

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

TEXTE ÜBERSETZEN MIT DEEPL, VBA UND EXCEL

Lerne das Tool DeepL und seine Programmierung per VBA kennen und lasse den Inhalt von Excel-Tabellen übersetzen.

SEITE 115

EXCEL-WORKBOOKS UND SHEETS PROGRAMMIEREN

Programmiere Excel-Workbooks und Worksheets sowie die in Bereichen und Zellen enthaltenen Daten per VBA.

SEITE 73

OUTLOOKMAILS NACH EINGANG VERARBEITEN

Fange E-Mails direkt am Eingang ab und verarbeite diese weiter. Alles per VBA!

SEITE 14



André Minhorst Verlag

Buttons in Excel

In anderen Artikeln zeigen wir, wie Du Makros in Excel aufzeichnest oder wie Du per VBA-Prozedur auf die verschiedenen Elemente einer Excel-Arbeitsmappe zugreifen kannst. Diese Makros sind durchaus praktisch, aber der Nutzen reduziert sich doch, wenn man zum Ausführen beziehungsweise aufrufen immer den VBA-Editor anzeigen muss. Also zeigen wir im vorliegenden Artikel, wie Du einem Arbeitsblatt in Excel eine Schaltfläche hinzufügst, mit der Du die für dieses Arbeitsblatt vorgesehenen VBA-Prozeduren beziehungsweise Makros viel schneller und komfortabler aufrufen kannst.

Wie wir im Artikel **VBA: Makros aufzeichnen** (www.vbentwickler.de/324) gezeigt haben, kannst Du einfache Abläufe in Excel mit der Aufzeichnen-Funktion als Makro aufzeichnen.

Im Bereich **Entwicklertools** des Ribbons, in dem sich auch die Aufzeichnen-Funktion befindet, können wir über den Befehl **Makros** den gleichnamigen Dialog öffnen, der dann alle aktuell erreichbaren Makros anzeigt (siehe Bild 1). Hier kannst Du die Schaltfläche **Ausführen** anklicken, um das aktuell in der Liste markierte Makro zu starten.

Das ist allerdings ohne weitere Anpassungen schon die einfachste Methode, um ein Makro zu starten. Wenn man bedenkt, dass ein Makro eigentlich das Ausführen verschiedener Aktionen vereinfachen oder beschleunigen soll, ist das recht kompliziert.

Besser wäre es, wenn man direkt an der Stelle, an der die Aktion gefragt ist, eine Möglichkeit zu ihrem Aufruf platzieren könnte. Dazu eignet sich beispielsweise eine Schaltfläche. Wie wir diese anlegen und damit das angegebene Makro aufrufen, zeigen wir in den folgenden Abschnitten.

Schaltfläche hinzufügen

Die erste Voraussetzung zum Hinzufügen einer Schaltfläche ist bereits gegeben, wenn Du über das Ribbon Makros aufzeichnen kannst. Dann hast Du bereits die

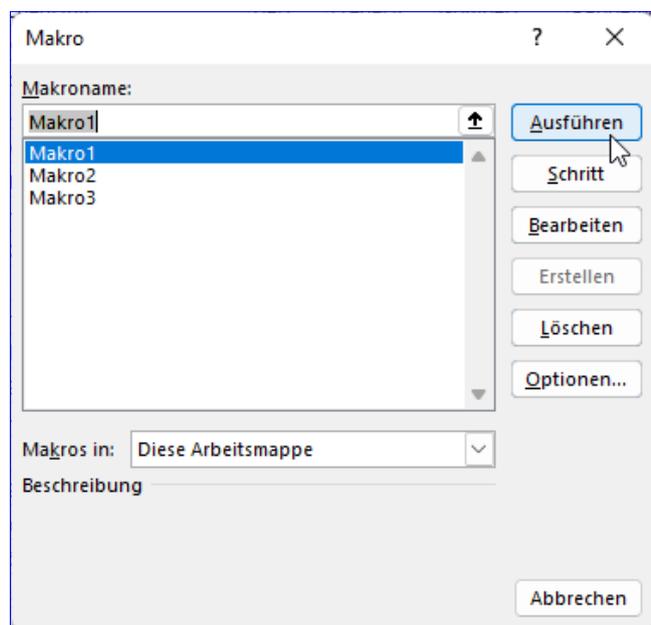


Bild 1: Aufruf eines Makros

Ribbon-Registerkarte **Entwicklertools** eingeblendet. Wenn Du nun die Arbeitsmappe geöffnet hast, der Du die Schaltfläche hinzufügen möchtest, kannst Du den Ribbon-Eintrag **Entwicklertools|Steuerelemente|Einfügen** aufrufen und findest die Liste der verfügbaren Steuerelemente vor. Diese liefert zwei Bereiche, wobei wir als Erstes die Schaltfläche aus dem Bereich **Formularsteuerelemente** anklicken (siehe Bild 2).

Danach können wir mit der Maus im Arbeitsblatt einen Rahmen aufziehen, der die Größe der zu erstellenden Schaltfläche angibt (siehe Bild 3).

Einfache Excel-Formulare erstellen

Excel kann nicht nur einfach Daten in Tabellenform anzeigen. Es bietet auch verschiedene Möglichkeiten, um Daten auszuwerten, beispielsweise in Diagrammen, oder auch Möglichkeiten zur komfortableren Eingabe von Daten. Wer zum Beispiel sonst mit Access arbeitet, dass man dem Benutzer dort üblicherweise Formulare zur Verfügung stellt, um die Daten einzugeben, zu betrachten und zu verwalten. Warum also sollte man den Anwender in Excel mit riesigen »Tapeten« von Daten quälen, statt im jeweils den Inhalt einer Zeile der Tabelle zum Betrachten und zum Bearbeiten anzubieten? Dieser Artikel zeigt eine sehr einfache Möglichkeit, um dem Benutzer ein ergonomischeres Erlebnis zu bieten.

Es gibt noch wesentlich ausgefeiltere und flexiblere Möglichkeiten, in Excel Formulare für verschiedene Zwecke anzuzeigen.

Bevor wir uns diese in weiteren Artikeln ansehen, wollen wir uns jedoch eine eher versteckte Variante anschauen.

Tabelle aus Bereich erstellen

Dazu benötigen wir zunächst eine Tabelle. Dabei zunächst zur Begriffsklärung der Hinweis, dass ein Arbeitsblatt (englisch **Worksheet**) nicht gleichzusetzen ist mit einer Tabelle – auch wenn der Registerreiter eines Arbeitsblattes unter dem Arbeitsblatt standardmäßig den Namen **Tabelle** enthält.

Wir gehen von einem einfachen Worksheet wie in Bild 1 aus. Hier markieren wir die Spaltenüberschriften und die Daten, die in dem noch zu erstellenden Formular angezeigt werden sollen. Dann betätigen wir den Ribbon-Befehl **Einfügen|Tabellen|Tabelle**.

Dies zeigt den Dialog **Tabelle erstellen** an. Hier können wir den Bereich für die Tabelle nochmal korrigieren und angeben, ob die Tabelle bereits Spaltenüberschriften enthält (siehe Bild 2).

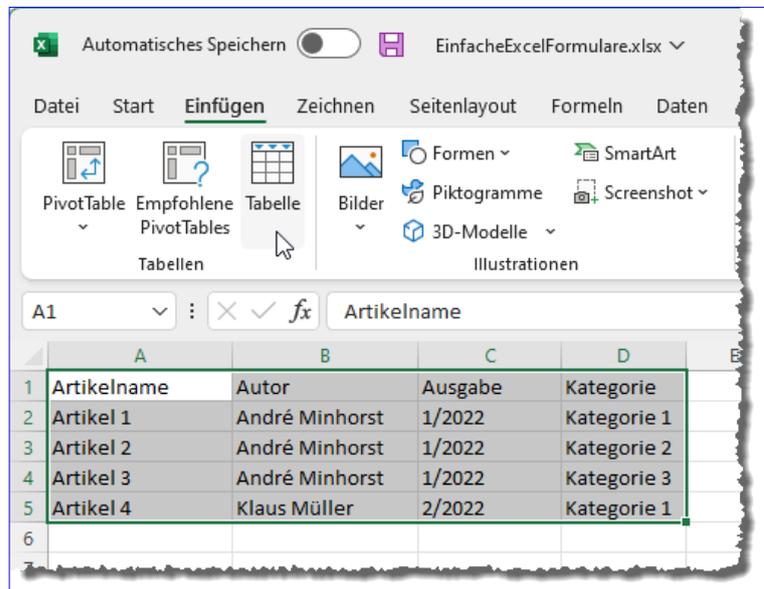


Bild 1: Markieren des Bereichs für die zu erstellende Tabelle

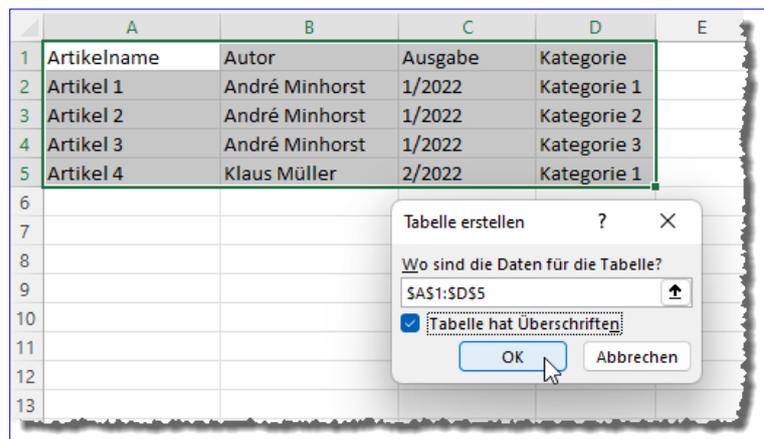


Bild 2: Erstellen einer Tabelle aus einem einfachen Worksheet

Excel: Sheet-Navigation per Button

Wenn man ein Excel-Workbook mit vielen verschiedenen Worksheets verwendet, möchte man beim Öffnen des Workbooks vielleicht eine Übersichtsseite präsentiert bekommen, von der aus man per Mausklick auf entsprechende Schaltflächen zu den übrigen Worksheets gelangt – und am besten von dort aus mit einer weiteren Schaltfläche wieder zurück zur Übersicht. In diesem Artikel zeigen wir, wie das gelingt, und warum wir noch nicht mal eine einzige Zeile VBA-Code für dieses Vorhaben benötigen.

In unserem Beispiel gehen wir davon aus, dass wir vier Worksheets nutzen. Das erste namens **Start** soll Schaltflächen enthalten beziehungsweise entsprechende Formen mit Beschriftungen. Die drei weiteren Worksheets namens **Tabelle 1**, **Tabelle 2** und **Tabelle 3** enthalten jeweils eine Schaltfläche beziehungsweise eine Form, um zur Startseite zurückzukehren.

Grundlegende Informationen zu Schaltflächen und Alternativen unter Excel findest Du im Artikel **Buttons in Excel** (www.vbentwickler.de/328).

Startseite gestalten

Dem als Startseite verwendeten Worksheet fügen wir als Erstes die drei Formen hinzu, über die wir die drei übrigen Worksheets ansteuern wollen. Dazu wechseln wir im Ribbon zum Bereich **Einfügen|Illustrationen** und wählen dort den Befehl **Formen|Rechteck: Abgerundete Ecken** aus (siehe Bild 1).

Auf diese Weise fügen wir drei Rechtecke hinzu, schreiben den Text **Tabelle 1**, **Tabelle 2** und **Tabelle 3** hinein und passen die Größe und Ausrichtung der Texte an (siehe Bild 2).

Ein Tipp dazu: Stelle zuerst eine Schaltfläche fertig und kopiere diese dann, damit Du die Texte nicht immer wieder neu anpassen musst.

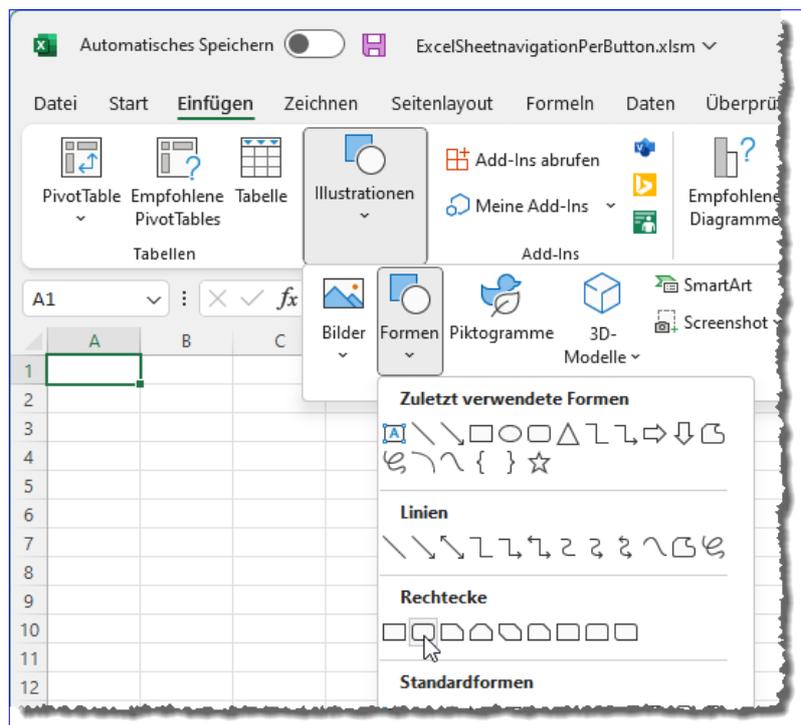


Bild 1: Hinzufügen der als Schaltfläche verwendeten Formen

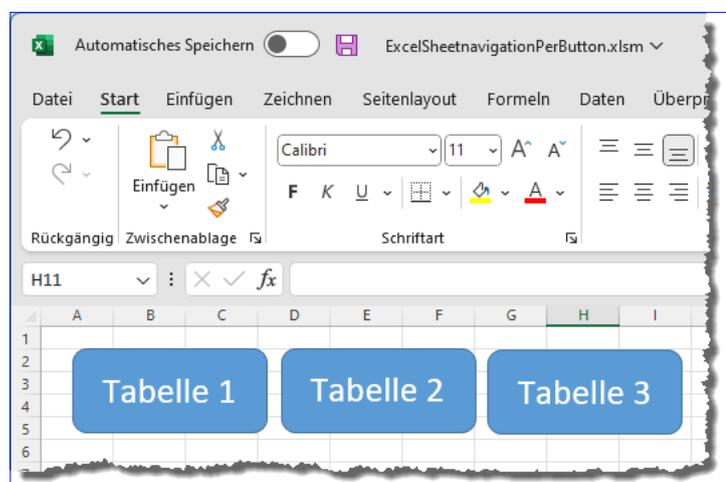


Bild 2: Die drei Schaltflächen zum Öffnen der weiteren Tabellen

Excel: Workbooks und Worksheets per VBA

Excel bietet eine ganze Reihe hierarchisch angeordneter Elemente, die wir uns in diesem Artikel ansehen. Dabei schauen wir uns auch gleich an, wie wir diese per VBA referenzieren können. Wir beginnen ganz oben in der Hierarchie mit der Excel-Anwendung und arbeiten uns dann über die verschiedenen Elemente bis hin zur einzelnen Zelle mit ihren Eigenschaften. Nach der Lektüre dieses Artikels kannst Du alle wichtigen Elemente eines Excel-Workbooks referenzieren, diese in Schleifen durchlaufen und je nach Objekttyp neue Elemente hinzufügen, bearbeiten oder entfernen können. Außerdem lernst Du noch einige Grundlagen des VBA-Editors kennen.

