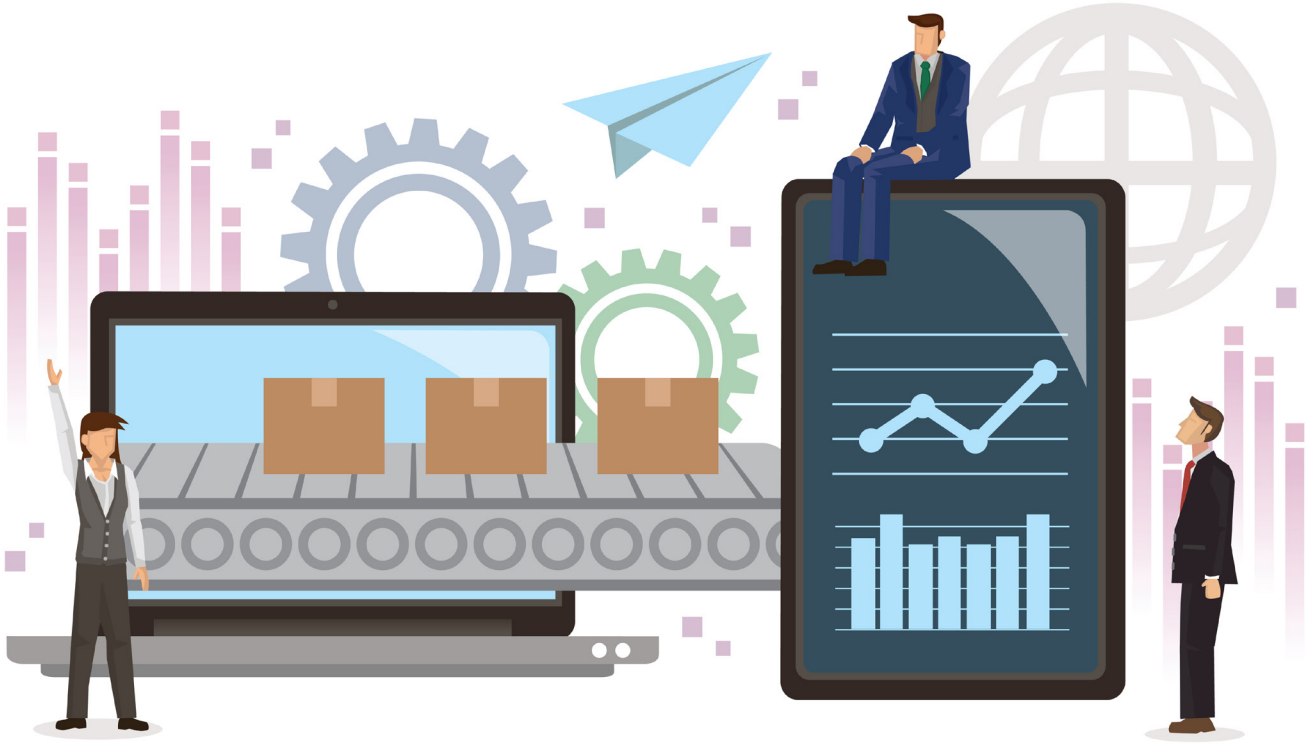
AUSGABE 02/2025

VISUAL BASIC ENTWICKLER

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



IN DIESEM HEFT:

ADODB: DATENZUGRIFF MIT RECORDSETS

Lerne alle Grundlagen zum Umgang mit der Recordset-Klasse der ADODB-Klasse.

SEITE 4

AUTOMATION MIT MAKE UND VBA

Erfahre, wie Du das mächtige Automationstool Make von VBA aus steuern kannst.

SEITE 21

MICROSOFT 365-E-MAILS MIT VBA UND MAKE

Automatisiere den E-Mail-Versand per VBA und nutze Make und Microsoft 365 dazu.

SEITE 37

André Minhorst Verlag

Outlook und Access clever automatisieren

Die Möglichkeiten, E-Mails zu automatisieren, sollen nach dem Wunsch von Microsoft demnächst in Outlook arg beschnitten werden. Das alte Outlook soll verschwinden, ein neues kommen – das aber nicht mehr VBA unterstützen wird. Deshalb wollen wir Möglichkeiten aufzeigen, mit denen Du die Automation zum Verwenden von E-Mails beibehalten kannst. Genau hier setzen die Artikel dieser Ausgabe an.

Den Auftakt macht der umfassende Beitrag **ADODB: Datenzugriff mit Recordsets** (ab Seite 4). Hier zeigen wir Dir in über 60 Funktionen und Methoden, wie Du Daten effizient verwaltest, filterst, sortierst und analysierst. Besonderes Highlight: die Nutzung von Ereignissen, um Änderungen abzufangen und Prozesse automatisch zu steuern – ideal für alle, die tiefer in die Welt des Recordsets einsteigen möchten.

Im Artikel **Automation mit VBA und Make** (ab Seite 21) lernst Du, wie Du Deine VBA-Projekte mit cloudbasierten Automationen verknüpfst. Du erfährst, wie Du per Webhook Abläufe startest, Ergebnisse verarbeitest und sogar Rückmeldungen in Echtzeit erhältst. Make.com wird hier zur Schaltzentrale für Deine Prozesse, ohne dass Du Dich mit komplexen API-Authentifizierungen auseinandersetzen musst.

Danach folgt **Microsoft 365-Mail mit Make und VBA ohne Outlook** (ab Seite 37). Dieser Beitrag zeigt Dir, wie Du den Versand von E-Mails aus VBA-Anwendungen heraus modern umsetzt – ohne den Umweg über das klassische Outlook. Stattdessen baust Du Dir einen robusten, flexiblen Workflow mit Make und Microsoft 365, der auch zukünftigen Änderungen standhält.

Zum Abschluss präsentieren wir **Microsoft 365 E-Mails mit Make per Klasse senden** (ab Seite 53). Hier lernst Du, wie Du eine eigene VBA-Klasse entwickelst, um E-Mails strukturiert und wiederverwendbar zu versenden. Du profitierst von sauberem Code, einfacher Erweiterbarkeit und der vollen Kontrolle über alle Parameter, Empfänger und Anhänge.

Ich bin sicher, dass diese Ausgabe viele neue Ideen für Deine Projekte liefern wird. Viel Freude beim Lesen, Ausprobieren und Weiterentwickeln!

Dein André Minhorst

ADODB: Datenzugriff mit Recordsets

Recordsets sind zentrale Bestandteile beim Zugriff auf Daten mit ADODB. Sie bieten umfangreiche Funktionen zur Navigation, Bearbeitung, Filterung und Analyse von Daten. In diesem Artikel zeigen wir eine vollständige Übersicht aller Eigenschaften und Methoden des Recordset-Objekts und erläutern diese jeweils ausführlich mit praktischen Beispielen. Besonderes Augenmerk legen wir auf die Ereignisse eines Recordsetes. Im Gegensatz zum DAO-Recordset bietet das ADODB-Recordset nämlich die Möglichkeit, auf verschiedene Ereignisse zu reagieren – beispielsweise auf Änderungen im Datensatz.

Beispieldatenbank

Die Beispiele dieses Artikels findest Du in der Beispieldatenbank **ADODB_Recordset_AlleEigenschaften.accdb**.

Übersicht: Eigenschaften und Methoden

Im Folgenden geben wir eine Kurzbeschreibung aller verfügbaren Eigenschaften und Methoden des **Recordset**-Objekts, bevor wir diese im Einzelnen detailliert erläutern:

- **AbsolutePage**: Liefert oder setzt die aktuelle Seite bei Seitennavigation.
- **AbsolutePosition**: Position des aktuellen Datensatzes im Recordset.
- **ActiveCommand**: Liefert das zugehörige **Command**-Objekt.
- **ActiveConnection**: Verbindung, mit der das Recordset verknüpft ist.
- **AddNew/Update/CancelUpdate**: Neue Datensätze hinzufügen und speichern.
- **BOF/EOF**: Gibt an, ob sich der Datensatzmarkierer vor dem ersten oder nach dem letzten Datensatz befindet.
- **Bookmark/CompareBookmarks**: Zum Speichern und Vergleichen von Positionen.
- **CacheSize**: Anzahl Datensätze, die lokal zwischengespeichert werden.
- **Cancel/CancelBatch**: Bricht aktuelle Operationen ab.
- **Clone**: Erstellt eine Kopie des Recordsets.
- **Close/Open**: Öffnet oder schließt das referenzierte Recordset.
- **CursorLocation/CursorType**: Steuerung der Navigation und Server-/Clientverarbeitung.
- **Delete**: Entfernt den aktuellen Datensatz.
- **EditMode**: Zeigt an, ob und wie ein Datensatz bearbeitet wird.
- **Filter/Find/Seek**: Filtert Datensätze oder sucht bestimmte Inhalte.
- **Fields**: Auflistung aller Felder eines Datensatzes.
- **GetRows/GetString**: Exportiert Daten als Array oder String.

- **LockType**: Steuerung der Sperrmechanismen bei Bearbeitung.
- **MoveFirst**: Navigiert zum ersten Datensatz im Recordset.
- **MoveLast**: Navigiert zum letzten Datensatz im Recordset.
- **MoveNext**: Navigiert zum nächsten Datensatz.
- **MovePrevious**: Navigiert zum vorherigen Datensatz.
- **Move:** Navigiert um eine bestimmte Anzahl Recordsets vor oder zurück.
- **NextRecordset**: Weiteres Recordset aus einem Stapel.
- **PageCount/PageSize**: Paginierung großer Datenmengen.
- **Properties**: Auflistung aller Eigenschaften.
- **RecordCount**: Anzahl der Datensätze.
- **Requery/Resync**: Aktualisiert das Recordset mit aktuellen Daten.
- **Save**: Speichert das Recordset als Datei.
- **Sort**: Sortiert die Datensätze.
- **Source/State/Status**: Informationsfelder zum Recordset.
- **StayInSync/Supports**: Steuerung von Synchronisierung und unterstützten Features.
- **UpdateBatch**: Speichert mehrere Änderungen gesammelt.

Detaillierte Beschreibung aller Funktionen

Im folgenden Abschnitt erläutern wir jede dieser Methoden und Eigenschaften einzeln.

Wir geben Anwendungsbeispiele, zeigen Einsatzmöglichkeiten in VBA und gehen auf Spezialfälle ein.

Beispielumgebung

In diesem Artikel verwenden wir durchgängig die Tabelle **tblKunden** mit den Feldern **KundenID**, **Nachname**, **Vorname** und **Ort**. Die Datenbank enthält fünf Testdatensätze mit Namen und Orten.

Diese erstellen wir mit einer eigenen Prozedur (siehe Listing 1).

Verbindung und Recordset öffnen

Alle folgenden Beispiele beginnen mit dem vollständigen Aufbau einer ADODB-Verbindung zu einer Access-Datenbank und dem Öffnen eines Recordsets auf Basis der Tabelle **tblKunden**.

Ein Beispiel hierzu sehen wir in Listing 2. Hier erstellen wir ein **Connection**-Objekt und füllen es mit einem Verweis auf die Connection der aktuellen Access-Datenbank (**CurrentProject.Connection**).

Danach erstellen und öffnen wir ein Recordset auf Basis der Tabelle **tblKunden**. Die Details beschreiben wir später. Schließlich durchlaufen wir in einer **Do While**-Schleife alle Datensätze dieses Recordsets.

EOF und BOF

Die Eigenschaften **EOF** (End of File) und **BOF** (Beginning of File) zeigen an, ob sich der Datensatzzeiger am Ende oder vor dem Anfang des Recordsets befindet.

Beide Eigenschaften sind vom Typ **Boolean** und spielen eine zentrale Rolle bei der sicheren Navigation durch Daten.

Insbesondere bei leeren Recordsets ist es wichtig, beide Eigenschaften zu prüfen: Ist **EOF** und **BOF** gleichzeitig **True**, enthält das Recordset keine Datensätze – weder vorne noch hinten

In Listing 3 erstellen wir ein Recordset, das keine Datensätze enthält, da das Kriterium (1=2) für keinen Datensatz erfüllt wird.

```
Public Sub TabelleKundenErstellenUndFüllen()
    Dim cnn As ADODB.Connection
    Dim strPfad As String
    Dim strSQL As String

    ' Verbindung zur aktuellen Access-Datenbank herstellen
    Set cnn = New ADODB.Connection
    strPfad = CurrentProject.FullName
    cnn.ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;" & _
                           "Data Source=" & strPfad & ";" & _
                           "Persist Security Info=False;"
    cnn.Open

    ' Bestehende Tabelle löschen (falls vorhanden)
    On Error Resume Next
    cnn.Execute "DROP TABLE tblKunden"
    On Error GoTo 0

    ' Tabelle erstellen
    strSQL = "CREATE TABLE tblKunden (" & _
             "KundenID AUTOINCREMENT PRIMARY KEY, " & _
             "Nachname TEXT(50), " & _
             "Vorname TEXT(50), " & _
             "Ort TEXT(50))"
    cnn.Execute strSQL

    ' Beispieldatensätze einfügen
    cnn.Execute "INSERT INTO tblKunden (Nachname, Vorname, Ort) VALUES ('Müller', 'Anna', 'Berlin')"
    cnn.Execute "INSERT INTO tblKunden (Nachname, Vorname, Ort) VALUES ('Schmidt', 'Peter', 'Hamburg')"
    cnn.Execute "INSERT INTO tblKunden (Nachname, Vorname, Ort) VALUES ('Lehmann', 'Julia', 'München')"
    cnn.Execute "INSERT INTO tblKunden (Nachname, Vorname, Ort) VALUES ('Weber', 'Klaus', 'Stuttgart')"
    cnn.Execute "INSERT INTO tblKunden (Nachname, Vorname, Ort) VALUES ('Klein', 'Maria', 'Düsseldorf')"

    ' Verbindung schließen
    cnn.Close
    Set cnn = Nothing

    MsgBox "Tabelle 'tblKunden' wurde erstellt und mit Daten gefüllt.", vbInformation
End Sub
```

Listing 1: Erstellen einer Tabelle mit Beispieldaten

```
Public Sub KundenRecordsetOeffnen()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "SELECT * FROM tblKunden", cnn, adOpenKeyset, adLockOptimistic
    Do Until rst.EOF
        Debug.Print rst!Vorname & " " & rst!Nachname
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    cnn.Close
    Set cnn = Nothing
End Sub
```

Listing 2: Prozedur zum Durchlaufen von Kundendatensätzen

Hier prüfen wir, ob **rst.EOF** und **rst.BOF** gleichzeitig **True** liefern. In diesem Fall gibt es keine Datensätze.

