

Verweise beim Start prüfen

Beim Einsatz von Access-Anwendungen gibt es manchmal Probleme, weil die Verweise auf Bibliotheken nicht korrekt gesetzt sind. Dann werden plötzlich VBA-Funktionen wie »Left« oder »Right« als fehlerhaft bewertet. Der Benutzer steht dann dumm da, weil er nicht weiß, was zu tun ist. Dieser Beitrag zeigt, wie Sie das Problem nachstellen können und welche Lösungsmöglichkeiten es gibt – von der manuellen Variante bis zur automatischen Korrektur des Verweises beim Start der Anwendung.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1904_VerweiseBeimStart.accdb**.

Fehlerhafter Verweis

Probleme wie die oben genannten treten meist auf, wenn Sie eine Anwendung auf Ihrem Rechner entwickeln und diese dann an einen Benutzer weitergeben, auf dessen Rechner nicht alle per Verweis referenzierten Bibliotheken vorliegen – oder diese nicht in der richtigen Version vorliegen.

Verweise finden Sie grundsätzlich im **Verweise**-Dialog, den Sie vom VBA-Editors aus (zu öffnen mit **Strg + G** oder **Alt + F11**) mit dem Menübefehl **Extras|Verweise** anzeigen. Dieser Dialog zeigt die Liste der eingebundenen Bibliotheken, die bei einer neuen, leeren Access-Anwendung auf Basis von Access 2016 etwa wie in Bild 1 aussieht. Die Bibliotheken hinter diesen vier Einträge sollten überall, wo Access 2016 installiert ist, auch vorhanden sein.

Bei umfangreicheren Anwendungen kommen meist noch Verweise auf weitere Bibliotheken hinzu, zum Beispiel um Word, Excel oder Outlook zu automatisieren, die Datenzugriffs-Bibliothek ADODB zu nutzen oder auch um die Befehle für die Verarbeitung von XML hinzuzufügen.

Wenn Sie nun einen Verweis auf Outlook hinzufügen, die Datenbank auf den Zielrechner übertragen und

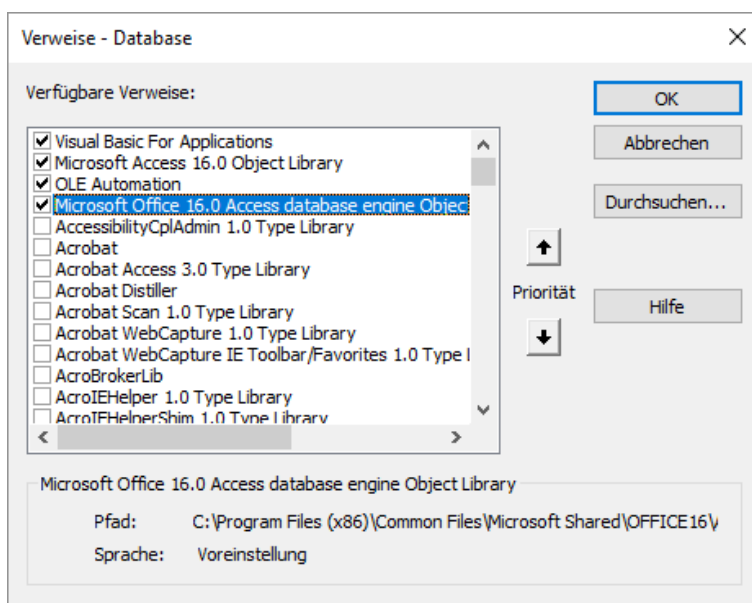


Bild 1: Der Verweise-Dialog von Access

dann aber Outlook gar nicht auf dem Zielrechner vorliegt, gibt es Probleme wegen des fehlenden Verweises.

Manchmal gibt es auch Probleme, wenn Sie mit einem Verweis auf die Objektbibliothek von Outlook 2016 programmieren und auf dem Zielrechner ist eine ältere Version von Outlook vorhanden.

