitarbeiterID	0	2000	0				
☐	AnredeID	AnredeID	1	2000	111	Table/Query	SELECT [tblAnr	1	
☐	Vorname	Vorname	2	2000	109				
☐	Nachname	Nachname	3	2000	109				
☐	Strasse	Strasse	4	2000	109				
☐	PLZ	PLZ	5	2000	109				
☐	Ort	Ort	6	2000	109				
☐	Bundesland	Bundesland	7	2000	109				
☐	LandID	LandID	8	2000	111	Table/Query	SELECT [tblLae	1	
☐	Geburtsdatum	Geburtsdatum	9	2000	0				
☐	Gehaltssteiger	Gehaltssteiger	10	2000	0				
☐	Foto	Foto	11	2000	126				
☐	Lebenslauf	Lebenslauf	12	2000	0				

Datensatz: 1 von 26 | Kein Filter | Suchen

Bild 5: Einstellen der anzuzeigenden Felder in der Datenblattansicht

Die nachfolgenden Eigenschaften **RowSourceType**, **RowSource**, **BoundColumn**, **ColumnCount** und **ColumnWidths** werden nur für die Darstellung von Nachschlagefeldern benötigt.

Schaltflächen konfigurieren

Haben wir die anzuzeigenden Felder eingestellt, kommen wir zu den Schaltflächen. Hier können wir zunächst festlegen, welche Schaltflächen überhaupt angezeigt werden sollen. Wir schauen uns das am Beispiel des **OK**-Buttons an (siehe Bild 6).

Hier sehen wir die folgenden Eigenschaften:

- **Button anzeigen**: Legt fest, ob diese Schaltfläche überhaupt angezeigt werden soll.
- **Beschriftung**: Stellt die Beschriftung des Buttons ein.
- **Code**: Legt den Code fest, der durch das **Beim Klicken**-Ereignis ausgelöst werden soll, im Bild beispielsweise eine einfache **Debug.Print**-Anweisung.
- **Icon**: Legt das anzuzeigende Icon fest.
- **Bei Enter-Taste auslösen**: Gibt an, ob diese Schaltfläche ausgelöst werden soll, wenn der Benutzer die Eingabetaste betätigt.

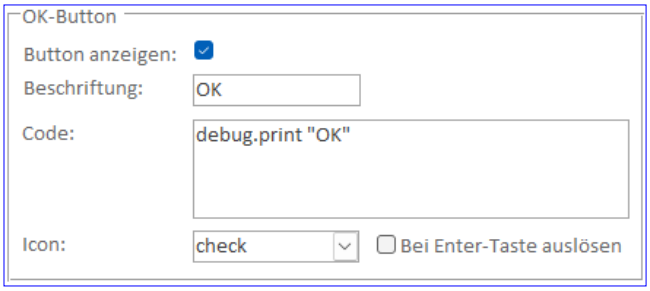


Bild 6: Einstellungen für den **OK**-Button

Die Ok-Schaltfläche wird immer unten angezeigt. Wir können hier einfach den folgenden Code eingeben, damit das Formular beim Klicken geschlossen wird:

```
DoCmd.Close acForm, Me.Name
```

Gegebenenfalls rücken wir den Code noch um vier Zeichen ein, damit dieser auch optisch korrekt abgebildet wird.

Anlegen-Button konfigurieren

Der **Anlegen**-Button kann je nach Bedarf verschiedene Aktionen auslösen. Die einfachste wäre, einfach einen neuen leeren Datensatz im Unterformular aufzurufen.

Dazu würden wir die folgenden Anweisungen einfügen:

```
Me!sfmMitarbeiter.SetFocus
DoCmd.GoToRecord Record:=acNewRec
```