In einem weiteren Beispiel nutzen wir **rst.EOF** bei einer Suche. Wir suchen mit der **Find**-Methode nach einem Datensatz mit einem bestimmten Kriterium. Nach der Suche prüfen wir, ob **rst.EOF** wahr ist. In diesem Fall wird die Suche als erfolglos gemeldet, anderenfalls wird der gefundene Datensatz ausgegeben:

```
...
rst.Find "Nachname = 'Meyer'"

If rst.EOF Then
```

```
Public Sub KundenDurchlaufen()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "SELECT * FROM tblKunden WHERE 1 = 2", cnn, adOpenKeyset, adLockOptimistic

    If rst.BOF And rst.EOF Then
        MsgBox "Keine Datensätze vorhanden."
    Else
        'Operationen für vorhandene Datensätze
    End If

    rst.Close
    Set rst = Nothing
    cnn.Close
    Set cnn = Nothing
End Sub
```

Listing 3: Prüfen, ob sich der Datensatzzeiger vor oder hinter dem ersten oder letzten Datensatz befindet.

```
    MsgBox "Kunde 'Meyer' wurde nicht gefunden.", _
        vbInformation
Else
    MsgBox "Kunde gefunden: " & rst!Vorname & " " _
        & rst!Nachname
End If
...
```

Auf Felder zugreifen mit Fields

Über die **Fields**-Auflistung kannst Du auf einzelne Spalten eines Datensatzes zugreifen. Du kannst sowohl den Feldnamen als auch den Index (beginnend bei **0**) verwenden.

```
Public Sub FeldZugriffBeispiel()
...
    rst.MoveFirst
    Debug.Print rst.Fields(„Vorname“).Value
    Debug.Print rst.Fields(2).Value
...
End Sub
```

EditMode

Mit **EditMode** lässt sich abfragen, ob das Recordset aktuell bearbeitet wird.

Diese Information ist besonders bei formulargebundenen oder interaktiv bearbeiteten Recordsets nützlich.

```
Public Sub BearbeitungsstatusPruefen()
    ...
    rst.Open "SELECT * FROM tblKunden", cnn, _
        adOpenKeyset, adLockOptimistic
    rst.MoveFirst
    rst!Ort = "Leipzig"
    If rst.EditMode = adEditInProgress Then
        Debug.Print "Bearbeitung aktiv."
    End If
    rst.Update
    ...
End Sub
```

Datensätze hinzufügen mit AddNew/Update

Neue Datensätze lassen sich mit **AddNew** erstellen. Anschließend musst Du mit **Update** speichern. Hier ein vollständiges Beispiel:

```
Public Sub KundeHinzufuegen()
    ...
    rst.AddNew
    rst!Vorname = "Lisa"
    rst!Nachname = "Neumann"
    rst!Ort = "Hannover"
    rst.Update
    ...
End Sub
```

Datensätze löschen mit Delete

Um einen Datensatz zu löschen, positionierst Du das Recordset auf den gewünschten Eintrag und rufst **Delete** auf. Die Änderung wird sofort übernommen:

```
Public Sub KundeLoeschen()
    ...
    Set rst = New ADODB.Recordset
    rst.Open "SELECT * FROM tblKunden WHERE Nachname = " _
        "'Neumann'", cnn, adOpenKeyset, adLockOptimistic

    If Not rst.EOF Then
        rst.Delete
    End If
    ...
End Sub
```

Recordset filtern mit Filter

Mit der Eigenschaft **Filter** kannst Du ein geöffnetes Recordset einschränken. Der Filter wirkt sich auf die Navigation und Methoden wie **MoveNext** aus.

```
Public Sub KundenInBerlin()
    ...
    rst.Open "SELECT * FROM tblKunden", cnn, _
```

```
        adOpenKeyset, adLockOptimistic

    rst.Filter = "Ort = 'Leipzig'"
    Do Until rst.EOF
        Debug.Print rst!Vorname & " " & rst!Nachname
        rst.MoveNext
    Loop
    ...
End Sub
```

Datensatz suchen mit Find

Mit **Find** lässt sich der erste passende Datensatz ermitteln. Die Suche erfolgt relativ zur aktuellen Position. Als Parameter geben wir ein Filterkriterium an, das genauso aussieht wie das für die **WHERE**-Condition:

```
Public Sub KundeFinden()
    ...
    rst.Find "Nachname = 'Schmidt'"
    If Not rst.EOF Then
        Debug.Print "Gefunden: " & rst!Vorname
    End If
    ...
End Sub
```

Datensätze sortieren mit Sort

Mit **Sort** kannst Du die aktuelle Reihenfolge der Datensätze im Recordset ändern – ohne erneute Datenbankabfrage. Hier ist es allerdings erforderlich, dass wir explizit die Eigenschaft **CursorLocation** auf **adUseClient** einstellen. Bisher haben wir diese Einstellung nicht vorgenommen, weil dies nicht notwendig war. Mehr zu dieser Einstellung weiter unten.

Im Beispiel rufen wir die **Sort**-Methode mit dem Parameter **Nachname DESC** auf. Dadurch sortieren wir absteigend nach den Nachnamen:

```
Public Sub KundenSortieren()
    ...
    Set rst = New ADODB.Recordset
    rst.CursorLocation = adUseClient
    rst.Open "SELECT * FROM tblKunden", cnn, adOpenKeyset,
adLockOptimistic

    rst.Sort = "Nachname DESC"
    rst.MoveFirst
    Do Until rst.EOF
        Debug.Print rst!Nachname
        rst.MoveNext
    Loop
    ...
End Sub
```

Wenn wir nicht nur nach einem Feld sortieren wollen, geben wir die Sortierkriterien durch Kommata getrennt an – genau wie in der **ORDER BY**-Klausel von SQL:

```
Public Sub KundenSortierenMehrereKriterien()
    ...
    rst.Sort = "Nachname DESC, Vorname ASC"
    rst.MoveFirst
    Do Until rst.EOF
        Debug.Print rst!Nachname, rst!Vorname
        rst.MoveNext
    Loop
    ...
End Sub
```

GetRows

Mit **GetRows** lassen sich Datensätze blockweise als Array auslesen – effizient für Auswertungen oder Übergabe an andere Routinen.

Im folgenden Beispiel füllen wir ein Array auf Basis der **GetRows**-Funktion des ADODB-Recordsets. Danach durchlaufen wir alle Zeilen und Spalten und geben diese im Direktbereich von Access aus:

```
Public Sub KundenAlsArray()
    Dim arr As Variant
```

```
    ...
    arr = rst.GetRows
    For j = LBound(arr, 2) To UBound(arr, 2) 'Datensätze
        For i = LBound(arr, 1) To UBound(arr, 1) 'Felder
            Debug.Print arr(i, j);
            If i < UBound(arr, 1) Then Debug.Print " | ";
        Next i
        Debug.Print
    Next j
    ...
End Sub
```

Dies gibt die Daten wie folgt aus:

```
1  | Müller | Anna | Leipzig
 2  | Schmidt | Peter | Hamburg
 3  | Lehmann | Julia | München
 4  | Weber | Klaus | Stuttgart
 5  | Klein | Maria | Düsseldorf
 7  | Klein | Nina | Düsseldorf
 8  | Klein | Andrea | Düsseldorf
```

Navigieren mit MoveFirst, MoveLast, MoveNext und MovePrevious

Diese Methoden ermöglichen die Navigation im Recordset. **MoveFirst** springt zum ersten Datensatz, **MoveLast** zum letzten. **MoveNext** und **MovePrevious** bewegen den Zeiger jeweils vorwärts oder rückwärts.

Das folgende Beispiel zeigt die verschiedenen Aufrufe:

```
Public Sub NavigationKunden()
...
rst.Open "SELECT * FROM tblKunden", cnn, _
    adOpenStatic,  adLockReadOnly
rst.MoveLast
Debug.Print "Letzter Kunde: " & rst!Nachname
rst.MoveFirst
Debug.Print "Erster Kunde: " & rst!Nachname
rst.MoveNext
Debug.Print "Zweiter Kunde: " & rst!Nachname
...
```

Datensatzzeiger beliebig verschieben mit Move

Die Methode **Move** ermöglicht es Dir, den Datensatzzeiger um eine beliebige Anzahl von Positionen vorwärts oder rückwärts zu verschieben. Der erste Parameter gibt an, wie viele Datensätze der Zeiger verschoben werden soll. Positive Zahlen bewegen den Zeiger vorwärts, negative rückwärts. Ein zweiter, optionaler Parameter gibt an, von welcher Startposition aus die Bewegung erfolgen soll, meistens wird dieser aber nicht verwendet.

Beispiel:

```
rst.Move 3 , Springt drei Datensätze vor
rst.Move -1 , Springt einen Datensatz zurück
```

Properties

Das **Properties**-Objekt enthält zusätzliche Informationen zum Recordset – etwa Provider-spezifische Einstellungen.

Hier durchlaufen wir alle **Property**-Elemente in einer **Do While**-Schleife und geben jeweils den **Property**-Namen und den Wert im Direktbereich des VBA-Editors aus:

```
Public Sub EigenschaftenAuflisten()
    ...
    Dim prp As ADODB.Property
    For Each prp In rst.Properties
        Debug.Print prp.Name & ": " & prp.Value
    Next prp
    ...
End Sub
```

RecordCount

RecordCount liefert die Anzahl der Datensätze. Achtung: Bei Verwendung des Wertes **adOpenForward-**

Only für den dritten Parameter der **Open**-Methode ist diese Information erst nach vollständigem Durchlauf verfügbar.

```
Public Sub KundenZaehlen()
    ...
    Debug.Print "Kundenanzahl: " & rst.RecordCount
    ...
End Sub
```

Requery

Requery lädt die Daten im Recordset neu – ideal nach Änderungen in der Datenquelle an anderer Stelle:

```
Public Sub KundenNeuLaden()
    ...
    rst.Requery
    Debug.Print "Neu geladen: " & rst.RecordCount _
        & " Datensätze"
    ...
End Sub
```

Resync

Mit der Methode **Resync** kannst Du den Inhalt eines Recordsets aktualisieren – entweder vollständig oder selektiv.

Das ist besonders nützlich, wenn sich die zugrunde liegenden Daten außerhalb des Recordsets geändert haben, zum Beispiel durch parallele Bearbeitung oder nach einem Abgleich mit dem Server.

Die Methode kann entweder für den aktuellen Datensatz oder das gesamte Recordset ausgeführt werden – je nach Parameterwahl:

- **adAffectCurrent**: nur aktueller Datensatz
- **adAffectGroup**: alle Datensätze mit gleichem Filter
- **adAffectAll**: gesamtes Recordset

Im folgenden Beispiel aktualisieren wir gezielt den aktuellen Datensatz, um Änderungen von außen zu übernehmen (zum Beispiel durch andere Benutzer):

```
Public Sub AktuellenKundenResyncen()
    ...
    rst.CursorLocation = adUseClient
    rst.Open "SELECT * FROM tblKunden", cnn, _
        adOpenKeyset, adLockOptimistic
    rst.MoveFirst
    MsgBox "Vor Resync: " & rst!Ort
    rst.Resync adAffectCurrent
    MsgBox "Nach Resync: " & rst!Ort
    ...
End Sub
```

Um alle Daten im Recordset auf den aktuellen Stand zu bringen, etwa nach Änderungen in der Tabelle durch andere Prozesse, verwendest Du:

```
Public Sub GesamtesRecordsetResyncen()
    ...
    rst.Resync adAffectAll
    MsgBox "Synchronisierung abgeschlossen."
    ...
End Sub
```

Änderungen auf einen Rutsch mit UpdateBatch

Die Methode **UpdateBatch** dient dazu, mehrere Änderungen im Recordset auf einmal an die zugrunde liegende Datenquelle zu übergeben. Diese Technik ist besonders nützlich, wenn Du im Hintergrund mehrere **AddNew**-, **Update**- oder **Delete**-Operationen ausführst und erst später gesammelt speichern willst.

Damit **UpdateBatch** verwendet werden kann, muss das Recordset im **BatchUpdate**-Modus geöffnet sein. Dafür sind zwei Voraussetzungen notwendig:

- **CursorLocation** = **adUseClient**

Automation mit VBA und Make

In vorherigen Artikeln haben wir bereits am Beispiel von Zapier gezeigt, wie wir Workflow Automation Tools nutzen können, um von VBA aus Automationen anstoßen können. Diese Automationen zielen auf Apps ab, die wir sonst über den Webbrowser steuern – GMail, CRMs, Zahlungsplattformen, Buchhaltungssoftware und viele mehr. Im vorliegenden Artikel nehmen wir einen weiteren Anbieter unter die Lupe, nämlich Make.com. Make ist grundsätzlich ähnlich zu nutzen wir Zapier, aber es bietet einen wichtigen Vorteil: Wir können damit nämlich Ergebnisse der Aufrufe der jeweiligen Anwendung sogar abfragen und diese auf unserem Client weiterverarbeiten. Wenn wir beispielsweise eine Mail über Google Mail versenden, können wir uns einen Link zu der gesendeten E-Mail zurückliefern lassen, um diese später per Mausklick im Browser zu öffnen. Oder wenn wir einen Datensatz anlegen, beispielsweise für einen neuen Kunden im CRM-Systen, können wir uns die ID des neuen Kunden zurückgeben lassen, um diesen als Referenz in der Datenbank zu speichern. In diesem Artikel zeigen wir zunächst, wie wir eine einfache Automation in Make.com anlegen und diese per VBA aufrufen und uns das Ergebnis zurückgeben lassen.