Manuelle Lösung

Vorausgesetzt, die Bibliothek, auf die der Verweis zeigen soll, ist auf dem Zielrechner installiert und registriert, können Sie wie folgt vorgehen, um den Verweis zu erneuern:

- Öffnen Sie den **Verweise**-Dialog.

- Entfernen Sie den mit **NICHT VORHANDEN** markierten Verweis.
- Fügen Sie den Verweis auf die gewünschte Bibliothek zur Liste der Verweise hinzu, indem Sie den entsprechenden Eintrag per Haken selektieren.
- Schließen Sie den **Verweise**-Dialog.

Danach sollte die Anwendung dann wie gewohnt funktionieren. Eine Ausnahme könnte sein, dass Sie in Ihrer Anwendung tatsächlich Funktionen einer neueren Bibliothek verwenden, die in der älteren, auf dem Zielrechner vorhandenen Version noch nicht verfügbar ist.

Wenn Sie eine Anwendung beim Kunden laufen haben, für die Sie regelmäßig neue Versionen liefern, dann ist es für den Kunden natürlich mühselig, immer die Verweise zu aktualisieren – daher stellen wir nun eine Alternative vor.

Verweise automatisch beim Start aktualisieren

Die Alternative lautet, beim Start einer Anwendung die Verweise zu prüfen und diese gegebenenfalls zu erneuern. Wenn ein Verweis nicht gesetzt werden kann, weil die entsprechende Bibliothek nicht verfügbar ist, soll eine Meldung erscheinen, die den Benutzer darüber informiert, damit dieser mit dem Entwickler der Software herausfinden kann, wo das Problem in diesem Fall liegt.

Zu Beispielzwecken nehmen wir an, dass wir Verweise auf die Bibliotheken für den Zugriff auf die Office-Anwendungen Excel, Outlook und Word hinzufügen sowie einen auf die Bibliothek **Microsoft XML, v6.0** (siehe Bild 2).

Bei den mit diesen Verweisen referenzierten Bibliotheken handelt es sich sämtlich um registrierte Bibliotheken, für die es eine GUID

gibt. Diese wollen wir nun zunächst ermitteln, indem wir diese per VBA-Routine im Direktbereich des VBA-Editors ausgeben:

```
Public Sub VerweisePruefenUndErneuern()
    Dim objReference As Reference
    Dim strGUID As String
    For Each objReference In References
        Debug.Print objReference.Name
        Debug.Print objReference.FullPath
        Debug.Print objReference.Guid
    Next objReference
End Sub
```

Das Ergebnis sieht dann so aus:

```
VBA
C:\Program Files (x86)\...\VBA7.1\VBE7.DLL
{000204EF-0000-0000-C000-000000000046}
Access
C:\Program Files (x86)\...\MSACC.OLB
{4AFC9A0-5F99-101B-AF4E-00AA003F0F07}
stdole
C:\Windows\SysWOW64\stdole2.tlb
{00020430-0000-0000-C000-000000000046}
DAO
C:\Program Files (x86)\...\OFFICE16\ACEDAO.DLL
```

