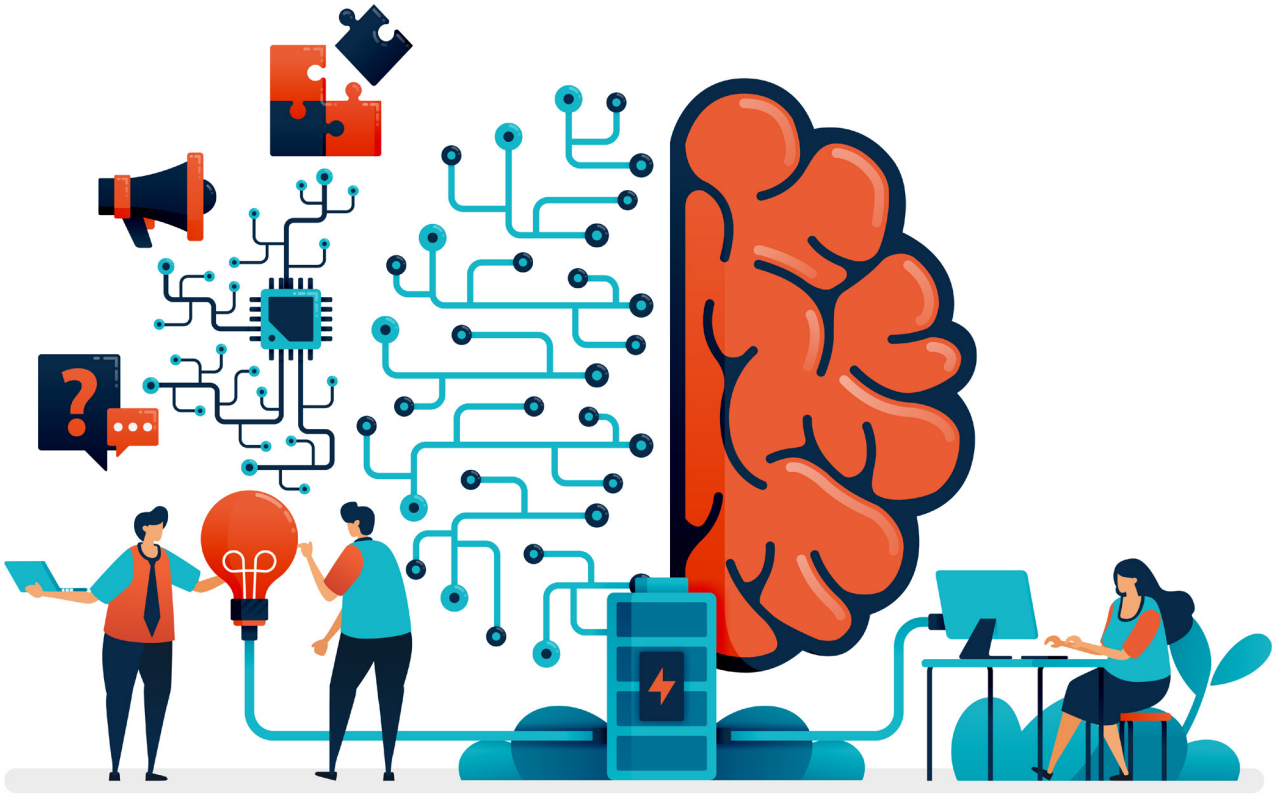
AUSGABE 03/2024

VISUAL BASIC ENTWICKLER

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



IN DIESEM HEFT:

André Minhorst Verlag

Aufgabenverwaltung vollständig im Griff

In dieser Ausgabe zeigen wir Dir nicht nur, wie Du Deine Projekte und Aufgaben auf einfachste Weise mit einer kostenlosen Microsoft-Lösung verwalten kannst. Wir gehen noch einen Schritt weiter und liefern Dir gleich noch das Know-how, mit dem Du die mit dieser Lösung verwalteten Aufgaben per VBA anlegen, bearbeiten, auslesen und löschen kannst. Das ist für VBA allein etwas viel, daher holen wir uns prominente Unterstützung – nämlich von Microsofts Automatisierungtool Power Automate.

Wenn Du ein einfaches Tool zur Verwaltung von Aufgaben suchst, das sowohl auf Deinem Windows-Rechner als App zur Verfügung steht als auch für mobile Endgeräte wie Smartphones, ist vielleicht **Microsoft To Do** das Richtige für Dich. Damit kannst Du einfache Aufgabenlisten verwalten, die Du noch in Gruppen und Listen hierarchisch unterteilen kannst. Der Clou: Die Listen kannst Du auch mit anderen Menschen teilen, was sowohl für die gemeinsame Einkaufsliste mit der Familie als auch für eine Aufgabenliste für Dich und Dein Entwicklerteam interessant ist. Mehr dazu erfährst Du im Artikel **Aufgaben mit Microsoft To Do verwalten** ab Seite 4.

Außerdem streuen wir mal wieder einen Grundlagenartikel zum Thema VBA ein. Unter **VBA Basics: Mit Arrays programmieren** erfährst Du ab Seite 10 alles, was Du über die Programmierung mit Arrays unter VBA und ähnlichen Programmiersprachen wissen musst.

Wenn Du mit VBA programmierst, kommst Du aktuell um den VBA-Editor noch nicht herum. Als Start einer kleinen Artikelreihe zu diesem Thema zeigen wir einmal, wie Du das Menüsystem im VBA-Editor optimal nutzen und für Deine Zwecke einstellen kannst. Mehr dazu unter dem Titel **Menüs im VBA-Editor anpassen** ab Seite 13.

Solltest Du gelegentlich mal eigene Klassen programmieren, kennst Du dieses Problem: Zum Definieren von Eigenschaften mit Setter und Getter ist viel manuelle Tipparbeit nötig. Wie Du das viel einfacher gestalten kannst, zeigen wir Dir ab Seite 23 im Artikel **VBA-Editor: Klasseneigenschaften per Mausklick**. Damit brauchst Du nur noch die Variable selbst zu deklarieren und mit einem Aufruf unserer Hilfsfunktion wird der Rest automatisch erledigt.

Wenn Du viel mit Word arbeitest, möchtest Du Deinen Dokumenten vielleicht die eine oder andere Automatisierung hinzufügen. In unserem Artikel **Word: Dokument mit Ribbon und VBA-Funktionen** (ab Seite 29) erfährst Du, wie das mit einem in das aktuelle Dokument integrierten Ribbon und den passenden VBA-Prozeduren gelingt.

Kommen wir schließlich zum Schwerpunkt der Ausgabe: Das bereits erwähnte Microsoft To Do können wir nämlich auch per VBA steuern. Dazu brauchen wir allerdings ein wenig Unterstützung, die wir uns bei dem Automatisierungsdienst Power Automate von Microsoft holen. Das ist zwar nicht kostenlos, aber liefert spannende Features. Damit können wir zum Beispiel per VBA Aktionen anstoßen, mit denen wir Aufgaben in Microsoft To Do anlegen, bearbeiten, auslesen und löschen können. In der heutigen Zeit, wo immer mehr Informationen in Online-Diensten verwaltet werden, ist es ein unfassbarer Vorteil, wenn wir von unseren gewohnten VBA-Anwendungen auf diese Daten zugreifen können! Mehr dazu in den Artikeln **To Do-Aufgabe mit Power Automate und VBA anlegen** (ab Seite 33) und **To Do mit VBA und Power Automate steuern** (ab Seite 44).

Nun viel Spaß beim Lesen!

Dein André Minhorst

Aufgaben mit Microsoft To Do verwalten

Auf der Suche nach einer einfachen Verwaltung für Aufgaben, sowohl privat als auch geschäftlich, bin ich wieder einmal über Microsoft To Do gestolpert. Was ich suchte, war eine App, in der ich Aufgaben einfach in Projekte strukturieren konnte und die mir die Möglichkeit gibt, diese mit einem Erledigungsdatum zu versehen. Außerdem wollte ich eine Übersicht über die heute zu erledigenden Aufgaben haben. Schließlich gibt es noch zwei weitere Anforderungen: Erstens sollte die App nicht nur auf dem Windows Desktop nutzbar sein, sondern auch von mobilen Geräten aus. Zweitens habe ich mir gewünscht, dass ich die Listen auch per VBA aus Excel-Tabellen oder auch einer Datenbank heraus befüllen kann. In einer Artikelreihe schauen wir uns an, wie all das funktioniert. In diesem Artikel betrachten wir erst einmal die Möglichkeiten von Microsoft To Do in der Windows App.

Microsoft To Do ist vor einigen Jahren entstanden, als Microsoft die beliebte App Wunderlist aufgekauft hat. Microsoft To Do steht dieser in nichts nach und bietet die folgenden Funktionen, die wir uns in den folgenden Abschnitten ansehen:

- Anlegen von Gruppen
- Anlegen von Listen, die den Gruppen untergeordnet werden
- Anlegen von Aufgaben, die jeweils einer Liste untergeordnet werden
- Festlegen von Schritten für jede Aufgabe

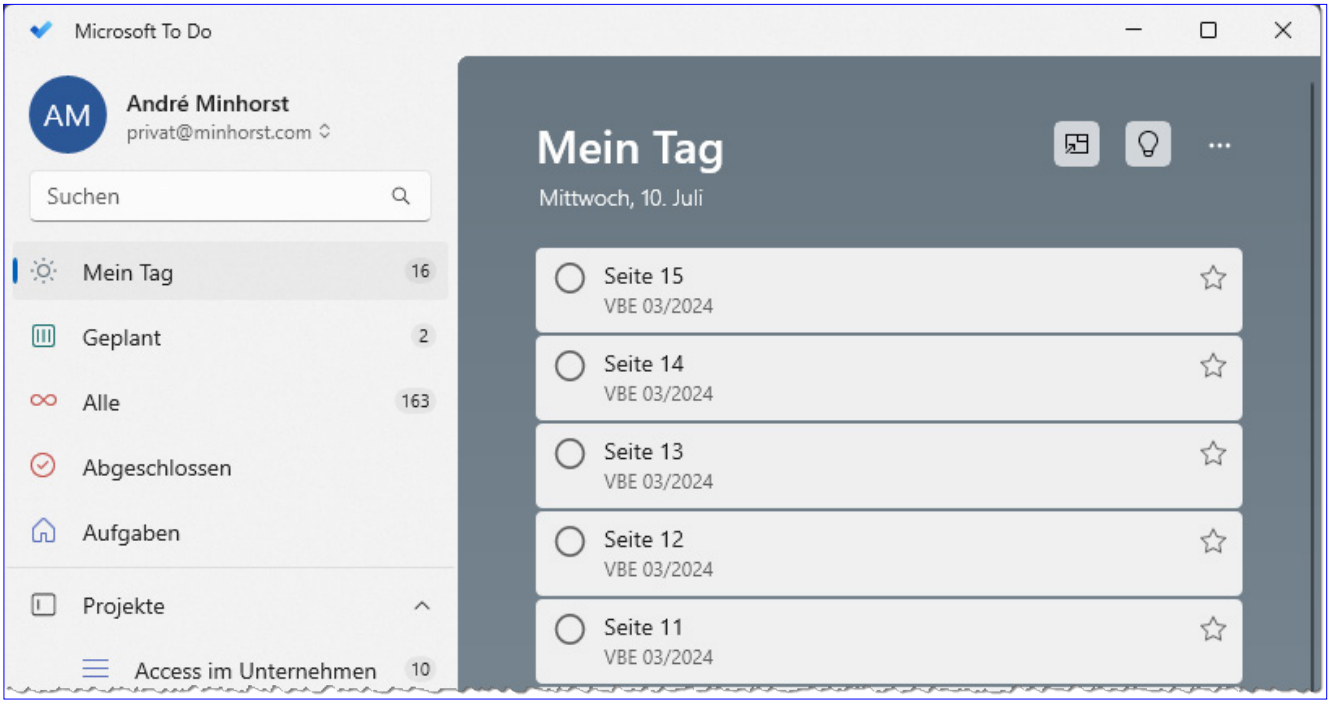


Bild 1: Der Bereich **Mein Tag** in Microsoft To Do

Kernstück für meine geplante Nutzung ist die Seite Mein Tag (siehe Bild 1). Hier erscheinen sowohl die Aufgaben, deren Fälligkeitsdatum den aktuellen Tag hat als auch die Aufgaben, die ich direkt hier hinzugefügt habe (ohne Bezug zu einer anderen Liste) oder die ich aus einer anderen Liste hier hineingezogen habe, weil ich diese heute erledigen möchte.

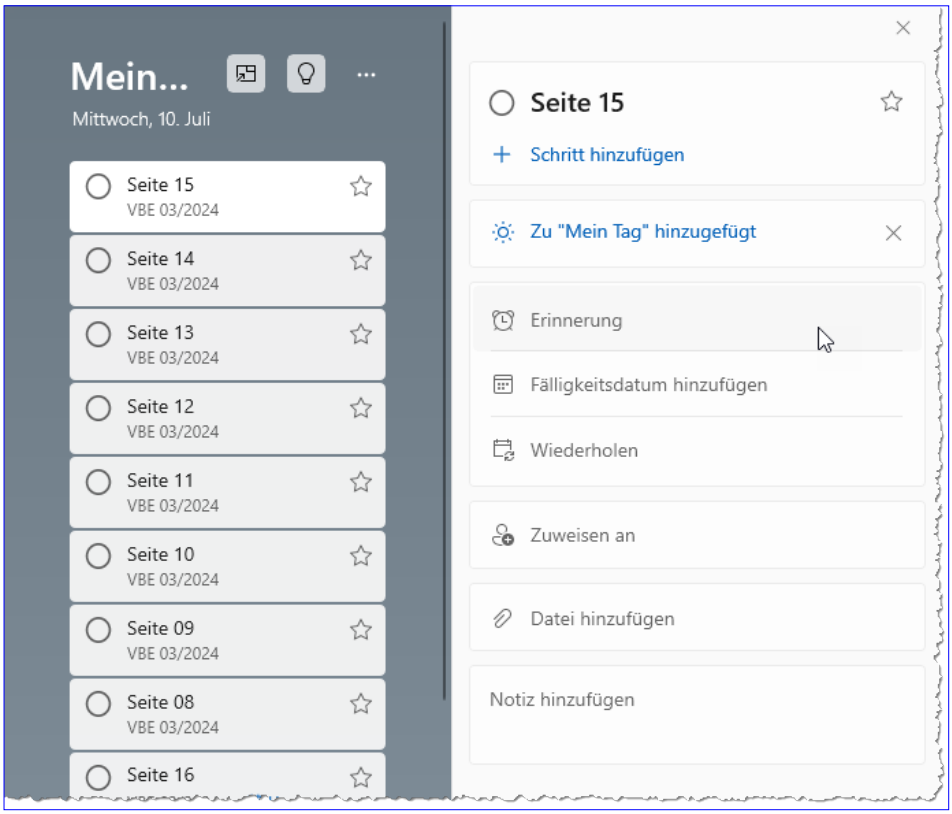


Bild 2: Funktionen für Aufgaben

Aufgaben bearbeiten

Für Aufgaben finden wir nach dem Anklicken die folgenden Optionen (siehe Bild 2):

- Hinzufügen von Schritten
- Festlegen von Erinnerungen
- Festlegen von Fälligkeitsdaten
- Anlegen von Wiederholungen für eine Aufgabe
- Zuweisen einer Aufgabe an Personen
- Hinzufügen von Dateien zu einer Aufgabe
- Eintragen von Notizen zu einer Aufgabe

Weitere Funktionen finden wir, wenn wir mit der rechten Maustaste auf eine Aufgabe klicken (siehe Bild 3).