Make.com-Konto erstellen

Wenn man sich noch nicht an den Gedanken gewöhnt hat: Der Trend in der IT geht dahin, dass man für die Nutzung von Software nicht mehr einmalig zahlt, sondern Abonnements abschließt, die man für die gesamte Nutzungsdauer monatlich oder jährlich (meist mit Rabatt) abschließt. Die Zeiten, in denen man sich einmalig eine Software wie ein Office-Paket gekauft hat und dieses dann genutzt hat, bis der Support des Herstellers beendet und die Software aus Sicherheitsgründen nicht mehr genutzt werden sollte, sind allmählich vorbei.

Auch Workflow Automation Tools wie Zapier und Make.com bieten ihre Dienste nach dem gleichen Prinzip an: Man zahlt monatlich und erhält dafür ein entsprechendes Kontingent an Zugriffen.

Zum Testen reicht im Falle von Make.com das kostenlose Paket aus, hier ist man allerdings auf einen Aufruf je 15 Minuten beschränkt. Wesentlich mehr erhält man bereits für 10,59 $ im Monat bei monatlicher Ab-

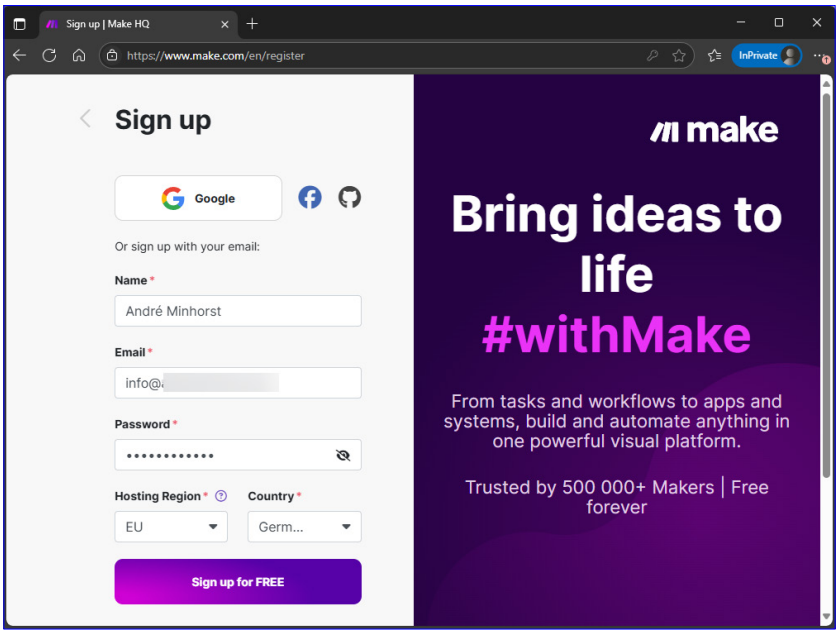


Bild 1: Erstellen eines **Make.com**-Kontos

rechnung beziehungsweise für 9,00 $ im Monat bei jährlicher Abrechnung.

Mehr Infos dazu findest Du unter folgendem Link:

```
https://www.make.com/en/pricing
```

Die Anmeldung ist schnell erledigt, nach der Angabe von Name, E-Mail und Kennwort brauchen wir nur noch den Bestätigungslink in der danach zugesendeten E-Mail zu betätigen (siehe Bild 1).

Automationen starten per Webhook

Automationen bestehen immer aus mindestens einem Trigger und einer Aktion. Der Trigger ist ein Element, das die Automation auslöst.

Meistens stammt der Trigger aus einer der Web-Anwendungen, die wir automatisieren wollen. Beispiele:

- eine neue Zeile in einem Google Sheet,
- eine neue Datei in Google Drive,
- eine neue E-Mail ist in Google Mail eingetroffen oder
- eine neue Datei wurde zu einem Dropbox-Ordner hinzugefügt.

In unserem Fall wollen wir den Trigger per VBA auslösen. Dafür findet sich logischerweise kein vorbereiteter Trigger, denn wir können VBA nicht so von Make.com aus referenzieren, dass wir dort auf ein Ereignis lauschen könnten, wie wir es mit einer Ereignisprozedur innerhalb eines VBA-Klassenmoduls tun.

Stattdessen nutzen wir einen speziellen Trigger namens Webhook. Das funktioniert so:

Make.com stellt uns eine bestimmten URL zur Verfügung, die wir per VBA aufrufen. Die URL ist einzigartig und nur uns bekannt, nachdem wir den Webhook-Trigger angelegt haben.

Diese URL rufen wir per VBA mit Unterstützung der Methoden der **XMLHTTP60**-Klasse auf.

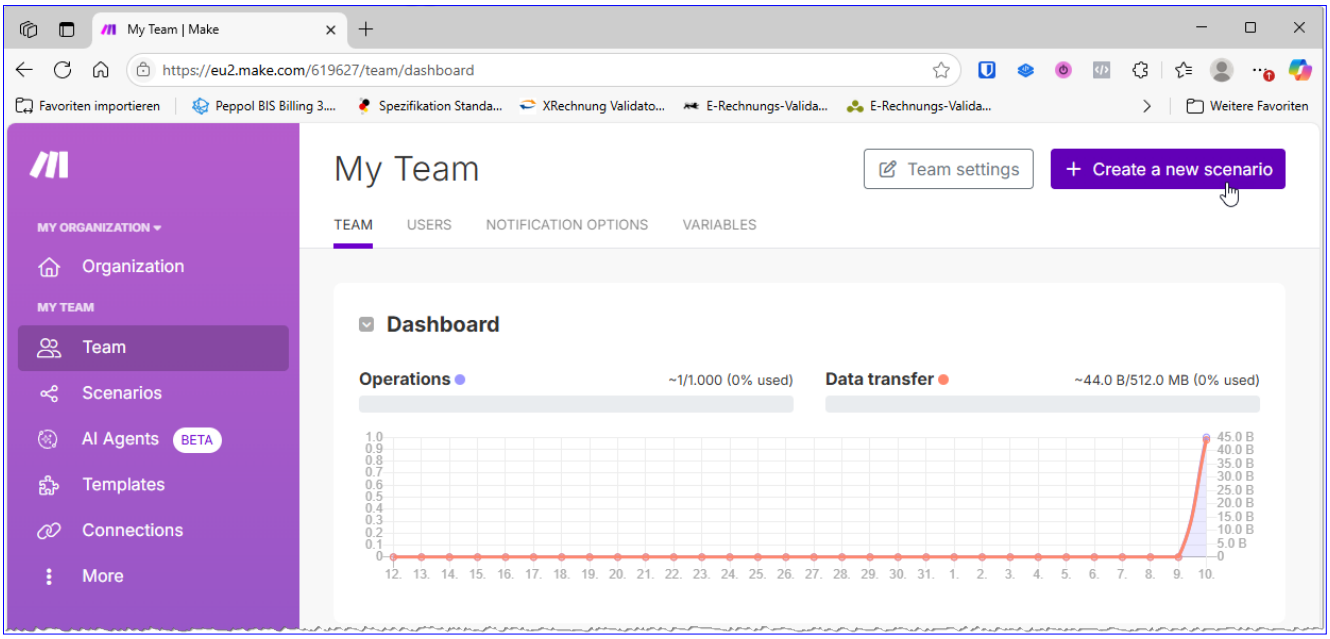


Bild 2: Anlegen eines neuen Scenarios

Mit dieser können wir alle notwendigen Informationen übergeben und auch das Ergebnis verarbeiten.

Ergebnis verarbeiten

Im Gegensatz zu Zapier bietet **Make.com** auch eine Aktion an, mit der wir das Ergebnis der Automationen an die Instanz zurückgeben können, die den Webhook aufgerufen hat. Neben den vielen anderen Aktionen, die wir durchführen können, finden wir also auch eine namens **Webhook response**. Diese soll im folgenden grundlegenden Beispiel auch die einzige Aktion sein, die wir in unserem Basisworkflow nutzen wollen.

Ein Scenario erstellen

Was unter Zapier **Zap** genannt wird, heißt unter Make.com **Scenario**. Von der Startseite können wir entweder direkt auf **Create a new scenario** klicken oder wir klicken links auf **Scenarios**, um die Übersicht aller bisher erstellten Automationen einzusehen (siehe Bild 2). Auch dort finden wir wieder die Schaltfläche **Create a new scenario**.

Trigger zum Scenario hinzufügen

Damit landen wir im Scenario-Designer. Hier sehen wir ein großes Plus-Icon und ein Menü mit der Übersicht der verfügbaren Trigger (siehe Bild 3).

Es lohnt sich, einmal durch die Liste zu scrollen und sich anzuschauen, durch welche Anwendungen eine solche Automation gestartet werden kann.

Klicken wir auf einen der Einträge, erscheint eine weitere Liste mit allen Ereignissen, die durch diese Anwendungen ausgelöst werden können. Im Falle von

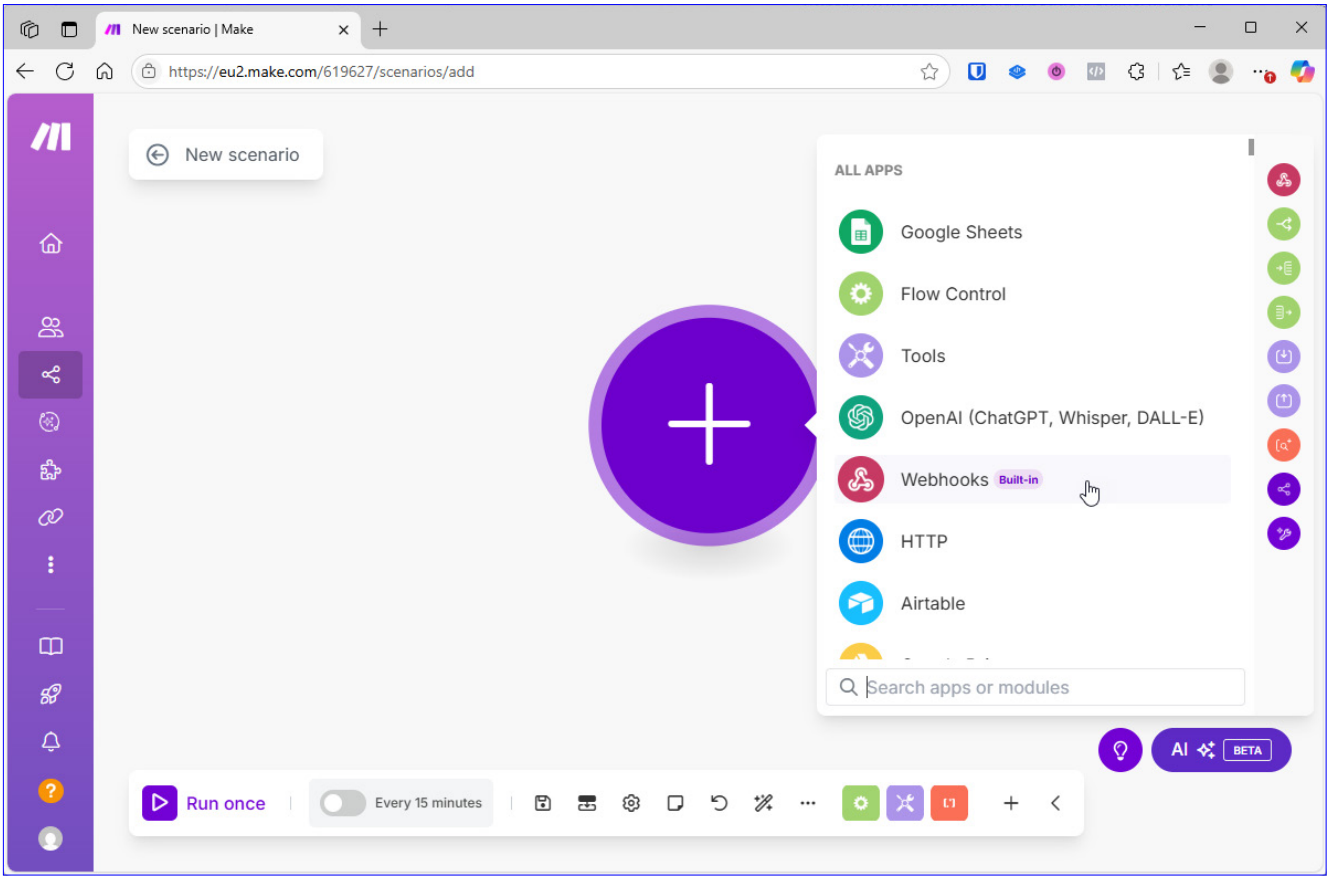


Bild 3: Ein noch leeres neues Scenario

Google Sheets sind das beispielsweise neu hinzugefügte Zeilen oder geänderte Zeilen (siehe Bild 4).

Webhook als Trigger verwenden

Wir wollen jedoch keine externe Anwendung als Trigger nutzen, sondern den Trigger per VBA von unserem VBA-Projekt aus auslösen.

Dazu kehren wir zur Hauptliste der Anwendungen zurück, die Trigger auslösen können, und klicken hier auf den Eintrag **Webhook**.

Dies liefert die drei Elemente aus Bild 5, wobei die ersten beiden Trigger sind und die dritte eine Aktion. Die beiden Trigger funktionieren wie folgt:

- **Custom mailhook**: Wenn wir einen Mailhook erstellen, erhalten wir eine E-Mail-Adresse. An diese E-Mail-Adresse können wir eine E-Mail senden. Kommt diese bei **Make.com** an, löst dies die Automation aus. In den nachfolgend zu definierenden Aktionen können wir auf die Eigenschaften der E-Mail wie Betreff, Inhalt, Datum, Absender et cetera zugreifen und diese weiterverarbeiten.

- **Custom webhook**: Erstellen wir einen **Custom webhook**, erhalten wir hingegen eine URL. Der Trigger wird ausgelöst, wenn wir diese URL aufrufen. Im einfachsten Fall geben wir diese einfach im Webbrowser ein.