Hinweis: Speichern als .xlsm-Datei nötig

Wir beschreiben in diesem Artikel einige Elemente, die zum VBA-Projekt einer Excel-Datei hinzugefügt werden. Damit die Änderungen an diesem VBA-Projekt mit der Excel-Datei gespeichert werden, musst Du diese unter einem anderen Dateityp speichern, nämlich als **Excel-Arbeitsmappe mit Makros (*.xlsm)**.

Dazu erscheint allerdings auch noch eine Meldung, bevor Du die Datei ohne diese Änderungen speicherst. Lies diese genau, um nicht versehentlich den hinzugefügten VBA-Code zu verlieren.

Ganz oben: Das Application-Objekt

Wenn wir ganz oben in der Hierarchie der Excel-Objekte beginnen wollen, dann sprechen wir über das **Application**-Objekt, also die eigentliche Anwendung.

Ein solches Objekt gibt es für alle Office-Anwendungen, die VBA-Unterstützung anbieten, also zum Beispiel Access, Excel, Outlook, PowerPoint und Word. Das **Application**-Objekt enthält

alle weiteren Objekte, die wir für die Automatisierung der jeweiligen Anwendung benötigen – zum Beispiel Eigenschaften, über die wir die aktive Arbeitsmappe (**Workbook**) oder das aktuelle Tabellenblatt (**Worksheet**) referenzieren und diese steuern können.

Application-Objekt: Von innen oder außen referenzieren?

Bei jeder Office-Anwendung gibt es verschiedene Möglichkeiten, das **Application**-Objekt der jeweiligen Anwendung zu referenzieren.

Dabei kommt es darauf an, von wo aus man darauf zugreifen möchte. Wir haben zum Beispiel die folgenden Optionen:

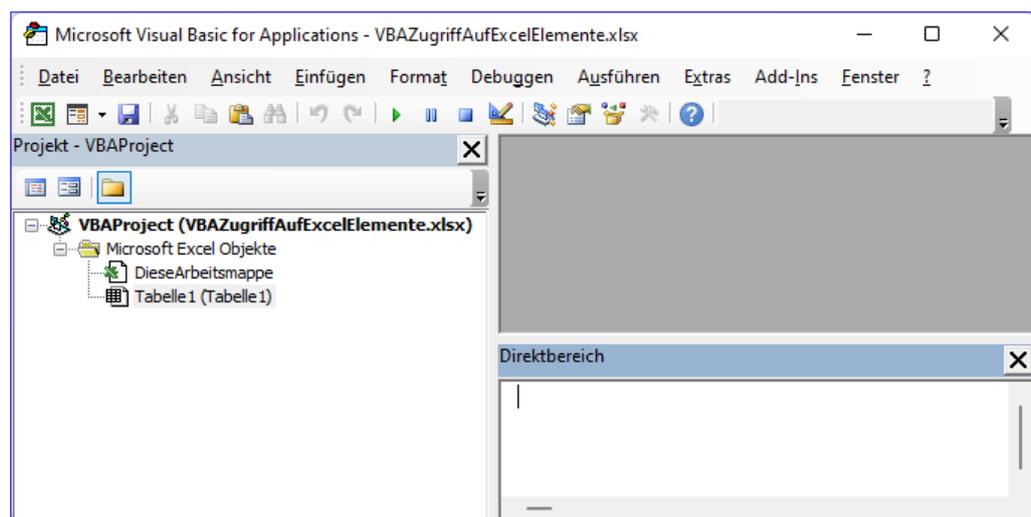


Bild 1: Der VBA-Editor für eine neue, leere Excel-Datei

Excel: Zellen und Bereiche per VBA

Im Artikel »Excel: Workbooks und Worksheets per VBA« haben wir uns angesehen, wie wir Arbeitsmappen und Arbeitsblätter mit VBA nutzen können. Im vorliegenden Artikel gehen wir einen Schritt weiter und nehmen uns die offensichtlichen Elemente eines Arbeitsblatts vor – die Zellen. Nicht weniger spannend sind allerdings die Bereiche, unter VBA »Range« genannt. Wie wir diese referenzieren, auslesen und bearbeiten können, zeigen wir auf den folgenden Seiten.

Worksheet referenzieren

Im oben genannten Artikel **Excel: Workbooks und Worksheets per VBA** (www.access-im-unternehmen.de/326) zeigen wir, wie Du **Workbook-** und **Worksheet-**Elemente referenzieren kannst. Im aktuellen Artikel verwenden wir oft das aktuelle **Worksheet-**Element als Basis für den Zugriff auf verschiedene Eigenschaften wie **Range** oder **Cells**.

Dieses können wir mit **ActiveSheet** referenzieren, allerdings bietet **ActiveSheet** kein IntelliSense an. Das liegt daran, dass **ActiveSheet** nicht nur ein **Worksheet-**Objekt, sondern auch ein **Chart-**Objekt zurückliefern könnte.

Um diese Einschränkung zu umgehen, referenzieren wir das **Worksheet-**Objekt jeweils mit folgenden Anweisungen:

```
Dim wks As Worksheet  
Set wks = ActiveSheet
```

wks ist nun explizit als **Worksheet-**Objekt deklariert und liefert folglich dessen Elemente per IntelliSense. Wir werden die Deklaration und Zuweisung des aktuellen Worksheets nicht in jedem Beispiel explizit ausführen, sondern nach dem ersten Beispiel einfach mit **wks** arbeiten.

Gleichwohl sei erwähnt, dass Du innerhalb des VBA-Projekts einer Excel-Arbeitsmappe auch direkt die **Range-** oder die **Cells-**Eigenschaft nutzen kannst. Im

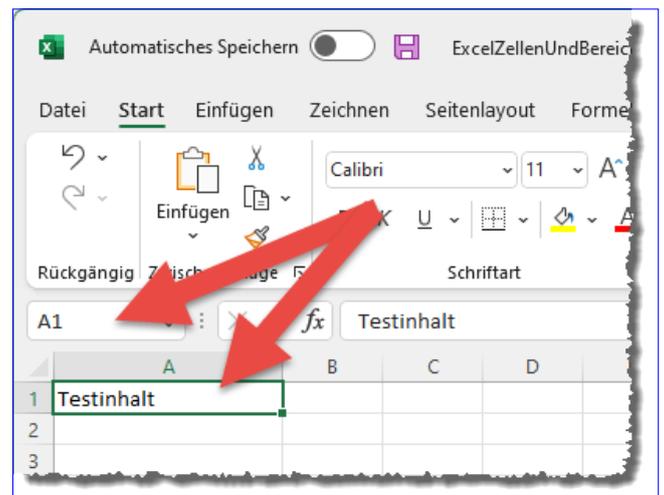


Bild 1: Die Bezeichnung der aktuellen Zelle, hier A1

Direktfenster erhältst Du also beispielsweise mit folgendem Ausdruck den Inhalt der Zelle mit der Adresse **A1**:

```
? Range("A1")
```

Da ich allerdings davon ausgehe, dass Du auch von anderen Anwendungen wie Access, Word oder Outlook oder auch von COM-Add-Ins auf die Inhalte von Excel-Arbeitsmappen zugreifen willst, verwenden wir immer die explizite Angabe der Objekte.

Einzelne Zelle über die A1-Notation per Range referenzieren

Im Excel-Arbeitsblatt erhalten wir schon Hinweise darauf, wie ein Weg zum Referenzieren einer Zelle aussehen könnte (siehe Bild 1).

Excel: Arbeitsblätter per Ribbon steuern

Wenn Du eine eigene Excel-Lösung mit einigen Arbeitsblättern erstellt hast, stört es Dich vielleicht, dass Du nicht schnell über die Registerreiter unten im Excel-Fenster auf alle Arbeitsblätter zugreifen kannst. Je nach Fenstergröße und Anzahl der Arbeitsblätter zeigt Excel dort nämlich nicht alle Arbeitsblätter an. Damit haben wir allerdings ein schönes Beispiel für den Einsatz des Ribbons in einer Excel-Arbeitsmappe. Diesem fügen wir ein Tab mit einem Button für jedes Arbeitsblatt, das schnell erreichbar sein soll, hinzu. Das Ribbon hat noch einen Vorteil: Wenn wir dort für die wichtigsten Arbeitsblätter je eine Schaltfläche hinzufügen, können wir diese auch noch mit einem Icon ausstatten, um das gesuchte Arbeitsblatt noch schneller zu finden.

Grundlagen zu Ribbons in Office-Dokumenten

Wie wir einer Excel-Arbeitsmappe ein eigenes Ribbon hinzufügen, haben wir grundlegend im Artikel **Ribbons in Office-Dokumenten** (www.vbentwickler.de/329) erläutert.

Hier haben wir das Tool **Office RibbonX Editor** vorgestellt, mit dem wir das Ribbon eines Office-Dokuments bearbeiten können.

Excel-Arbeitsmappe mit VBA-Code erstellen

Da wir für die geplanten Ribbon-Buttons VBA-Code hinterlegen wollen, können wir direkt eine Excel-Arbeitsmappe mit der Dateiendung **.xlsm** anlegen. Diese nennen wir **Excel_ArbeitsblaetterPerRibbon.xlsm**.

Der Arbeitsmappe fügen wir neben dem vorhandenen Arbeitsblatt noch weitere Arbeitsblätter hinzu. Anschließend soll das Register am unteren Rand wie in Bild 1 aussehen. Hier ist anhand der

drei Punkte (...) zu erkennen, dass in dieser Ansicht nicht alle Registerreiter angezeigt werden können.

Ribbondefinition zusammenstellen

Die Ribbondefinition ist nicht besonders komplex, da sie lediglich ein **tab**-, ein **group**- und einige **button**-Elemente enthält, wobei wir für das **tab**-Element noch per Attribut festlegen müssen, an welcher Position es angezeigt werden soll. Vielleicht möchtest Du auch alle eingebauten Ribbon-Tabs ausblenden und nur dieses **tab**-Element anzeigen – das ist noch einfacher zu realisieren.

Bevor wir mit dem Zusammenstellen beginnen, öffnen wir die Excel-Arbeitsmappe mit dem Tool **Office RibbonX Editor**. Hier klicken wir mit der rechten Maus-

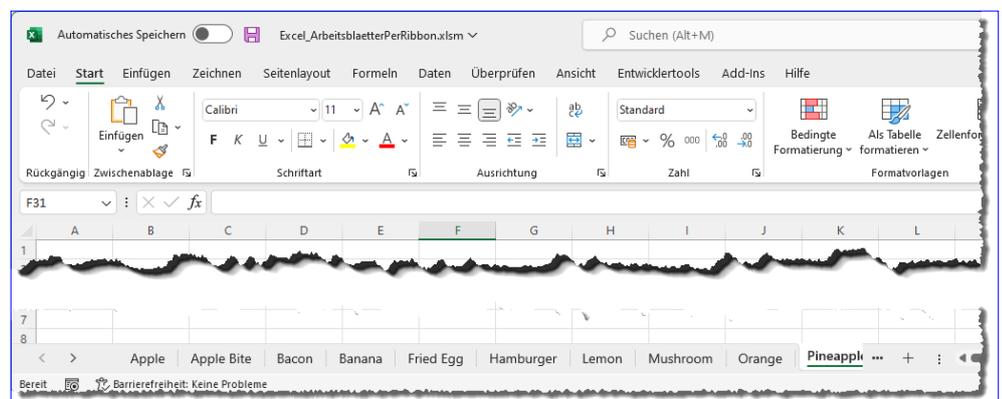


Bild 1: Arbeitsmappe mit einigen Arbeitsblättern

Outlook: E-Mail per Drag and Drop nach Access

Wenn Du eine Kundendatenbank mit Access programmiert hast und die Kommunikation per E-Mail über Outlook läuft, möchtest Du vielleicht einem Kundendatensatz die E-Mails dieses Kunden zuweisen. Dafür gibt es verschiedene Möglichkeiten. In diesem Artikel schauen wir uns eine an, bei der Du die E-Mail per Drag and Drop auf einen Bereich in einem Access-Formular ziehst. Dort verarbeiten wir die E-Mail und speichern bestimmte Daten in einer Tabelle, damit die E-Mail bei Bedarf vom Kunden-Formular aus wieder in Outlook angezeigt werden kann. Wer sich schon mit dem Thema beschäftigt hat, weiß, dass man E-Mails eigentlich nicht nach Access ziehen kann. Deshalb umschiffen wir dieses Problem mit einem kleinen Trick.

Beschreibung der Lösung

Die Lösung soll folgenden Vorgang abbilden: In einer Access-Datenbank sind die Daten von Kunden gespeichert, die in einem Formular angezeigt werden können. Über Outlook kommunizieren wir per E-Mail mit diesem Kunden. Die E-Mails wollen wir nicht nur in Outlook sehen, sondern wir wollen auch in der Access-Kundendatenbank die E-Mails des Kunden sehen und diese gegebenenfalls in Outlook anzeigen.

Dabei wollen wir aber keineswegs alle E-Mails von diesem Kunden und an diesen Kunden automatisch in der Access-Datenbank speichern, sondern der Benutzer soll entscheiden, welche E-Mails des Kunden in der Datenbank gespeichert werden sollen.

Dazu benötigen wir eine einfache Möglichkeit, um die E-Mail zur Kundentabelle hinzuzufügen. Eine der intuitivsten Möglichkeiten, um Elemente von einer Anwendung zur anderen zu bewegen, ist Drag and Drop – also das Ziehen des Elements, in diesem Fall der E-Mail, mit der Maus.