- Hinzufügen zur Liste **Mein Tag** oder Entfernen aus dieser Liste, wenn die Aufgabe schon dort enthalten ist

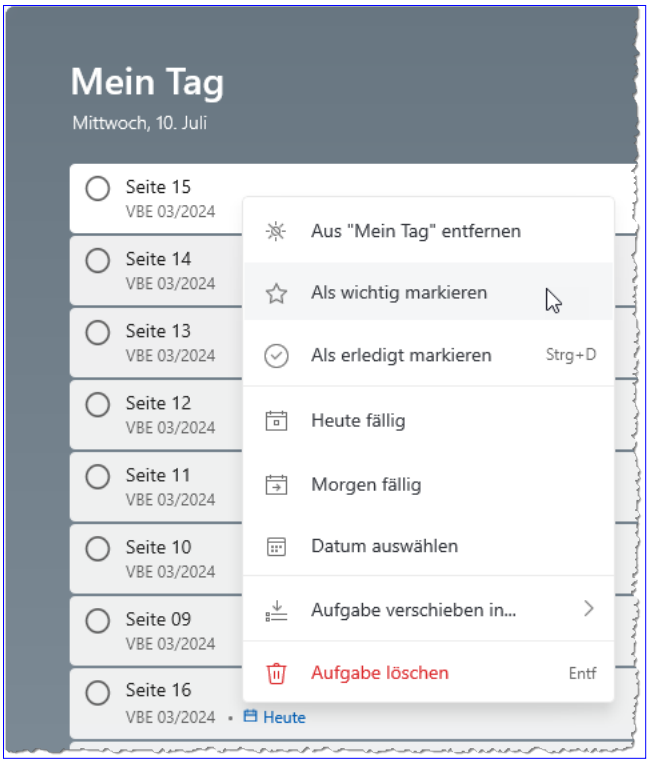


Bild 3: Weitere per Kontextmenü verfügbare Funktionen

- Aufgabe als wichtig markieren (gelingt auch durch Anklicken des Sterns rechts in der Aufgabe)
- Aufgabe als erledigt markieren (gelingt auch durch Setzen eines Hakens an die Aufgabe)
- Fälligkeitsdatum auf heute oder morgen einstellen oder ein anderes Datum auswählen
- Aufgabe in eine andere Liste verschieben
- Aufgabe löschen

Strukturieren von Aufgaben

Schauen wir in den Navigationsbereich von Microsoft To Do, finden wir verschiedene Ebenen für unsere Aufgaben (siehe Bild 4).

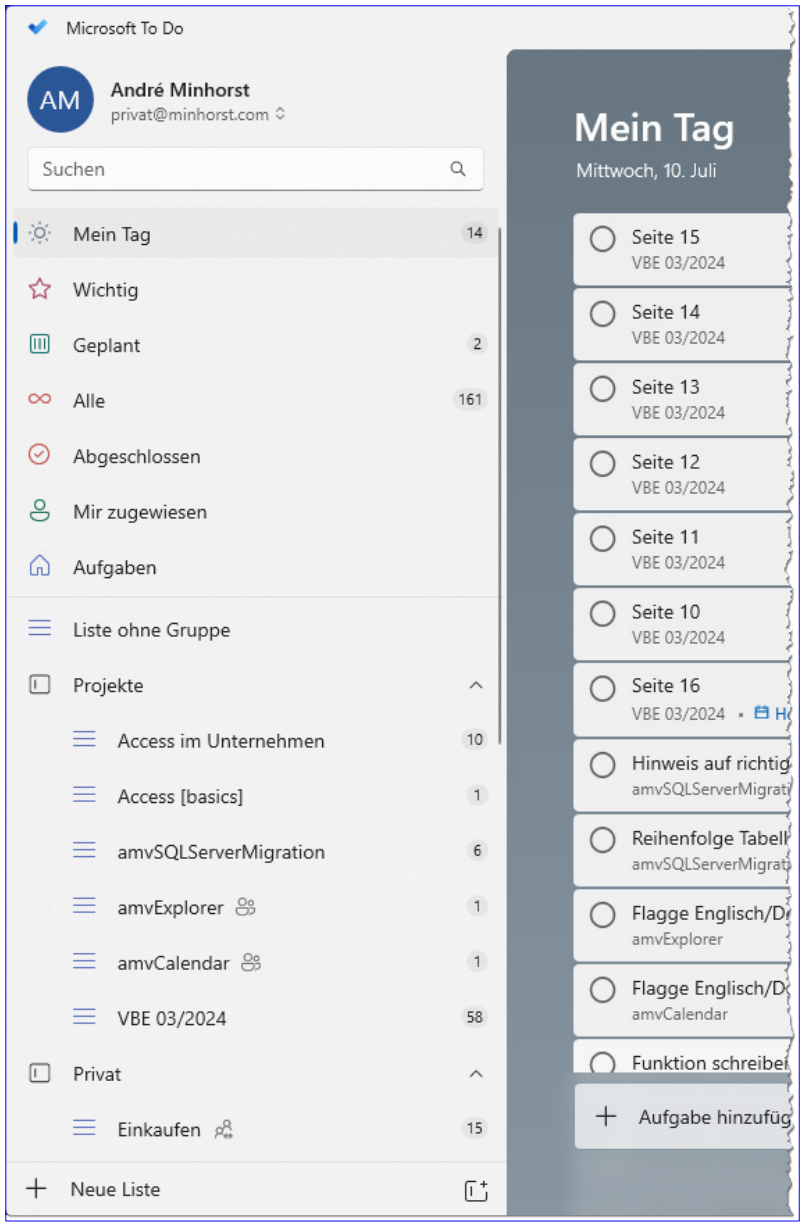


Bild 4: Verschiedene Hierarchie-Ebenen

Ganz oben sehen wir einige sogenannte intelligente Listen:

- **Mein Tag**: Aufgaben, die am aktuellen Datum fällig sind oder dieser Liste manuell hinzugefügt wurden
- **Wichtig**: Zeigt alle als wichtig markierten Einträge an.
- **Geplant**: Aufgaben, die ein Fälligkeitsdatum haben
- **Alle**: Alle Aufgaben, gruppiert nach Liste
- **Abgeschlossen**: Alle abgeschlossenen Aufgaben, ebenfalls gruppiert nach Liste
- **Mir zugewiesen**: Alle mir zugewiesenen Aufgaben

Darunter finden wir die benutzerdefinierten Listen. Der erste Eintrag **Liste ohne Gruppe** ist eine Liste, die keiner Gruppe zugeordnet wurde. Eine solche Liste fügen wir hinzu, indem wir unten auf den Befehl **Neue Liste** klicken.

Diese Listen können wir in Gruppen zusammenfassen. Eine solche Gruppe fügen wir mit einem Klick auf das Icon unten rechts hinzu.

VBA Basics: Mit Arrays programmieren

Arrays sind eine einfache Möglichkeit, mit VBA-Bordmitteln mehrere Werte unter einem einzigen Namen zu speichern und effizient auf diese Werte zuzugreifen. In diesem Artikel geben wir eine umfassende Einführung in die Verwendung von Arrays in VBA einschließlich der Definition, Deklaration, Manipulation und der Anwendung in verschiedenen Szenarien.

Arrays sind Datenstrukturen, die Elemente desselben Datentyps speichern. Dabei werden die Elemente in einer oder mehreren Dimensionen abgespeichert und können über den Index der verschiedenen Dimensionen abgefragt werden.

Normalerweise gibt man den Datentyp eines Arrays explizit an (zum Beispiel **Integer** oder **String**), wir können aber auch den Datentyp **Variant** zum Speichern beliebiger Informationen verwenden.

Bevor man Daten in ein Array schreiben kann, muss man diese deklarieren und dabei die Menge der aufzunehmenden Elemente definieren. Diese Menge kann man auch im Nachhinein anpassen. Die Deklaration von Arrays unterscheidet sich durch das Hinzufügen eines Klammernpaares zum Namen der Variablen von der Deklaration einfacher Variablen:

```
Dim strBeispiel(2) As String
```

Index 0- oder 1-basiert

Der Wert **2** in Klammern gibt den Wert des Indexes des letzten Elements an. Damit ist nicht eindeutig angegeben, wie viele Elemente das Array aufnehmen kann. Dies hängt davon ab, ob der Index 0- oder 1-basiert ist. Standardmäßig ist der Index 0-basiert. Wir können dies explizit festlegen, indem wir in dem jeweiligen Modul ganz oben eine entsprechende Angabe machen. Um sicherzustellen, dass wir tatsächlich einen 0-basierten Index verwenden, geben wir dort folgende Zeile an:

```
Option Base 0
```

Wir können auch einen 1-basierten Index verwenden:

```
Option Base 1
```

In diesem Fall kann das Array mit der Angabe **2** in Klammern maximal zwei Elemente aufnehmen, sonst drei.

Untersten und obersten Index-Wert herausfinden

Meist für das Durchlaufen von Array in einer Schleife wollen wir den Wert für den untersten und den obersten Index ermitteln. Dazu dienen die folgenden beiden Funktionen:

- **LBound**: Ermittelt den Wert des untersten Indexes.
- **UBound**: Ermittelt den Wert des obersten Indexes.

Wir testen diese Funktionen für ein Array mit 0-basiertem Index und dem größte Indexwert von 2:

```
Dim strBeispiel(2) As String
Debug.Print "Unterer Indexwert: " & LBound(strBeispiel)
Debug.Print "Oberer Indexwert: " & UBound(strBeispiel)
```

Das Ergebnis lautet:

```
Unterer Indexwert: 0
Oberer Indexwert: 2
```

Stellen wir **Option Base 1** ein, erhalten wir:

```
Unterer Indexwert: 1
Oberer Indexwert: 2
```

Individuelle Indexwerte

Wir können die Indexwerte auch komplett selbst festlegen. Das folgende Array enthält die Indexwerte **2** und **3**:

```
Dim strEigene(2 To 3)
```

Array füllen

Um das Array zu füllen, können wir direkt über den Indexwert auf das jeweilige Element zugreifen:

```
strBeispiel(0) = "Text 0"
strBeispiel(1) = "Text 1"
strBeispiel(2) = "Text 2"
```

Die Inhalte fragen wir wie folgt ab:

```
Debug.Print strBeispiel(0)
Debug.Print strBeispiel(1)
Debug.Print strBeispiel(2)
```

Array mit Werten initialisieren

Diese Werte können wir auch direkt in einer Zeile zuweisen. Dazu nutzen wir die **Array**-Funktion.

Da sie das Ergebnis als **Variant**-Array liefert, müssen wir das Array entsprechend deklarieren. Außerdem dürfen wir die Anzahl der Elemente hier nicht vordefinieren:

```
Dim varBeispiel() As Variant
varBeispiel = Array("Text 0", "Text 1", "Text 2")
```

Array per Schleife auslesen

Mithilfe der oben vorgestellten Funktionen **LBound** und **UBound** können wir die Elemente auch in einer Schleife durchlaufen und ausgeben:

```
Dim i As Integer
For i = LBound(strBeispiel) To UBound(strBeispiel)
    Debug.Print strBeispiel(i)
Next i
```

Array-Größe zur Laufzeit anpassen

Nicht immer wissen wir bereits vorher, wie viele Elemente das Array aufnehmen soll. Deshalb können wir es zunächst einmal ohne Angabe der Menge deklarieren und dann zur Laufzeit die Anzahl der Elemente festlegen. Dabei können wir entscheiden, ob bei dieser Änderung die vorhandenen Inhalte beibehalten oder verworfen werden sollen.

Wir deklarieren das Array zuerst ohne Elemente:

```
Dim strBeispiel() As String
```

Dann stellen wir die Größe auf den maximalen Index von **0** ein, also für ein Element, und weisen diesem einen Wert hinzu:

```
ReDim strBeispiel(0)
strBeispiel(0) = "Text 0"
```

Nun wollen wir zwei weitere Werte hinzufügen, aber den Wert im ersten Element behalten. Dazu stellen wir wieder mit **ReDim** den höchsten Index auf **2** ein und verwenden zusätzlich das Schlüsselwort **Preserve**, um die Inhalte

```
ReDim Preserve strBeispiel(2)
strBeispiel(1) = "Text 1"
strBeispiel(2) = "Text 2"
```

Wenn wir **Preserve** nicht setzen würden, wäre **strBeispiel(0)** anschließend leer beziehungsweise würde eine leere Zeichenkette enthalten.

Hier wird dann der Standardwert für den jeweiligen Datentyp gesetzt, für **String** also eine leere Zeichenkette oder für Zahlendatentypen der Wert **0**.

Mehrdimensionale Arrays

Wir können Arrays auch mehrdimensional gestalten. Hier deklarieren wir ein Array, das in der ersten Di-

Menüs im VBA-Editor anpassen

Viele Themen in diesem Magazin drehen sich um die Programmierung des VBA-Editors. Damit erweitern wir das wichtigste Werkzeug für Programmierer, die sich um die Automation von Anwendungen wie Access, Excel, Outlook oder Word beschäftigen. Ein wichtiger Teil des VBA-Editors sind die Menüleisten, Symbolleisten und Kontextmenüs. Was sind diese drei Elemente überhaupt und wie können wir diese anpassen – sowohl über die Benutzeroberfläche als auch per VBA? Dieser Artikel beleuchtet die wichtigsten Möglichkeiten und zeigt, wie Du das Menüsystem nutzen kannst, um einen optimalen Workflow zu gewährleisten und auch Deine eigenen Erweiterungen, beispielsweise in Form von COM-Add-Ins, an der richtigen Stelle einzubauen.

Grundlagen zu Menüsystem des VBA-Editors

Im Menüsystem des VBA-Editors gibt es drei verschiedene Arten von Menüs.

Wer schon vor 2007 Office-Anwendungen wie Access, Excel oder Word programmiert hat, kennt das hier verwendete Menüsystem auch noch von diesen Anwendungen. Office 2003 war die letzte Office-Version, die mit dem alten Menüsystem ausgestattet war. Dann hat Microsoft mit Office 2007 das Ribbon als neues Menüsystem eingeführt.

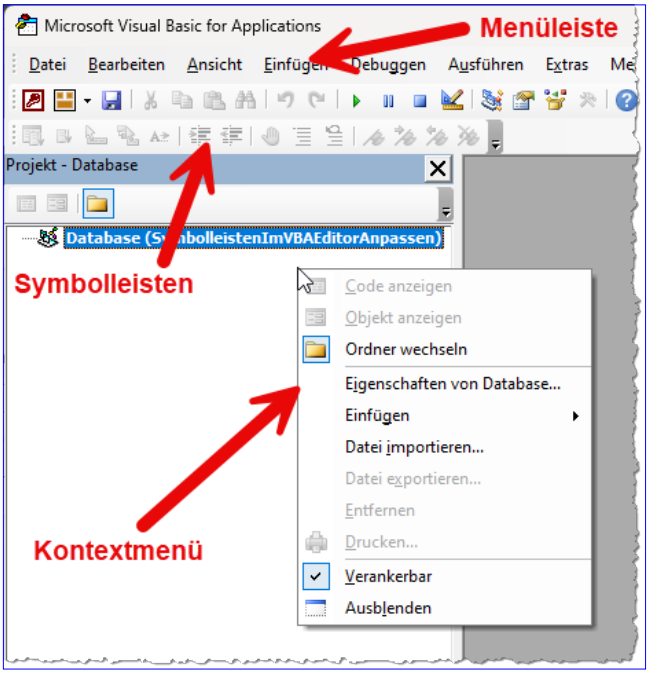


Bild 1: Alle drei Arten von Menüs

Das alte Menüsystem war, was die Anpassbarkeit und die Programmierbarkeit anging, durchaus einfacher handzuhaben. Wer erst später in die Office-Programmierung eingestiegen ist und sich nun mit der Anpassung und Programmierung des Menüsystems des VBA-Editors beschäftigt, erhält also nun einen Einblick und kann entscheiden, ob »früher alles besser war«. Tatsächlich spielt das keine Rolle, denn wer einigermaßen auf dem Stand der Technik bleiben will, arbeitet nicht mehr mit Microsoft Office 2003 und älter (auch wenn es in der Praxis immer wieder Kunden gibt, die das dennoch tun).

Bild 1 zeigt alle drei Arten von Menüs:

- Jede Anwendung enthält eine Menüleiste. Das ist das Hauptmenü, über das man in der Regel thematisch sortierte Untermenüpunkte aufklappen und aufrufen kann.

- Außerdem kann eine Anwendung beliebig viele Symbolleisten enthalten. Diese werden normalerweise unterhalb der Menüleiste angeordnet, können aber auch freischwebend platziert werden.

- Schließlich finden wir verschiedene Kontextmenüs vor. Diese können wir durch einen rechten Mausklick auf verschiedene Elemente öffnen. Es wird jeweils das Kontextmenü mit den für das geöffnete Element passenden Befehlen geöffnet.