Die Aktion **Webhook response** stellt die entscheidende Besonderheit von **Make.com** gegenüber Zapier dar: Darin finden wir eine Möglichkeit, das Ergebnis einer durch einen Webhook aufgerufenen Automation auch wieder an die aufrufende Instanz zurückzugeben.

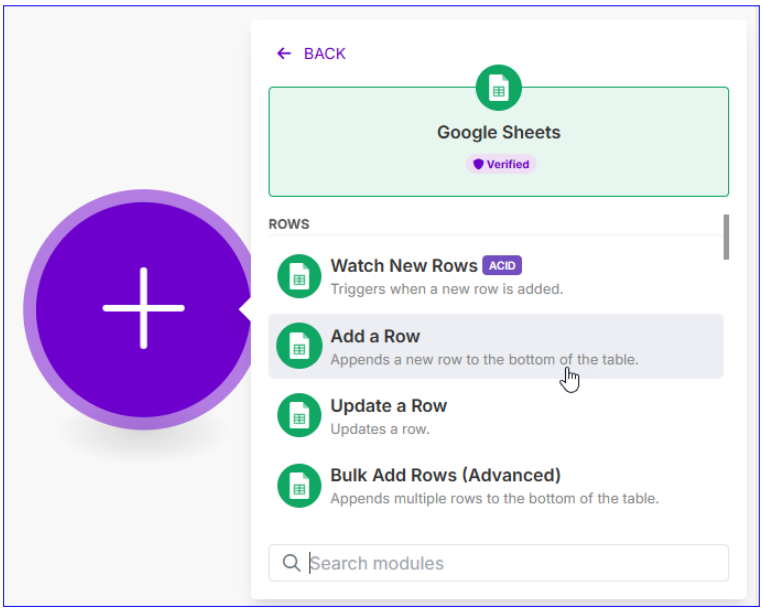


Bild 4: Betrachten der verschiedenen Trigger einer Anwendung

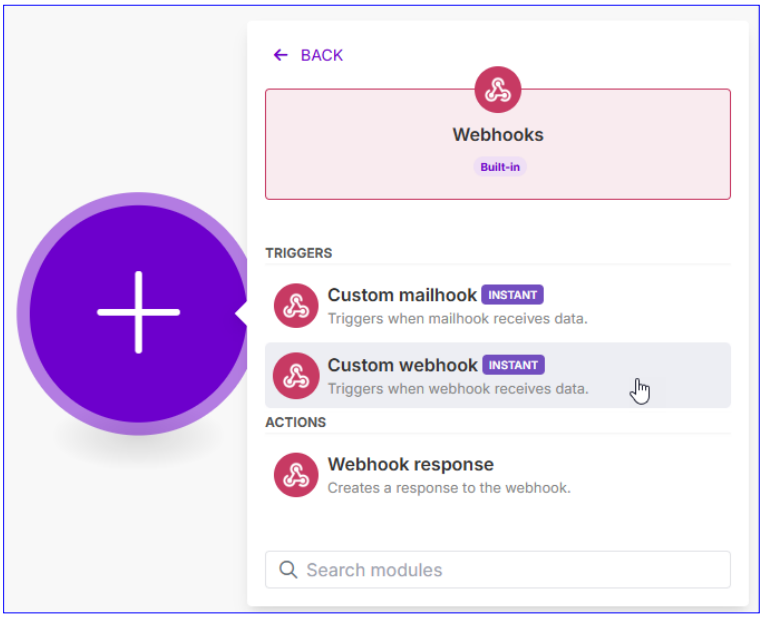


Bild 5: Hinzufügen eines Triggers namens **Custom webhook**

Trigger anlegen

Um den Trigger anzulegen, klicken wir auf **Custom webhook**.

Dies öffnet ein Popup, indem wir einen bestehenden Webhook auswählen oder einen neuen anlegen können (siehe Bild 6).

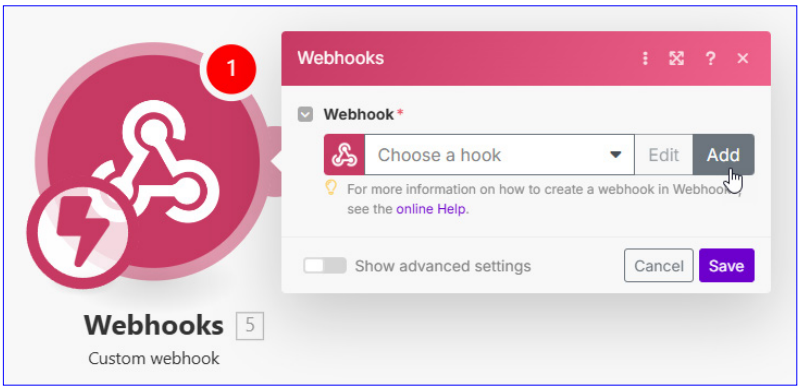


Bild 6: Anlegen und Konfigurieren des Webhooks

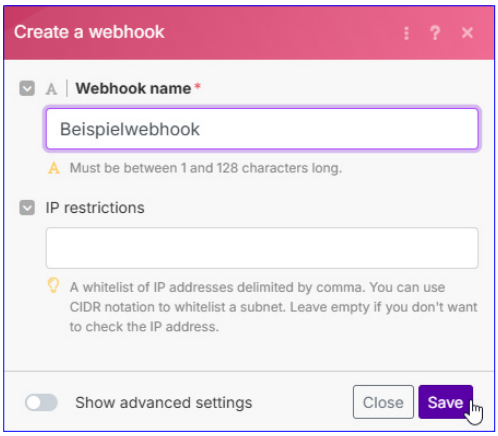


Bild 7: Einstellen des Namens für den Webhook

Wir benötigen hier einen neuen Hook und klicken auf die Schaltfläche **Add**.

Nun erscheint der Dialog **Create a webhook**, in dem wir den Namen für unseren Webhook eintragen – in diesem Fall **Beispielwebhook** (siehe Bild 7).

Danach enthält der Webhook weitere Informationen, zum Beispiel eine URL (siehe Bild 8). Diese kopieren wir mit einem Klick auf **Copy address to clipboard** in die Zwischenablage.

An der Schaltfläche **Stop** erkennen wir, dass der Webhook bereits aktiv ist und auf Aufrufe wartet.

Erster Test des Webhooks

Damit können wir den Webhook nun bereits testen – ohne größeren Aufwand wie beispielsweise eine VBA-Prozedur. Wir fügen die Adresse einfach in einen Browser ein und rufen diese auf. Das Ergebnis ist der Text **Accepted** (siehe Bild 9). Das sieht sehr gut aus – der Aufruf funktioniert.

Damit wollen wir den Webhook nun speichern und klicken dazu auf die Schaltfläche **Save**.

Antwort benutzerdefiniert gestalten

Damit kommen wir zum zweiten Schritt unseres Beispiels. Hier wollen wir nun die Antwort **Accepted** durch eine benutzerdefinierte Antwort ersetzen. Dazu fügen wir eine Aktion hinzu, indem wir auf die kleine **Plus**-Schaltfläche rechts von unserem Trigger klicken.

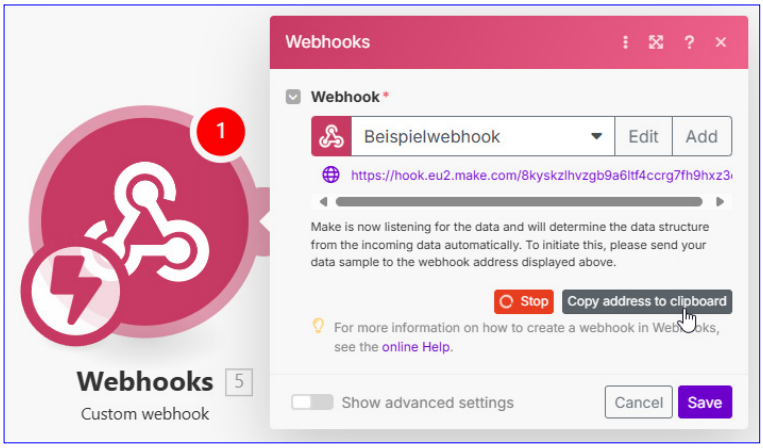


Bild 8: Details des Webhooks

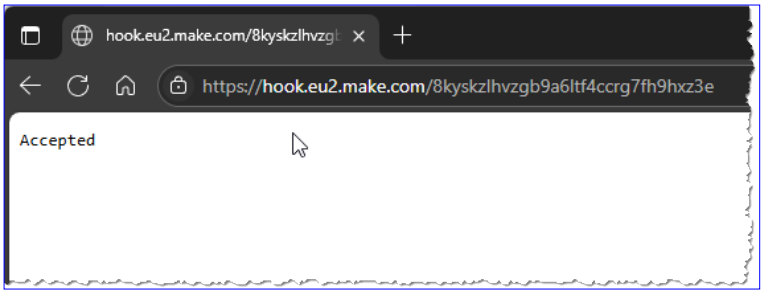


Bild 9: Aufruf des Webhooks über den Webbrowser

Dies öffnet die gleiche Auswahl wie zuvor (siehe Bild 10). Klicken wir hier erneut auf **Webhooks**, finden wir lediglich noch den Eintrag **Webhook response** vor. Logisch, denn wir können an dieser Stelle keinen weiteren Trigger hinzufügen – wir benötigen eine Aktion. Also wählen wir **Webhook response** aus.

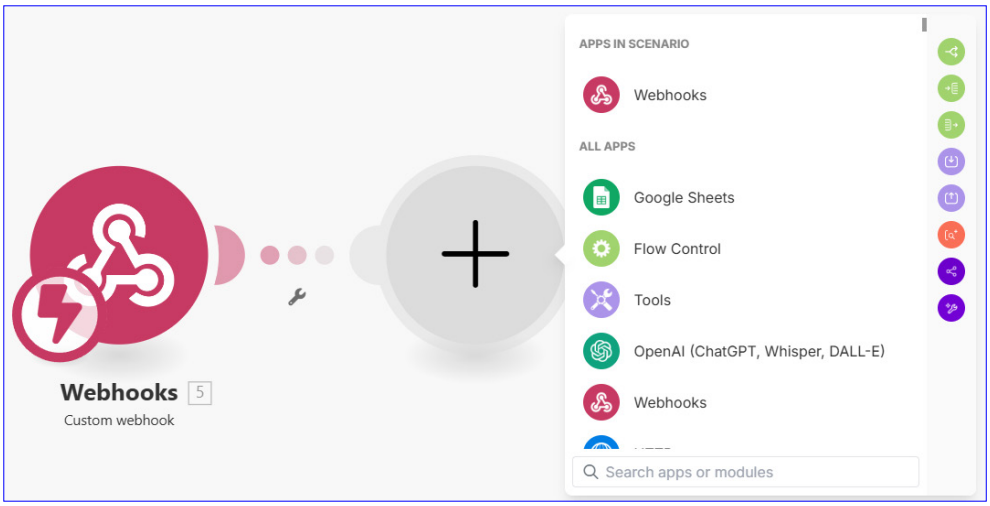


Bild 10: Hinzufügen einer Aktion zum Workflow

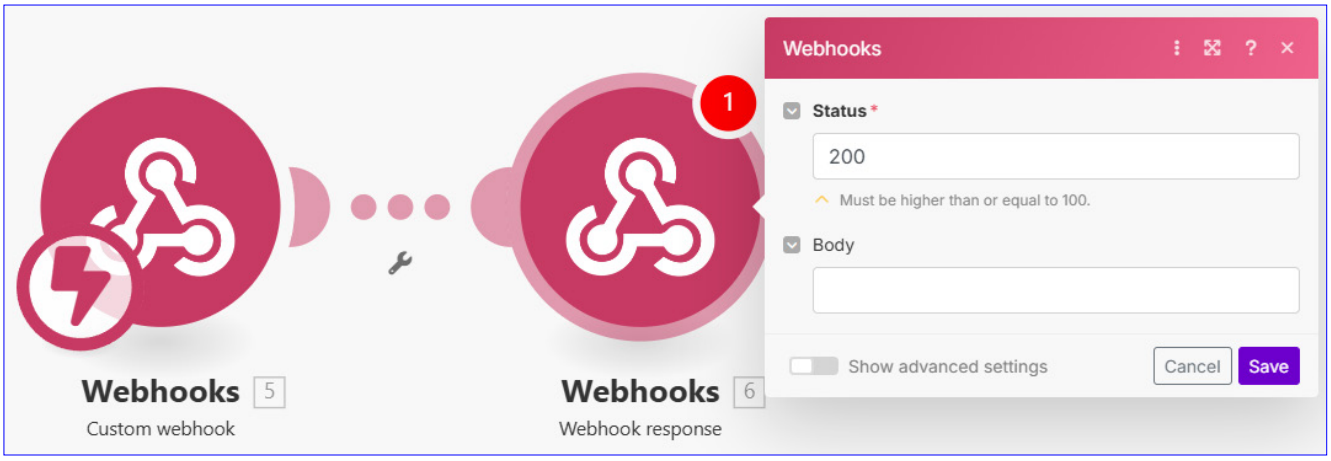


Bild 11: Die neue Aktion im Scenario

Dies fügt das Element **Webhook response** hinzu und zeigt gleich seine Eigenschaften an (siehe Bild 11).

Wenn wir das Feld **Body** anklicken, erscheint ein weiteres Popup.