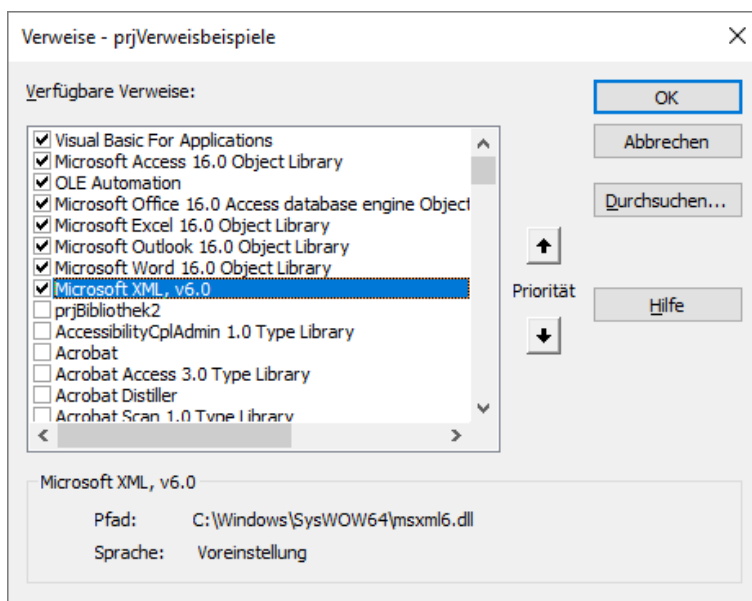


Bild 2: Verweise-Dialog mit zusätzlichen Verweisen

```
{4AC9E1DA-5BAD-4AC7-86E3-24F4CDCECA28}
Excel
C:\Program Files (x86)\...\EXCEL.EXE
{00020813-0000-0000-C000-000000000046}
Outlook
C:\Program Files (x86)\...\MSOUTL.OLB
{00062FFF-0000-0000-C000-000000000046}
Word
C:\Program Files (x86)\...\MSWORD.OLB
{00020905-0000-0000-C000-000000000046}
MSXML2
C:\Windows\SysWOW64\msxml6.dll
{F5078F18-C551-11D3-89B9-0000F81FE221}
```

Nun wollen wir ja gar nicht alle Einträge prüfen, sondern nur diejenigen, die nicht eingebaut sind. Das prüfen wir durch eine **If...Then**-Bedingung, mit der wir den Wert der Eigenschaft **BuiltIn** untersuchen. Ist diese **True**, handelt es sich um eine eingebaute Eigenschaft, die wir nicht ersetzen müssen, da diese meist vorhanden sind. Hat **BuiltIn** den Wert **False**, müssen wir handeln: Dann könnte es sein, dass ein falscher Verweis zu Fehlern führt. Mit der folgenden geänderten Version der Prozedur **VerweisePruefen-**

```
Public Sub VerweisePruefenUndErneuern()
    Dim objReference As Reference
    Dim objReferenceNew As Reference
    Dim strReference As String
    Dim strGUID As String
    For Each objReference In References
        If objReference.BuiltIn = False Then
            If objReference.IsBroken = True Then
                strGUID = objReference.Guid
                strReference = objReference.Name
                References.Remove objReference
                Set objReferenceNew = References.AddFromGuid(strGUID, 0, 0)
                If objReference Is Nothing Then
                    MsgBox "Der Verweis auf die Bibliothek '" & strReference & "' fehlt."
                End If
            End If
        End If
    Next objReference
End Sub
```

Listing 1: Fehlende Verweise werden direkt erneuert

UndErneuern geben wir nur noch die Verweise aus, die wir erneuern können:

```
Public Sub VerweisePruefenUndErneuern()
    Dim objReference As Reference
    Dim strGUID As String
    For Each objReference In References
        If objReference.BuiltIn = True Then
            Debug.Print objReference.Name
            Debug.Print objReference.FullPath
            Debug.Print objReference.Guid
        End If
    Next objReference
End Sub
```

Damit erhalten wir dann ein ähnliches Ergebnis wie weiter oben, nur dass diesmal die eingebauten Verweise nicht berücksichtigt werden.

Was fangen wir nun damit an? Wir könnten direkt in dieser Prozedur ein paar Zeilen unterbringen, die dafür sorgen, dass die Verweise auf Gültigkeit überprüft und gegebenenfalls erneuert werden. Das würde dann etwa wie in Listing 1 aussehen. Hier

durchlaufen wir wieder alle **Reference**-Objekte, schließen aber durch die ersten beiden **If...Then**-Bedingungen zuerst die eingebauten und dann die vorhandenen Verweise aus.