Menüsystem im VBA-Editor über die Benutzeroberfläche anpassen

Wir können verschiedene Aspekte des Menüsystems des VBA-Editors über die Benutzeroberfläche anpassen.

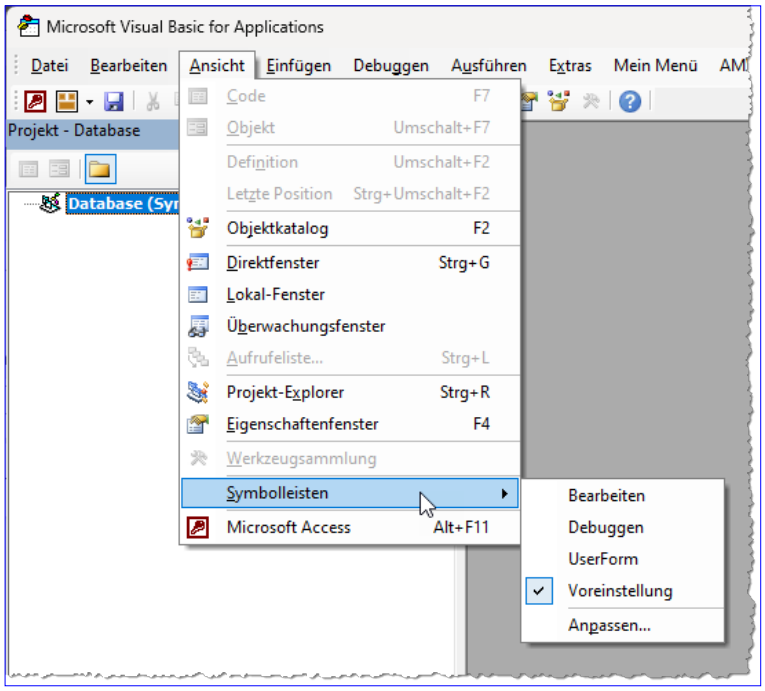


Bild 2: Ein- und Ausblenden von Menüleisten

Der erste Weg und einfachste Weg beschränkt sich auf die Elemente, die wir normalerweise gar nicht sehen, sondern die erst per Rechtsklick auf ein Element auftauchen – die Kontextmenüs. Klicken wir auf dem Menüpunkt **Ansicht|Symbolleisten**, finden wir eine Liste aller verfügbaren Symbolleisten und können hier einzelne Einträge ein- und ausblenden. Außerdem finden wir hier noch einen Eintrag namens **Anpassen...** – diesen schauen wir uns später an (siehe Bild 2).

Klicken wir die noch nicht aktivierten Einträge **Bearbeiten**, **Debuggen** und **UserForm** an, sehen wir, dass diese entweder direkt fest verankert oder freischwebend. eingeblendet werden (siehe Bild 3). Diese können wir nun frei bewegen und an der Stelle platzieren, wo wir diese gerade benötigen, oder wir bewegen diese in Richtung Menüleiste, wo diese dann »eingeklinkt« werden. Wenn wir freischwebende Symbolleisten bewegen wollen, können wir diese mit der Maus an der Titelleiste greifen und verschieben.

Wenn wir eine einmal eingeklinkte Symbolleiste an eine andere Stelle verschieben wollen, können wir diese am linken Rand an den drei vertikal angeordneten Punkten greifen und entweder wieder freischwebend platzieren oder einfach ihre Position im oberen Bereich ändern.

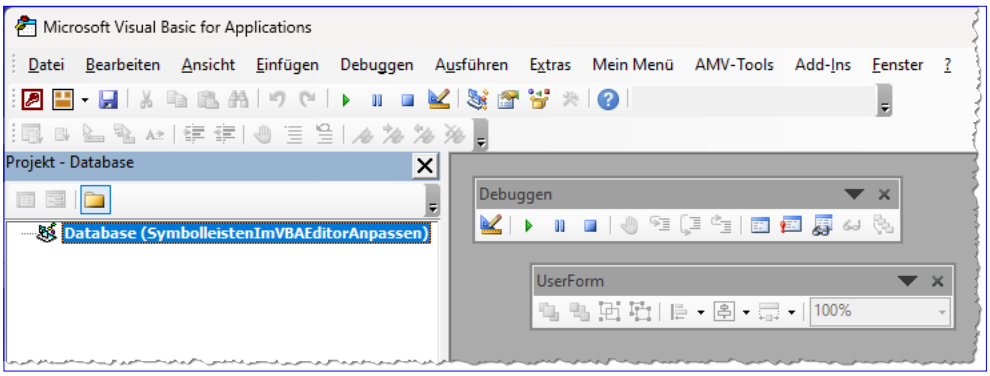


Bild 3: Alle Symbolleisten sind nun eingeblendet.

Wir können die Symbolleisten alle untereinander platzieren (siehe Bild 4) oder diese in mehreren Zeilen nebeneinander anordnen. Wie man das macht, hängt unter anderem vom verfügbaren Platz auf dem Bildschirm ab.

Eingebaute Symbolleisten

Die eingebauten Symbolleisten liefern uns die folgenden Funktionen:

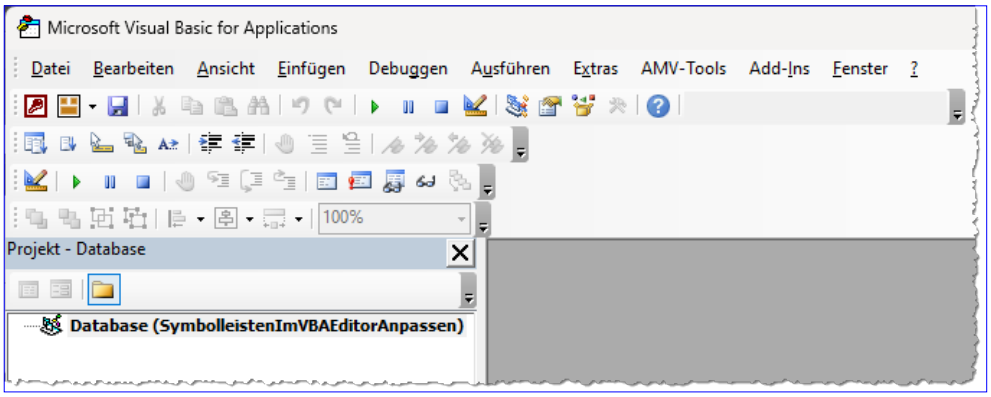


Bild 4: Alle Symbolleisten sind nun verankert.

- **Bearbeiten**: Diese Symbolleiste sollte aus unserer Sicht standardmäßig eingeblendet sein, denn sie enthält zwei sehr wichtige Elemente – nämlich die zum Auskommentieren und Einkommentieren aller aktuell markierten Codezeilen. Außerdem finden wir hier zum Beispiel Befehle zum Setzen von Haltepunkten oder zum Aktivieren von IntelliSense. Diese Befehle würde man jedoch aus Effizienzgründen eher per Tastenkombination ausführen.

- **Debuggen**: In dieser Symbolleiste finden wir alle Elemente, die beim Debuggen von Code wichtig sind. Hier können wir den Code wiederum starten, pausieren oder stoppen und wir können diesen auf bestimmte Arten durchlaufen. Außerdem lassen sich hier weitere Elemente des VBA-Editors wie **Lokal-Fenster**, **Direktfenster** oder **Überwachungs-Fenster** einblenden.

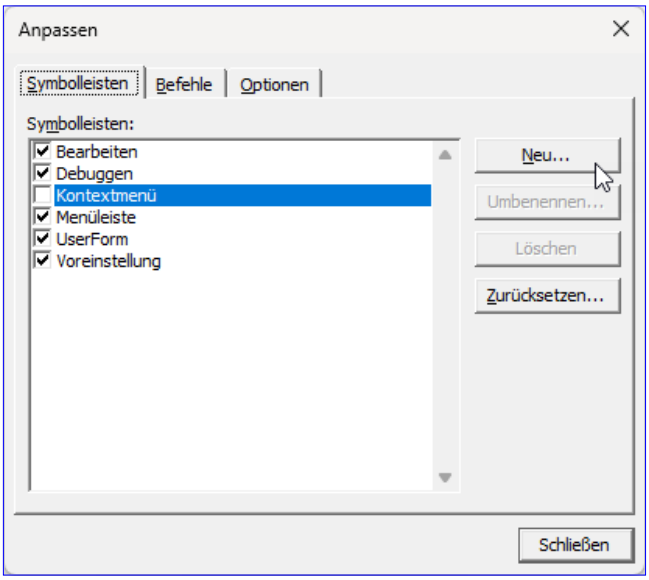


Bild 5: Der Dialog zum Anpassen des Menüsytems

- **UserForm**: Diese Symbolleiste benötigen wir, wenn wir UserForms anlegen und bearbeiten wollen.

- **Voreinstellung**: Dies ist die Symbolleiste, die standardmäßig eingeblendet ist. Sie enthält Befehle wie zum Speichern, zum Bedienen der Zwischenablage, zum Rückgängig machen von Aktionen, zum Starten und Stoppen von Prozeduren und zum Einblenden wichtiger Elemente wie Projekt-Explorer, Eigenschaftenfenster und Objektkatalog.

Individuelle Anpassungen

Wenn uns das Ein- und Ausblenden und das Anordnen von Symbolleisten nicht ausreicht, können wir noch einen Schritt weitergehen. Unter **Ansicht|Symbolleisten** gibt es den bereits weiter oben erwähnten Befehl mit der Beschriftung **Anpassen...**, den wir nun betätigen.

Damit blenden wir den Dialog aus Bild 5 ein. Dieser zeigt erst einmal die Symbolleisten an, die wir bereits kennengelernt haben. Außerdem finden wir hier aber auch noch einen Eintrag namens **Kontextmenü** und einen namens **Menüleiste**. Diese Darstellung liefert

uns einen Hinweis, wie das Menüsystem intern aufgebaut ist: Wir arbeiten nämlich eigentlich nur mit Symbolleisten. Diese kommen allerdings in verschiedenen Ausprägungen, nämlich neben den eigentlich Symbolleisten noch als Menüleiste (eben jene Leiste, die wir auch Hauptmenü nennen könnten) und als Kontextmenüleiste. Wie wir später bei der Programmierung per VBA sehen werden, sind es lediglich die Werte von Eigenschaften, die den Unterschied ausmachen.

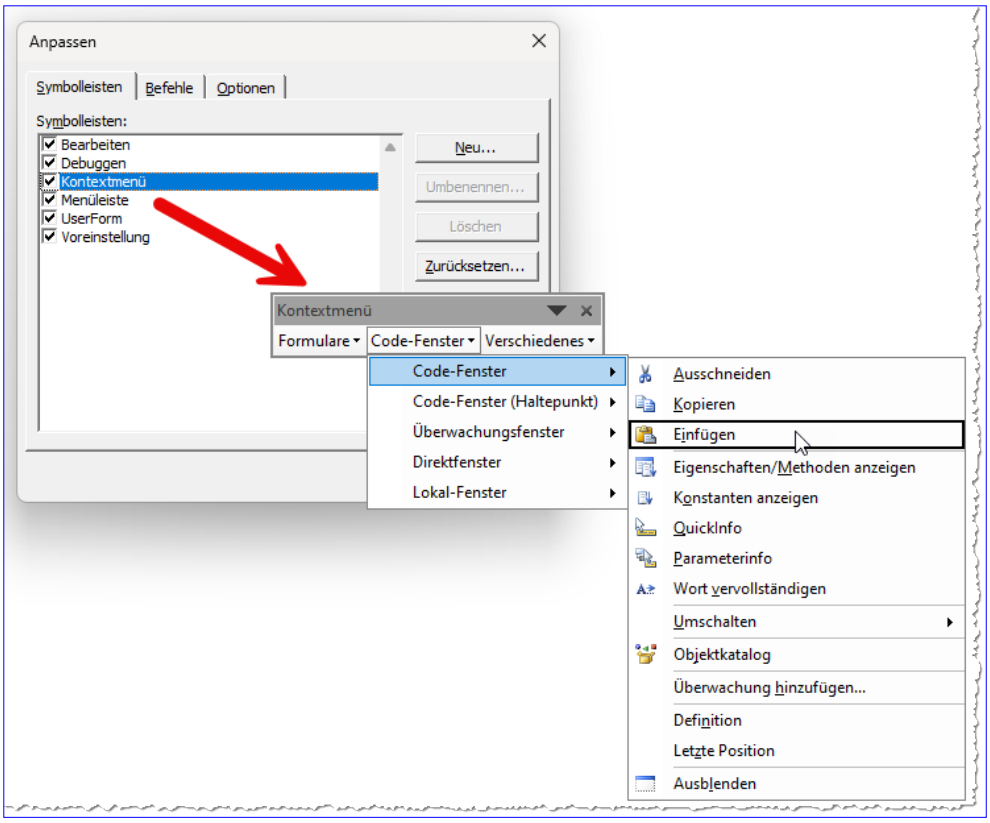


Bild 6: Bearbeiten von Kontextmenüs über die Benutzeroberfläche

Auch im Dialog **Anpassen** können wir die »normalen« Symbolleisten durch Setzen eines Hakens in den Kontrollkästchen ein- und ausblenden.

Wenn wir das beim Eintrag **Menüleiste** probieren, gelingt das jedoch nicht. Das zeigt: Die Anwendung muss eine Menüleiste haben, die auf diesem Wege nicht ausgeblendet werden kann.

Was aber geschieht, wenn wir den Eintrag **Kontextmenü** anklicken? Blenden wir damit die Kontextmenüs dauerhaft ein? Nein: Insgesamt dient das Öffnen des **Anpassen**-Dialogs eher dazu, das gesamte Menüsystem in einen Bearbeitungszustand zu versetzen. Und während wir die Menü- und Symbolleisten ja ohnehin schon sehen können, dient der Eintrag **Kontextmenü** dazu, die sonst nur per Rechtsklick sichtbaren Kontextmenüs zu Bearbeitungszwecken sichtbar zu machen.

Aktivieren wir diesen Eintrag, erscheint eine weitere Symbolleiste mit dem Titel **Kontextmenü**. Diese enthält drei Kategorien für verschiedene Kontextmenüs. In der zweiten namens **Code-Fenster** finden wir einen weiteren gleichnamigen Eintrag, der beim Anklicken alle Elemente des Kontextmenüs anzeigt, die beim Rechtsklick im Code-Fenster erscheinen (siehe Bild 6).

Einstellungen eines Menüeintrags anpassen

Hier können wir nun ebenfalls mit der rechten Maustaste ein Kontextmenü öffnen, dass uns alle Möglichkeiten zum Anpassen eines Menüeintrags anzeigt (siehe Bild 7).

Dieses Kontextmenü können wir nicht nur für die Menüeintrage der Kontextmenüs anzeigen. Auch ein Rechtsklick auf die Einträge in der Menüleiste oder in

eine der Symbolleisten zeigt das Kontextmenü zum Bearbeiten des jeweiligen Eintrags an (siehe Bild 8).

Wichtig ist nur, dass der Dialog Anpassen geöffnet und die Symbolleisten damit in den Bearbeitungszustand versetzt wurden.

In diesem Zustand können wir die eigentlichen Funktionen der Menüeinträge auch nicht aufrufen, sondern nur die Funktionen zum Anpassen des Menüsystems.