Hier sehen wir zum Beispiel die Variable **executionId**, welche eine ID für

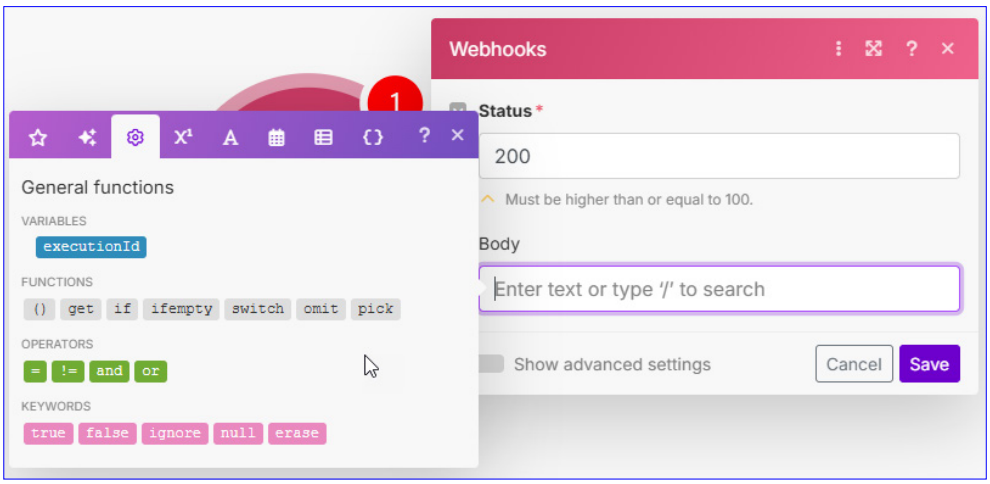


Bild 12: Verfügbare Elemente für die Antwort

die aktuelle Ausführung der Automation enthält (siehe Bild 12).

Wir tragen testweise einfach einmal den Text **Benutzerdefinierte Antwort**, gefolgt von der Variablen **executeId** in das Feld **Body** ein. **executeId** ziehen wir dazu per Drag and Drop aus der Liste in das Feld **Body**, sodass wir das Ergebnis aus Bild 13 erhalten.

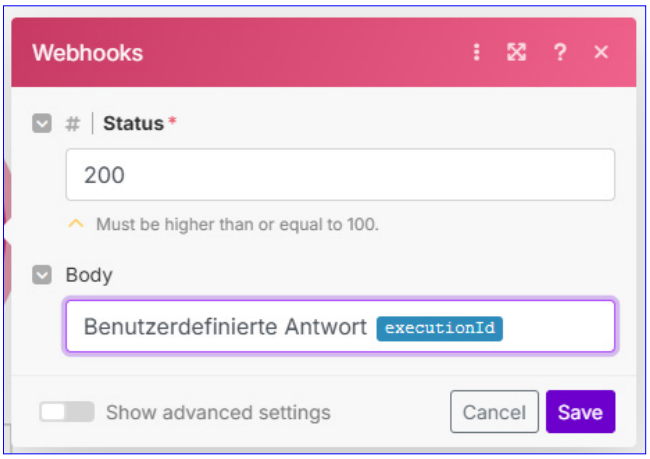


Bild 13: Hinzufüge nder **executionId**

Mit einem Klick auf die Schaltfläche **Save** speichern wir das neue Element.

Nun lauscht der Webhook allerdings nicht mehr auf unsere Anfragen, sodass wir im Browser zunächst diese Antwort erhalten:

```
There is no scenario listening for this webhook.
```

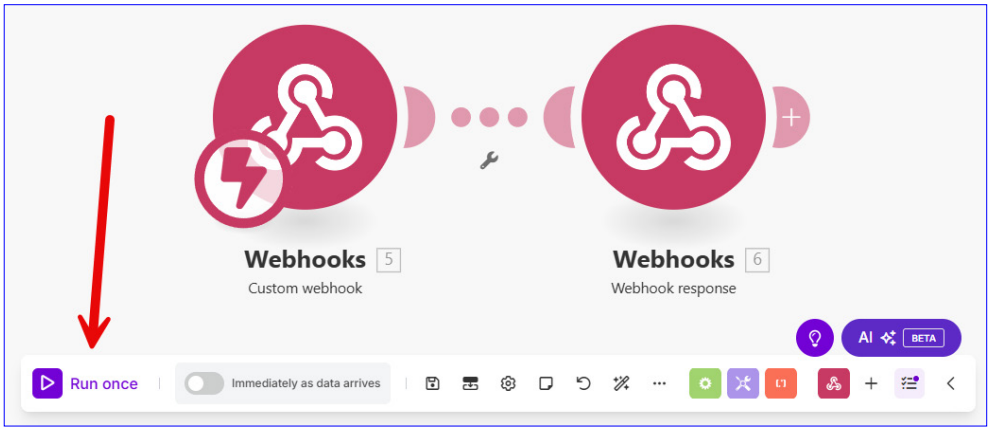


Bild 14: Starten der Automation

Deshalb klicken wir nun auf die Schaltfläche **Run once** (siehe Bild 14). Die Schaltfläche wandelt sich in eine **Stop**-Schaltfläche um und die Automation lauscht wieder auf Aufrufe.

Rufen wir die URL des Webhooks nun erneut im Browser auf, erhalten wir das Ergebnis aus Bild 15. Die Rückgabe einer individuell in der Automation festgelegten Antwort funktioniert also.

Nächste Schritte

Damit haben wir das Grundgerüst geschaffen. Wir werden uns nun ansehen, wie wir die Automation per VBA starten können. Das Ziel ist es, die Automation zu starten und das Ergebnis, dass wir nun im Browser erhalten haben, per VBA auszulesen und zu verarbeiten.

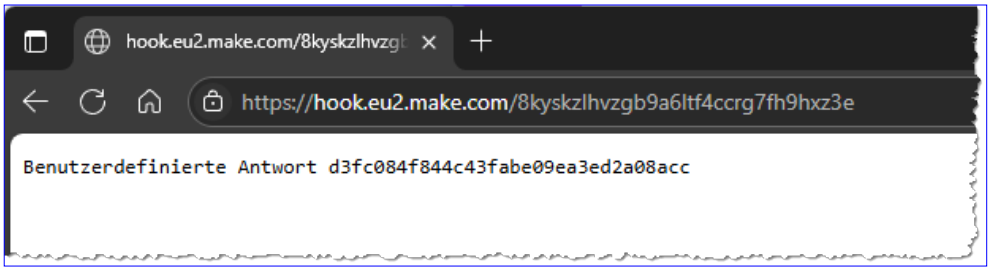


Bild 15: Erfolgreicher Test der Automation

Make.com-Automation per VBA aufrufen

Um die Automation per VBA zu starten und das Ergebnis zu erhalten, benötigen wir ein VBA-Projekt, für

Microsoft 365-Mail mit Make und VBA ohne Outlook

Am liebsten wäre den meisten Microsoft-Anwendern vermutlich, wenn sie einfach ihre vorhandenen VBA-Automationen auf Basis von Microsoft Outlook verwenden könnten. Allerdings zeigt Microsoft aktuell kein Zeichen, dass das klassische Outlook für Desktop mit seiner VBA-Programmierbarkeit über das Jahr 2029 hinaus unterstützt werden wird. Die neue Outlook-Version ist allerdings noch nicht so weit, dass wir damit so arbeiten können, wie es mit der klassischen Version möglich ist. Genau genommen wird das, wenn man sich die Pläne von Microsoft ansieht, auch zumindest per VBA nicht mehr möglich sein. Um zumindest das Versenden von E-Mails von VBA aus zu realisieren, benötigen wir also eine Alternative. In diesem Artikel schauen wir uns an, wie wir Outlook.com über den Automatisierungdienst Make dazu bringen, E-Mails zu versenden – mit allen Funktionen, die wir von Outlook gewohnt sind.

Microsoft Access und Microsoft Outlook in der klassischen Version sind ein großartiges Gespann. Wer von Access aus per VBA eine E-Mail versenden wollte, konnte dies mit wenigen Zeilen Code realisieren. Outlook starten (wenn auch unsichtbar), ein neues **MailItem**-Objekt erstellen, Betreff, Inhalt und gegebenenfalls noch Anlagen und weitere Eigenschaften einstellen und los geht es. Die so vorbereitete E-Mail können wir mit der **Send**-Methode direkt absenden oder mit der **Display**-Methode so öffnen, als ob wir die E-Mail soeben selbst erstellt hätten – und brauchen dann nur noch auf **Senden** zu klicken, nachdem wir den Entwurf gegebenenfalls noch angepasst haben.

In Zukunft können wir verschiedene Alternativen nutzen:

- E-Mails unter Versendung der Bibliothek CDO versenden: Das ist jedoch aufwendig, weil sich die Parameter je nach dem zu verwendenden E-Mail-Anbieter unterscheiden. Außerdem ist auch diese Bibliothek bereits abgekündigt.

- E-Mails mit der Graph-API von Microsoft versenden: Dies haben wir uns bereits in einem Artikel in Access im Unternehmen angesehen – **Mails senden mit der Microsoft Graph API** (**www.access-im-unternehmen.de/1532**). Allerdings ist es sehr aufwendig, dies zu realisieren, da sehr viel Vorbereitung notwendig ist – speziell für die Authentifizierung am entsprechenden Outlook-Konto.

- Office-JavaScript-API: Es soll auch für das neue Outlook eine API geben. Allerdings ist hier noch unklar, ob und wie man damit Mails mit VBA erstellen und verschicken kann.

Make.com als Schnittstelle

Also zeigen wir, wie wir mit alternativen Techniken E-Mails über Outlook.com versenden können – und dazu mit Make.com ein Tool zur Workflow-Automation nutzen.

Im Artikel **Automation mit VBA und Make** (**www.vbentwickler.de/463**) haben wir bereits gezeigt, wie wir Make.com von VBA aus ansteuern können. Dort haben wir uns angesehen, wie man ein Automations-Scenario anlegen, wie wir dieses von VBA aus starten und wie das Ergebnis der Automation abgefragt und ausgewertet werden kann.

Auf den dort beschriebenen Techniken setzen wir nun auf und wollen darin den Versand einer E-Mail mit Outlook.com einbetten.

Warum aber sollten wir dazu **Make.com** nutzen und nicht über die Microsoft Graph API auf Outlook.com zugreifen? Weil auch hier die Authentifizierung relativ aufwendig zu realisieren ist und **Make.com** uns dabei eine Menge Arbeit abnimmt.

Wir gehen dabei in den folgenden Schritten vor:

- Programmierung des Mailversands in Make.com
- Aufruf des Mailversands in eine VBA-Prozedur

Mails versenden in Make.com

Zunächst programmieren wir einen Mailversand in **Make.com**, ohne dass wir Betreff, Inhalt, Empfänger und so weiter von VBA aus übergeben.

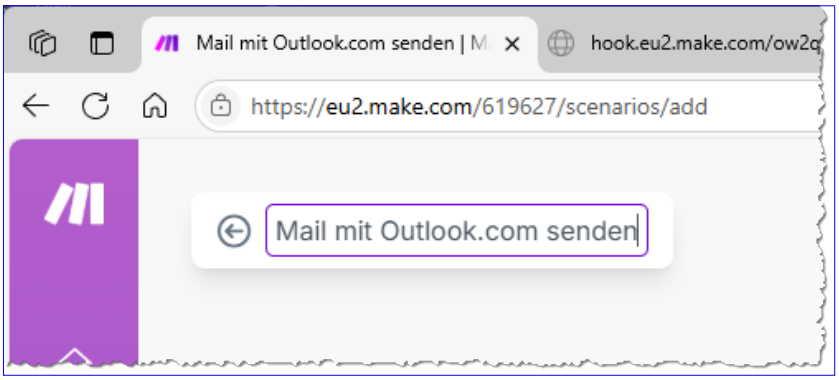


Bild 1: Scenario umbenennen

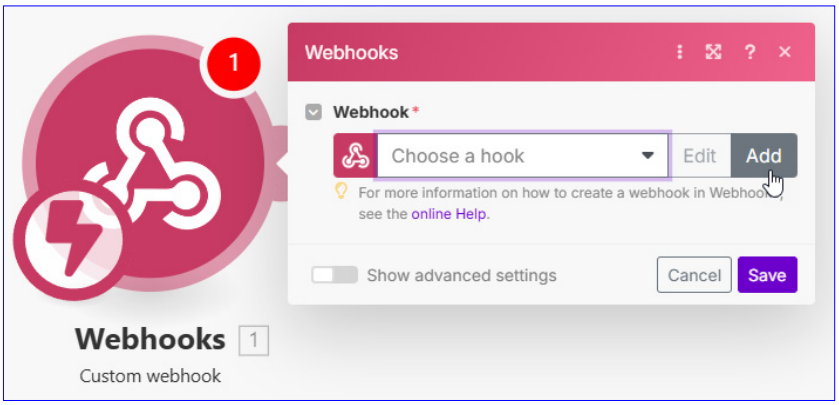


Bild 2: Trigger hinzufügen und Hook anlegen

Wie wir ein **Make.com**-Konto anlegen und wie wir grundsätzlich Scenarios anlegen und damit arbeiten, haben wir im oben genannten Artikel beschrieben.

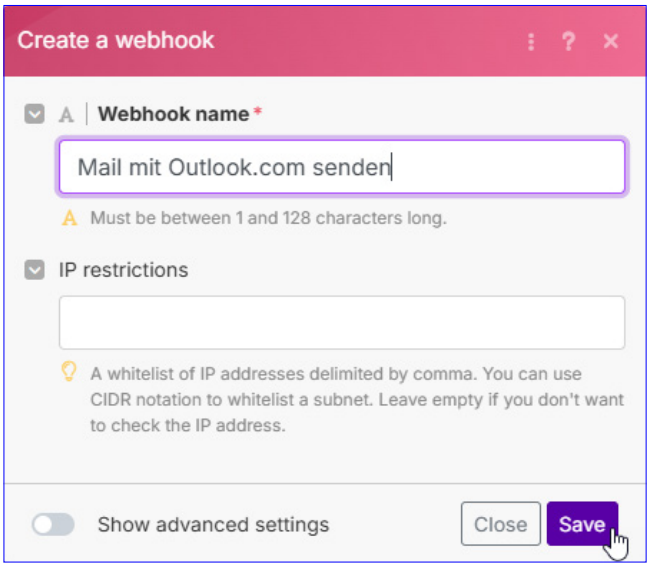


Bild 3: Name des Webhooks festlegen

Dazu legen wir in **Make.com** ein neues Scenario an, indem wir auf der Seite **All scenarios** auf **Create a new scenario** klicken. Im neuen Scenario ändern wir den Namen gleich oben links auf **Mail mit Outlook.com senden** (siehe Bild 1).