Die E-Mail soll nach dem Ziehen in einen bestimmten Bereich im Formular mit den Kundendaten in einem Unterformular angezeigt werden. Im Hintergrund nutzen wir dazu neben der Kundentabelle eine Tabelle, die alle relevanten Daten zu den E-Mails des Kunden

speichert. Darunter befindet sich auch der eindeutige Schlüssel dieser E-Mail in Outlook, mit dem wir diese in Outlook von der Kundendatenbank aus öffnen können.

Aufbau des Datenmodells

Das Datenmodell zu unserer Lösung umfasst lediglich zwei Tabellen:

- **tblKunden:** Diese Tabelle enthält rudimentäre Kundendaten – eben so viele, dass wir diese in einem Kundenformular darstellen können.
- **tblMailItems:** Diese Tabelle enthält ein Fremdschlüsselfeld zur Tabelle **tblKunden**, womit wir die

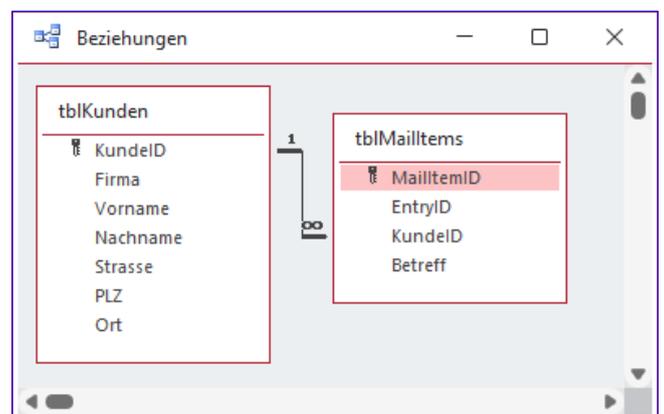


Bild 1: Die Tabellen der Datenbank in der Übersicht

Outlook: E-Mails nach Eingang verarbeiten

Die verschiedenen Klassen von Outlook bieten eine ganze Reihe von Ereignissen an. Diese werden durch unterschiedliche Aktionen ausgelöst. Eine dieser Aktionen ist das Eintreffen einer oder mehrerer neuer E-Mails. Dieses Ereignis mit einer geeigneten Ereignisprozedur abzufangen ist ein sinnvolles Beispiel für die Beschreibung der Programmierung von Ereignissen unter Outlook. Mit E-Mails kann man nach dem Eingang eine Menge anstellen – diese löschen, in einen anderen Ordner verschieben, die Message-Datei im Dateisystem sichern oder auch die Inhalte auslesen, um diese etwa in eine Datenbank zu schreiben. In diesem Artikel erläutern wir erst einmal allgemein, wie wir überhaupt mit VBA-Code auf den Eingang einer E-Mail reagieren können.

Outlook bietet alles, was man braucht – E-Mails werden abgerufen und landen im Posteingangs-Ordner. Man kann verschiedene Regeln aufstellen, nach denen die eingehenden E-Mails automatisch nach den gewünschten Kriterien in verschiedene Ordner verschoben werden, und wenn es auch der Ordner **Gelöschte Objekte** oder der Spam-Ordner ist. Und natürlich können wir die E-Mails auch manuell in andere Ordner verschieben oder auch Aufgaben auf Basis einer E-Mail erstellen. Noch schöner wäre es allerdings, wenn wir die gewünschten Schritte individuell programmieren könnten.

Welches Objekt und welches Ereignis?

Dazu müssen wir erst einmal wissen, wie wir eine Ereignisprozedur programmieren, mit der wir auf den Eingang einer E-Mail reagieren können. Der erste Schritt ist dabei, herauszufinden, welches Ereignis welcher Klasse wir überhaupt nutzen können. In weiteren Artikeln zur Programmierung von Outlook per VBA haben wir bereits gesehen, dass es unterschiedliche Klassen gibt, mit denen wir die verschiedenen Elemente der Benutzeroberfläche referenzieren können. Mit dem **Application**-Objekt greifen wir auf die Anwendung selbst zu, mit dem **Explorer**-Objekt auf die verschiedenen Ansichten von Outlook und mit dem **Inspector**-Objekt auf von Outlook aus geöffnete Fenster beispielsweise zur Anzeige einer E-Mail. Außerdem gibt es noch Klassen,

mit denen wir die einzelnen Ordner oder die darin enthaltenen Elemente wie E-Mails, Kontakte, Termine oder Aufgaben referenzieren können.

Wenn wir neu im Thema sind und nicht genau wissen, welche Klasse welche Ereignisse bietet, hilft uns ein Blick in den Objektkatalog. In unserem Beispiel, wo wir auf den Eingang einer E-Mail reagieren wollen, gibt es unterschiedliche Ideen, welche Klasse ein passendes Ereignis bietet. Vielleicht gibt es ein Ereignis, das ausgelöst wird, wenn einem Ordner ein neues Element hinzugefügt wird? Oder wird ein solches Ereignis von einer anderen Klasse zur Verfügung gestellt?

Haben wir Outlook geöffnet, starten wir mit **Alt + F11** den VBA-Editor. Hier zeigen wir mit **F2** den Objektkatalog an. Oben im Fenster wählen wir statt dem Eintrag **Alle Bibliotheken** die Bibliothek **Outlook** aus, direkt darunter geben wir als Suchbegriff **Mail** ein.

In der Liste der Suchergebnisse suchen wir nun nach Elementen, die durch ein Blitz-Icon markiert werden und stoßen schnell auf die beiden Ereignisse **NewMail** und **NewMailEx** (siehe Bild 1). Klicken wir **NewMail** an, sehen wir die Beschreibung dieses Ereignisses im unteren Bereich. Außerdem sehen wir, dass es sich hierbei um ein Ereignis der Klasse **Outlook.Application** handelt.

Outlook: Application_Startup feuert nicht

Das Ereignis »Startup« des »Application«-Objekts von Outlook ist für viele benutzerdefinierte Erweiterungen von Outlook essenziell, da es die Möglichkeit bietet, direkt beim Starten von Outlook VBA-Code auszuführen. Damit lassen sich für verschiedene Anwendungen wichtige Automatismen anstoßen – zum Beispiel das Deklarieren und Initialisieren von Objektvariablen, für die Ereignisse implementiert werden sollen. Leider passiert es gelegentlich, dass die Ereignisprozedur `Application_Startup` beim Starten von Outlook nicht aufgerufen wird. Woran das liegt und wir dies ändern, zeigt der vorliegende Artikel.

Du hast das Ereignis `Application_Startup` im Modul `ThisOutlookApplication` implementiert und wenn Du Outlook startest, wird diese nicht ausgelöst? Das kann verschiedene Gründe haben:

- Die Makroeinstellungen für Outlook erlauben die Ausführung von Makros nicht. Der Klassiker, der gerade bei neu installierten Office-Anwendungen auftritt.
- Outlook ist bereits geöffnet, gegebenenfalls auch unsichtbar. In diesem Fall kann `Application_Startup` nicht ausgelöst werden, weil Outlook ja gar nicht neu startet. Wenn Outlook per Code initialisiert wird, feuert `Application_Startup` übrigens nicht.
- Ein spezieller Registry-Eintrag, den Outlook beim Starten abfragt, weist einen Wert auf, der das Ausführen von `Application_Startup` unterbindet.

In den folgenden Abschnitten schauen wir uns die Lösungen für diese Probleme an.

Makroeinstellungen kontrollieren und anpassen

Wenn `Application_Startup` beim Öffnen von Outlook nicht ausgeführt wird, solltest Du als Erstes die Makroeinstellungen prüfen. Dazu brauchst Du gar nicht erst den Optionen-Dialog zu öffnen – wechsle ein-

fach zum VBA-Editor (**Alt + F11**), öffne das Modul `ThisOutlookApplication`, platziere die Einfügemarke in der Prozedur `Application_Startup` und betätige die Taste **F5**.

Wenn die Prozedur nun auch nicht ausgeführt wird, sondern die Meldung aus Bild 1 erscheint, hast Du schon die Lösung des Problems gefunden.

Um Makros zu aktivieren, öffnest Du mit dem Ribbonbefehl **Datei|Optionen** den Optionen-Dialog von Outlook.

Hier wechselst Du zum Bereich **Trust Center** und klickst dort auf **Einstellungen für das Trust Center...**, was den Dialog **Trust Center** öffnet.

Dort aktivierst Du im Bereich **Makroeinstellungen** eine der Optionen **Benachrichtigungen für alle Makros** oder **Alle Makros aktivieren**.

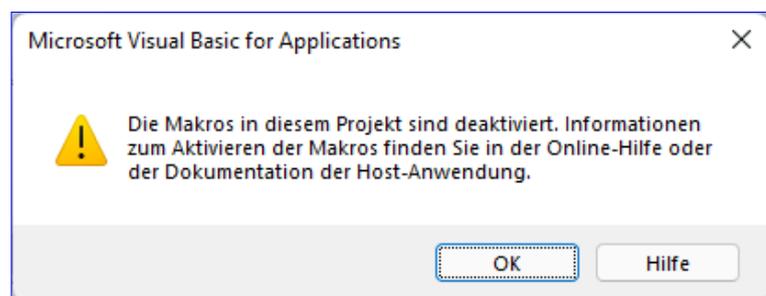


Bild 1: Meldung beim Versuch, ein Makro trotz Deaktivierung auszuführen

Ribbons in Office-Dokumenten

Word, Excel und PowerPoint sind die Office-Anwendungen, mit denen Du Dokumente anzeigen und bearbeiten, aber auch automatisieren kannst. Wenn Du beispielsweise einer Excel-Arbeitsmappe eigene, per VBA programmierte Funktionen hinzufügen möchtest, musst Du diese irgendwie aufrufen. Eine Schaltfläche in einem Excel-Arbeitsblatt ist eine Möglichkeit. Die Alternative ist, einen entsprechenden Befehl im Ribbon zu platzieren. Wie das gelingt, zeige ich im vorliegenden Artikel. Dabei erfährst Du auch einige Dinge rund um die Office Open XML-Formate und wie die Daten in diesen Dokumenten gespeichert sind – darunter auch die Definitionen von Ribbons und die darin anzuzeigenden Icons.

Grundlagen: Das Office Open XML-Format

Jede Datei, die in einem der **Office Open XML**-Formate gespeichert ist, besteht aus einem Dateicontainer, der auf einer einfachen, komponentenbasierten und komprimierten ZIP-Dateiformatspezifikation aufbaut. Das gilt zum Beispiel für Dateien mit der Dateierweiterung **.docx**, **.xlsx** oder **.pptx**, aber natürlich auch für verwandte Dokumentformate.

Kern der **Office Open XML**-Formate ist die Verwendung von XML-Referenzschemas und eines ZIP-Containers. Jede Datei setzt sich aus einer Auflistung einer beliebigen Anzahl von Komponenten zusammen. Diese Auflistung definiert das Dokument.

Dokumentkomponenten werden mithilfe des ZIP-Formats in der Containerdatei beziehungsweise dem Paket gespeichert. Bei den meisten Komponenten handelt es sich um XML-Dateien, die in der Containerdatei gespeicherte Anwendungsdaten, Metadaten und sogar Kundendaten beschreiben.

Im Containerpaket können andere, Nicht-XML-Komponenten einschließlich Komponenten wie Binärdateien, die im Dokument eingebettete Bilder oder OLE-Objekte darstellen, enthalten sein. Zusätzlich werden durch Beziehungskomponenten die Beziehungen zwischen Komponenten festgelegt.

Dieser Entwurf stellt die Struktur für Office-Dateien dar. Während sich der Inhalt der Datei aus Komponenten zusammensetzt, beschreiben die Beziehungen, wie die einzelnen Komponenten zusammenarbeiten.

Die Zusammensetzung einer Datei nach dem **Office Open XML**-Standard kannst Du Dir ansehen, indem

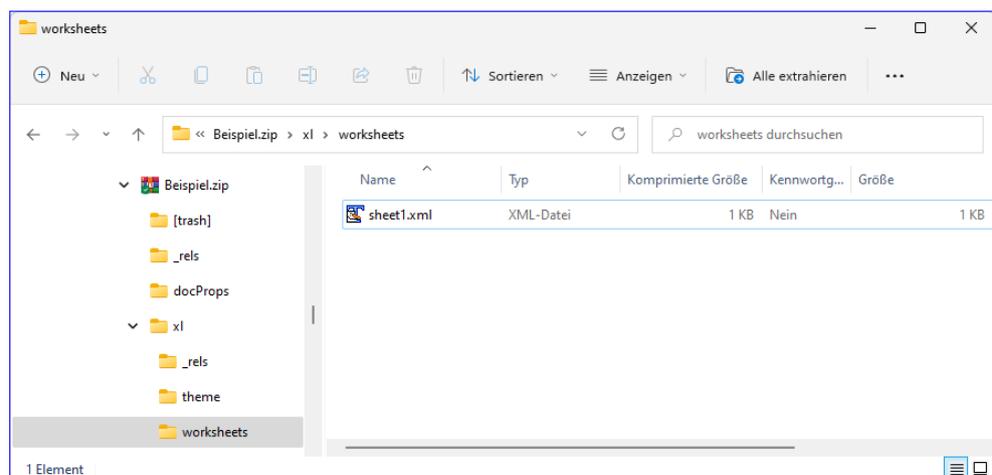


Bild 1: Komprimierter Zip-Container einer Excel-Arbeitsmappe

Texte übersetzen mit DeepL

Zum automatischen Übersetzen von Texten gibt es viele Anlässe. Vielleicht möchtest Du die Texte in einer Anwendung automatisch übersetzen lassen, damit Du selbst die Übersetzung nur noch prüfen und gegebenenfalls anpassen musst. Oder Du hast Texte in einer anderen Sprache, die Du gern in die deutsche Sprache übersetzen möchtest, um diese leichter lesen zu können. Wie auch immer: Es gibt zwar Dienste wie Google Translate, mit denen man das im Browser erledigen kann, aber wenn man viele oder umfangreiche Texte übersetzen lassen möchte, ist diese Lösung unbefriedigend. In diesem Fall bietet sich eine Automation des Vorgangs an. Und wie das geht, zeigen wir anhand eines der aktuell besten Übersetzungstools, nämlich DeepL. DeepL bietet eine API an, die wir per VBA oder mit anderen Programmiersprachen ansteuern können. Dieser Artikel stellt die Grundlagen dazu vor.

Übersetzen mit DeepL

Die erste Anlaufstelle für die Arbeit mit der API des Übersetzungstools **DeepL** ist die Webseite <https://www.deepl.com>. Hier finden wir gleich zwei Textfelder, deren linkes wir für die Eingabe des zu übersetzenden Textes nutzen können. Geben wir hier einen Text ein, erkennt DeepL automatisch die verwendete Sprache und übersetzt den Text in die Sprache, die für das rechte Textfeld ausgewählt ist. Das geht auch recht schnell, sodass das Ergebnis wie in Bild 1 aussieht.