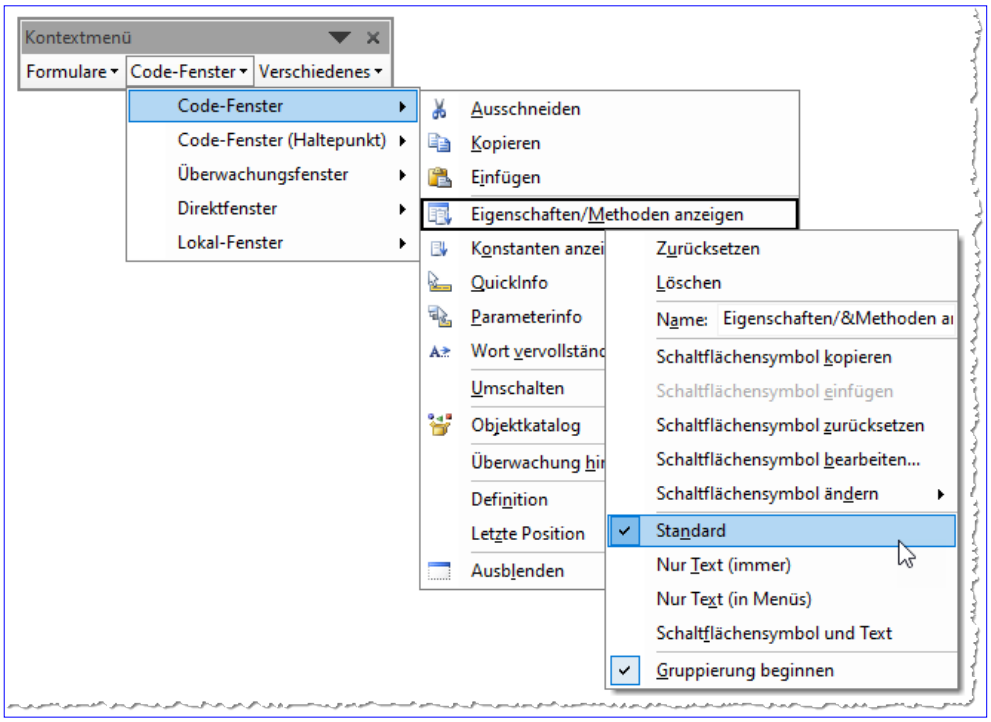


Bild 7: Einstellungen eines Menüeintrags anpassen

Welche Möglichkeiten bietet uns dieses Kontextmenü nun? Wichtig: Wir können diese Eigenschaften für die eingebauten Elemente nach Lust und Laune ausprobieren.

Mit einem Klick auf den Befehl **Zurücksetzen** können wir die Eigenschaften von Steuerelementen und Menüs wieder in den Anfangszustand zurückversetzen:

- **Zurücksetzen**: Setzt die Eigenschaften des aktuellen Steuerelements auf den Standardzustand zurück. Das bezieht sich vor allem auf Eigenschaften wie **Name** und **Schaltflächensymbol**.

- **Löschen**: Löscht das Steuerelement aus der Symbolleiste.

- **Name**: Stellt den Namen des Steuerelements ein.

- **Schaltflächensymbol kopieren**: Kopiert das Symbol des aktuellen Steuerelements.

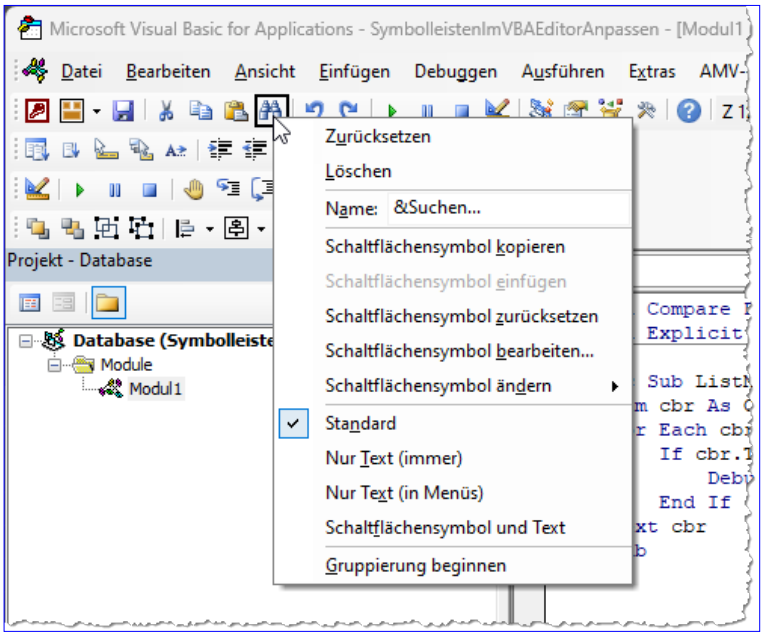


Bild 8: Auch die übrigen Menüpunkte können wir per Kontextmenü anpassen.

VBA-Editor: Klasseneigenschaften per Mausklick

Wenn wir im VBA-Editor benutzerdefinierte Klassen programmieren wollen, verwenden wir für Eigenschaften üblicherweise eine private Variable, die wir über eine öffentliche Property Set/Let-Prozedur mit einem Wert füllen und mit einer Property Get-Prozedur auslesen können. Das sind mindestens sieben Zeilen je Eigenschaft, was jede Menge Tipparbeit und Aufwand bedeutet und außerdem noch fehleranfällig ist. Selbst Copy und Paste macht diese Aufgabe nicht wesentlich angenehmer. Wohl dem, der weiß, wie er den VBA-Editor programmiert, sodass er solche Aufgaben mit wenigen Mausklicks automatisieren kann. Hier gibt es verschiedene Ansätze, die wir uns in diesem Artikel ansehen und auch umsetzen.

Die Programmierung von Office-Anwendungen mit VBA ist die eine Aufgabe. Diese wird üblicherweise durch die Anforderungen des Kunden oder des Benutzers gesteuert und einfach Schritt für Schritt abgearbeitet. Als Entwickler sollten wir jedoch immer darauf bedacht sein, Abkürzungen bei der Arbeit zu suchen, um schneller und zuverlässiger ans Ziel zu kommen.

Das gelingt unter Office anwendungsübergreifend im Bereich der VBA-Programmierung. Wir können uns auch die eine oder andere Prozedur bauen, um Aufgaben in Access, Excel, Outlook, PowerPoint oder Word zu automatisieren. Schließlich gibt es in einigen dieser Anwendungen dazu extra den Makro-Rekorder.

Unter VBA ist das jedoch noch ein wenig spannender, denn hier können wir für alle Office-Anwendungen gleichermaßen Vereinfachungen programmieren. Und so machen wir uns auch gleich ans Werk und schauen uns an, welche Möglichkeiten wir hier so haben.

Ziel: Private Member-Variable plus Get/Set/Let-Prozedur

Aber von welchen verschiedenen Ansätzen haben wir in der Einleitung gesprochen? Nun, es gibt verschiedene Ausgangssituationen. Wir schauen uns zuerst einmal an, was wir überhaupt haben wollen. Für eine vollwertige Eigenschaft, deren Wert gelesen und gesetzt werden kann, benötigen wir als Erstes eine private Member-Variable, die wir beispielsweise wie folgt deklarieren:

```
Private m_Firma As String
```

Damit wir von außen auf diese Variable zugreifen können, um ihren Wert einzustellen, nutzen wir eine öffentliche **Property Let**-Prozedur:

```
Public Property Get Firma() As String
    Firma = m_Firma
End Property
```

Schließlich wollen wir eine Eigenschaft auch von außen setzen können, also fügen wir noch eine entsprechende **Property Let**-Prozedur hinzu:

```
Public Property Let Firma(str As String)
    m_Firma = str
End Property
```

Und wenn unsere Eigenschaft keine skalare Variable aufnimmt (also zum Beispiel **Long** oder **String**), sondern einen Objektverweis, müssen wir die Schreibweise ein wenig anpassen. Die Variable deklarieren wir dann etwa so:

```
Private m_Recordset As DAO.Recordset
```

Auch in der **Get**-Prozedur benötigen wir eine Änderung, denn für die Zuweisung von Objektverweisen ist das Schlüsselwort **Set** nötig:

```
Public Property Get Recordset() As DAO.Recordset
    Set Recordset = m_Recordset
End Property
```

Dieses Schlüsselwort benötigen wir auch beim Setzen des Objektverweises. Außerdem verwenden wir hier keine **Property Let**-Prozedur, sondern eine **Property Set**-Prozedur:

```
Public Property Set Recordset(rst As DAO.Recordset)
    Set m_Recordset = rst
End Property
```

Die nächste Vorgabe ist, dass die **Property Let/Set/Get**-Prozeduren im Modul frühestens nach der letzten Deklarationszeile hinzugefügt werden darf. Nach der ersten Prozedur darf keine alleinstehende Deklarationszeile mehr folgen.

Prozedur zum Ersetzen von öffentlichen Eigenschaften durch Get/Let/Set-Property

Wir können mit den Methoden der Bibliothek **Microsoft Visual Basic for Applications 5.3 Object Library** auf die Elemente des VBA-Editors zugreifen und diese manipulieren. Dies nutzen wir in diesem Falle, um die gewünschten Änderungen in einem Klassenmodul vorzunehmen. Den benötigten Verweis auf diese Bibliothek setzen wir auf gewohnte Weise über den **Verweise**-Dialog.

Dann fügen wir die Prozedur **VariableToProperty** zu einem Standardmodul hinzu (siehe Listing 1).

Diese Prozedur deklariert zunächst einige Variablen und stellt dann die Variable **objCodePane** mit der Funktion **ActiveCodePane** auf das aktuelle **CodePane**-Objekt ein, was dem aktuellen Codefenster entspricht.

Wir gehen davon aus, dass der Benutzer eine oder mehrere Zeilen mit öffentlichen Variablen markiert hat und dass die Prozedur diese alle in private Variablen umwandeln soll, die über eine **Get Property** verfügbar gemacht und mit **Set/Let Property** gesetzt werden können sollen.

Davon ausgehend erfassen wir die aktuelle Markierung im Codefenster mit der **GetSelection**-Funktion des **CodePane**-Objekts. Diese liefert uns für die Variablen **lngStartline**, **lngStartcolumn**, **lngEndline** und **lngEndcolumn** die Koordinaten der aktuellen Markierung.

Damit ausgestattet durchlaufen wir die erste Zeile der Markierung (**lngStartline**) bis zur letzten Zeile (**lngEndline**) in einer **For...Next**-Schleife mit **lngLine** als Laufvariable.

Darin lesen wir die aktuelle Zeile in die Variable **strLine** ein und entfernen mit der **Trim**-Funktion direkt führende und folgende Leerzeichen.

Wenn **strLine** nun keine leere Zeichenkette ist, teilen wir die Zeile mit der **Split**-Funktion an den Leerzeichen auf und ermitteln das zweite und das vierte Element (mit den Indexwerten **1** und **3** wegen des **0**-basierten Indexes dieser Funktion). Wir gehen von einer Zeile wie der folgenden aus:

```
Public Field As DAO.Field
```

Dann landet der Variablenname **Field** in der Variablen **strVariable** und **DAO.Field** landet in **strDatatype**.

Bevor wir beginnen, den notwendigen Code zusammenzustellen, wollen wir noch den Datentyp der Va-

```
Public Sub VariableToProperty()
    Dim objCodePane As CodePane
    Dim objCodeModule As CodeModule
    Dim lngStartline As Long
    Dim lngStartcolumn As Long
    Dim lngEndline As Long
    Dim lngEndcolumn As Long
    Dim lngLine As Long
    Dim strLine As String
    Dim strVariable As String
    Dim strDatatype As String
    Dim strVariables As String
    Dim strProperties As String
    Dim strPrefix As String

    Set objCodePane = VBE.ActiveCodePane
    objCodePane.GetSelection lngStartline, lngStartcolumn, lngEndline, lngEndcolumn
    Set objCodeModule = objCodePane.CodeModule
    For lngLine = lngStartline To lngEndline
        strLine = Trim(objCodeModule.Lines(lngLine, 1))
        If Not Len(strLine) = 0 Then
            strVariable = Split(strLine, " ")(1)
            strDatatype = Split(strLine, " ")(3)
            strPrefix = GetPrefix(strDatatype)
            strVariables = strVariables & "Private m_" & strVariable & " As " & strDatatype & vbCrLf
            Select Case strPrefix
                Case "obj"
                    strProperties = strProperties & "Public Property Set " & strVariable & "(" & strPrefix & " As " _
                        & strDatatype & ")" & vbCrLf
                    strProperties = strProperties & "    Set m_" & strVariable & " = " & strPrefix & vbCrLf
                Case Else
                    strProperties = strProperties & "Public Property Let " & strVariable & "(" & strPrefix & " As " _
                        & strDatatype & ")" & vbCrLf
                    strProperties = strProperties & "    m_" & strVariable & " = " & strPrefix & vbCrLf
            End Select
            strProperties = strProperties & "End Property" & vbCrLf & vbCrLf
            strProperties = strProperties & "Public Property Get " & strVariable & " As " & strDatatype & vbCrLf
            strProperties = strProperties & "    " & strVariable & " = m_" & strVariable & vbCrLf
            strProperties = strProperties & "End Property " & vbCrLf & vbCrLf
        End If
    Next lngLine
    strVariables = strVariables & vbCrLf
    objCodeModule.DeleteLines lngStartline, lngEndline - lngStartline + 1
    lngStartline = objCodeModule.CountOfDeclarationLines + 1
    objCodeModule.InsertLines lngStartline, strVariables & strProperties
End Sub
```

Listing 1: Prozedur zum Erstellen von Property Get/Let/Set-Prozeduren für Variablen

Word: Dokument mit Ribbon und VBA-Funktionen

Nicht jeder, der seine Word-Dokumente mit ein paar zusätzlichen VBA-Funktionen ausstatten möchte, will direkt ein COM-Add-In dafür programmieren. Das ist auch nicht nötig, denn wir können solche Funktionen auch einfach zu einem Word-Dokument hinzufügen und die Funktionen in einem integrierten Ribbon verfügbar machen. Wie das gelingt, zeigen wir an einem einfachen Beispiel. Dabei wollen wir das aktuelle Dokument als PDF-Dokument in das gleiche Verzeichnis wie das Dokument exportieren. Einem zweiten Ribbonbefehl fügen wir noch einen Schritt hinzu, der das frisch erstellte PDF-Dokument direkt in die Zwischenablage kopiert, damit es beispielsweise gleich in eine E-Mail eingefügt werden kann.

Wenn Du eine VBA-Funktion programmiert hast, die Du dauerhaft in einem Word-Dokument verfügbar machen willst, gibt es verschiedene Möglichkeiten. Die VBA-Funktion zum Dokument hinzuzufügen, ist schnell erledigt: Du öffnest per **Alt + F11** den VBA-Editor, wechselst dort zum VBA-Projekt dieses Word-Dokuments und öffnest das VBA-Modul **ThisDocument**.

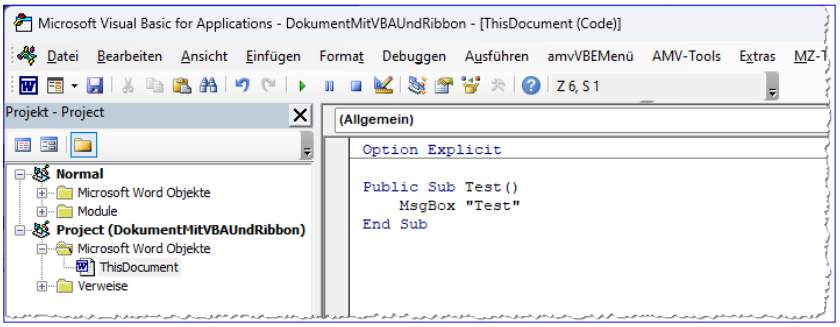


Bild 1: Anlegen einer VBA-Prozedur für das aktuelle Dokument

Hier können wir direkt eine Prozedur wie in Bild 1 hinzufügen und ausführen.

Nun möchtest Du nicht immer in den VBA-Editor wechseln, um die Funktion aufzurufen, sondern diese gegebenenfalls direkt vom Ribbon aus oder über ein Kontextmenü starten. Das Anlegen eines entsprechenden Ribbon-Eintrags ist nicht besonders aufwendig und dieser kann auch mit dem aktuellen Dokument gespeichert werden.

Ein Kontextmenü können wir nur mit VBA anlegen, aber auch dies schauen wir uns später an.

Zunächst wollen wir die Funktion per Ribbonbefehl verfügbar machen. Dazu verwenden wir ein separates Tool, mit dem wir das Dokument öffnen. Also müssen wir das Dokument zunächst schließen.

Beim Schließen erscheint jedoch der Hinweis, dass der enthaltene VBA-Code nur gespeichert werden kann, wenn wir das Dokument mit dem Typ **Word-Dokument mit Makros (.dotm)** speichern, was wir direkt erledigen.

Ribbon-Eintrag hinzufügen

Den Ribbon-Befehl wollen wir in einem eigenen Tab unterbringen, das die Beschriftung **Mein Dokument** erhalten soll. Darin fügen wir eine Gruppe ein, die schließlich eine Schaltfläche zum Aufrufen unseres noch zu erstellenden VBA-Befehls enthält.