Danach fügen wir den Trigger ein. Wir wollen den Mailversand per VBA anstoßen, wozu wir ein **XMLHTTP60**-Objekt nutzen werden.

Dazu benötigen wir ein **Webhook**-Element als Trigger, das wir nun hinzufügen. Dazu klicken wir auf das große Plus-Icon im Scenario und wählen **Webhook|Custom webhook** aus. Hier klicken wir nun auf **Add**, um einen neuen Hook hinzuzufügen (siehe Bild 2).

Anschließend legen wir den Namen des Webhooks fest und speichern diesen mit **Save** (siehe Bild 3).

Die URL des Webhooks speichern wir in der Zwischenablage und fügen diese dann in einem Browser ein. Nach dem Betätigen der Eingabetaste sollte der Text **Accepted** erscheinen.

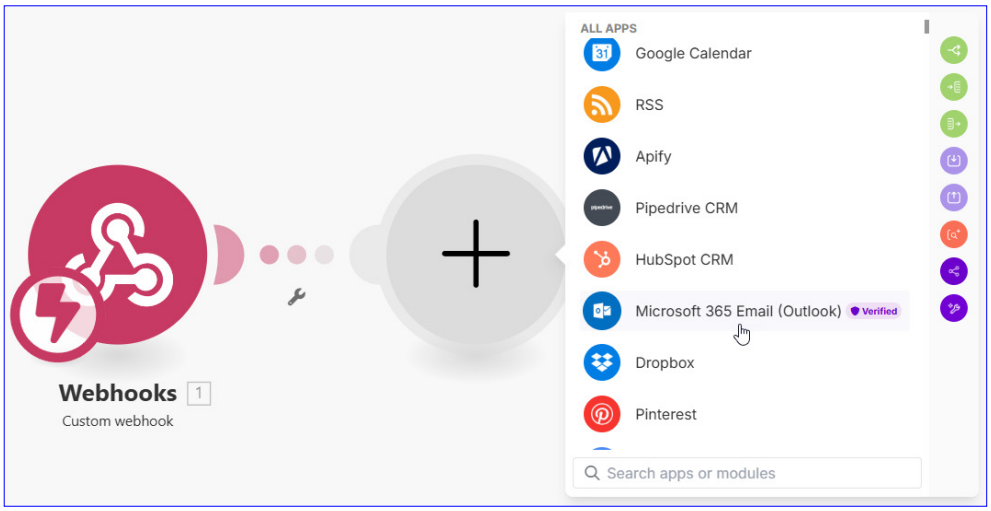


Bild 4: Microsoft 365 E-Mail (Outlook)-Aktion hinzufügen

Das erste Element des Scenarios können wir damit speichern.

Aktion zum Versenden von E-Mails hinzufügen

Nun klicken wir auf das Plus-Zeichen rechts vom Trigger-Element. Im Popup suchen wir den Eintrag **Microsoft 365 Email (Outlook)** (siehe Bild 4).

Nach dem Anklicken erscheinen alle Aktionen, die diese App bereitstellt. Wir sehen gleich, dass wir eine ganze Menge Aktionen durchführen können, die mit E-Mails in Microsoft 365 Mail zu tun haben.

Uns interessiert jedoch zunächst die Aktion **Create and Send a Message** (siehe Bild 5).

Verbindung zu Microsoft 365 herstellen

Klicken wir diese an, wird diese Aktion angelegt und es erscheint ein Popup zur Auswahl oder zum Anlegen der Connection zu **Microsoft 365** (siehe Bild 6).

An dieser Stelle gehen wir davon aus, dass Du bereits ein Microsoft 365-Konto hast. Da dieser vermutlich noch nicht als Connection zu **Make.com** hinzugefügt ist, klicken wir hier auf die Schaltfläche **Add**.

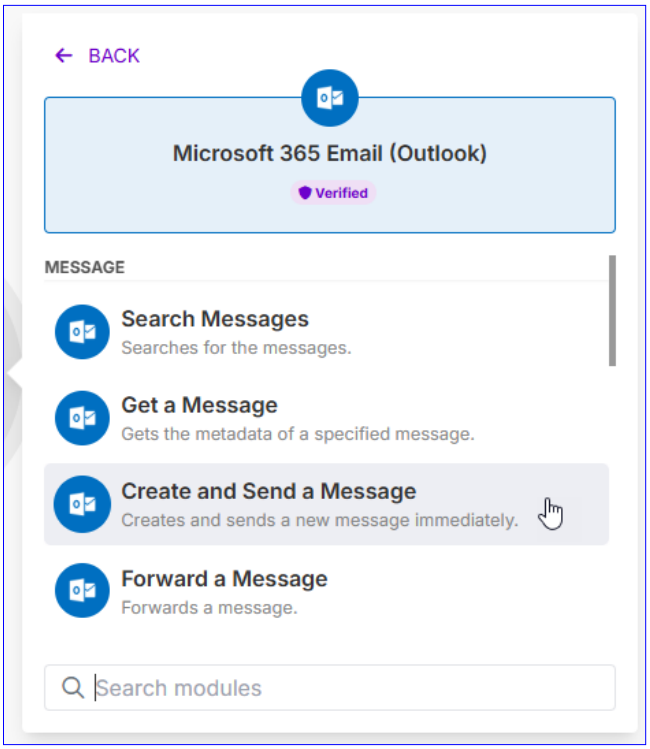


Bild 5: Aktion zum Versenden einer E-Mail auswählen

Im nun erscheinenden Popup geben wir eine Bezeichnung für die zu verwendende Connection an (siehe Bild 7). Eine Bezeichnung ist sinnvoll, weil wir gegebenenfalls mehrere Microsoft 365-Konten haben. Wenn wir diese noch in anderen Scenarios für den Versand von E-Mails nutzen wollen, finden wir die Connec-

tion schnell wieder, wenn wir einen aussagekräftigen Namen vergeben haben. Wir verwenden hier zum Beispiel **Microsoft 365-Connection**.

Danach klicken wir unten auf **Save**.

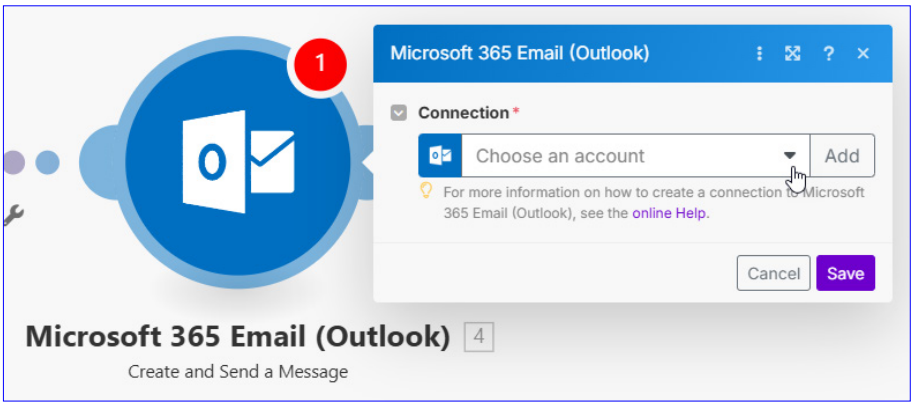


Bild 6: Auswählen oder Hinzufügen der Connection

Es erscheint nun ein Microsoft-Dialog, wo wir nach der Authentifizierung die Berechtigungen von Make für den Zugriff auf unser Microsoft-Konto akzeptieren müssen (siehe Bild 8).

Eigenschaften der E-Mail einstellen

Schließlich landen wir in dem Dialog **Microsoft 365 Email (Outlook)** aus Bild 9, in dem die zuvor gewählte Connection bereits angezeigt wird.

Hier finden wir die üblichen Felder für das Verwenden einer E-Mail.

Also tragen wir Werte in einige dieser Felder ein, zum Beispiel in **Subject**, **Body Content** und **To Recipients**.

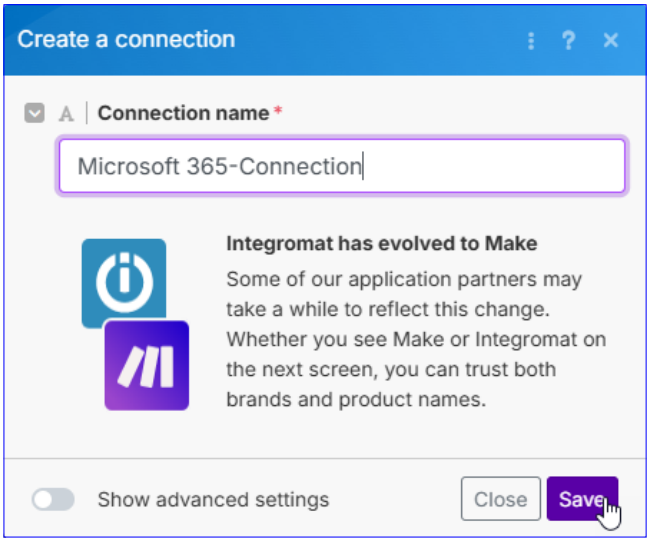


Bild 7: Angeben einer Bezeichnung für die Connection

Noch sind diese Werte statisch, später werden wir diese aus dem per VBA abgesetzten Aufruf entnehmen. Wir wollen allerdings erst einmal ausprobieren, ob das Versenden überhaupt funktioniert.

Bild 8: Akzeptieren der Berechtigungen

Nach dem Eingeben der Eigenschaften klicken wir auf die Schaltfläche **Save**.

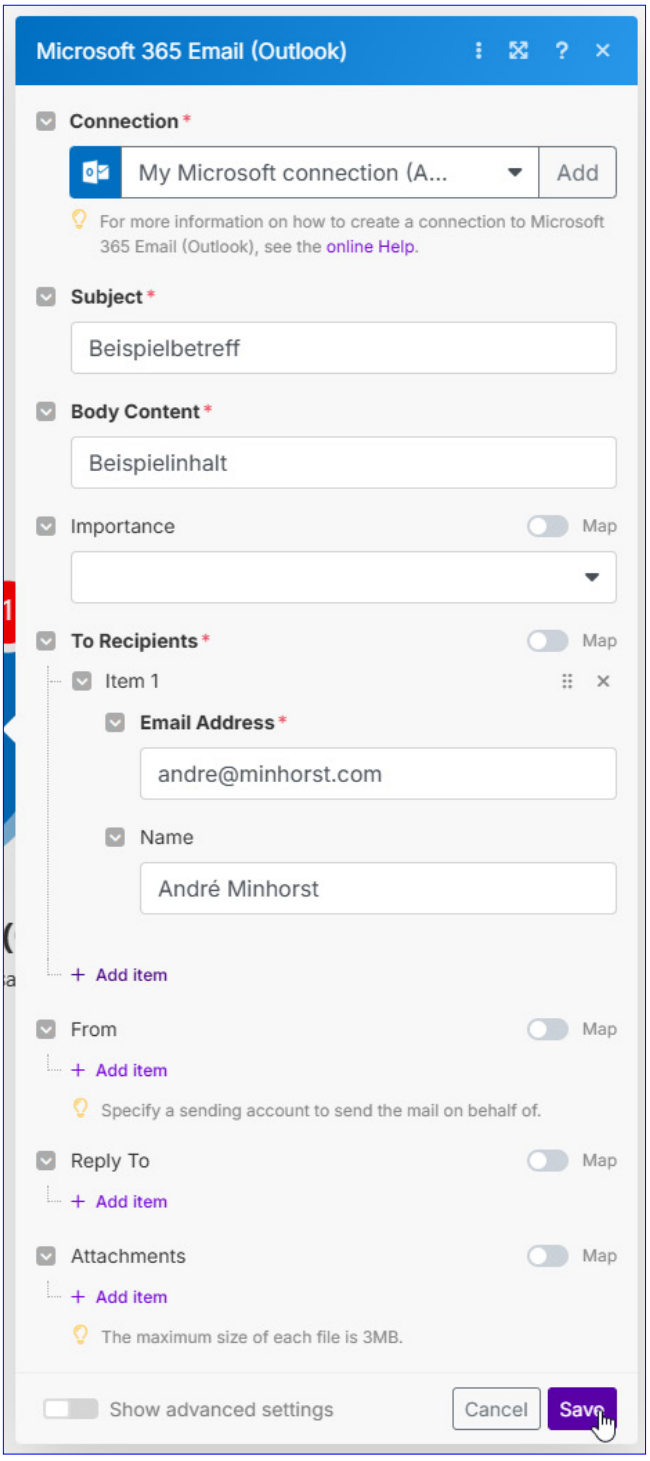


Bild 9: Anlegen einer Beispiel-E-Mail

Antwort-Element hinzufügen?

Im Beispiel aus dem oben genannten Artikel haben wir noch ein **Response**-Element angehängt, mit dem wir eine entsprechende Meldung zurückgegeben haben. Hier lohnt es sich eigentlich nicht, ein solches Element einzufügen – wenn wir die Connection korrekt aufgesetzt haben und die E-Mail-Daten korrekt gefüllt haben, sollten keine Fehler auftreten.

Das kann jedoch dennoch der Fall sein, zum Beispiel wenn wir extern die Berechtigung für Make für den Zugriff auf Microsoft 365 entziehen. Dann sollte beim Versuch, im Kontext unserer Connection auf Microsoft 365 zuzugreifen, der Status 401 etc. zurückgeliefert werden.

Leider können wir diesen Status aber nicht auslesen, da dieser vom Modul **Create and Send a Message** nicht weitergegeben wird.