Sind beide **If...Then**-Bedingungen wahr, merken wir uns in den Variablen **strGUID** und **strReference** die GUID und den Namen der referenzierten Bibliothek. Dann fügen wir die Referenz auf eine Bibliothek über die Methode **AddFromGUID** wieder zur Liste **References** hinzu. Dabei

übergeben wir als Parameter die GUID der Bibliothek sowie den Wert 0 für die beiden Parameter **Major** und **Minor**.

Eine fehlende Bibliothek sollte damit wieder referenziert werden können.

Diese Prozedur rufen wir am besten beim Start der Anwendung auf, damit keine Fehler durch fehlende Verweis auftreten. Wie das gelingt, zeigen wir am Ende des Beitrags.

Benötigte Verweise an anderer Stelle speichern

Das Problem ist nur: Wenn der Benutzer auf irgendeine Weise einen der Verweise entfernt, ohne diesen wieder hinzuzufügen, treten in der Folge Probleme auf an den Stellen, wo im Code Elemente dieser Bibliothek verwendet werden.

Wenn wir dann beim Starten der Anwendung die Prozedur aufrufen, durchläuft diese wieder alle Verweise und prüft die Verweise mit der Eigenschaft **IsBroken**. Wenn jedoch ein Verweis auf eine Bibliothek gar nicht mehr in der Liste der Verweise enthalten ist, kann er mit der hier vorliegenden Version der Prozedur auch nicht ersetzt werden – die Prozedur erfährt ja gar nicht, dass ein Verweis fehlt. Das wird erst später offensichtlich, wenn die Anwendung das erste Mal auf Elemente der mit dem Verweis referenzierten Bibliothek zugreift und dadurch Laufzeitfehler ausgelöst werden.

Also wollen wir, bevor wir die Anwendung an den Benutzer weitergeben, die Verweise, die überprüft werden sollen, in irgendeiner Form speichern. Dabei gibt es zwei Möglichkeiten:

- Wir speichern die GUIDs fest im Code.
- Wir legen eine Tabelle an, der wir die Informationen über die Verweise hinzufügen. Diese Tabelle sollte dann im Frontend der Datenbankanwendung liegen, wenn diese in Frontend und Backend aufgeteilt ist.

Die Basis für beide Varianten sieht beispielsweise wie in Listing 2 aus. Diese Funktion nimmt den Parameter **strGUID** entgegen. Im Direktbereich könnten wir die Funktion wie folgt aufrufen:

```
? VerweiseErneuern("{00062FFF-0000-0000-C000-000000000046}")
```

Die Funktion stellt die Variablen **bolReferenceBroken** auf **False** und **bolReferenceMissing** auf **True** ein. Dann versucht sie, einen Verweis auf die Bibliothek mit der mit **strGUID** übergebenen GUID zu finden. Dies geschieht in einer **For Each**-Schleife über alle Elemente der Auflistung **References**. Wenn der aktuell mit **objReference** referenzierte Verweis die gesuchte GUID enthält, können wir die Variable **bolReferenceMissing** schon einmal auf **False** einstellen, denn der Verweis ist ja zumindest vorhanden.

Dann prüft die Prozedur mit der Eigenschaft **IsBroken** noch, ob der Verweis als **NICHT VORHANDEN** markiert ist. Das ist etwas missverständlich, denn **NICHT VORHANDEN** gibt ja nicht an, dass der Verweis fehlt, sondern die referenzierte Bibliothek. Jedenfalls stellt die Funktion die Variable **bolReferenceBroken** auf **True** ein, wenn **IsBroken** den Wert **True** aufweist. Sobald der Verweis passend zu **strGUID** gefunden wurde, verlassen wir dann auch mit **Exit For** die Schleife.

objReference enthält somit auch einen Verweis auf das **Reference**-Objekt. Damit gehen wir in eine **If...Then**-Bedingung, in der wir prüfen, ob entweder **bolReferenceMissing** oder **bolReferenceBroken** den Wert **True** hat. Das ist nun entweder der Fall, wenn der Verweis gar nicht vorhanden ist oder dieser »broken« ist.