Um dem Dokument die dafür notwendige Ribbon-Definition hinzuzufügen, nehmen wir ein Tool namens

Office RibbonX Editor zu Hilfe, das wir hier herunterladen können (sollte der Link nicht mehr verfügbar sein, hilft eine Suche nach dem Produktnamen):

```
https://github.com/fernandreu/office-ribbonx-editor/releases/tag/v1.9.0
```

Nach der Installation öffnen wir das Tool und öffnen das soeben angelegte Word-Dokument damit. Dieses erscheint nun in der linken Liste.

Mit einem Rechtsklick zeigen wir das Kontextmenü an, aus dem wir den Eintrag **Office 2010+ Custom UI-Abschnitt einfügen** auswählen (siehe Bild 2). Dies fügt einen neuen Untereintrag namens **customUI14.xml** zum Dokument hinzu.

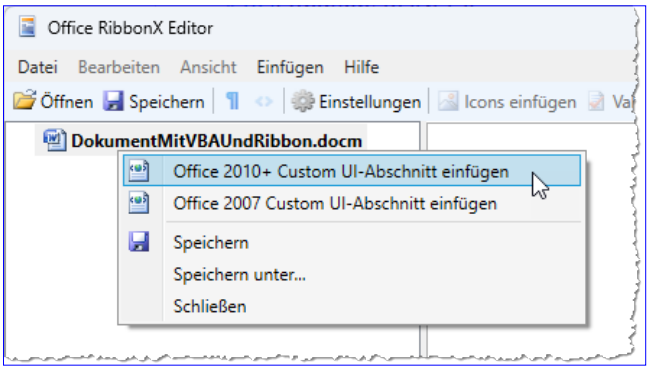


Bild 2: Einfügen eines Custom UI-Abschnitts

Ein Doppelklick zeigt im rechten Bereich den Inhalt dieses Abschnitts an, den wir nun mit dem gewünschten XML-Code füllen:

```
<customUI xmlns="http://schemas.microsoft.com/
    office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab" label="Mein Dokument">
        <group id="grp" label="PDF">
          <button id="btnExportPDF" label="PDF exportieren"
              size="large" image="PDF"
              onAction="ExportPDF"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Damit auch noch ein hübsches Icon angezeigt wird, fügen wir dieses über den Kontextmenübefehl wie in Bild 3 hinzu.

Den Namen, der nun unter dem Eintrag **customUI14.xml** für die Ribbondefinition angezeigt wird, fügen wir für das Attribut **image** des **button**-Elements ein.

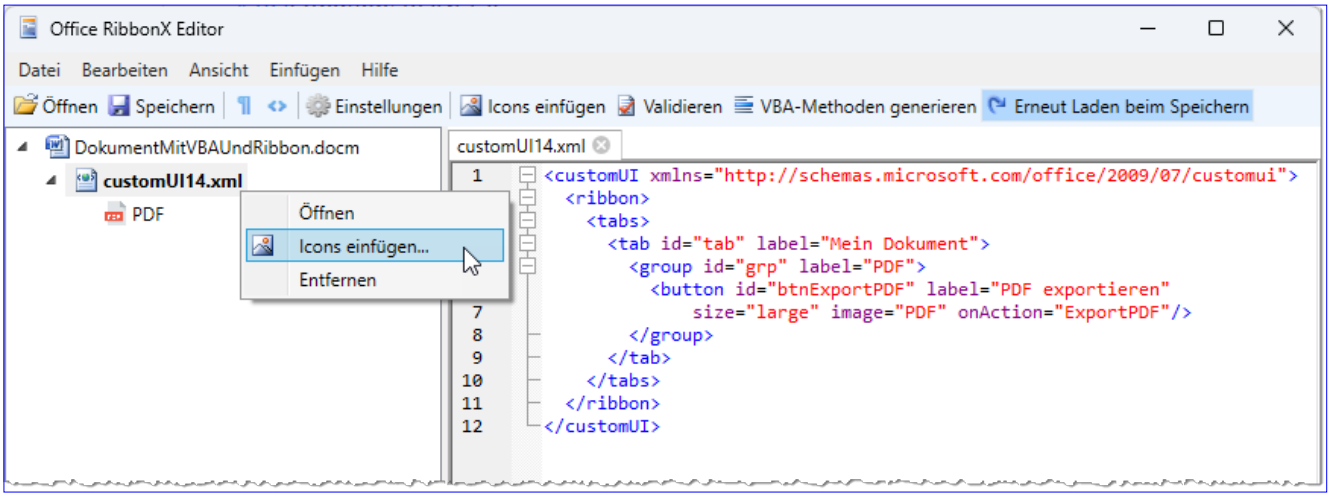


Bild 3: Einfügen des Icons und des XML-Codes

To Do-Aufgabe mit Power Automate und VBA anlegen

Power Automate ist ein cloudbasierter Service von Microsoft, der es Benutzern ermöglicht, Automatisierungen und Workflows zwischen verschiedenen Apps und Diensten zu erstellen. Früher bekannt als Microsoft Flow, bietet Power Automate eine benutzerfreundliche Oberfläche zum Erstellen von Automatisierungen ohne die Notwendigkeit von tiefgreifenden Programmierkenntnissen. Mit ein paar Tricks können wir Power Automate auch per VBA steuern. Ein guter Grund, trotz der anfallenden monatlichen Gebühren einmal einen Blick auf diese Technologie zu werfen. In diesem Artikel schauen wir uns an, welche Voraussetzungen es für Power Automate gibt und wie wir einen unverbindlichen Test damit durchführen können. Außerdem nutzen wir diesen Test für ein praxisnahes Beispiel: Wir wollen der Anwendung Microsoft To Do einen Termin hinzufügen, indem wir per VBA einen Power Automate-Flow triggern, der dann eine Aktion zum Anlegen der Aufgabe auslöst.

Was ist Power Automate überhaupt? Hier sind einige Schlüsselmerkmale und Funktionen von Power Automate:

- **Integrationen**: Es ermöglicht die nahtlose Integration mit einer Vielzahl von Microsoft- und Drittanbieteranwendungen wie Office 365, SharePoint, Dynamics 365, Google Drive, Salesforce, Twitter und so weiter.

- Workflow-Erstellung: Nutzer können Workflows visuell durch Drag-and-Drop von Aktionen erstellen. Diese Aktionen können verschiedene Aufgaben wie das Senden einer E-Mail, das Erstellen eines Eintrags in einer Datenbank, das Verschieben von Dateien et cetera ausführen.

- **Automatisierung**: Routineaufgaben können automatisiert werden, um die Produktivität zu steigern und menschliche Fehler zu reduzieren. Zum Beispiel können Genehmigungsworkflows erstellt werden, die auf bestimmte Bedingungen reagieren.

- **Benutzerfreundlichkeit**: Die Plattform ist so konzipiert, dass sie von Benutzern mit unterschiedlichem technischem Hintergrund genutzt werden kann, von Anfängern bis zu fortgeschrittenen Benutzern.

- **Auslöser und Bedingungen**: Automatisierungen können basierend auf Ereignissen (wie beispielsweise das Eingehen einer E-Mail, das Hinzufügen einer Datei in OneDrive) oder bestimmten Bedingungen gestartet werden.

- **Berichterstellung und Überwachung**: Es bietet Einblicke in die Ausführung von Workflows und ermöglicht die Überwachung der Leistung sowie die Fehlerbehandlung.

- **Sicherheit und Compliance**: Power Automate unterstützt Sicherheitsmaßnahmen wie Datenverschlüsselung und Compliance-Standards wie GDPR.

Insgesamt ist Power Automate ein leistungsstarkes Werkzeug zur Automatisierung von Geschäftsprozessen, das Unternehmen hilft, effizienter zu arbeiten und die Interaktion zwischen verschiedenen Apps und Diensten zu optimieren. Auf den nächsten Seiten schauen wir uns an, wie wir es nutzen können.

Beispielanwendung: Microsoft To Do

Ehrlich gesagt sind wir auf Power Automate schon länger aufmerksam geworden, aber am einfachsten fällt das Schreiben von Artikeln über bestimmte Themen doch, wenn man die vorgestellte Technik auch einmal praktisch einsetzen kann.

In diesem Fall haben wir nach einer Möglichkeit gesucht, um Microsoft To Do von außen mit Daten zu füttern, um beispielsweise Aufgaben aus einer Datenbankanwendung heraus schreiben zu können.

Wir haben uns natürlich auch die Rest API angesehen, die Microsoft To Do liefert. Allerdings wäre der Aufwand, die Authentifizierung zu programmieren, zu hoch gewesen. Auf der Suche nach einer Alternative sind wir über Power Automate gestoßen. Wir wollen hier eine bestimmte URL mit passenden Informationen aufrufen, um Einträge zu Microsoft To Do hinzuzufügen. Power Automate bietet eine solche Schnittstelle, sodass wir hier dieses Werkzeug nutzen wollen.

Geschäftliches Microsoftkonto erforderlich

Wir benötigen ein Geschäftskonto bei Microsoft, um mit Power Automate zu arbeiten. Mit einem privaten Konto funktionierte dies zum Zeitpunkt der Erstellung dieses Artikels nicht. Wir starten unter folgender URL:

```
https://www.microsoft.com/de-de/power-platform/products/
power-automate?market=de
```

Hier klicken wir auf **Kostenlos testen** (siehe Bild 1).

Danach landen wir auf einer Seite, auf der wir unsere Geschäfts-, Schul- oder Uni-E-Mail-Adresse eingeben können, mit der wir ein Konto bei Microsoft haben (siehe Bild 2).

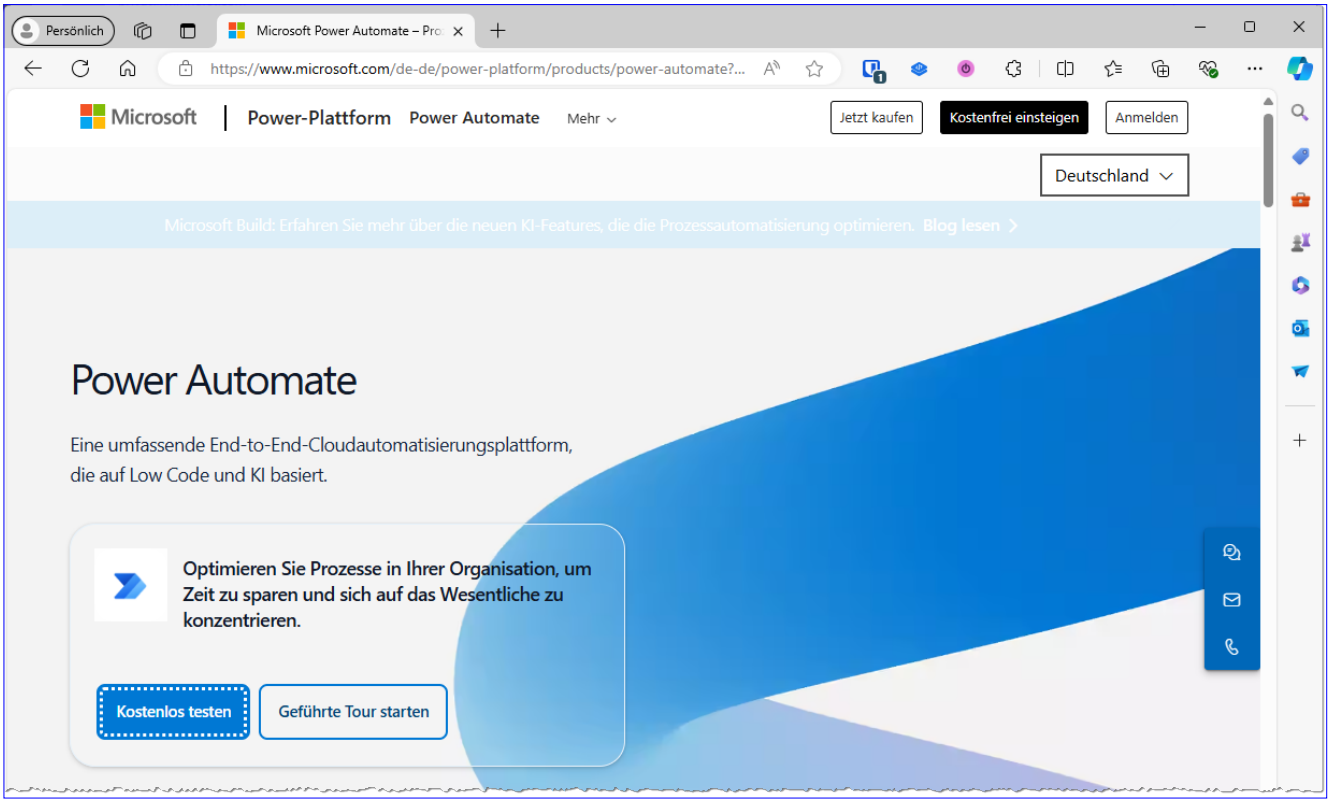


Bild 1: Einstieg in PowerAutomate

Im nächsten Schritt geben wir an, dass wir die Adresse von unserer Organisation erhalten haben.

Einen Schritt weiter fragt Microsoft unsere Telefonnummer ab, um per SMS oder Anruf einen Prüfcode zu übermitteln. Diesen geben wir nach Erhalt ein und gelangen zu einer weiteren Seite, auf der wir nun die Daten zur Erstellung unseres Kontos hinterlegen.

Nach dem erneuten Eintragen eines Prüfcodes, der diesmal per E-Mail versendet wird, klicken wir auf **Weiter**.

Nach der Registrierung können wir zu der zuerst aufgerufenen Seite zurückkehren und diese aktualisieren. Wir landen dann auf der Seite aus Bild 3.

Hier klicken wir nun auf die Schaltfläche **Erstellen**.

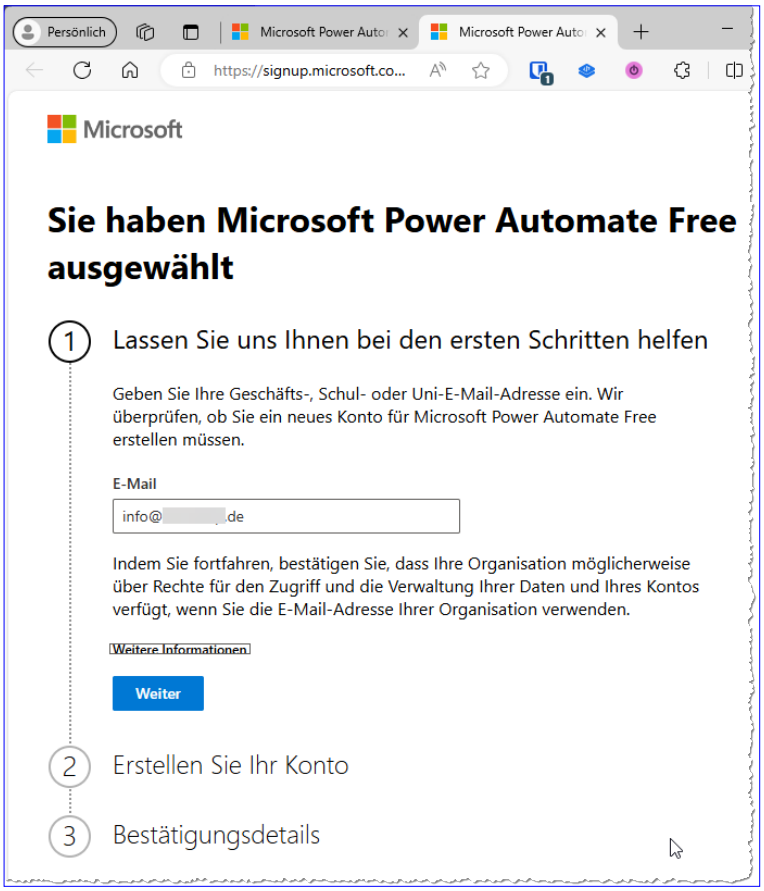


Bild 2: Eingeben der E-Mail-Adresse

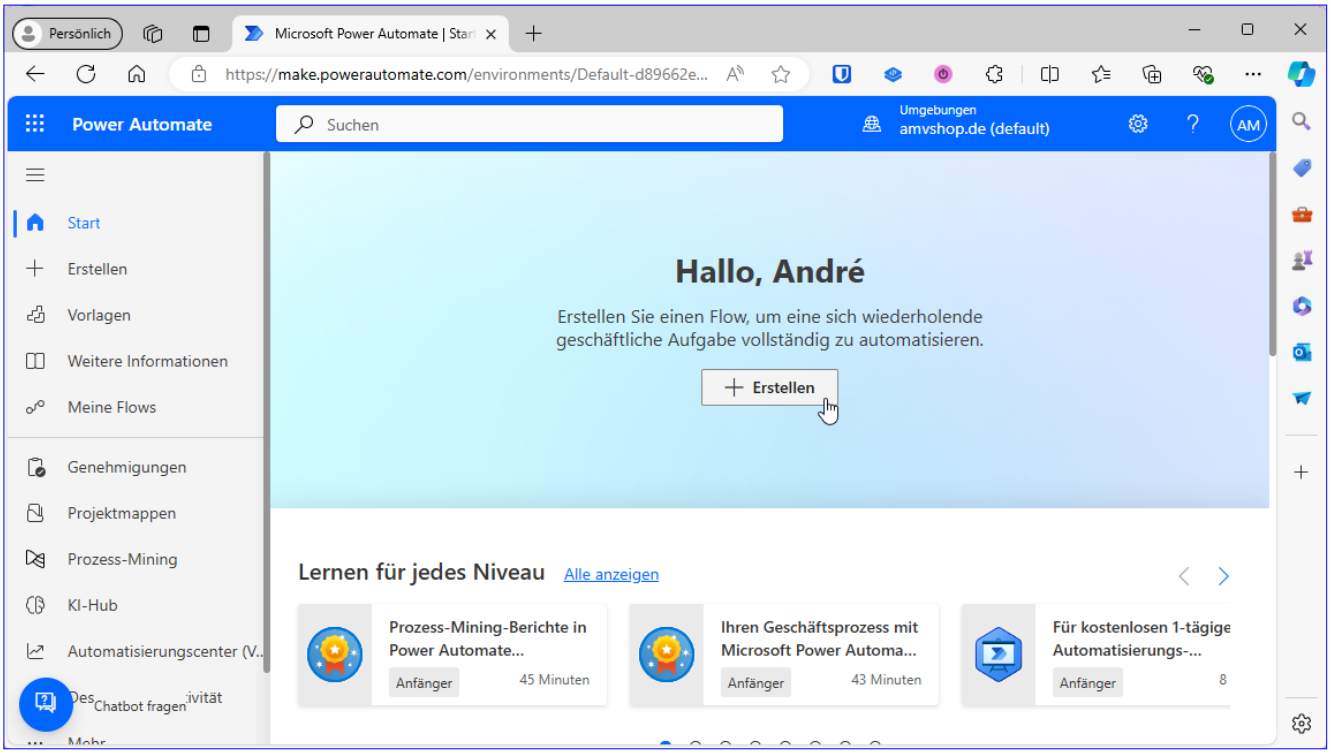


Bild 3: Zurück auf der PowerAutomate-Plattform

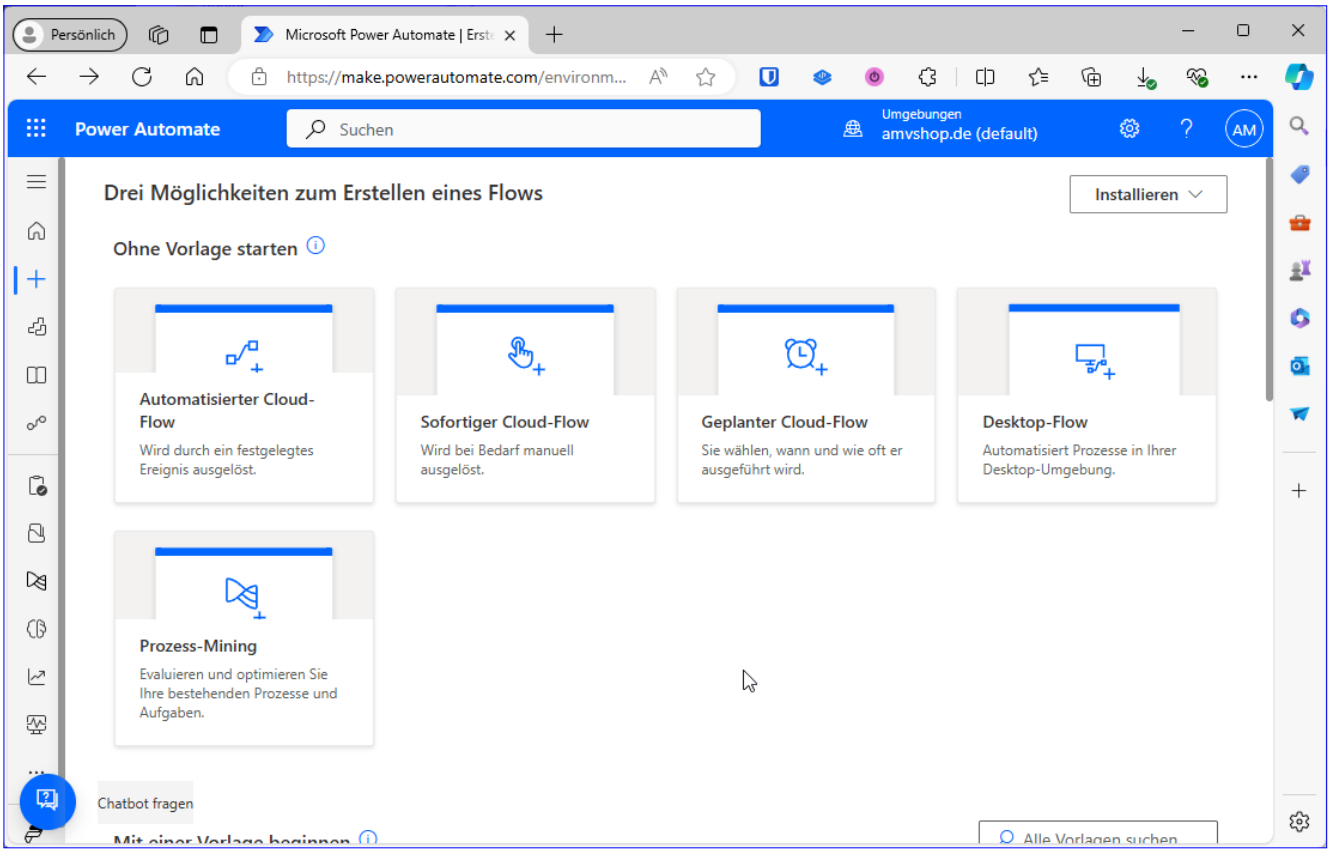


Bild 4: Erstellen eines neuen Flows

Hier sehen wir nun unter anderem die Möglichkeit, Power Automate für Desktop herunterzuladen und zu installieren. Wir wollen jedoch zunächst mit der Webversion arbeiten und schauen uns daher die verfügbaren Möglichkeiten an (siehe Bild 4).

Auf den ersten Blick ist man erst einmal überwältigt und weiß nicht, wo man anfangen soll. Die fünf hier vorgestellten Elemente dienen dazu sind allgemeine Vorlagen.

Weiter unten finden wir einige konkrete Vorlagen. Diese sind ähnlich aufgebaut wie bei Tools wie Zapier oder Make: Es gibt immer mindestens zwei beteiligte Anwendungen, von denen die eine den Trigger stellt und die andere liefert das Ziel für die nun durchzuführende Aktion. Hier kannst Du Dir einmal einen Überblick über die grundsätzlichen Möglichkeiten verschaffen.

Weiter unten findest Du die Möglichkeit, einen Connector zum Beginnen auszuwählen. Das bedeutet, dass Du die Anwendung wählst, die den Flow auslösen soll.

Hier finden wir nach einem Klick auf **Alle Connectors** viele Einträge, zum Beispiel auch Office 365 Outlook, SharePoint, Microsoft Teams, Outlook.com, SQL Server, Power BI, One Note (Business), Excel Online, Trello und auch Microsoft To Do. Wir können aber auch externe Anwendungen wie X, Google Drive, Slack, YouTube und viele mehr adressieren.

Wir starten für unseren Anwendungsfall jedoch mit der Vorlage **Sofortiger Cloud-Flow**.

Im nun erscheinenden Fenster wählen wir den Eintrag **Beim Empfang einer HTTP-Anforderung** aus (siehe Bild 5) und klicken auf **Erstellen**.

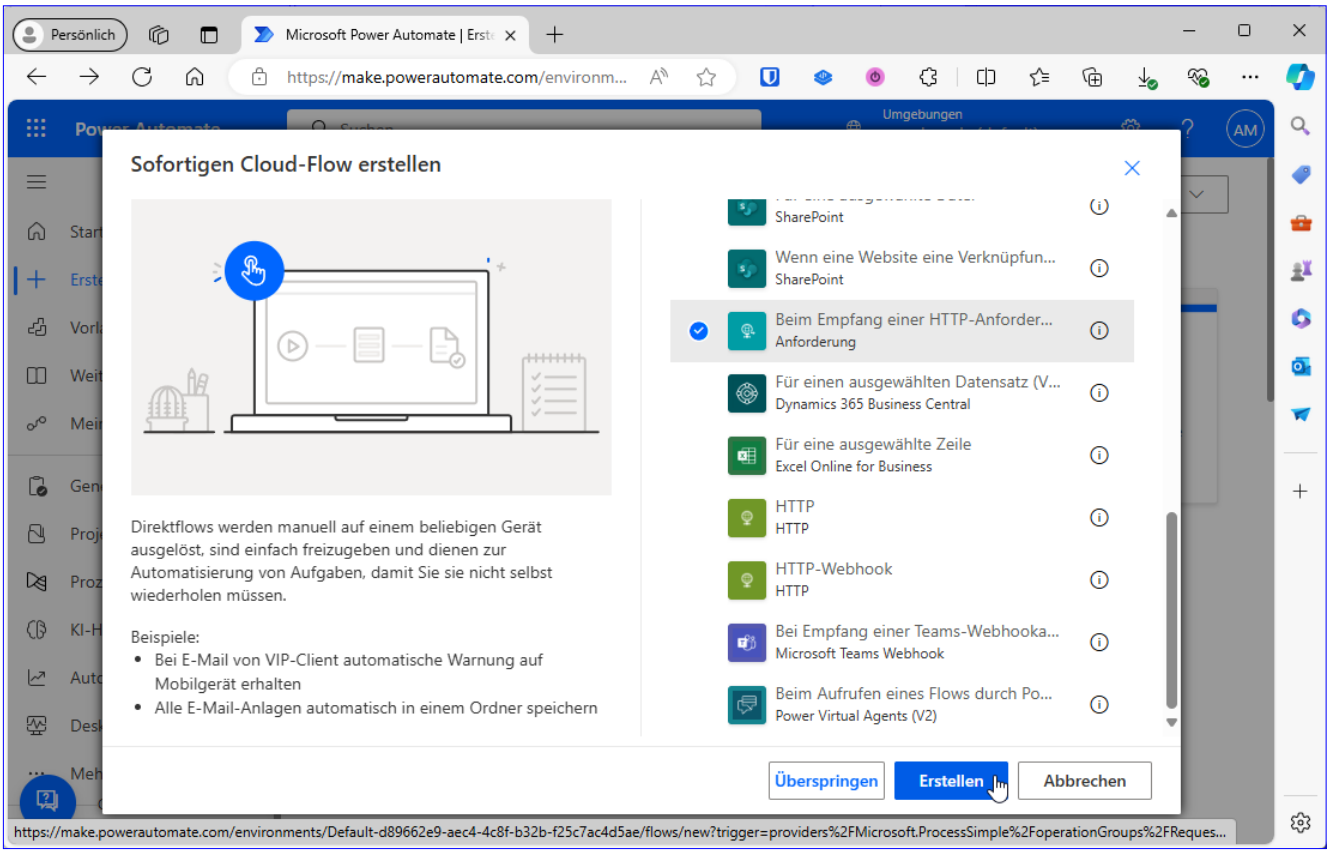


Bild 5: Hinzufügen eines Direktflows

Hier klicken wir nun auf **Manual** und füllen das nun erscheinende Formular wie in Bild 6 aus.

Unter **Parameter** stellen wir unter **Wer kann den Flow auslösen** den Wert **Jeder** ein. Das ist wichtig, weil wir sonst einen Fehler 401 erhalten. Darunter fügen wir das folgende vorläufige JSON-Schema ein, das wir später noch anpassen werden:

```
{
    "type": "object",
    "properties": {
        "title": {
            "type": "string"
        },
        "dueDate": {
```

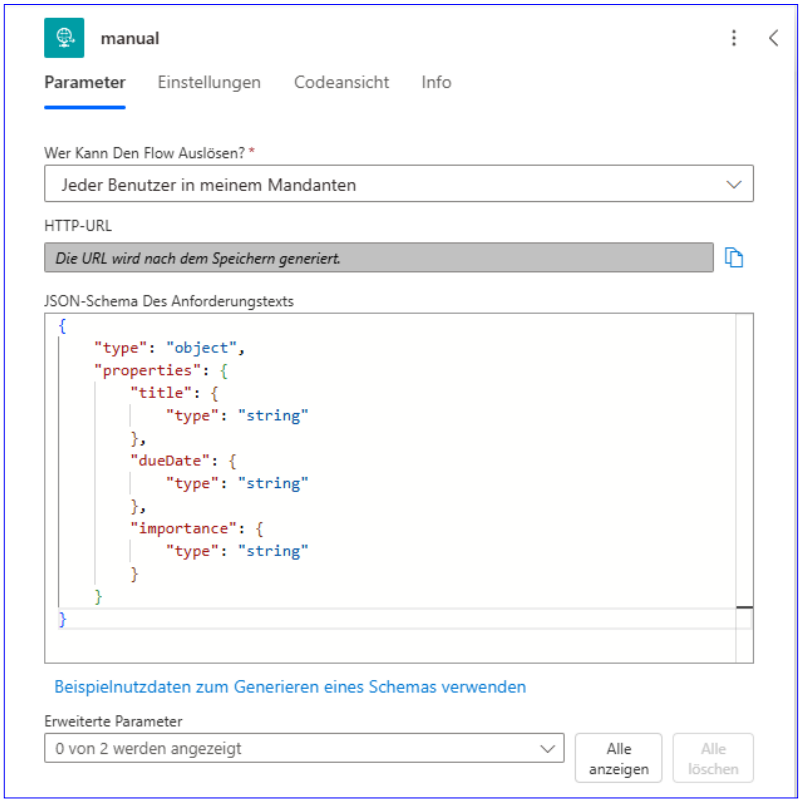


Bild 6: Eintragen eines JSON-Schemas

To Do mit VBA und Power Automate steuern

In einem weiteren Artikel namens »To Do-Aufgabe mit Power Automate und VBA anlegen« (www.vbentwickler.de/431) haben wir die Grundlagen zur Steuerung von Microsoft To Do mit VBA über Power Automate beschrieben und einen ersten Anwendungsfall vorgestellt – das Anlegen einer Aufgabe für eine vorgegebene Liste. In diesem Artikel haben wir die wichtigsten Vorbereitungen getroffen, nämlich das Anlegen eines Power Automate Kontos und das Freigeben des Zugriffs auf das To Do-Konto, dessen Listen und Aufgaben wir verwalten wollen. Dabei schauen wir uns in diesem Artikel an, wie wir die Aufrufe noch genauer gestalten können, um beispielsweise die Liste einzustellen, der wir eine neue Aufgabe hinzufügen. Außerdem schauen wir uns an, wie wir Aufgaben auslesen, bearbeiten oder löschen können und wie wir Listen auslesen, anlegen, bearbeiten oder löschen können. Es gibt viel zu tun!

Wir starten direkt dort, wo wir im oben genannten Artikel aufgehört haben, nämlich beim Anlegen einer Aufgabe. In dem dort verwendeten Code haben wir beispielsweise noch nicht die Liste dynamisch angegeben, in die wir die neue Aufgabe übertragen wollen. Wir konnten diese nur aus den vorhandenen Listen auswählen, was wir aber gern direkt von der VBA-Anwendung aus erledigen würden. Dazu müssen wir die Listen und ihre Eigenschaften auswählen, was wir nun erledigen.

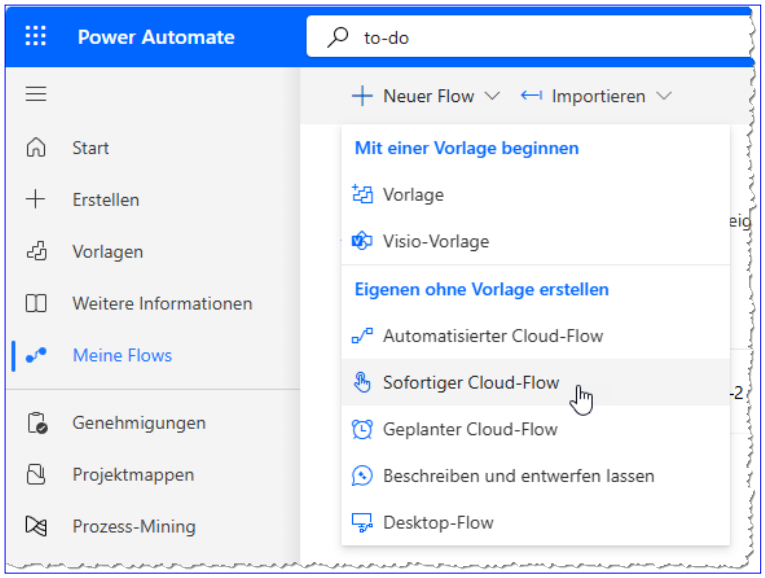


Bild 1: Anlegen eines sofortigen Cloud-Flows

Neuen Flow anlegen

Dazu legen wir einen neuen Flow mit der Vorlage **Sofortiger Cloud-Flow** (siehe Bild 1) und dem Typ **Beim Empfang einer HTTP-Anforderung** an.

Hier brauchen wir nichts weiter festzulegen – wir wollen alle Listen von To Do abfragen, dazu benötigen wir keine Requestparameter. Danach fügen wir mit einem Klick auf das Plus-Zeichen und anschließender Auswahl von Aktion hinzufügen die Aktion zum Abfragen der To Do-Listen hinzu (siehe Bild 2). Hier sehen wir bereits das Ergebnis.

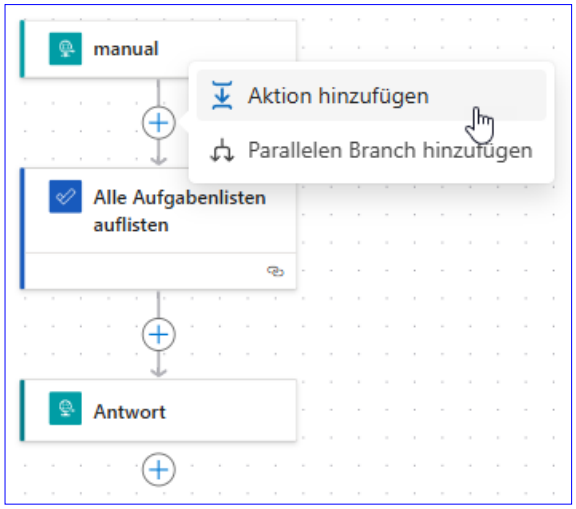


Bild 2: Hinzufügen einer Aktion

Vorher wählen wir jedoch noch aus dem Bereich **Eine Aktion hinzufügen** die gewünschte Aktion hinzu.

Dazu geben wir als Suchbegriff **Aufgabe** ein und scrollen dann nach unten, bis wir den Connector **Microsoft To-Do** finden. Hier klicken wir auf **Mehr anzeigen** und finden alle Aktionen für Microsoft To-Do vor (siehe Bild 3).

Hier wählen wir den Eintrag **Alle Aufgabenlisten auflisten** aus. Dessen Eigenschaften erscheinen nun auf der linken Seite. Auf der Seite Parameter erhalten wir wie bereits angedeutet den Hinweis, dass keine zusätzlichen Informationen nötig sind. Im unteren Bereich sollten wir, wenn wir bereits die Schritte aus dem eingangs erwähnten Artikel ausgeführt haben, die Information vorfinden, dass wir bereits mit Microsoft To-Do verbunden sind. Wir brauchen auch bei diesem Schritt nichts weiter zu tun (siehe Bild 4).

Wir können den Flow nun speichern und das wird auch ohne Probleme gelingen. Damit können wir nun zur ersten Aktion beziehungsweise zum Trigger mit der Beschriftung **manual** zurückkehren. Klicken wir diesen an, sehen wir nun unter **HTTP-URL** die URL, die wir aufrufen müssen. Hier müssen wir außerdem noch prüfen, ob **Wer kann den Flow auslösen** den Wert **Jeder** aufweist (siehe Bild 5).

Im Gegensatz zum Beispiel aus dem eingangs erwähnten Artikel wollen wir hier eine Antwort erhalten, die über einen Status mit einer Erfolgsmeldung hinausgeht. Dazu fügen wir noch eine weitere Aktion hinzu. Diese heißt **Antwort** und wir finden diese wie in Bild 6.

Antwort als Eigenschaft hinzufügen

Hier müssen wir allerdings Hand anlegen. Die Parameter der **Antwort**-Aktion sehen wir in Bild 7. Wichtig sind hier die Eigenschaften **Status Code** und **Body**. Wenn wir den Flow aufrufen, ohne diese letzte Aktion hinzuzufügen, bekom-

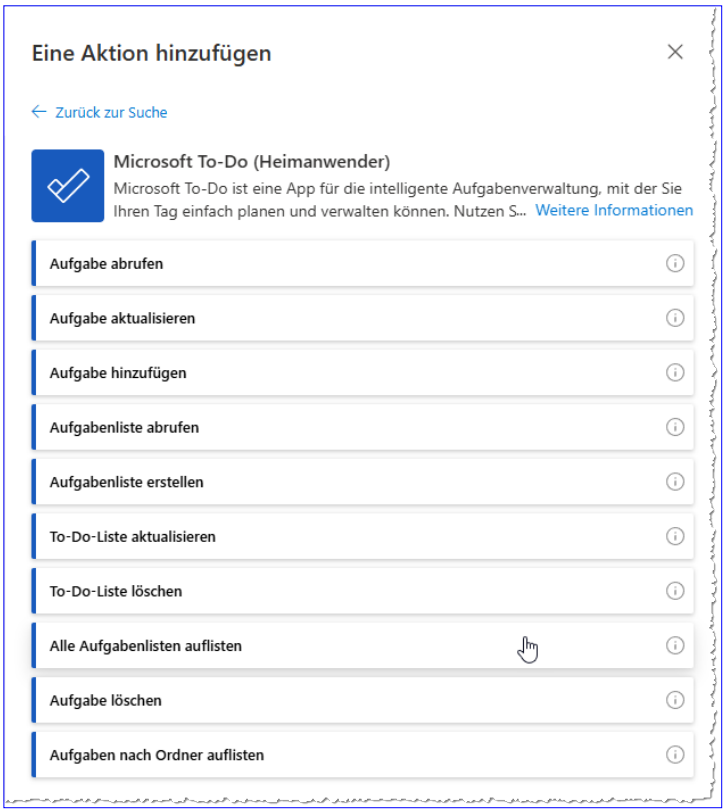


Bild 3: Alle Microsoft To-Do-Aktionen

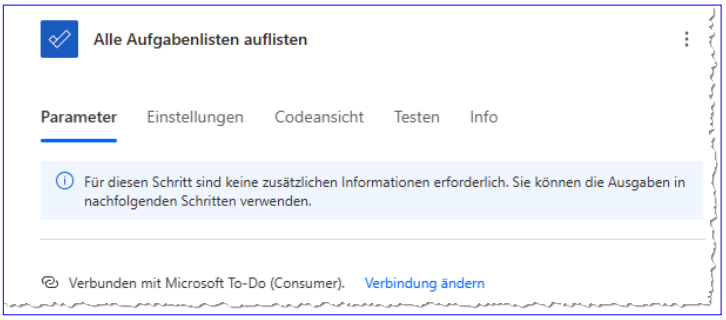


Bild 4: Einstellungen für die neue Aktion

Bild 5: Prüfen des Auslösers und Abholen der URL

Bild 6: Anlegen der **Antwort**-Aktion

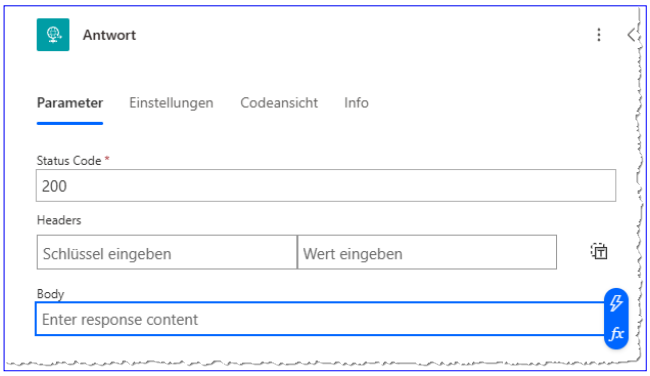


Bild 7: Eintragen des **Body**-Wertes

men wir immer den Status **202** zurück – ohne einen **Response**-Text. Hier legen wir nun fest, dass wir den Status **200** erhalten, wenn dieser Schritt erreicht wird, und mit der Eigenschaft **Body** definieren wir, welchen Inhalt wir zurückerhalten wollen. Deshalb klicken wir auf die Schaltfläche mit dem Blitz und dem Text **fx**, um die verfügbaren Elemente anzuzeigen.

Diese sehen wir nun in Bild 8. Hier sehen wir alle möglichen Ergebnisse der Aktion **Alle Aufgabenlisten auflisten**. Wir wählen den Eintrag **Textkörper** aus. Mit allen anderen Einträgen konnten wir den Flow nicht ohne Fehlermeldungen speichern. Nun gelingt dies jedoch, und wir können uns dem Aufruf per VBA zuwenden.

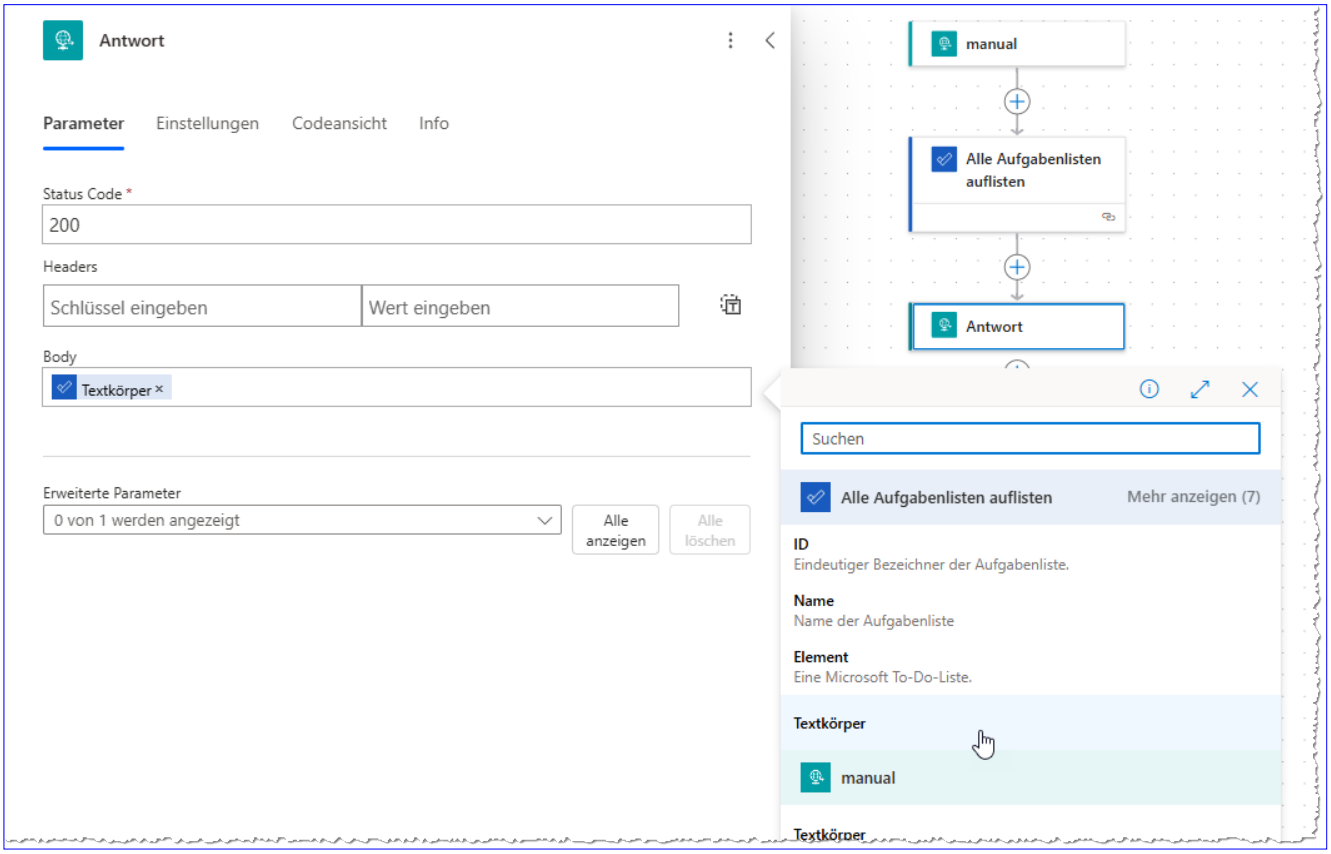


Bild 8: Hinzufügen des Elements **Textkörper** zur Eigenschaft **Body** der Antwort

Alle Aufgabenlisten per VBA ermitteln

Nachdem wir uns so praktisch unseren eigenen maßgeschneiderten Webservice gebaut haben, können wir diesen per VBA aufrufen. Um diese optimal verarbeiten zu können, wollen wir jedoch noch zwei Voraussetzungen schaffen. Als Erstes benötigen wir einen Verweis auf die Bibliothek **Microsoft Scripting Runtime**. Außerdem fügen wir zwei Module namens **mdlJSON** und **mdlJSONDOM** zum VBA-Projekt hinzu. Diese enthalten wichtige Funktionen zum Auswerten der Antwort des Webservices.

In der Prozedur **GetToDoLists** aus Listing 1 rufen wir den Webservice auf. Hier deklarieren wir einige Variablen und weisen dann der Variablen **strURL** die URL zu, die wir weiter oben dem Element **manual** entnommen haben. Dann rufen wir diese URL. Liefert diese den Wert **200** als Status zurück, war der Aufruf erfolgreich. Dann können wir den Inhalt der Eigenschaft **responseText** des **XMLHTTP**-Objekts in die Variable **strResponse** schreiben. Da es sich hierbei um ein JSON-Dokument handelt, verwenden wir die Funktion **GetJSONDOM**, um die Referenzen zu den einzelnen Inhalten des Dokuments im Direktbereich auszugeben. Dieser sieht für das erste Element wie folgt aus:

```
objJSON.Item(1).Item("@odata.etag"): W/"5kiLK5VJy0..."
objJSON.Item(1).Item("displayName"): Tasks
objJSON.Item(1).Item("isOwner"): Wahr
objJSON.Item(1).Item("isShared"): Falsch
objJSON.Item(1).Item("wellknownListName"): defaultList
objJSON.Item(1).Item("id"): AQMkADAwATNiZmYAZC05M2I2LTRi-
YTgtMDACLTAwCgAuAAADKTYDS538L0CXniiMkbeMfQEA5kiLK5VJy0C-
ml8itxoUMYQAAAgESAAAA
```

Uns interessieren primär der Anzeigename (**displayName**) und die **id**. Letztere benötigen wir, um gezielt die Aufgaben einer Liste zu bearbeiten.