Fehlerbehandlung einbauen

Allerdings gibt es auch in Make eine Fehlerbehandlung. Diese aktivieren wir, indem wir mit der rechten

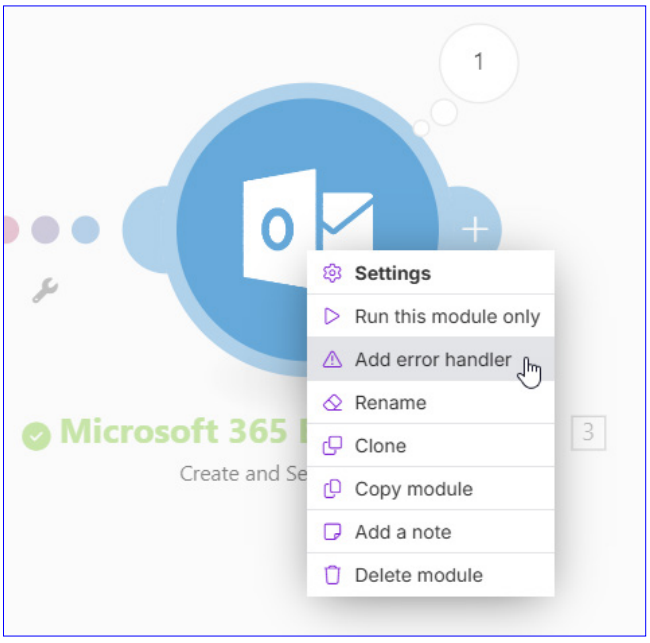


Bild 10: Hinzufügen einer Fehlerbehandlung

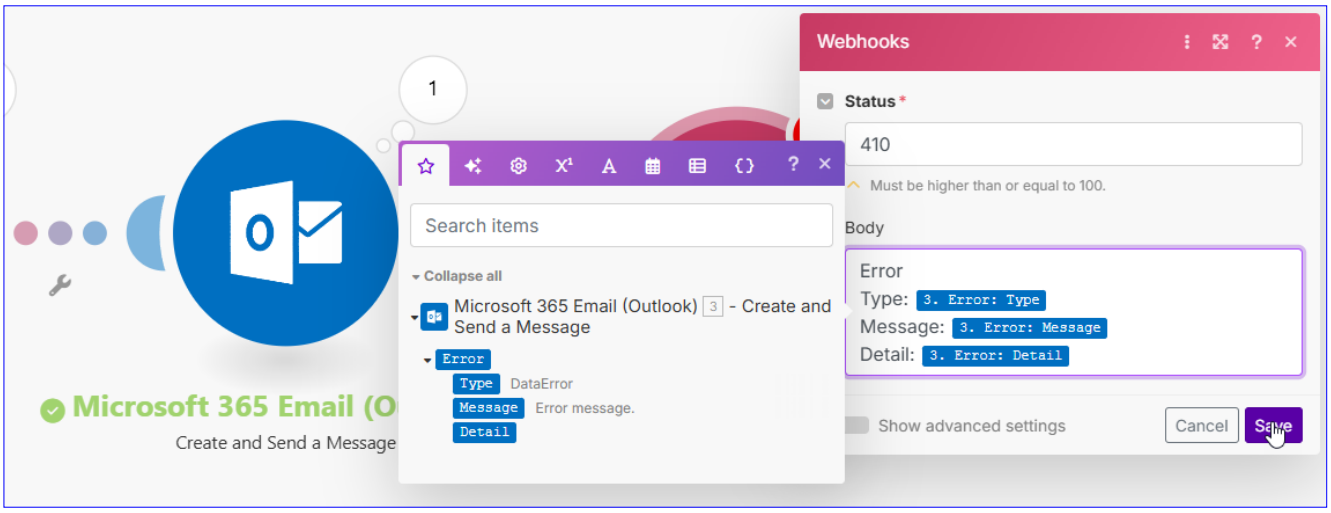


Bild 11: Hinzufügen einer Error-Response

Maustaste auf das Modul **Create and Send a Message klicken** und dort den Befehl **Add error handler** aufrufen (siehe Bild 10).

Es erscheint ein neues Modul mit dem Plus-Zeichen, für das wir das Element **Webhooks|Webhook response** einfügen. Hier klicken wir in das Feld **Body** und fügen die Elemente aus der Auswahl wie in Bild 11 hinzu. Außerdem stellen wir Status auf den Wert **410** ein, der sonst normalerweise nicht verwendet wird.

Response im Erfolgsfall einbauen

Allerdings wollen wir auch noch eine Response einbauen, die beim erfolgreichen Mailversand ausgelöst wird und ein entsprechendes Ergebnis zurückliefert.

Dazu klicken wir im **Create and Send a Message**-Modul auf das Pluszeichen rechts. Hier fügen wir ein weiteres Element des Typs **Webhooks|Webhook response** hinzu. Diesmal stellen wir **Status** auf **201** und **Body** auf **Erfolgreich versendet** ein.

Außerdem benennen wir die beiden **Response**-Elemente auch direkt um in **Webhook Success** und **Webhook Error**.

Anschließend sieht unser Scenario wie in Bild 12 aus.

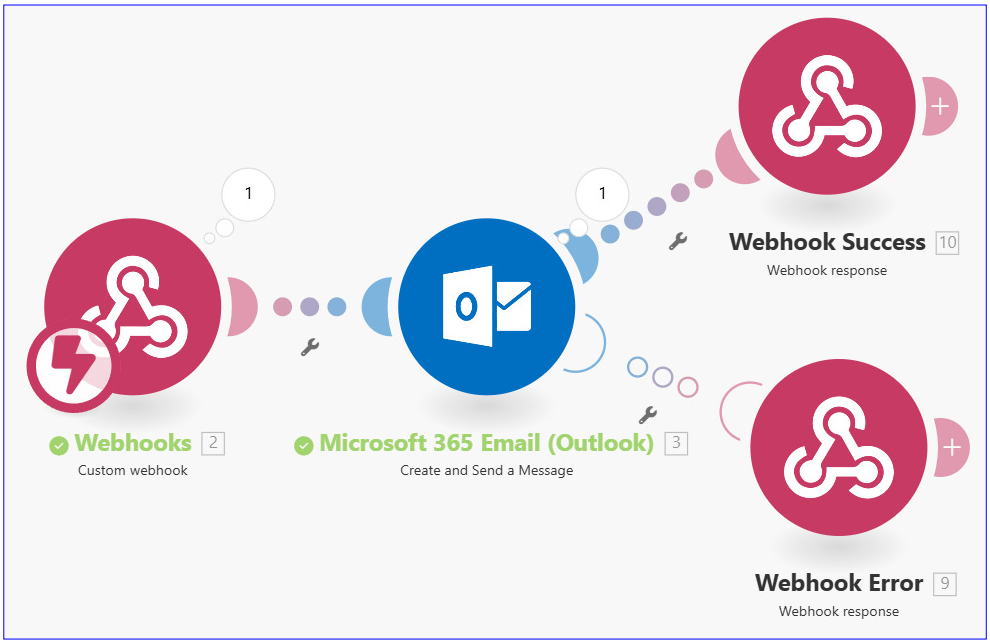


Bild 12: Die Automation im Überblick

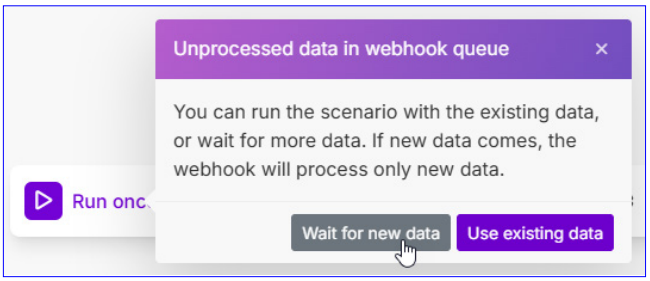


Bild 13: **Run once** sorgt für das Lauschen auf einen Aufruf

Ausführen des Mailversands

Bevor wir die Automation per VBA aufrufen wollen, testen wir diese einmal. Dazu klicken wir unten links im Make-Fenster in der Menüleiste auf **Run once** (siehe Bild 13) und dann auf **Wait for new data**.

Dadurch lauscht die Automation auf einen Aufruf. Den führen wir aus, indem wir die Prozedur aus Listing 1 in unserer Access-Anwendung ausführen.

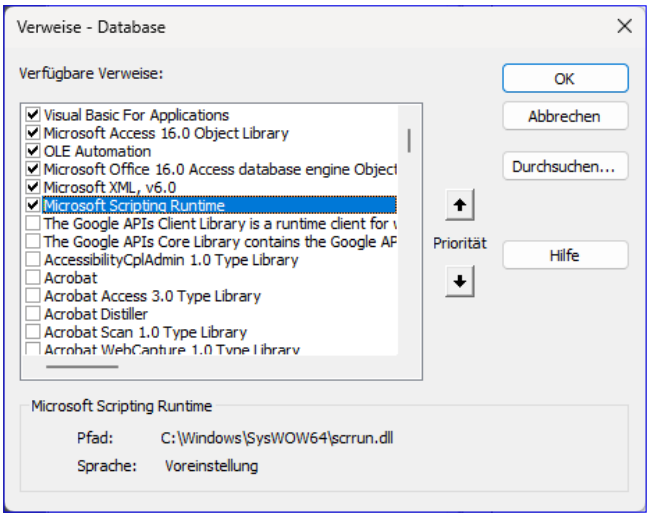


Bild 14: Verweise für das VBA-Projekt

Diese erfordert das Vorhandensein eines Verweises auf die Bibliothek **Microsoft XML, v6.0**. Im gleichen Zuge fügen wir auch noch einen Verweis auf die Bibliothek **Microsoft Scripting Runtime** hinzu (siehe Bild 14).

```
Public Sub Microsoft365_MailVersenden()
    Dim strWebhookURL As String
    Dim objXMLHTTP As MSXML2.XMLHTTP60
    Dim strJSON As String
    strWebhookURL = "https://hook.eu2.make.com/5yauwb5ehk63mudiohao3xf4mkc7lwe2"
    strJSON = ""
    Set objXMLHTTP = New MSXML2.XMLHTTP60
    objXMLHTTP.Open "POST", strWebhookURL, False
    objXMLHTTP.setRequestHeader "Content-Type", "application/json"

    objXMLHTTP.send strJSON

    Select Case objXMLHTTP.status
        Case 201
            MsgBox "Erfolgreich gesendet."
        Case 200
            MsgBox "Erfolgreich aufgerufen, aber nicht gesendet."
       Case 400
            MsgBox "JSON ungültig."
        Case Else
            MsgBox "Fehler beim Senden: " & objXMLHTTP.status & " - " & objXMLHTTP.statusText
    End Select
End Sub
```

Listing 1: Die Prozedur führt einen ersten Aufruf unserer Automation aus.

Microsoft 365 E-Mails mit Make per Klasse senden

In unserem Artikel Google Mail mit Make.com und VBA statt Outlook (www.vbentwickler.de/464) haben wir detailliert gezeigt, wie man per VBA die Informationen zum Versenden einer E-Mail zusammenstellt und diese dann über den Automatisierungsdienst Make.com an Microsoft 365 E-Mail sendet. Hier sind wir allerdings nur so weit gekommen, dass wir die notwendige JSON-Datei mit den eigentlichen Daten der E-Mail Zeile für Zeile zusammengestellt und mit den nötigen Daten gefüllt haben. Für einen ersten Schritt nicht schlecht, aber wenn man diesen Weg professionell nutzen möchte, sollte man sich dazu eine ordentliche Klasse bauen. Eine, die man schnell initialisieren und mit den Daten wie Empfänger, Betreff, Inhalt, CC/BCC-Empfängern und Anlagen füllen kann und die eine einfache Send-Methode enthält. Ganz genau so, wie es auch mit VBA und Outlook geht – mit dem Unterschied, dass die hier vorgestellte Methode vermutlich wesentlich bessere Chancen hat, die nächsten fünf Jahre zu überleben.

Das Versenden von E-Mails per VBA könnte so einfach sein. Dazu benötigen wir lediglich eine Prozedur wie die aus Listing 1.

Hier erstellen wir ein neues **clsMail**-Objekt und füllen es mit den für eine E-Mail benötigten Daten wie Empfänger, Betreff, Inhalt und Anlagen. Ein Aufruf der **Send**-Methode erledigt den Rest und die E-Mail landet im Postfach des angegebenen Empfängers.

Dazu brauchen wir allerdings noch ein wenig mehr – aber das ist nur einmal einzurichten und läuft dann, gegebenenfalls auch in all Deinen Anwendungen aufrufbar. Was wir noch benötigen:

```
Public Sub MailVersenden()
    Dim objMail As clsMail
    Dim strResponse As String
    Set objMail = New clsMail
    With objMail
        .Subject = "Testmail von André"
        .ContentType = PlainText
        .Body = "Dies ist eine Test-E-Mail." & vbCrLf & vbCrLf & "Viele Grüße - André"
        .AddToRecipient "andre@minhorst.com", "André Minhorst"
        .AddAttachmentPath CurrentProject.Path & "\test.pdf"
        If .Send(strResponse) = True Then
            MsgBox strResponse
        Else
            MsgBox strResponse
        End If
    End With
End Sub
```

Listing 1: Einfacher Versand einer E-Mail per Klasse

- Ein Microsoft 365-Konto, über das Du auch E-Mails versenden kannst,
- ein Konto bei Make.com, um dieses Konto auf einfache Weise und ohne aufwendige Authentifizierung anzusteuern,
- eine Automation bei Make.com, die Deine zu versendenden Informationen entgegennimmt und sie an Microsoft 365 sendet und
- die hier vorgestellten VBA-Klassen, welche die Daten für Deine E-Mail im entsprechenden Format zusammenstellen und sie an Make.com schicken, damit dieses den Rest der Arbeit erledigt.

Wie die ersten Schritte gelingen, zeigen wir Dir in den folgenden Artikeln. Danach stellen wir Dir ausführlich die Ableitung einer Klasse daraus vor:

- **Automation mit VBA und Make** (**www.vbentwickler.de/463**)
- **Google Mail mit Make.com und VBA statt Outlook** (**www.vbentwickler.de/464**)

Im ersten Artikel geht es um die Grundlagen zur Steuerung von Webprodukten wie Microsoft 365 et cetera mit Make.com. Der zweite Artikel zeigt, wie wir Make.com dazu nutzen, eine E-Mail über Microsoft 365 zu versenden. Dort lernst Du, wie Du eine entsprechende Automation zusammenstellst. Auf diese Automation bauen wir im vorliegenden Artikel auf.