Die DeepL-API

Mit einem Klick auf das Menü rechts oben finden wir schnell den Eintrag **API**. Klicken wir diesen an, landen wir auf einer Seite, die eine kostenlose Registrierung anbietet.

Wählen wir diese Option, landen wir

auf der Seite mit den verschiedenen Angeboten (siehe Bild 2). Die gute Nachricht ist: die kostenlose Variante reicht zum Ausprobieren der API wie in diesem Artikel beschrieben völlig aus.

Wenn Du nicht auf mehr als 500.000 Zeichen im Monat kommst, wählst Du einfach diese Option – und wenn es mehr werden, wäre die zweite Option sinnvoll. Hier fallen allerdings nach aktuellem Stand 20

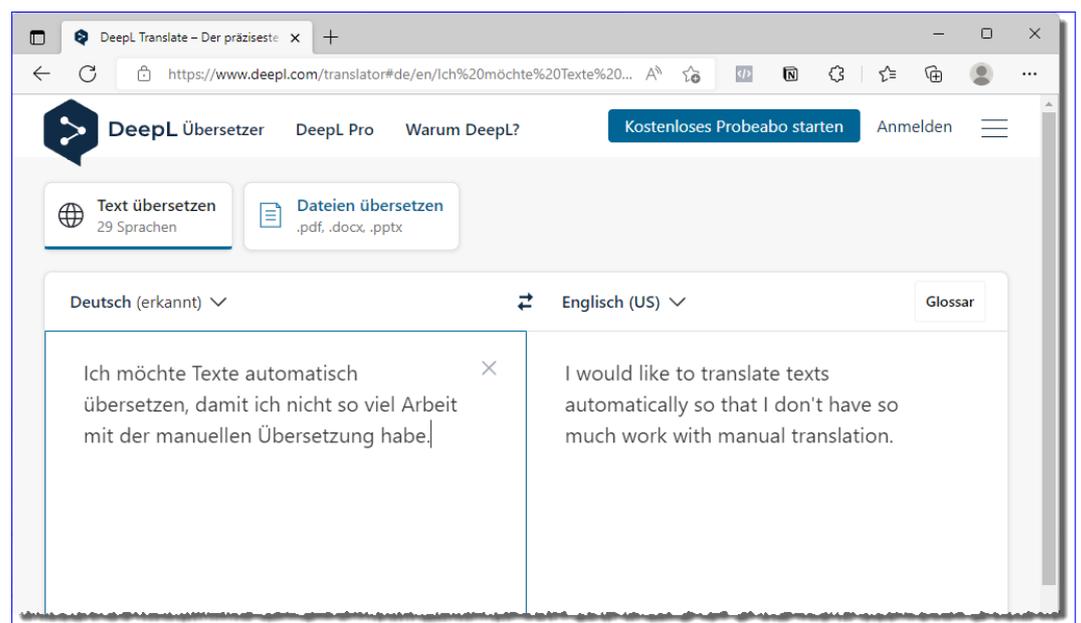


Bild 1: DeepL in Aktion

VBA Basics: Makros, Prozeduren, Funktionen und Co.

VBA-Code in VBA-Projekten von Office-Anwendungen landet zuerst einmal in Modulen. Darunter gibt es einige weitere Strukturen, auf welche die Anweisungen aufgeteilt werden. Während Deklarationen von Variablen auch direkt in einem Modul angelegt werden können, müssen ausführbare Codezeilen zwingend in Konstrukten untergebracht werden, die je nach Anwendung Makros oder Prozeduren genannt werden. Außerdem gibt es noch Funktionen. Was es mit all diesen Begriffen auf sich hat und wieso diese nicht einheitlich definiert sind, erläutern wir in diesem Artikel.

Makros vs. Prozeduren

Wer erstmal mit VBA unter Excel, Word oder Power-Point in Kontakt kommt, erledigt dies in der Regel über das Aufzeichnen von Makros – mehr dazu im Artikel **VBA: Makros aufzeichnen** (www.access-im-unternehmen.de/324). Bei der Aufzeichnung kommt ein sogenanntes Makro heraus. Schaut man sich das im VBA-Editor an, findet man jedoch eine Sub-Prozedur vor. Wir könnten nun einfach die Begriffe Makro und Sub-Prozedur synonym verwenden, wenn da nicht noch Access wäre: Hier gibt es nämlich neben VBA noch einen eigenen Objekttyp zum Erstellen von Automatisierungen, der wiederum Makro heißt.

Allerdings hat das Makro unter Access nichts mit VBA zu tun, sondern es handelt sich dabei eher um eine Möglichkeit zum »Zusammenklicken« von Automatisierungen. Microsoft hat also den Begriff Makro in verschiedenen Office-Anwendungen für unterschiedliche Techniken verwendet, was erstens bei Benutzern und Programmierern für Verwirrung sorgt.

Allerdings ist das nicht nur dort problematisch, sondern auch für unser Magazin **Visual Basic Entwickler**. Wir wollen hier ja über den Einsatz von **Visual Basic for Applications (VBA)** und verwandten Sprachen wie **Visual Basic.NET** oder **twinBASIC** berichten, und zwar für alle Office-Anwendungen.

Um Verwechslungen auszuschließen, werden wir also den Begriff »Makro« nur noch dort verwenden, wo

wir tatsächlich die Funktion zum Aufzeichnen von Makros nutzen. An allen anderen Stellen bezeichnen wir alle Code-Konstrukte, die in der ersten Zeile das Schlüsselwort **Sub** tragen, als Prozedur.

Prozeduren und Funktionen

Neben den Prozeduren gibt es noch zwei weitere Konstrukte, die zur Ausführung bestimmte Codezeilen aufnehmen können. Das erste ist die sogenannte Funktion, das zweite sind die **Property...**-Methoden. Diese treten allerdings erst in Zusammenhang mit der objektorientierten Programmierung von Klassenmodulen auf, sodass wir in diesem Artikel nicht darauf eingehen wollen.

Eigentlich sind die Bezeichnungen »Prozedur« und »Funktion« auch wieder nicht ganz konsequent, denn eigentlich sind beide Prozeduren – eine mit dem Schlüsselwort **Sub** und eine mit dem Schlüsselwort **Function**. Die korrekteren Bezeichnungen sind wohl **Sub-Prozedur** und **Function-Prozedur**. Allerdings denke ich, dass auch die Bezeichnungen **Prozedur** und **Funktion** verständlich sind, solange sie konsequent verwendet werden.

Prozeduren und Funktionen führen beide eine oder mehrere Anweisungen aus. Der wichtigste Unterschied zwischen beiden ist, dass die Funktionen ein Funktionsergebnis direkt zurückgeben kann. Dazu zwei Beispiele. Eine Prozedur sieht im einfachsten Fall wie folgt aus:

VBA Basics: Bedingungen

Wenn man in VBA-Routinen bestimmte Anweisungen in Abhängigkeit von einem Wert einer Variablen, einer Eingabe oder anderen Bedingungen ausführen lassen möchte, verwendet man sogenannte Bedingungen. Unter VBA gibt es dazu die **If...Then**-Bedingung und die **Select Case**-Bedingung. Streng genommen gibt es noch einige VBA-Funktionen, die auch Bedingungen enthalten. Diese schauen wir uns aber in einem anderen Artikel an. Hier geht es zunächst um die beiden genannten Konstrukte.

Manche Prozeduren oder Funktionen unter VBA erfordern es, dass abhängig von bestimmten Voraussetzungen unterschiedliche Anweisungen ausgeführt werden sollen. Ein einfaches Beispiel: Wenn Du per **MsgBox**-Funktion vom Benutzer einen der Werte **vbYes** oder **vbNo** abfragst, dann wirst Du im Code auch für die beiden Eingaben jeweils eigene Anweisungen ausführen wollen. Wenn nur diese beiden Möglichkeiten bestehen, bist Du mit einer einfachen **If...Then**-Bedingung gut vorbereitet. Es kann auch sein, dass es noch mehr verschiedene Auswahlmöglichkeiten gibt. Dann würde man eine **If...Then**-Bedingung mit mehr als nur zwei Zweigen verwenden – oder vielleicht sogar eine **Select Case**-Bedingung, die das Verwalten mehrerer Zweige oft einfacher und übersichtlicher macht.

Die If...Then-Bedingung

Eine **If...Then**-Bedingung kann nur aus einem einzigen Zweig bestehen, dessen Anweisungen nur bei der Erfüllung der angegebenen Bedingung ausgeführt werden. Die Bedingung ist im einfachsten Fall wie folgt aufgebaut:

```
If [Bedingung] Then
    [Anweisungen]
End If
```

[**Bedingung**] kann dabei ein beliebiger Ausdruck sein, der den Wert **True** oder **False** zurückliefert. Das Ergebnis des für [**Bedingung**] angegebenen Ausdrucks muss also den Datentyp **Boolean** aufweisen. Wenn

[**Bedingung**] den Wert **True** ergibt, werden die zwischen **If...Then** und **End If** enthaltenen Anweisungen ausgeführt. Das sieht im einfachsten Fall wie folgt aus:

```
Dim bolBedingung As Boolean
bolBedingung = True
If bolBedingung Then
    MsgBox "bolBedingung ist True"
End If
```

Viele Entwickler schreiben die **If...Then**-Zeile in diesem Fall wie folgt:

```
If bolBedingung = True Then
    ...
```

Das ist nicht grundsätzlich falsch, aber **bolBedingung** enthält bereits einen **Boolean**-Wert, der Vergleich mit **True** hat hier keinen zusätzlichen Nutzen und kostet lediglich Rechenzeit. Wenn Du allerdings im **If...Then**-Zweig prüfen willst, ob die Bedingung nicht erfüllt ist, dann benötigen wir mindestens einen weiteren Operator. Die erste Möglichkeit ist der Vergleich mit dem Wert **False**:

```
If bolBedingung = False Then
    MsgBox "bolBedingung ist False"
End If
```

Die zweite Möglichkeit ist der Einsatz des Operators **Not**:

VBA Basics: Module, Klassen und Co.

Unter VBA strukturieren wir den Code in verschiedene Elemente. Die übergeordneten Elemente sind die Module. Hier unterscheiden wir zwischen Klassenmodulen und Standardmodulen. Darunter können wir Variablen und ähnliche deklarieren sowie auszuführende Anweisungen unterbringen. Diese Anweisungen müssen eine Voraussetzung erfüllen: Sie müssen in einer Sub- oder Function-Prozedur oder innerhalb einer Property-Methode eingetragen werden. In diesem Artikel schauen wir uns die Grundlagen von Modulen, Klassenmodulen und Objektmodulen im Detail an.

Module unter VBA

Module sind im Prinzip Textdokumente, die Code aufnehmen können. Allerdings sind diese Textdokumente nicht in Form einzelner Dateien verfügbar, wie es beispielsweise bei anderen Entwicklungsumgebungen beziehungsweise Programmiersprachen der Fall ist, sondern sie sind alle in einer einzigen Datei gespeichert. Unter Access befinden sich die Module gemeinsam mit den übrigen Elementen wie Tabellen, Abfragen, Formularen und Berichten gleich in der **.accdb**-Datei. Unter Excel, Word und PowerPoint landen sie im jeweiligen Dokument. Unter Outlook, wo es keine Dateien wie bei den anderen Office-Anwendungen gibt, liegt das VBA-Projekt sogar in Form einer eigenen Datei vor.

Module lassen sich allerdings auch von den übrigen Office-Anwendungen als Textdateien exportieren und so kannst Du sie auch in anderen Office-Dokumenten beziehungsweise Access-Datenbanken wieder importieren.

Modularten

Es gibt eigentlich nur zwei Modularten:

- **Klassenmodule:** Dies sind Module mit einigen besonderen Eigenschaften. In der Regel müssen diese erst ini-

tialisiert und mit einer Objektvariablen referenziert werden, damit man mit den enthaltenen Elementen arbeiten kann. In manchen Fällen kann man jedoch auch direkt auf diese zugreifen, ohne dass man sie explizit initialisieren muss. Beispiele für Klassenmodule sind die Module von Formularen und Berichten unter Access, die Module für die Arbeitsmappe und die einzelnen Tabellen unter Excel, das Modul **ThisDocument** zu einem Word-Dokument oder auch das Modul **ThisOutlookSession** unter Outlook.

- **Standardmodule:** Standardmodule kann man den VBA-Projekten aller Office-Anwendungen hinzufügen. Sie müssen im Gegensatz zu Klassenmodulen nicht initialisiert werden, daher stehen die darin enthaltenen Variablen, **Sub**- und **Function**-Prozeduren und anderen Elemente jederzeit zur Verfü-

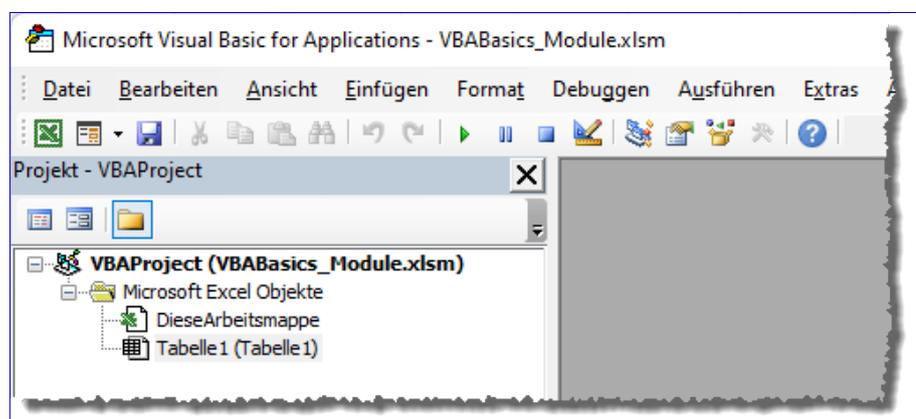


Bild 1: Verwalten von Modulen im Projekt-Explorer

VBA: Makros aufzeichnen

Das Programmieren von Office-Automatisierungen beispielsweise in der Sprache VBA kann für Einsteiger herausfordernd sein. Allerdings gibt es Möglichkeiten, sich hier und da zu behelfen: Die Anwendungen Excel, Word und PowerPoint bieten nämlich einen sogenannten Makro-Rekorder, mit dem man einfache Abläufe innerhalb der Anwendung aufzeichnen kann. Das Ergebnis ist eine VBA-Prozedur, die Du anschließend erneut aufrufen kannst. In vielen Fällen reicht dies bereits aus, um die gewünschten Schritte zu automatisieren, in anderen Fällen möchtest Du das Ergebnis des Makro-Recordsets vielleicht noch anpassen. Wie Du diesen Makro-Rekorder einsetzt und wie Du die Ergebnisse anpassen kannst, zeigt der vorliegende Artikel.