Dann versucht die Funktion, den Verweis mit der **Remove**-Methode der **References**-Auflistung zu entfernen und diesen dann über die Methode **AddFromGUID** erneut hinzuzufügen, wobei ein Verweis auf das neue **Reference**-Objekt in der Variablen **objReference** landet – aber auch nur, wenn das Hinzufügen erfolgreich war.

Wenn etwa die Datei hinter der GUID nicht vorhanden ist, gelingt dies nicht und **objReference** bleibt leer.

Ist **objReference** danach leer, wird der Wert **False** mit dem Funktionswert zurückgegeben.

Von einer Prozedur aus könnten wir diese Funktion wie etwa in Listing 3 aufrufen. Dies würde, wenn die Outlook-Bibliothek nicht vorhanden oder fehlerhaft ist, eine Meldung liefern.

Reparieren eines Verweises testen

Das wollen wir nun ausprobieren. Solange der Verweis nicht auf eine nicht vorhandene Datei zeigt, was man nach dem Öffnen des **Verweise**-Dialog schnell durch eine Prüfung auf den Hinweis **NICHT VORHANDEN** prüfen kann.

Ist der Verweis fehlerhaft, sieht der Eintrag im **Verweise**-Dialog wie in Bild 3 aus.

```
Public Function VerweiseErneuern(strGUID As String) As Boolean
    Dim objReference As Reference
    Dim bolReferenceMissing As Boolean
    Dim bolReferenceBroken As Boolean
    bolReferenceBroken = False
    bolReferenceMissing = True
    VerweiseErneuern = True
    For Each objReference In References
        If objReference.Guid = strGUID Then
            bolReferenceMissing = False
            If objReference.IsBroken = True Then
                bolReferenceBroken = True
            End If
            Exit For
        End If
    Next objReference
    If bolReferenceMissing = True Or bolReferenceBroken = True Then
        On Error Resume Next
        References.Remove objReference
        Set objReference = References.AddFromGuid(strGUID, 0, 0)
        If objReference Is Nothing Then
            VerweiseErneuern = False
        End If
    End If
End Function
```

Listing 2: Erneuern des Outlook-Verweises

Gegebenenfalls erhalten Sie beim Öffnen des **Verweise**-Dialogs auch gleich eine Meldung wie in Bild 4.

Wie aber wollen Sie das Reparieren eines Verweises testen, wenn sich auf Ihrem System überhaupt kein Probleme verursachender Verweis befindet? Dann

```
Public Sub VerweiseErneuernAufruf()
    Dim strGUID As String
    Dim strReference As String
    strGUID = "{00062FFF-0000-0000-C000-000000000046}"
    strReference = "Microsoft Outlook x.0 Object Library"
    If VerweiseErneuern(strGUID, strReference) = False Then
        MsgBox "Der Verweis auf die Bibliothek '" & strReference & "' ist fehlerhaft und wurde entfernt oder ist nicht '" & strReference & "' vorhanden. Fügen Sie den Verweis auf die Bibliothek manuell hinzu."
    End If
End Sub
```

Listing 3: Aufruf der Funktion **VerweiseErneuern**

können Sie selbst ein solches Problem schaffen. Dazu suchen Sie beispielsweise die Datei **MSOUTL.OLB** auf, unter Windows 10 und Office 2016 etwa im Verzeichnis **C:\Program Files (x86)\Microsoft Office\root\Office16** zu finden (den Pfad finden Sie aber auch über die **FullPath**-Eigenschaft des **Reference**-Objekts).

Diese benennen wir einfach in **_MSOUTL.OLB** um. Die dabei erscheinende Meldung, die nach Administratorberechtigungen verlangt, können Sie bestätigen. Vergessen Sie nicht, die Datei später wieder umzubenennen!