Die Automation besteht aus wenigen, einfachen Schritten. **Webhooks|Custom webhook** nimmt die Anfrage entgegen. Microsoft 365 Email liest die übermittelten Daten aus und versendet die E-Mail. Das obere Response-Element sendet eine Rückgabe, wenn die E-Mail

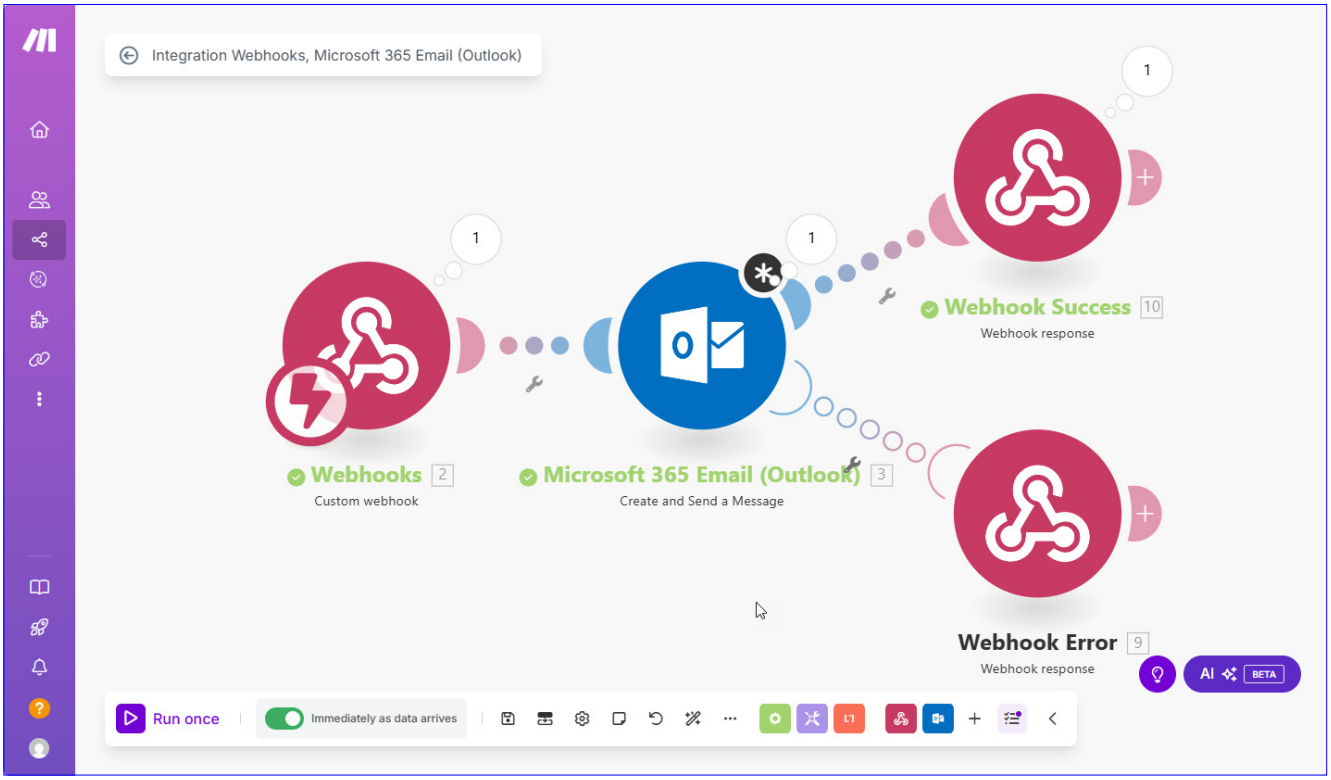


Bild 1: Diese Automation steuern wir über die hier vorgestellten Klassen an.

```
{
  "subject": "Testmail von Andr\u00E9",
  "body": {
    "content": "Dies ist eine Test-E-Mail.\r\n\r\nViele Gr\u00FC\u00DFe - Andr\u00E9",
    "contentType": "text"
  },
  "toRecipients": [
    {
      "address": "andre@minhorst.com",
      "name": "Andr\u00E9 Minhorst To"
    }
  ],
  "attachments": [
    {
      "filename": "test.pdf",
      "data": "JVBERi0xLjYNJeLjz9MNCjE2IDAgb2J..."
    }
  ],
  "importance": "Normal"
}
```

Listing 2: JSON-Dokument für das Versenden einer E-Mail

erfolgreich versendet wurde, die untere eine Rückgabe mit Informationen bezüglich auftretender Fehler (siehe Bild 1).

Benötigtes JSON-Dokument

Im Folgenden werden dazu zum Beispiel das JSON-Dokument zusammensetzen, das über den Webhook an Microsoft 365 gesendet wird, damit es von dort verschickt werden kann.

Dies sieht für unseren Aufruf von oben wie in Listing 2 aus.

Dieses Beispiel enthält nur die wichtigsten Elemente. Wir könnten noch weitere To-Empfänger einfügen, CC- und BCC-Empfänger, Reply-Adressen oder weitere Attachments.

Mapping der JSON-Daten im Make.com

Die so gelieferten Informationen können wir in Make.com wie in Bild 2 verarbeiten. Hier klicken wir einfach nur in eines der vorhandenen Eigenschaftsfelder und wählen die im JSON-Dokument enthaltenen Informationen aus.

Vorbereitungen

Da wir hier viel mit JSON-Dokumenten arbeiten werden, holen wir uns die beiden Module **mdlJSON** und **mdlJSONDOM** hinzu. Diese können wir zum Zusammenstellen und Parsen von JSON-Dokumenten nutzen.

Außerdem benötigen wir zwei Verweise auf die Bibliotheken **Microsoft XML, v6.0** und **Microsoft Scripting Runtime**. Letztere steuert das in **mdlJSON** und **mdlJSONDOM** verwendete **Dictionary**-Element bei, das wir auch in der Klasse vermehrt einsetzen.

Klassenmodul erstellen

Die Hauptklasse unseres Projekts heißt **clsMail**. Sie soll per IntelliSense alle benötigten Eigenschaften, Funktionen und Methoden anbieten (siehe Bild 3). Dazu definieren wir einige Elemente, die extern sichtbar sein sollen und einige interne Elemente. Die internen

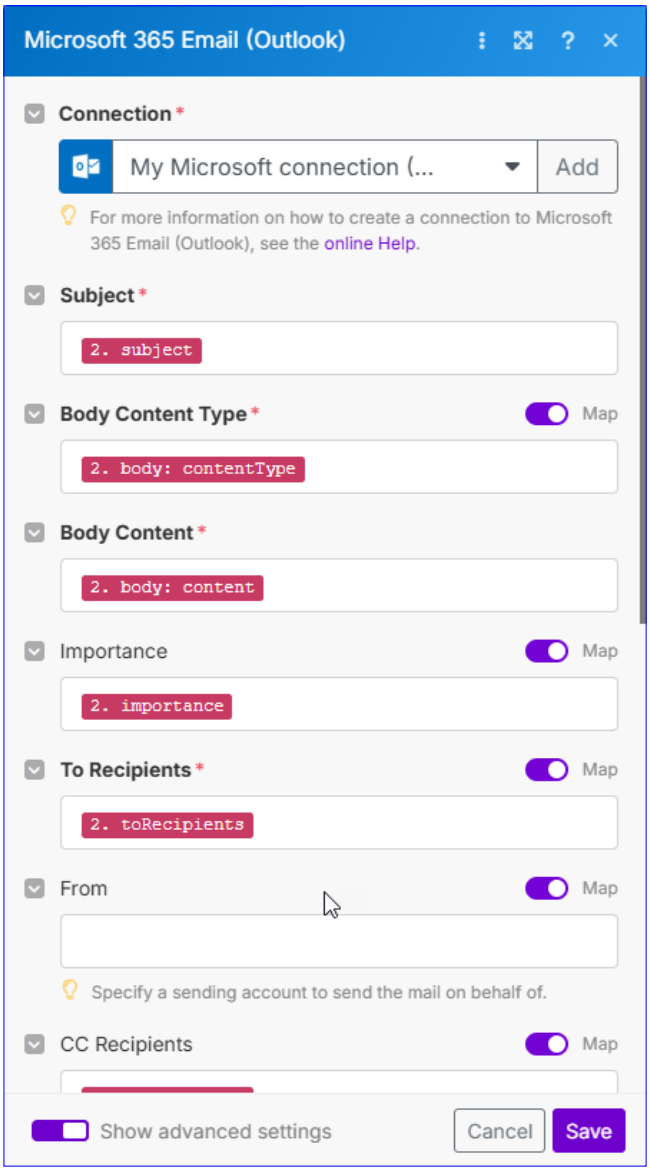


Bild 2: Die Informationen aus dem JSON-Dokument können hier gemappt werden.

Elemente sind zum Beispiel die folgenden **Collection**-Elemente, mit denen wir die verschiedenen Mail-Adressen für Empfänger und Absender sammeln sowie für die Attachments:

```
Private colFrom As Collection
Private colToMailContacts As Collection
Private colCCMailContacts As Collection
Private colBCCMailContacts As Collection
Private colReplyToMailContacts As Collection
```

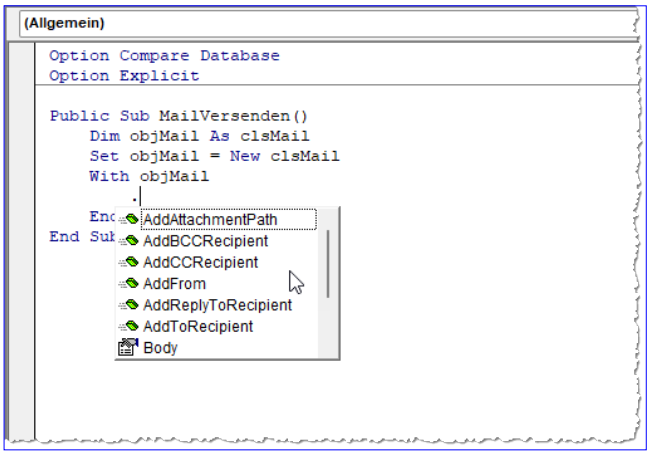


Bild 3: Auswahl der Elemente der Klasse **clsMail**

```
Private colAttachmentPaths As Collection
```

Außerdem gibt es einige Eigenschaften, die wir über **Property Let/Get**-Prozeduren nach außen veröffentlichen. Die damit erfassten Werte sollen in den wie folgt deklarierten privaten Variablen erfasst werden:

```
Private m_ContentType As ContentTypeEnum
Private m_Subject As String
Private m_Body As String
Private m_Importance As ImportanceEnum
```

Klasse initialisieren

Damit die Collections mit den nachfolgend beschriebenen **Add...**-Methoden gefüllt werden können, erstellen wir jeweils eine Instanz dieser **Collection**-Objekte. Das erledigen wir in der Ereignisprozedur **Class_Initialize**, das automatisch beim Erstellen der Klasse ausgelöst wird:

```
Private Sub Class_Initialize()
    Set colFrom = New Collection
    Set colToMailContacts = New Collection
    Set colCCMailContacts = New Collection
    Set colBCCMailContacts = New Collection
    Set colReplyToMailContacts = New Collection
    Set colAttachmentPaths = New Collection
End Sub
```

```
Public Sub AddToRecipient(strEMail As String, strName As String)
    Dim objMailContact As New clsMailContact
    With objMailContact
        .Address = strEMail
        .Name = strName
    End With
    colToMailContacts.Add objMailContact
End Sub

Public Sub AddCCRecipient(strEMail As String, strName As String)
    Dim objMailContact As New clsMailContact
    With objMailContact
        .Address = strEMail
        .Name = strName
    End With
    colCCMailContacts.Add objMailContact
End Sub

Public Sub AddBCCRecipient(strEMail As String, strName As String)
    Dim objMailContact As New clsMailContact
    With objMailContact
        .Address = strEMail
        .Name = strName
    End With
    colBCCMailContacts.Add objMailContact
End Sub

Public Sub AddReplyToRecipient(strEMail As String, strName As String)
    Dim objMailContact As New clsMailContact
    With objMailContact
        .Address = strEMail
        .Name = strName
    End With
    colReplyToMailContacts.Add objMailContact
End Sub

Public Sub AddFrom(strEMail As String, strName As String)
    Dim objMailContact As clsMailContact
    With objMailContact
        .Address = strEMail
        .Name = strName
    End With
    colFrom.Add objMailContact
End Sub

Public Sub AddAttachmentPath(strPath As String)
    colAttachmentPaths.Add strPath
End Sub
```

Listing 3: Prozeduren zum Hinzufügen von E-Mail-Kontakten und Attachments

Methoden zum Hinzufügen von Mailkontakten und Anlagen

In Listing 3 sehen wir verschiedene **Add...**-Methoden. Wir schauen uns diese am Beispiel der Methode **AddToRecipient** an. Diese soll einen Absender-E-Mail-Kontakt zur Klasse hinzufügen.

Ein E-Mail-Kontakt, wie wir ihn als Absender, Empfänger, CC oder BCC angeben können, erfordert aber immer die Angabe von E-Mail-Adresse und Name. Damit wir diese gemeinsam in einer Collection unterbringen können, haben wir dazu noch eine Klasse erstellt. Diese heißt **clsMailContact** uns wird in Listing 4 abgebildet.

In Prozeduren wie **AddToRecipient** nehmen wir nun die E-Mail-Adresse und den Namen per Parameter entgegen, erstellen eine neue Instanz der Klasse **clsMailContact** und weisen ihren Eigenschaften **Address** und **Name** die benötigten Daten hinzu. Danach hängen wir das neue **clsMailContact**-Objekt an die jeweilige Collection an, in diesem Fall **colToMailContacts**.

Da diese Collections klassenweit deklariert sind, gehen ihre Inhalte nicht verloren und wir können Schritt für Schritt weitere Elemente hinzufügen.

Anlagen hinzufügen

Noch etwas einfacher gelingt dies bei den Anlagen. Die Prozedur **AddAttachmentPath** nimmt den Pfad der