Werkzeuge zum Aufzeichnen von Makros einblenden

Bevor wir das erste Makro aufzeichnen können, ist zunächst eine kleine Anpassung erforderlich. Die Werkzeuge zum Aufzeichnen von Makros sind nämlich standardmäßig gar nicht sichtbar. Also schauen wir

uns anhand einer neuen, leeren Excel-Arbeitsmappe an, wie wir diese einblenden.

Diese Einstellung nehmen wir im Optionen-Dialog der jeweiligen Anwendung vor. Unter Excel klicken wir dazu auf den Registerreiter **Datei** und dann auf

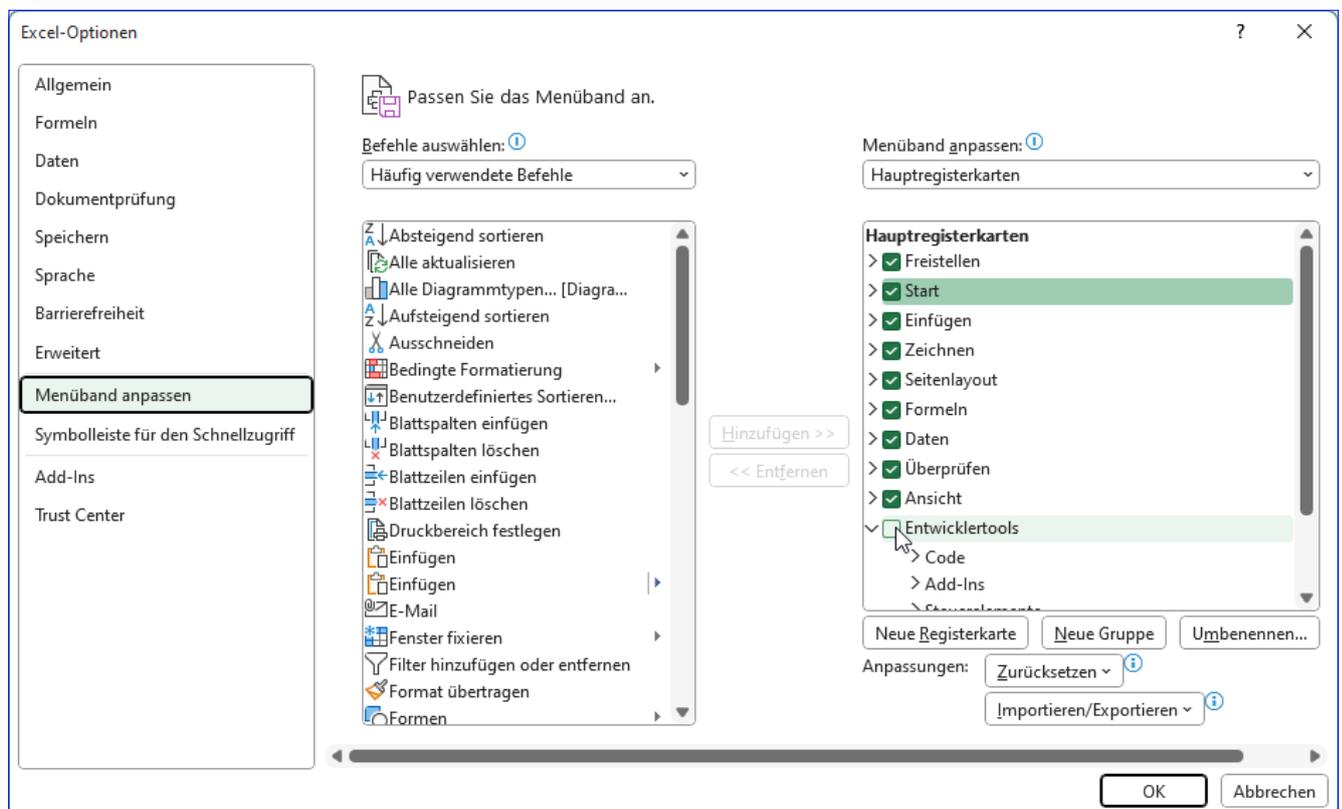


Bild 1: Einblenden der Entwicklertools und damit der Werkzeuge zum Aufzeichnen von Makros

VBA: MsgBox- und InputBox-Funktion

Die beiden Funktionen `MsgBox` und `InputBox` ermöglichen die schnelle Abfrage von Benutzerfeedback. Soll eine Datei überschrieben werden? Wie soll die neue Kategorie heißen? Das sind nur zwei von vielen Beispielen für den Einsatz dieser beiden Funktionen. Dieser Artikel beschreibt die beiden so kompakt wie möglich, damit Du sie schnell in Deine eigenen Anwendungen integrieren kannst.

Die MsgBox-Funktion

Die erste der beiden Funktionen hat zwei Haupteinsatzzwecke:

- Ausgabe von Informationen oder Warnungen an den Benutzer
- Ermitteln von einfachen Antworten des Benutzers wie OK oder Abbrechen durch entsprechende Schaltflächen

MsgBox zur Anzeige von Meldungen

Wenn Du einfach nur eine Meldung mit Informationen anzeigen möchtest, reicht der einfache Aufruf der `MsgBox`-Funktion unter Angabe der anzuzeigenden Informationen und gegebenenfalls eines Symbols.

Um dem Benutzer mehrere Auswahlmöglichkeiten für eine Reaktion zu geben, kannst Du verschiedene Schaltflächen anzeigen lassen und diese abfragen.

Der einfachste `MsgBox`-Aufruf gibt einfach nur einen Text aus und zeigt eine **OK**-Schaltfläche an:

```
MsgBox "Dies ist eine Beispielmeldung."
```

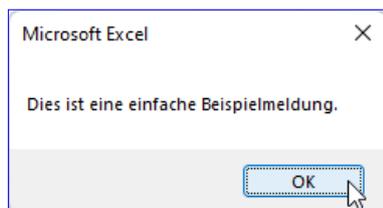


Bild 1: Einfache Meldung

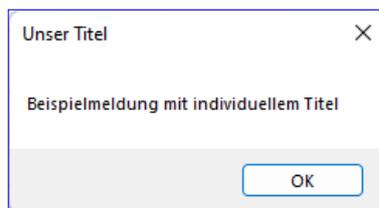


Bild 2: Meldung mit Titel

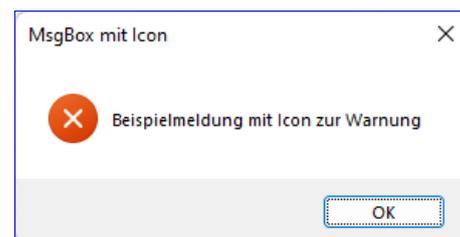


Bild 3: Meldung mit Icon

Diese können wir über den Direktbereich des VBA-Editors aufrufen oder auch in eine Prozedur oder Funktion einbinden. Das Ergebnis siehst Du in Bild 1.

MsgBox mit Titel

Die oben erzeugte Meldung zeigt den gewünschten Text an, und auch die Titelzeile ist gefüllt. Wenn Du den dortigen Text ersetzen willst, der standardmäßig den Namen der übergeordneten Anwendung anzeigt, brauchst Du nur den dritten Parameter der `MsgBox`-Funktion mit dem gewünschten Text zu füllen:

```
MsgBox "Beispielmeldung mit individuellem Titel", , _  
"Unser Titel"
```

Das Ergebnis siehst Du in Bild 2. Alternativ und um die unübersichtliche Anzeige der Kommata zwischen den Parametern zu entschärfen kannst Du auch benannte Parameter verwenden, wenn sich die Parameter nicht an der vorgesehenen Stelle befinden (für den ersten Parameter ist dies daher nicht erforderlich):

```
MsgBox "Beispielmeldung mit individuellem Titel", _  
Title:="Unser Titel"
```

Umfangreiche Texte in Code integrieren

Es gibt Aufgaben, um die schlägt man sich nicht. Eine davon ist es, größere Texte hart im VBA-Code zu verdrahten, sodass diese später weiterverarbeitet werden kann. Ein Beispiel ist das Zusammenstellen eines XML-Dokuments, von dem man eine Vorlage hat, und das man mit eigenen Werten füllen möchte, um es dann beispielsweise als Anfrage an einen Webservice zu schicken. Oder man möchte den Inhalt einer Mail per VBA an Outlook schicken und versenden, nachdem man den Mailtext noch um individuelle Inhalte wie Anrede oder Name des Empfängers ergänzt hat. Sprich: Uns liegt ein mehrzeiliger Text vor, den wir irgendwie in eine Variable packen wollen – und zwar ausschließlich per VBA-Code. Wie das gelingt, zeigt der vorliegende Artikel.

Mein aktuelles Beispiel, das mich zum Entwickeln der in diesem Artikel vorgestellten Lösung veranlasst hat, ist das Zusammenstellen eines XML-Dokuments. Wer schon einmal ein XML-Dokument mit der Anfrage an einen Webservice geschickt hat, kennt das vielleicht – in der Dokumentation des Webservice-Betreibers finden wir ein umfassendes XML-Dokument, das mit Beispieldaten gefüllt ist und das wir nur noch an unsere eigenen Zwecke angepasst werden soll. Eine Zeile darin lautet dann beispielsweise:

```
<password>pass</password>
```

Wir wollen diese samt den restlichen Zeilen des XML-Dokuments mit unseren realen Daten anpassen und das resultierende Dokument in einer Variablen speichern, um es dann an den Webservice zu schicken. Nur für diese eine Zeile würden wir dann eine Prozedur wie die folgende programmieren:

```
Public Sub XMLZusammenstellen()  
    Dim strText As String  
    strText = "<password>pass</password>" & vbCrLf  
    Debug.Print strText  
End Sub
```

Im nächsten Schritt würden wir dann das hier das fest im Code untergebrachte Kennwort **pass** durch einen Parameter ersetzen. Diese Version sieht wie folgt aus:

```
Public Sub XMLZusammenstellen(strPassword As String)  
    Dim strText As String  
    strText = "<password>" & strPassword _  
        & "</password>" & vbCrLf  
    Debug.Print strText  
End Sub
```

Dann rufen wir die Prozedur mit einem Parameterwert wie folgt auf:

```
XMLZusammenstellen "MeinKennwort"
```

Das liefert dann:

```
<password>MeinKennwort</password>
```

Für eine einzige Zeile ist der Aufwand zum Parametrisieren überschaubar, für ein paar weitere Zeilen vielleicht auch noch. Aber das XML-Dokument, das zu füllen war, sah in den VBA-Code kopiert zunächst wie in Bild 1 aus. Das heißt, wir müssen jede Zeile mit Ausnahme der ersten wie folgt ergänzen, also vorn um die Zuweisung des bisher in der Variablen gespeicherten Textes plus der neuen Zeile an die Variable:

```
strXML = strXML & " <soapenv:Header>" & vbCrLf
```

Wenn wir das für einige zig Zeilen erledigen müssen, wird es Zeit, über eine Automatisierung nach-

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

DHL-ETIKETTEN PER VBA ERSTELLEN

Programmiere die API von DHL für Geschäftskunden und erstelle Etiketten zum Beispiel direkt aus einer Datenbank heraus.

SEITE 52

DATEIEN AUSWÄHLEN LEICHT GEMACHT

Rufe die Dateiauswahl-Dialoge von Office per VBA auf und lerne die verschiedenen Optionen kennen.

SEITE 11

OUTLOOK-ANLAGEN VERARBEITEN

E-Mails kommen oft mit Datei-Anlagen. Wie Du diese automatisch im Dateisystem speicherst, zeigt dieser Artikel!

SEITE 34

SCHWERPUNKT WORD-PROGRAMMIERUNG

Grundlagen der Wordprogrammierung – vom Öffnen von Word über das Anlegen von Dokumenten und dem Bearbeiten von Texten, Tabellen bis zu Lösungen zum Übersetzen von Texten und dem Stellen von Rechnungen

SEITE 69



André Minhorst Verlag

COM-Add-In zum Übersetzen von Word-Dokumenten

Das Internet bietet für beinahe jeden Zweck einen Service. Auch für das Übersetzen von Texten. Ein Anbieter heißt DeepL. Er bietet nicht nur eine Benutzeroberfläche, in die man Texte eingeben kann, die direkt übersetzt werden, sondern auch noch eine API. Dieser können wir programmgesteuert Texte übergeben und die Übersetzung entgegennehmen. Das ist sehr praktisch für diejenigen Bereiche der Programmierung, wo man Texte in einer anderen oder in mehreren Sprachen braucht. In diesem Artikel wollen wir uns jedoch ansehen, wie wir aus den Techniken, die wir in weiteren Artikeln vorgestellt haben, ein COM-Add-In bauen, das sich nahtlos in die Word-Benutzeroberfläche eingliedert und auf Knopfdruck komplette Dokumente oder auch nur markierte Bereiche in die gewünschte Sprache übersetzt.

Übersetzen mit DeepL

Die Artikel, in denen wir auf die Grundlagen zu DeepL eingehen, findest Du unter **Excel: Übersetzungen mit DeepL** (www.vbentwickler.de/325), **Texte übersetzen mit DeepL** (www.vbentwickler.de/322) und **Übersetzen mit Word und DeepL** (www.vbentwickler.de/346).

Da DeepL seine API für bis zu 500.000 Zeichen pro Monat kostenlos anbietet, nutzen wir es gern als Beispiel für den Zugriff auf eine REST API. Wenn Du mehr als diese Anzahl Zeichen übersetzen möchtest, kannst Du den kostenpflichtigen Pro-Account nutzen. Beide schließt Du unter folgendem Link ab:

<https://www.deepl.com/de/pro#developer>

twinBASIC

Auch das Tool, das wir zum Erstellen des COM-Add-Ins nutzen, ist zumindest in der Variante, die das Erstellen von 32-Bit-Dateien erlaubt, kostenlos. Du findest weitere Informationen dazu zum Beispiel in **twinBASIC: Visual Basic für die Zukunft** (www.vbentwickler.de/310), **COM-Add-Ins mit twinBASIC** (www.vbentwickler.de/311) und **Ribbon-Signaturen für VBA, VB6 und twinBASIC** (www.vbentwickler.de/314).