Wenn Sie nun die Beispieldatenbank starten und in den **Verweise**-Dialog schauen, finden Sie einen als **NICHT VORHANDEN** markierten Eintrag der Bibliothek **Microsoft Outlook 16.0 Object Library**.

Diesen können Sie dann versuchen, mit der obigen Funktion wiederherzustellen. Das wird in diesem Fall scheitern, da der Verweis tatsächlich ins Leere zeigt und die hinter der GUID angegebene Datei **MSOUTL.OLB** nicht vorhanden ist.

Den Fall, dass die Outlook-Bibliothek nicht in der gewünschten Version vorhanden ist und es dadurch einen Fehler gibt, konnten wir mangels anderer Outlook-OLB-Datei nicht nachstellen. Die Prozedur sollte jedoch in diesem Fall den alten

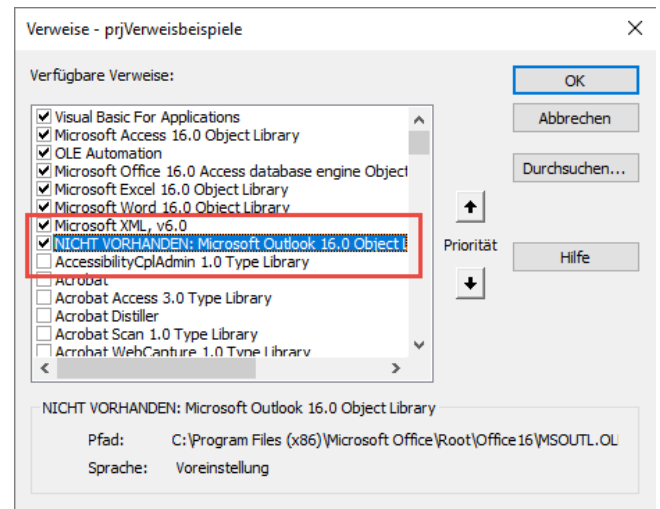


Bild 3: Der mit NICHT VORHANDEN markierte Eintrag

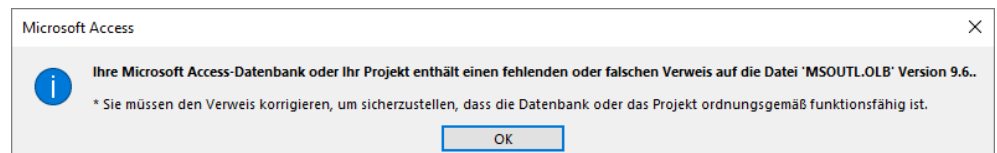


Bild 4: Hinweis auf ein Verweis-Problem

```
Public Sub VerweiseErneuernAufrufMehrere()
    Dim strGUID As String
    Dim strReference As String
    Dim strMeldung As String
    strGUID = "{00062FFF-0000-0000-C000-000000000046}" 'Outlook
    strReference = "Microsoft Outlook x.0 Object Library"
    If VerweiseErneuern(strGUID, strReference) = False Then
        strMeldung = strMeldung & strReference & vbCrLf
    End If
    strGUID = "{00020813-0000-0000-C000-000000000046}" 'Excel
    strReference = "Microsoft Excel x.0 Object Library"
    If VerweiseErneuern(strGUID, strReference) = False Then
        strMeldung = strMeldung & strReference & vbCrLf
    End If
    ... weitere Verweise
    If Len(strMeldung) > 0 Then
        strMeldung = "Ein oder mehrere Verweise konnten nicht wiederhergestellt werden:" & _
            & vbCrLf & vbCrLf & strMeldung
        strMeldung = strMeldung & vbCrLf & "Fügen Sie diese manuell hinzu oder kontaktieren " & _
            & "Sie den Entwickler."
        MsgBox strMeldung, , "Fehlerhafte Verweise"
    End If
End Sub
```

Listing 4: Erneuern des Outlook-Verweises

Verweis löschen und den auf die korrekte Version hinzufügen können.