```
Sub GetToDoLists()
    Dim objXMLHTTP As MSXML2.ServerXMLHTTP60
    Dim strUrl As String
    Dim strResponse As String
    Dim objJSON As Object
    Set objXMLHTTP = New MSXML2.ServerXMLHTTP60
    strUrl = "https://prod2-25.germanywestcentral.logic.azure.com:443/workflows/..."
    With objXMLHTTP
        .Open "POST", strUrl, False
        .setRequestHeader "Content-Type", "application/json"
        .send "{}"
    End With
    Select Case objXMLHTTP.status
        Case 200
            strResponse = objXMLHTTP.responseText
            Set objJSON = ParseJson(strResponse)
            Debug.Print GetJSONDOM(strResponse, True)
        Case Else
            MsgBox "Fehler: " & objXMLHTTP.status & " - " & objXMLHTTP.statusText
    End Select
    Set objXMLHTTP = Nothing
End Sub
```

Listing 1: Prozedur zum Ermitteln aller Aufgabenlisten von To Do

Wir haben den Inhalt mit der Funktion **ParseJSON** ausgelesen und in eine Objektstruktur geschrieben, die wir mit der Variablen **objJSON** referenzieren. Über diese können wir die enthaltene Werte nun mit den von **GetJSONDOM** gelieferten Referenzen ermitteln.

Wenn wir **GetJSONDOM** nun durch die folgende Schleife ersetzen, erhalten wir alle Anzeigenamen plus ID im Direktbereich:

```
Dim i As Integer
For i = 1 To objJSON.Count
    Debug.Print objJSON.Item(i).Item("displayName"), _
       objJSON.Item(i).Item("id")
Next i
```

Damit können wir arbeiten – wir können die Informationen nun beispielsweise in eine Access- oder Excel-Tabelle schreiben, um weiter damit zu arbeiten.

Aufgaben einer Aufgabenliste abrufen

Im nächsten Schritt wollen wir zu einer dieser Aufgabenlisten alle Aufgaben ausgeben. Dazu legen wir einen neuen Flow an. Das ist bereits deutlich anspruchsvoller, wie wir gleich sehen werden.

Grundsätzlich ist der Aufbau des Flows ähnlich: Wir verwenden wieder das **manual**-Objekt zum Start, dann eines, das die Einträge liefert und am Ende die Antwort. Allerdings können wir die Einträge nicht so einfach wie zuvor als Antwort zurückschicken, sondern müssen diese zwischendurch noch prozessieren. Das erscheint gleich in der Beschreibung leicht, aber da wir im Web kein Beispiel dafür gefunden haben und alles selbst herausfinden mussten, sind dafür einige Stunden draufgegangen.

Der vollständige Flow wird später wie in Bild 9 aussehen. Wir starten mit dem Request-Element, dem wir mitteilen, welche Liste ausgelesen werden soll und wie viele Einträge wir maximal erhalten wollen. Dann folgt

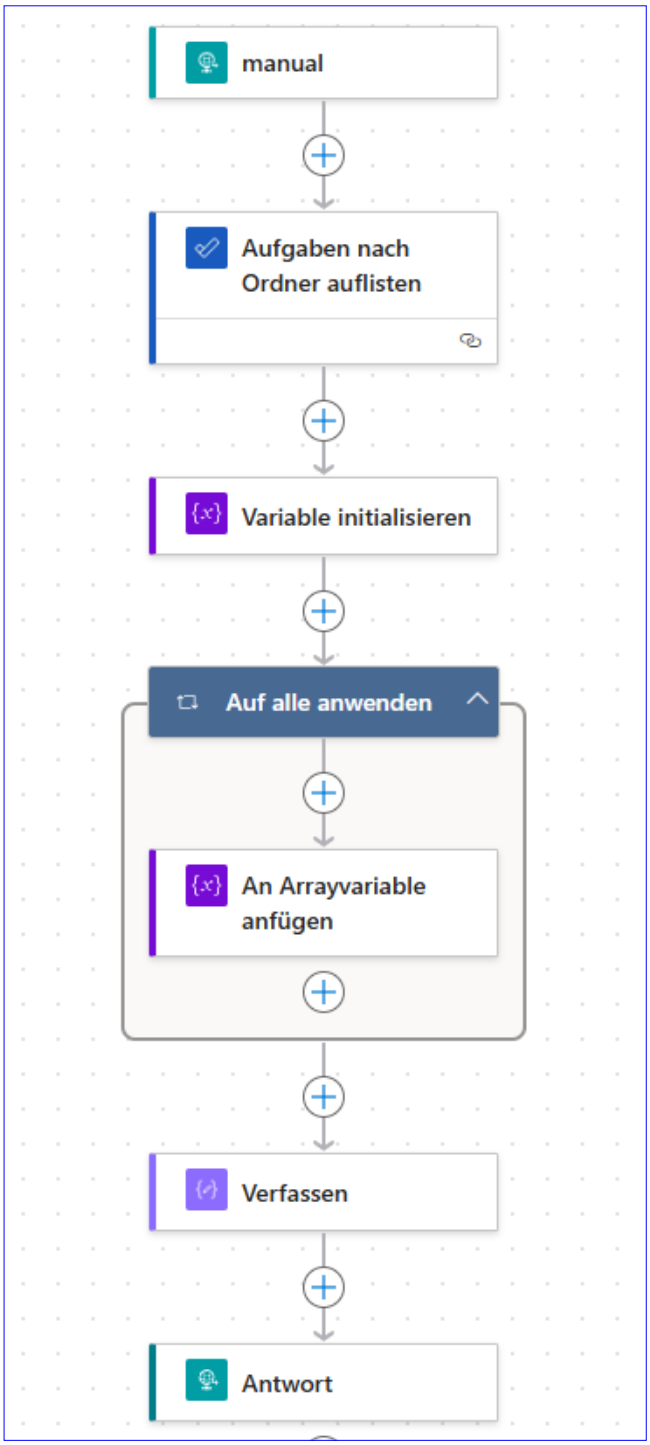


Bild 9: Vollständiger Flow zum Abfragen der Aufgaben einer Aufgabenliste

das eigentliche Auslesen durch die Aktion **Aufgaben nach Ordner auflisten**. Die damit erhaltenen Informationen müssen wir allerdings noch verarbeiten. Dazu

erstellen wir eine Array-Variable, die wir in der folgenden Schleife mit den Elementen der gefundenen Aufgaben füllen. Mit der Verfassen-Aktion bereiten wir die Inhalte zur Rückgabe als Response vor und geben diese schließlich mit dem Response-Element zurück.

Dies schauen wir uns nun im Detail an.

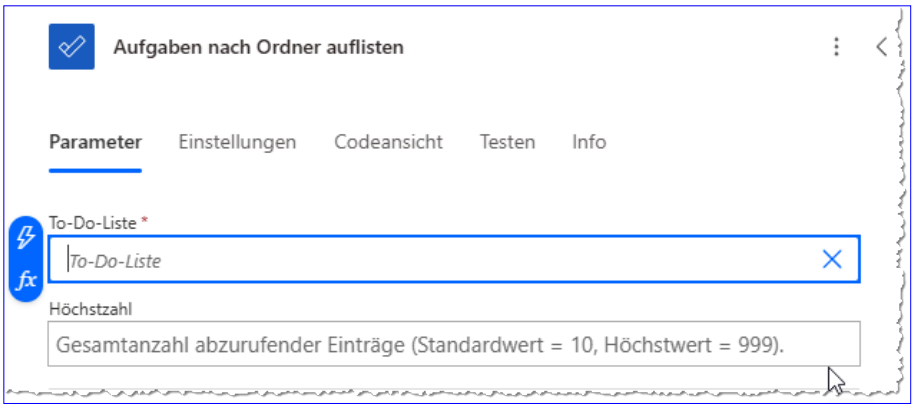


Bild 10: Weitere Auswahlmöglichkeiten

Trigger für den Request hinzufügen

Wir starten mit einem neuen Flow, für den wir wieder den Typ **Beim Empfang einer HTTP-Aufforderung**. Für diesen legen wir unter **Wer kann den Flow auslösen** wieder **Jeder** fest.

Für JSON-Schema des Anforderungstexts hinterlegen wir das folgende JSON-Dokument, mit dem wir diesmal nicht nur die **listId** abfragen, sondern mit **count** auch die maximale Anzahl der zurückzugebenden Elemente:

```
{
  "properties": {
    "listId": {
      "type": "string"
    },
    "count": {
      "type": "integer"
    }
  },
  "type": "object"
}
```

Aktion zum Abfragen der Aufgaben hinzufügen

Über das Pluszeichen fügen wir nun ein neues Element hinzu. Leider ist die Suche nach Aktionen so schlecht, dass wir uns behelfen, indem wir das Suchwort **Aufgaben** eingeben und etwas nach unten scrollen, bis wir die Gruppe **Microsoft To-Do** finden. Diese enthält auch die Aktion **Aufgaben nach Ordner auflisten**, die wir durch Anklicken hinzufügen.

Hier finden wir in den Eigenschaften unter Parameter die beiden Parameter **To-Do-List** und **Höchstzahl**. Wenn wir das Nachschlagfeld für **To-Do-Liste** öffnen, sehen wir alle To-Do-Listen für das angegebene Konto und ganz unten den Eintrag **Benutzerdefinierten Wert eingeben**. Dies blendet die Schaltfläche aus Bild 10 ein.

Klicken wir auf den Blitz oben, erhalten wir die aus den bisherigen Elementen verfügbaren Parameter, hier **listId** und **count** aus dem **manual**-Element (siehe Bild 11). Hier wählen wir folglich das Element **listId** für **To-Do-Liste** aus und **count** für **Höchstanzahl** und haben so schon alle Einstellungen für diese Aktion festgelegt.

Du solltest an dieser Stelle auch noch prüfen, ob du mit einem Microsoft To-Do-Konto verbunden bist und dies gegebenenfalls an der Stelle aus Bild 12 erledigst.

Array-Variable initialisieren

Im vorherigen Beispiel konnten wir an dieser Stelle bereits die **Antwort**-Aktion hinzufügen. Das ist hier noch nicht möglich, denn wir müssen die Elemente

des Ergebnisses der Aufgaben nach **Ordner auflisten**-Aktion noch aufbereiten.

Dazu geben wir als Suchbegriff **Variable** an und wählen dort den Eintrag **Variable initialisieren** aus (siehe Bild 13).

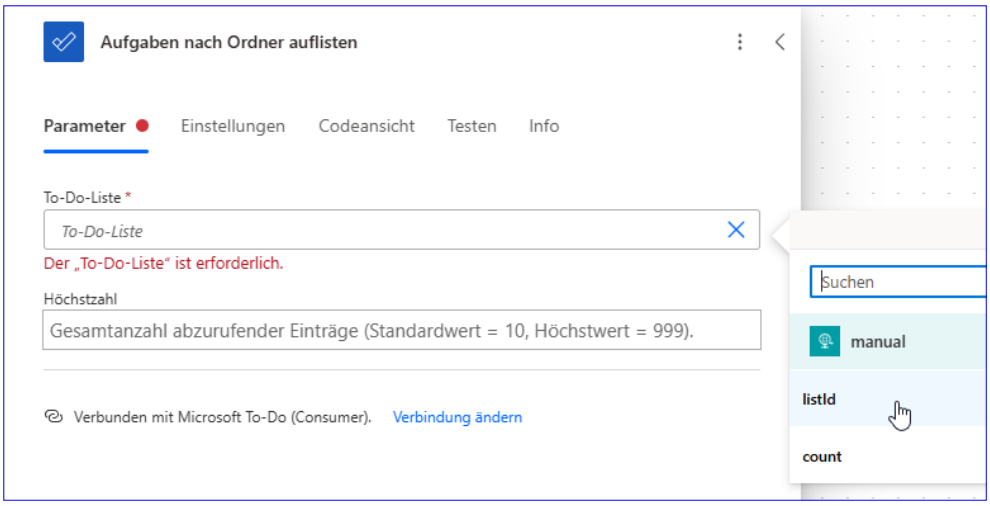


Bild 11: Hinzufügen der Parameter

Für dieses neue Variable-Element stellen wir nun die Parameter ein (siehe Bild 14). Hier finden wir den Parameter **Name**, dem wir den Wert **tasksArray** zuweisen. Der Parameter **Type** erhält den Wert **Array** und für **Value** stellen wir mit einem eckigen Klammernpaar ein leeres Array ein.

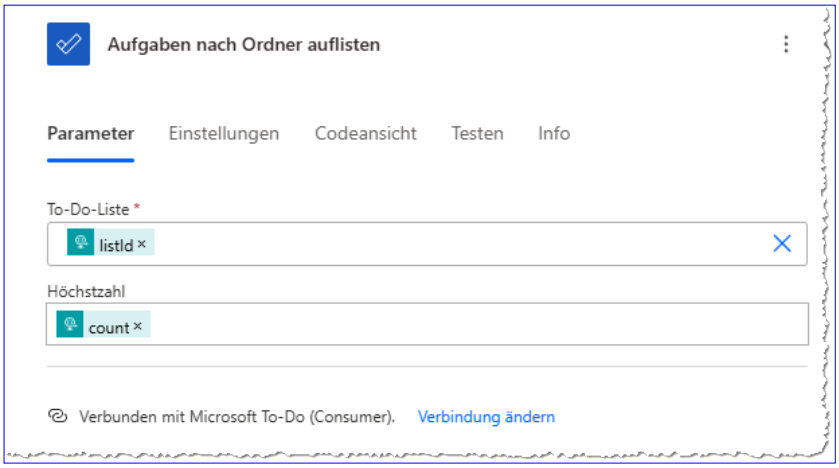


Bild 12: Prüfen der Verbindung mit Microsoft To-Do

Schleife über alle Aufgaben

Nun benötigen wir eine Schleife, in der wir alle Aufgaben in die Array-Variable eintragen. Diese finden wir unter dem Namen **Auf alle anwenden** im Bereich **Control** (siehe Bild 15).

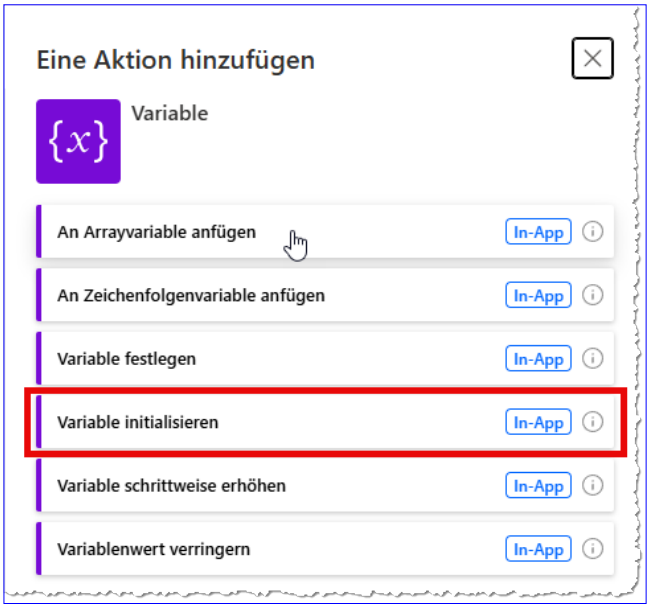


Bild 13: Hinzufügen der Aktion **Variable initialisieren**

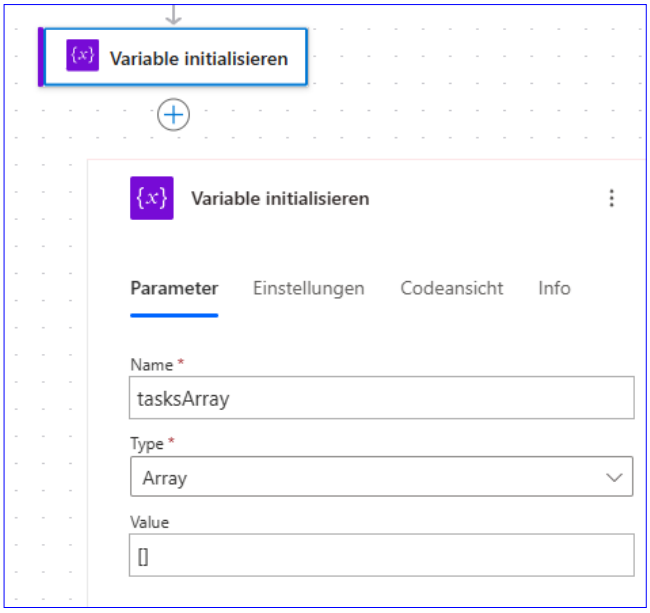


Bild 14: Einstellen der Parameter