Ziel des Artikels

Bevor wir in die Programmierung einsteigen, schauen wir uns den Plan an. Das Ziel ist es, dass wir per Mausklick auf einen Ribbon-Eintrag einen Übersetzungsvorgang starten können. Dieser Vorgang soll entweder den aktuell markierten Text übersetzen oder, wenn keine Markierung vorhanden ist, den Absatz übersetzen, in dem sich aktuell die Einfügemarke befindet.



Bild 1: Auswahl einer Vorlage

Rechnungen mit Word und Access

Office bietet verschiedene Möglichkeiten, um Rechnungen zu erstellen. Du kannst ein Worksheet in Excel dazu nutzen, eine Rechnung zu erstellen, die Berichte unter Access eignen sich recht gut dafür, aber auch Word lässt sich prima zum Erstellen von Rechnungen nutzen. Alle haben Vor- und Nachteile: Unter Excel gerät das Berechnen von Beträgen zum Kinderspiel, während mehrseitige Rechnungen eher schwierig zu realisieren sind, unter Access können Berichte recht flexibel realisiert werden, während man sich gerade für mehrseitige Berichte etwas einarbeiten muss – und unter Word kann man das Layout sehr einfach anpassen, während hier die Berechnung nicht so intuitiv erfolgt. Wir schauen uns eine Kombination aus Access und Word an: Die Daten stammen aus Word-Tabellen, während wir die Rechnung selbst in Word erstellen. So lassen sich auch nach dem Erstellen noch Feinheiten anpassen.

Rechnungsdaten von Access nach Word

Es gibt verschiedene Wege, wie wir Rechnungsdaten von Access aus in ein Word-Dokument bewegen können. Das eine Extrem wäre, das Word-Dokument aus der Access-Datenbank heraus von Grund auf neu zu erstellen und mit den gewünschten Inhalten zu füllen. Das andere Extrem lautet, das Word-Dokument so weit wie möglich vorzubereiten und mit Textmarken auszustatten, sodass wir nur noch die eigentlichen Texte aus der Datenbank heraus zum Word-Dokument schicken müssen.

Der erste Weg wäre sicherlich eine schöne Herausforderung für einen Software-Entwickler. Allerdings wollen wir Lösungen entwickeln, mit denen der Benutzer der Anwendung optimal arbeiten kann. Und wenn wir das Word-Dokument mit einer Rechnung komplett per Code generieren, gäbe es für den Benutzer nur noch nachträglich die Möglichkeit, das Dokument überhaupt anzupassen. Wenn er jedoch Änderungen vornehmen möchte, die sich in jedem Dokument niederschlagen, dann müsste er diese für jedes Dokument nach dessen Erstellung erneut durchführen.

Aus Benutzersicht wäre also der zweite Weg hilfreicher, denn je mehr Inhalte bereits in der zu verwendenden

Word-Vorlage enthalten sind, desto mehr Elemente kann der Benutzer selbst direkt an der Vorlage anpassen – und wenn die Änderungen einmal durchgeführt sind, wirken sich diese auf alle Dokumente aus, die wir von Access aus erstellen. Auf diese Weise können wir dem Benutzer sogar ermöglichen, mehr als eine Vorlage zu erstellen und in der Access-Anwendung auszuwählen, welche Vorlage für welche Rechnung verwendet werden soll.

Aber welche Elemente können wir überhaupt bereits in der Dokumentvorlage für die Rechnung unterbringen? Wir gehen davon aus, dass die Rechnungen nur im Namen einer einzigen Person oder eines einzigen Unternehmens gestellt werden.

Dann können wir zumindest den Briefkopf und die Absenderadresse in dem Bereich, der später im Fenster des Briefumschlags sichtbar sein soll, anpassen.

Außerdem kann der Benutzer die Anordnung der übrigen Elemente wie Betreff, Rechnungstext, Block mit den Rechnungspositionen und die Zahlungsbedingungen nach den eigenen Anforderungen festlegen – und damit auch die Formatierungen wie Schriftart, Schriftgröße et cetera definieren.

Rechnungen mit Word und Access

Office bietet verschiedene Möglichkeiten, um Rechnungen zu erstellen. Du kannst ein Worksheet in Excel dazu nutzen, eine Rechnung zu erstellen, die Berichte unter Access eignen sich recht gut dafür, aber auch Word lässt sich prima zum Erstellen von Rechnungen nutzen. Alle haben Vor- und Nachteile: Unter Excel gerät das Berechnen von Beträgen zum Kinderspiel, während mehrseitige Rechnungen eher schwierig zu realisieren sind, unter Access können Berichte recht flexibel realisiert werden, während man sich gerade für mehrseitige Berichte etwas einarbeiten muss – und unter Word kann man das Layout sehr einfach anpassen, während hier die Berechnung nicht so intuitiv erfolgt. Wir schauen uns eine Kombination aus Access und Word an: Die Daten stammen aus Access-Tabellen, während wir die Rechnung selbst in Word erstellen. So lassen sich auch nach dem Erstellen noch Feinheiten anpassen.

Rechnungsdaten von Access nach Word

Es gibt verschiedene Wege, wie wir Rechnungsdaten von Access aus in ein Word-Dokument bewegen können. Das eine Extrem wäre, das Word-Dokument aus der Access-Datenbank heraus von Grund auf neu zu erstellen und mit den gewünschten Inhalten zu füllen. Das andere Extrem lautet, das Word-Dokument so weit wie möglich vorzubereiten und mit Textmarken auszustatten, sodass wir nur noch die eigentlichen Texte aus der Datenbank heraus zum Word-Dokument schicken müssen.

Der erste Weg wäre sicherlich eine schöne Herausforderung für einen Software-Entwickler. Allerdings wollen wir Lösungen entwickeln, mit denen der Benutzer der Anwendung optimal arbeiten kann. Und wenn wir das Word-Dokument mit einer Rechnung komplett per Code generieren, gäbe es für den Benutzer nur noch nachträglich die Möglichkeit, das Dokument überhaupt anzupassen. Wenn er jedoch Änderungen vornehmen möchte, die sich in jedem Dokument niederschlagen, dann müsste er diese für jedes Dokument nach dessen Erstellung erneut durchführen.

Aus Benutzersicht wäre also der zweite Weg hilfreicher, denn je mehr Inhalte bereits in der zu verwendenden

Word-Vorlage enthalten sind, desto mehr Elemente kann der Benutzer selbst direkt an der Vorlage anpassen – und wenn die Änderungen einmal durchgeführt sind, wirken sich diese auf alle Dokumente aus, die wir von Access aus erstellen. Auf diese Weise können wir dem Benutzer sogar ermöglichen, mehr als eine Vorlage zu erstellen und in der Access-Anwendung auszuwählen, welche Vorlage für welche Rechnung verwendet werden soll.

Aber welche Elemente können wir überhaupt bereits in der Dokumentvorlage für die Rechnung unterbringen? Wir gehen davon aus, dass die Rechnungen nur im Namen einer einzigen Person oder eines einzigen Unternehmens gestellt werden.

Dann können wir zumindest den Briefkopf und die Absenderadresse in dem Bereich, der später im Fenster des Briefumschlags sichtbar sein soll, anpassen.

Außerdem kann der Benutzer die Anordnung der übrigen Elemente wie Betreff, Rechnungstext, Block mit den Rechnungspositionen und die Zahlungsbedingungen nach den eigenen Anforderungen festlegen – und damit auch die Formatierungen wie Schriftart, Schriftgröße et cetera definieren.

Übersetzen mit Word und DeepL

Nachdem wir schon gezeigt haben, wie Du Texte in einer Excel-Tabelle übersetzen kannst, schauen wir uns nun an, wie die DeepL-Techniken für die Gegenüberstellung der englischen und der deutschen Version eines Textes nutzen können. Dazu wollen wir bei einem Text beginnen, der sich in einem Word-Dokument befindet und diesen in ein neues Dokument übertragen, wo wir die einzelnen Absätze des Textes in jeweils eine Zelle in der linken Spalte einer Word-Tabelle einfügen. Nachdem wir den Text so auf die Zellen einer Tabelle aufgeteilt haben, wollen wir in der rechten Spalte per Aufruf der DeepL-REST-API die Übersetzung in der gewünschten Sprache unterbringen. Das Ergebnis: Die Übersetzung eines Textes, wobei jeweils ein Absatz in der einen Sprache dem gleichen Absatz in der anderen Sprache gegenübergestellt wird.

Den VBA-Code, mit dem wir die gewünschten Aufgaben erledigen, fügen wir zunächst in das Klassenmodul **ThisDocument** des aktuellen Dokuments ein. Hier wollen wir einen Dateiauswahldialog unterbringen, mit dem der Benutzer die Quelldatei für den Ausgangstext selektieren kann.

Diese enthält einfache Absätze mit Texten wie beispielsweise wie in Bild 1. Anschließend soll die Prozedur die Inhalte dieses Dokuments einlesen und in eine neue Tabelle im aktuellen Dokument schreiben – und zwar auf die linke Seite. Eine zweite Prozedur soll alle in der linken Spalte der Tabelle enthaltenen Texte einlesen und mit einem Aufruf der REST-API von DeepL den Text in der gewünschten Sprache ermitteln. Die Übersetzung des aktuellen Absatzes soll dann in die zweite Spalte der Tabelle neben dem Originaltext eingefügt werden.

Hauptprozedur zur Steuerung des Ablaufs

Die Hauptprozedur, die wir einfach aufrufen und welche die übrigen Funktionen und Prozeduren startet, heißt

Uebersetzen. Sie ruft als Erstes die Funktion **Dateiauswählen** auf, welche den Dateiauswahldialog öffnet und mit dem der Benutzer die Word-Datei auswählt, aus welcher der zu übersetzende Text eingelesen werden soll (mehr dazu im Anschluss).

Der Pfad zu dieser Datei wird in der Variablen **strPfad** gespeichert. Die Prozedur **Uebersetzen** prüft anschlie-

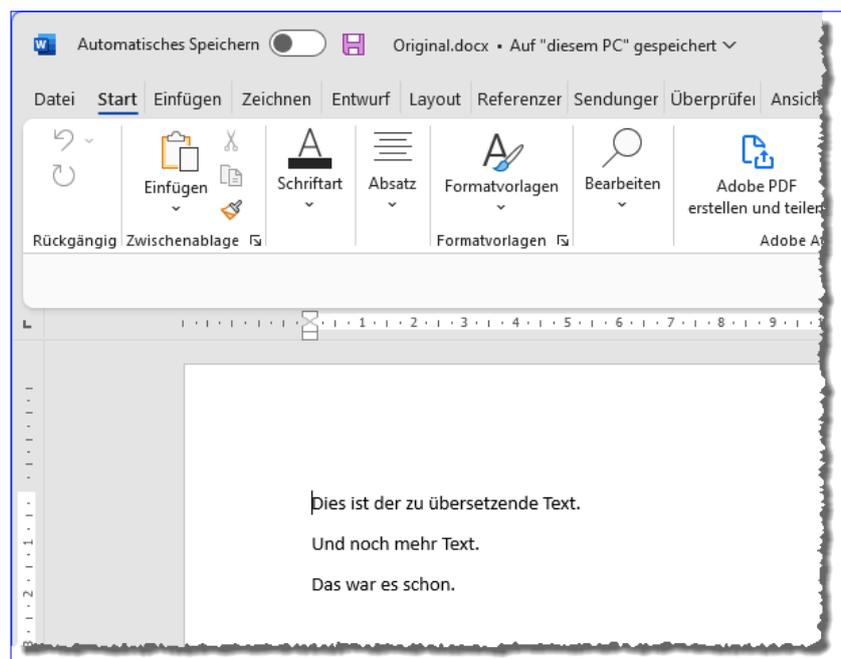


Bild 1: Beispiel für ein Ausgangsdokument

VBA-Ereignisse in Word programmieren

Microsoft Word bietet für das Application- und für das Document-Objekt einige sehr interessante Ereignisse an. Diese können wir implementieren und damit auf verschiedene Benutzeraktionen wie das Öffnen, Speichern oder Schließen eines Dokuments oder auch auf andere Aktionen zu reagieren. In diesem Artikel stellen wir die Ereignisse vor und zeigen, wie Du diese implementieren und mit eigenem Code füllen kannst. Damit lernst Du, wie Du auf das Öffnen, Schließen oder Anlegen von Dokumenten reagieren kannst oder wie sich Ereignisse programmieren lassen, die beim Drucken oder Speichern ausgelöst werden. Auch Aktionen wie ein Doppelklick oder ein Rechtsklick auf die Inhalte des Word-Dokuments lassen sich damit steuern. Wie das gelingt, zeigen wir in diesem Artikel.

Voraussetzung: Das WithEvents-Schlüsselwort

Wenn wir Ereignisprozeduren für die Ereignisse einer Word-Instanz implementieren wollen, benötigen wir vor allem eines: Die Deklaration einer Objektvariablen für die Word-Instanz, und zwar mit dem Schlüsselwort **WithEvents** versehen:

```
Private WithEvents objWord As Word.  
Application
```

Diese Anweisung können wir nur in einem Klassenmodul platzieren, nicht jedoch in einem Standardmodul.

Das Modul **ThisDocument** des VBA-Projekts eines Word-Dokuments beispielsweise ist jedoch ein Klassenmodul, in dem wir diese Anweisung unterbringen können.