Funktion für mehrere Verweise aufrufen

Wenn Sie gleich für mehrere Verweise prüfen wollen, ob diese noch vorhanden und einsatzbereit sind, können Sie die Funktion auch für mehrere Verweise in einer Routine aufrufen (siehe Listing 4).

Hier übergeben wir nacheinander die Parameter für die Prüfung und gegebenenfalls Erneuerung verschiedener Verweise. Immer wenn die Funktion den Wert **False** zurückliefert, wird der Variablen **strMeldung** eine Zeile mit der Bezeichnung der jeweils nicht reparierten Referenz hinzugefügt. Am Ende prüft die Prozedur, ob mindestens eine Referenz nicht repariert werden konnte und ergänzt die Meldung aus **strMeldung** aus eine führende und eine folgende Zeile, die dem Benutzer mitteilen, welche Schritte nun nötig sind.

Speichern der Referenzen in einer Tabelle

Wir können die Daten für die Verweise statt im Code auch in einer Tabelle speichern, was ein späteres Hinzufügen oder Entfernen der Verweise erleichtert. Diese Tabelle heißt **tblReferences** und sieht nach dem Füllen etwa wie in Bild 5 aus.

Um uns die Arbeit zu erleichtern, schreiben wir eine Prozedur, mit der wir die Daten der Verweise in die Tabelle eintragen. Diese sieht wie in Listing 5 aus. Hier durchlaufen wir alle Verweise der aktuellen Datenbank und prüfen, welche davon nicht eingebaut sind. Für diese fügen wir dann per **INSERT INTO**-Anweisung jeweils einen Datensatz zur Tabelle **tblReferences** hinzu.

Vorher leeren wir die Tabelle noch per **DELETE**-Anweisung.

ReferenceID	ReferenceGUID	ReferenceName
1	{00020430-0000-0000-C000-000000000046}	stdole
2	{4AC9E1DA-5BAD-4AC7-86E3-24F4CDCECA28}	DAO
3	{00020813-0000-0000-C000-000000000046}	Excel
4	{00020905-0000-0000-C000-000000000046}	Word
5	{F5078F18-C551-11D3-89B9-0000F81FE221}	MSXML2
6	{00062FFF-0000-0000-C000-000000000046}	Outlook
*	(Neu)	

Bild 5: Tabelle zum Speichern der Verweisinformationen

Sie können nach dem Durchlaufen dieser Prozedur noch manuell die Bezeichnungen der Verweise anpassen, also etwa **Microsoft Outlook x.0 Object Library** statt **Outlook**. Das Ergebnis sieht dann wie in Bild 6 aus.

Um die Funktion **VerweiseErneuern** für jeden Eintrag der Tabelle einmal aufzurufen, verwenden wir die Prozedur aus Listing 6. Diese erstellt ein Recordset auf Basis der Tabelle **tblReferences**. Danach durchläuft sie alle Datensätze dieses Recordsets und ruft für

ReferenceID	ReferenceGUID	ReferenceName
1	{00020430-0000-0000-C000-000000000046}	OLE Automation
2	{4AC9E1DA-5BAD-4AC7-86E3-24F4CDCECA28}	Microsoft Office x.0 Access database
3	{00020813-0000-0000-C000-000000000046}	Microsoft Excel x.0 Object Library
4	{00020905-0000-0000-C000-000000000046}	Microsoft Word
5	{F5078F18-C551-11D3-89B9-0000F81FE221}	Microsoft XML, v6.0
6	{00062FFF-0000-0000-C000-000000000046}	Microsoft Outlook x.0 Object Library
*	(Neu)	