Die Änderung durch das **WithEvents**-Schlüsselwort ist nicht direkt sichtbar. Wenn wir allerdings auf das linke Auswahlfeld oben im Code-Fenster klicken, sehen wir nun die auf diese Weise deklarierte Variable **objWord**. Die Auswahl dieses Eintrags legt auto-

matisch eine Prozedur für das Standard-Ereignis dieses Objekts an, in diesem Fall **objWord_Quit**:

```
Private Sub objWord_Quit()  
  
End Sub
```

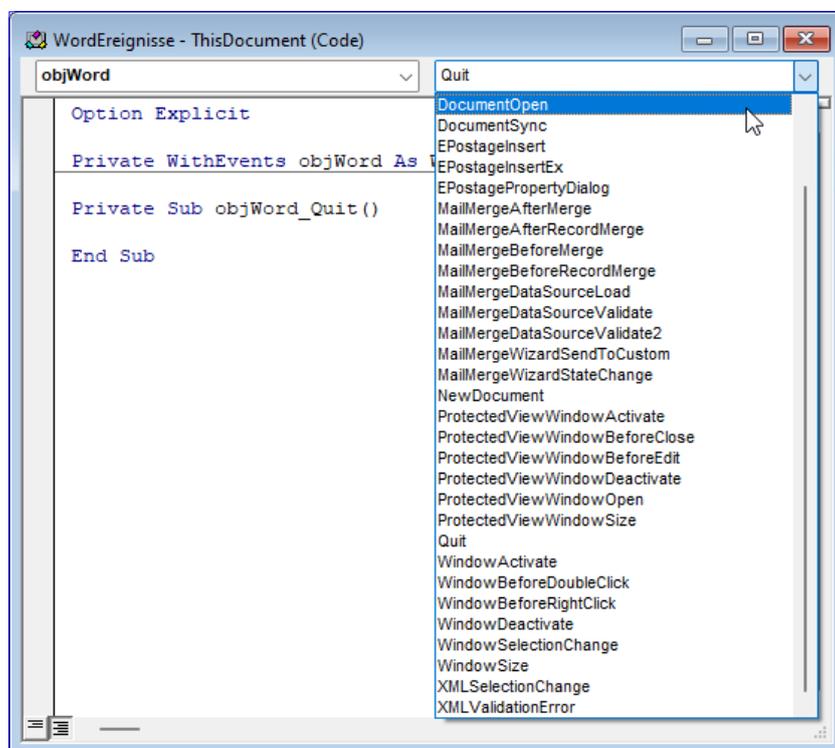


Bild 1: Auswahl der Ereignisse des Application-Objekts

Word: Lesen und Schreiben per VBA

Nachdem wir in zwei weiteren Artikeln beschrieben haben, wie man auf Word-Dokumente zugreift und die Ereignisse von Word und den angezeigten Dokumenten nutzt, gehen wir nun einen Schritt weiter: Wir schauen uns an, wie wir die Inhalte von Word-Dokumenten lesen und schreiben können. Das ist nicht ganz trivial, denn natürlich enthält ein Word-Dokument nicht einfach nur Text, den wir in eine Variable übertragen und weiterverarbeiten können. Stattdessen gibt es eine Unterteilung in Absätze, Bereiche, Tabellen und viele weitere Elemente. Der Fokus dieses Artikels liegt daher auf der Untersuchung dieser Elemente und wie wir diese am besten per VBA programmieren können.

Vorbereitung für die Programmierung von Dokumentinhalten

Da wir bisher per VBA weder Texte in ein Word-Dokument geschrieben noch gelesen haben, starten wir systematisch – wir schreiben diese zuerst in ein Dokument und lesen die Inhalte dann ebenfalls per VBA aus.

Die nachfolgenden Prozeduren wollen wir nicht von einer anderen Anwendung aus ausführen, sondern diese einfach in das VBA-Projekt des geöffneten Word-Dokuments eintragen und dort aufrufen. Wir greifen dabei mit dem Verweis **ThisDocument** auf das aktuelle Word-Dokument zu.

Dokument explizit referenzieren

Wenn Du die nachfolgenden Beispiele auf ein anderes Dokument anwenden möchtest als das mit **ThisDocument** referenzierte Dokument, weil Du beispielsweise von einer anderen Anwendung aus arbeiten möchtest, verwendest Du eine wie folgt deklarierte Variable dazu:

```
Dim objDocument As Word.Document
```

Dieses referenzierst Du bei einem neu zu erstellenden Dokument beispielsweise wie folgt:

```
Dim objWord As Word.Application
```

```
Set objWord = New Word.Application  
Set objDocument = objWord.Documents.Add
```

Weitere Möglichkeiten zum Referenzieren eines Word-Dokuments zeigen wir im Artikel **Word mit VBA programmieren** (www.vbentwickler.de/348).

Text in ein Dokument einfügen

Die einfachste Möglichkeit, einen Text in das neue Word-Dokument einzufügen, ist die folgende Anweisung:

```
ThisDocument.Range.Text = "Dies ist ein erster Text."
```

Wir greifen also über **ThisDocument** auf das **Document**-Objekt zu, referenzieren dann das **Range**-Objekt des Dokuments und legen seinen Text fest.

Diese und die folgenden Anweisungen kannst Du in den meisten Fällen innerhalb einer Prozedur ausführen, aber auch als einzelne Anweisung in den Direktbereich des VBA-Editors eingeben und ausführen. Wir können übrigens auch die abgekürzte Form der obigen Anweisung nutzen, denn **Text** ist die Standardeigenschaft des **Range**-Objekts und kann somit weggelassen werden:

```
ThisDocument.Range = "Dies ist ein erster Text."
```

Word mit VBA programmieren

Genau wie in Outlook oder Excel gibt es auch in Word ausreichende Möglichkeiten, um sich wiederholende Vorgänge zu automatisieren oder zu vereinfachen. Dazu benötigen wir die Programmiersprache VBA und das Objektmodell zur Programmierung von Word und den darin angezeigten Dokumenten. Im vorliegenden Beitrag schauen wir uns zunächst einmal an, wie wir von außen, also beispielsweise von einer anderen Anwendung aus, eine Word-Instanz initialisieren und diese nutzen können. Bei den weiteren Beispielen starten wir jedoch direkt vom VBA-Projekt des aktuell geöffneten Word-Dokuments aus.

Vorbereitung

Die meisten der nachfolgend beschriebenen Techniken kannst Du sowohl von einem externen VBA-Projekt, also beispielsweise von Excel, Access oder Outlook oder auch von einer anderen Word-Instanz aus aufrufen.

Wenn Du jedoch eine andere Anwendung als Word nutzt, benötigst Du für den Early Binding-

Zugriff, der die Nutzung von IntelliSense ermöglicht, einen Verweis auf die Bibliothek **Microsoft Word 16.0 Object Library**. Um diesen im jeweiligen VBA-Projekt einzurichten, aktivierst Du den VBA-Editor mit **Alt + F11**, wählst den Menüeintrag **Extras|Verweise** und fügst dort den genannten Eintrag hinzu.

Wenn Du im VBA-Projekt eines Word-Dokuments programmierst, musst Du diesen Verweis nicht hinzufügen, da er standardmäßig bereits enthalten ist.

Word-Instanz erstellen

Word ist wie Excel und im Gegensatz zu Outlook eine Anwendung, von der wir mehrere Instanzen erstellen

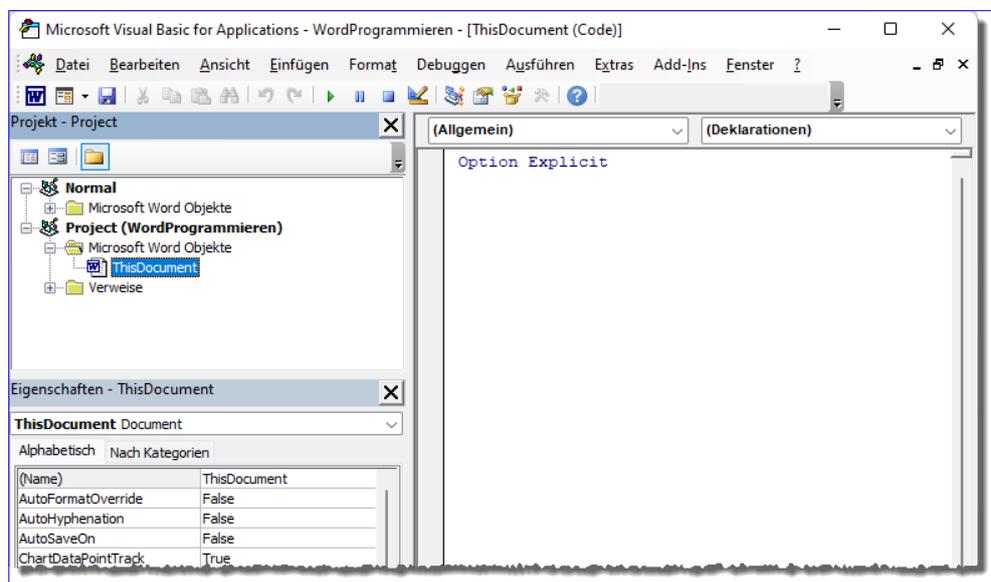


Bild 1: Der VBA-Editor mit einem Word-Projekt

können. Wenn Outlook bereits geöffnet ist und wir versuchen, mit **Set objOutlook = New Outlook.Application** eine neue Instanz von Outlook zu erstellen, erhalten wir lediglich einen Verweis auf die aktuell geöffnete Instanz. Wenn wir hingegen wie folgt eine Word-Objektvariable definieren und dieser mit dem Schlüsselwort **New** eine neue Instanz zuweisen, erhalten wir tatsächlich eine weitere Instanz von Word.

Diese machen wir schließlich auch noch sichtbar:

```
Dim objWord As Word.Application
Set objWord = New Word.Application
objWord.Visible = True
```

Word: Tabellen mit VBA programmieren

Tabellen sind ein wichtiges Element zur Strukturierung in Dokumenten – so auch in Microsoft Word. Tabellen fügt man dabei normalerweise über die Benutzeroberfläche ein und füllt diese manuell mit den gewünschten Daten. Manchmal jedoch sollen Dokumente und damit auch Tabellen automatisiert erstellt werden – beispielsweise, wenn man Daten aus Anwendungen wie Access oder Excel in einem Word-Dokument weiterverarbeiten möchte. Damit lassen sich beispielsweise gut die Rechnungspositionen in einer Rechnung oder auch Katalogdaten abbilden. In einer Lösung in einem weiteren Artikel nutzen wir eine Tabelle, um die einzelnen Absätze verschiedener Übersetzungen eines Textes anzuzeigen. Im vorliegenden Artikel zeigen wir, wie wir einem Word-Dokument eine Tabelle hinzufügen und diese mit den gewünschten Zeilen und Spalten versehen können – und welche Techniken noch sinnvoll sein können, wenn Du Tabellen programmierst.

Word-Tabellen per Benutzeroberfläche

Üblicherweise fügt man Tabellen in Word über den Ribboneintrag **Einfügen|Tabellen|Tabelle** ein, wo man im oberen Bereich direkt die gewünschte Anzahl von Zeilen und Spalten auswählen kann (siehe Bild 1).

Unter VBA ist dies auch gar nicht viel komplizierter – wie es genau geht, schauen wir uns in den folgenden Abschnitten an.

Vorbereitung

Für die Beispiele dieses Artikels brauchen eine **.dotm**-Datei, also ein Word-Dokument mit VBA-Modul.

In den meisten Beispielen dieses Artikels gehen wir davon aus, dass wir die Beispielaufrufe und Prozeduren im VBA-Projekt zum geöffneten Word-Dokument ausführen.

Dabei verwenden wir zum Referenzieren des Word-Dokuments beispielsweise **ThisDocument**. Wenn Du die

Beispiele von einer anderen Anwendung aus ausprobieren möchtest, musst Du noch einen Verweis auf die Word-Instanz und das Dokument setzen – weitere Informationen dazu findest Du im Artikel **Word mit**

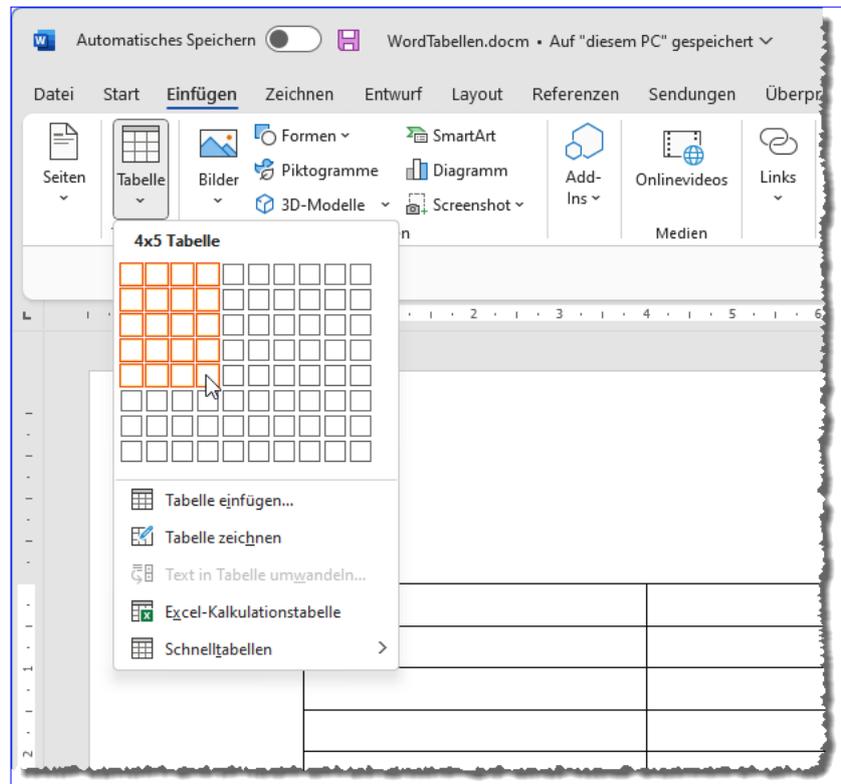
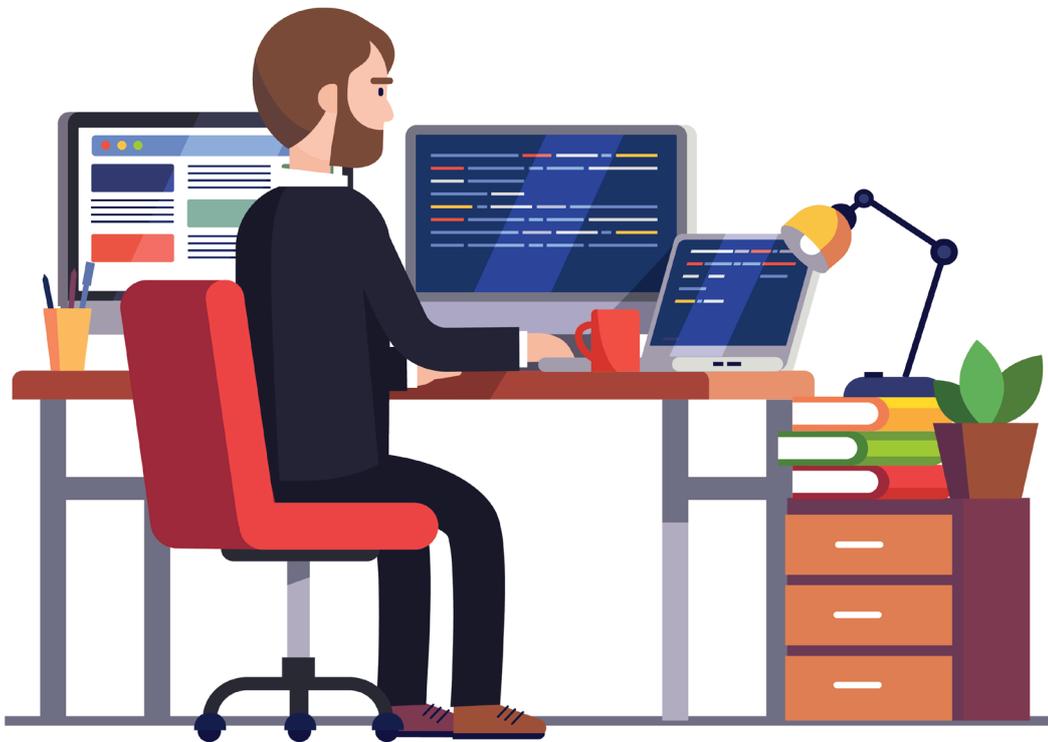


Bild 1: Hinzufügen einer Word-Tabelle über die Benutzeroberfläche

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

EXCEL ERWEITERN

Lerne, wie Du benutzerdefinierte Funktionen als Formeln verwendest und wie Du eigene Prozeduren über das Ribbon verfügbar machst.