Bild 6: Verweisinformationen mit ausgeschriebenen Bibliotheken

```
Public Sub VerweiseInDB()
    Dim objReference As Reference
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "DELETE FROM tblReferences", dbOpenDynaset
    For Each objReference In References
        If objReference.BuiltIn = False Then
            db.Execute "INSERT INTO tblReferences (ReferenceGUID, ReferenceName) " _
                & "VALUES(' " & objReference.Guid & "', ' " _
                & objReference.Name & " ')", dbFailOnError
        End If
    Next objReference
End Sub
```

Listing 5: Eintragen der Verweisdaten in die Tabelle **tblReferences**

```
Public Sub VerweiseErneuernAusTabelleAufruf()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblReferences", dbOpenDynaset)
    Dim strMeldung As String
    Do While Not rst.EOF
        If VerweiseErneuern(rst!ReferenceGUID) = False Then
            strMeldung = strMeldung & rst!ReferenceName & vbCrLf
        End If
        rst.MoveNext
    Loop
    If Len(strMeldung) > 0 Then
        strMeldung = "Ein oder mehrere Verweise konnten nicht wiederhergestellt werden:" & vbCrLf & vbCrLf & strMeldung
        strMeldung = strMeldung & vbCrLf & "Fügen Sie diese manuell hinzu oder kontaktieren Sie den Entwickler."
        MsgBox strMeldung, , "Fehlerhafte Verweise"
    End If
End Sub
```

Listing 6: Verweise aus der Tabelle erneuern

jeden Datensatz die Funktion **VerweiseErneuern** auf und übergibt den Wert des Feldes **ReferenceGUID** an die Funktion.

Liefert die Funktion den Wert **False** zurück, was bedeutet, dass ein fehlerhafter Verweis nicht repariert werden konnte, wird die Zeichenkette **strMeldung** um den Namen der Bibliothek in jeweils einer eigenen Zeile ergänzt. Ist mindestens ein Verweis nicht wiederherstellbar, erscheint eine Meldung wie aus Bild 7.

Prozedur beim Start der Anwendung aufrufen

Um diese Prozedur beim Start einer Anwendung aufzurufen, haben Sie zwei Möglichkeiten:

- Sie legen das **AutoExec**-Makro an und fügen den Namen der Prozedur für eine neue Makroaktion namens **AusführenCode** hinzu. Dazu ändern Sie die Prozedur **VerweiseErneuernAusTabelleAufruf** in eine Funktion um (Entwurf des Makros siehe Bild 8).
- Oder Sie fügen einem Formular, dass Sie für die Eigenschaft **Formular anzeigen** in den Access-

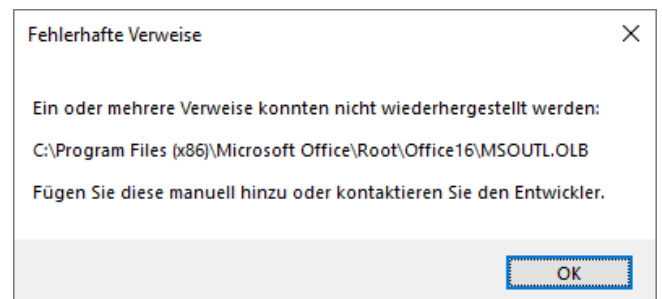


Bild 7: Ein Verweis konnte nicht wiederhergestellt werden.

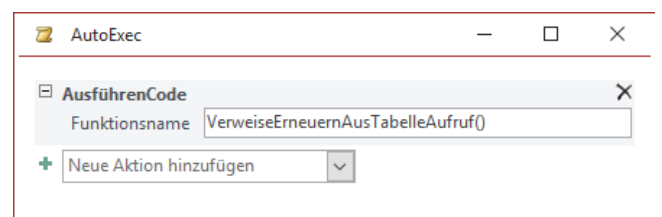


Bild 8: Entwurf des AutoExec-Makros

Optionen der Anwendung einstellen, eine **Form_Load**-Prozedur hinzu, welche die Prozedur **VerweiseErneuernAusTabelleAufruf** startet.