SEITE 4

OPENAI UND VBA

Stelle per VBA Fragen an die künstliche Intelligenz und werte sie aus. Nutze dazu unsere Lösung für das Analysieren von JSON-Dateien!

SEITE 12

EINFACHE .NET-ANWENDUNGEN

Programmiere oft wiederkehrende Aufgaben mit .NET und erstelle eine .exe-Datei daraus, die Du überall nutzen kannst.

SEITE 24



André Minhorst Verlag

AfterUpdate für WPF-Steuerelemente mit VB.NET

Wenn man einfache Anwendungen wie im Artikel »Standalone-Apps mit .NET programmieren« erstellt, stößt man relativ schnell an Grenzen. Eine davon sind die verfügbaren Ereignisse für Steuerelemente. Wer einmal mit Access gearbeitet hat, weiß, dass es für jedes Steuerelement ein Ereignis wie »Nach Aktualisierung« gibt. Unter WPF ist das nicht der Fall, was an der zugrunde liegenden Philosophie liegt. Diese lautet, dass Benutzeroberfläche und Anwendungslogik so weit wie möglich getrennt werden sollen. In diesem Artikel zeigen wir anhand eines Beispiels, wie sich dies in der Praxis auswirkt.

Im Artikel **Standalone-Apps mit .NET programmieren** (www.vbentwickler.de/358) haben wir grundlegend gezeigt, wie man .exe-Dateien zum Erledigen kleiner oder auch größerer Aufgaben mit Visual Studio programmieren kann. In einem weiteren Artikel namens **Anwendungsdaten speichern per VB.NET** (www.vbentwickler.de/359) haben wir darauf aufbauend eine kleine Anwendung erzeugt, mit der wir die Daten in Textfeldern und anderen Steuerelementen als Anwendungskonfigurationsdaten speichern können.

Allerdings haben wir diese immer nur komplett entweder nach einem Klick auf eine dafür vorgesehene **Speichern**-Schaltfläche oder beim Schließen der Anwendung gespeichert.

Dort haben wir auch festgestellt, dass es unter WPF nicht für alle Steuerelemente Ereignisse gibt, die nach der Aktualisierung des Inhalts ausgelöst werden. Bei einem **TextBox**-Element können wir zwar das **TextChanged**-Ereignis nutzen, das nach der Eingabe eines jeden Zeichens ausgelöst wird, aber den Inhalt nach jeder nicht durch die Eingabetaste bestätigten Änderung zu speichern, wäre doch etwas übertrieben.

Also schauen wir uns im vorliegenden Artikel an, wie man solche Aufgaben unter Berücksichtigung der Philosophie von WPF löst. Und dabei steigen wir nicht in die Tiefe ein – das würde bedeuten, das **MVVM**-Entwurfsmuster zu erläutern (**Model-View-ViewModel**).

Wir wollen nun die Daten der Steuerelemente ohne Ereignisse der Steuerelemente selbst und nur mit Datenbindung in der Konfigurationsdatei speichern und diese beim Starten der Anwendung wiederherstellen.

Elemente bei der WPF-Datenbindung

Bei der WPF-Datenbindung benötigen wir die folgenden Elemente:

- Ein Steuerelement, dessen Eigenschaft, über die der Inhalt festgelegt wird, an eine Eigenschaft des Code behind-Moduls gebunden wird.
- Eine Eigenschaft im Code behind-Modul, die ausgelesen und beschrieben werden kann.
- Die Angabe, an welche Klasse das XAML-Fenster gebunden wird und woher es seine Daten beziehen soll – in unserem Fall die Code behind-Klasse.

Nachfolgend schauen wir uns die dazu notwendigen Schritte an. Dabei gehen wir davon aus, dass Du bereits ein neues Projekt des Typs **WPF-App** (.NET Framework) erstellt hat und das Fenster **MainWindow.xaml** in der Entwurfsansicht angezeigt wird.

Außerdem kannst Du bereits einmal im Projektmappen-Explorer den Eintrag **MainWindow.xaml** erweitern, sodass Du hier die Code behind-Datei **MainWindow.xaml.vb** siehst. Diese öffnen wir per Doppelklick.

Excel: Add-In mit Ribbon-Button erstellen

Wir können Excel nicht nur um benutzerdefinierte Funktionen erweitern, die wir in eine Add-In-Datei schreiben und die dann überall als Formel verfügbar sind. Wir können auch Funktionen hinzufügen, mit denen wir die Anwendung um selbst programmierte Abläufe erweitern. In diesem Artikel zeigen wir, wie Du einem Excel-Add-In ein eigenes Ribbontab hinzufügst sowie eine Funktion, die durch einen Button des Ribbontabs ausgelöst wird.

Wenn Du nicht nur eigene Funktionen benötigst, die Du als Formeln in Excel nutzen möchtest, sondern auch noch VBA-Prozeduren programmiert hast, die Du gern in allen Excel-Workbooks auf Deinem Rechner einsetzen willst, findest Du in diesem Artikel die benötigten Vorgehensweise. Wir werden:

- eine Excel-Add-In-Datei erstellen,
- diese verfügbar machen,
- der Datei ein Ribbon mit einer Schaltfläche und einem Icon hinzufügen sowie
- eine Prozedur anlegen, die durch diese Schaltfläche ausgelöst wird.

Excel-Add-In-Datei erstellen

Das Erstellen der Add-In-Datei ist in wenigen Sekunden erledigt:

- Lege ein neues Excel-Workbook an.
- Speichere es unter dem Namen **ExcelAddInMitRibbon.xlam** als **Excel-Add-In (*.xlam)**.

Fertig! Die neue Add-In-Datei wird automatisch in das Verzeichnis `C:\Users\[Benutzername]\AppData\Roaming\Microsoft\AddIns` verschoben.

Excel-Add-In verfügbar machen

Um das Excel-Add-In in Excel auf dem aktuellen Rechner verfügbar zu machen, nehmen wir eine Einstellungen in den Optionen von Excel vor:

- Klicke im Ribbon auf **Datei|Optionen**.
- Wechsle zum Bereich **Add-Ins**.
- Klicke unter **Verwalten** für den Eintrag **Excel-Add-Ins** auf die Schaltfläche **Los...**

- Suche den Eintrag **Exceladdinmitribbon** aus und setze einen Haken vor diesen Eintrag (siehe Bild 1) und schließe den Dialog und die Optionen wieder.

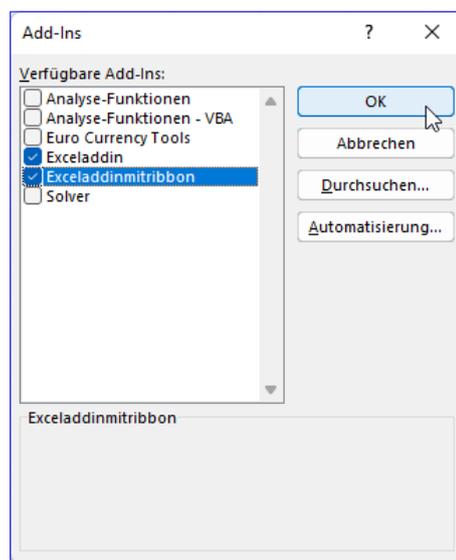


Bild 1: Aktivieren des Add-Ins

Ribbon hinzufügen

Nun schließen wir die Add-In-Datei und öffnen ein Tool namens **Office RibbonX Editor** – mehr dazu siehe im Artikel **Ribbons in Office-Dokumenten** (www.vbentwickler.de/329).

- Hier öffnen wir zuerst die frisch erstellte Excel-Add-In-Datei.

- Danach erscheint im linken Bereich des **Office RibbonX Editors** ein Eintrag mit dem Namen der Daten.

Excel: Benutzerdefinierte Funktionen per Add-In

Benutzerdefinierte VBA-Funktionen lassen sich leicht zu einem Excel-Workbook hinzufügen. Sie sind dann aber normalerweise nur in dem entsprechenden Workbook verfügbar. Was aber, wenn Du richtig coole Funktionen entwickelt hast, die Du nicht nur in einem Workbook nutzen möchtest, sondern in verschiedenen Dateien – und Du hast keine Lust, den VBA-Code immer wieder in das VBA-Projekt neuer Workbooks zu kopieren? In diesem Fall gibt es gute Nachrichten: Excel bietet den Dateityp Excel-Add-In an. Darin kannst Funktionen definieren, die immer verfügbar sind.

UDF oder Userdefined Functions

Was wir in der Überschrift **Benutzerdefinierte Funktionen** genannt haben, findest Du auf den englischen Excel-Seiten unter der Bezeichnung **User-defined Functions**, kurz **UDF**. Wir verwenden in diesem und auch in anderen Artikeln jedoch die deutsche Bezeichnung **Benutzerdefinierte Funktion**.

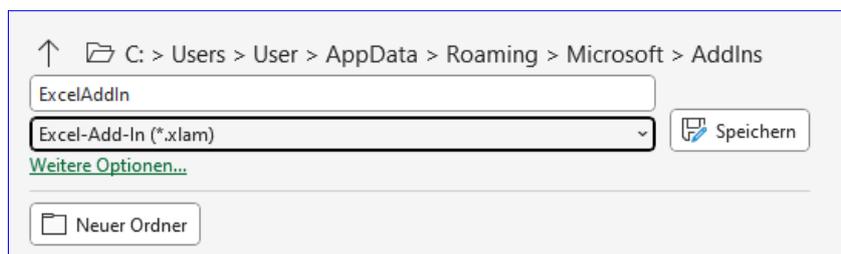


Bild 1: Speichern als Excel-Add-In

Anlegen eines Excel-Add-Ins

Das Erstellen eines Add-Ins besteht erst einmal in einem ganz kleinen Schritt: Dem Speichern einer neuen,

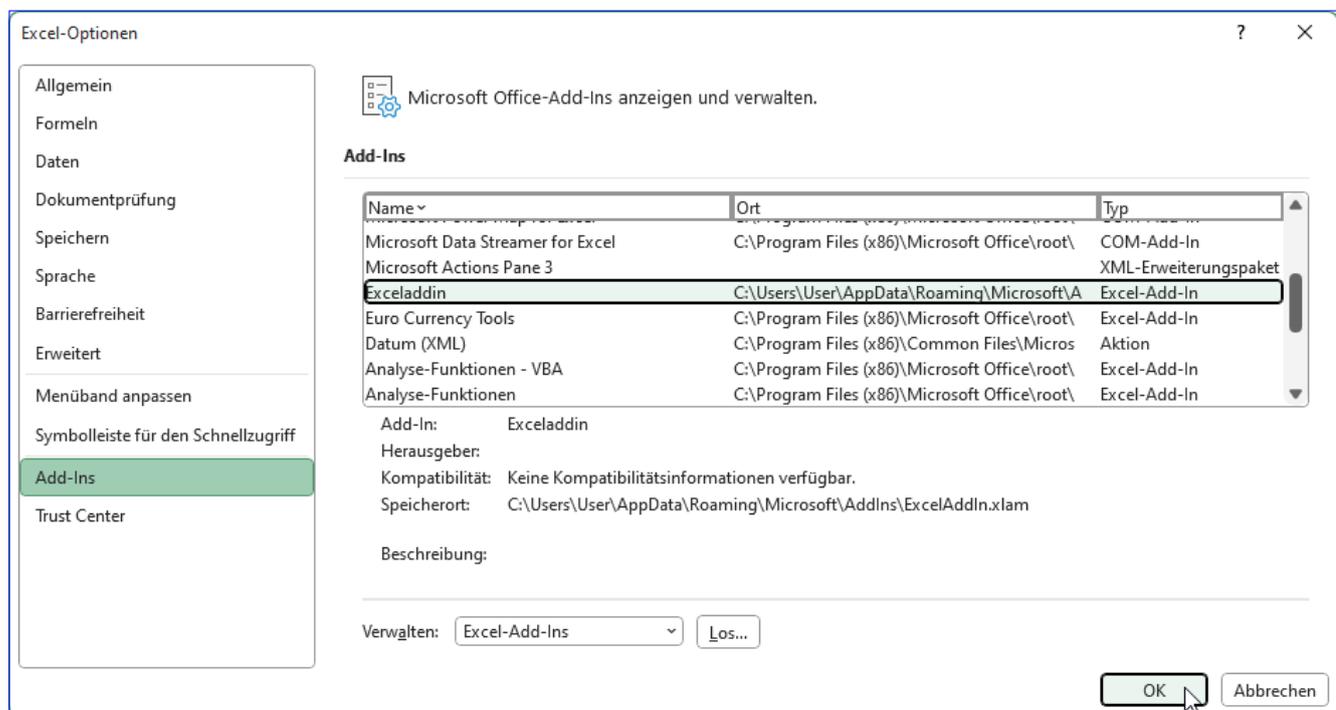


Bild 2: Übersicht der Office-Add-Ins

Mit JSON arbeiten

JSON heißt JavaScript Object Notation und ist wie XML ein Format, mit dem Daten strukturiert gespeichert werden können. Als Visual Basic-Entwickler hat man üblicherweise nicht viele Berührungspunkte mit dieser Notation. Wenn man jedoch gelegentlich mit Webservices beziehungsweise Rest APIs arbeitet, findest du Datenaustausch entweder mit XML oder JSON statt. Während es für den Zugriff auf den Inhalt von XML-Dokumenten die »Microsoft XML, vX.0«-Bibliothek gibt, ist man bei JSON auf Lösungen von Drittherstellern angewiesen. In diesem Fall nutzen wir eine Bibliothek von Tim Hall und eine Erweiterung im Eigenbau, mit der wir relativ einfach auf die Daten in JSON-Dokumenten zugreifen können.

Wer mit Rest APIs wie mit dem aktuell gefragten Webservice von OpenAI arbeitet, bekommt von diesen Antworten entweder im XML- oder im JSON-Format. Wie Du die Rest API von OpenAI ansprichst, um die dahinter stehende, sogenannte »künstliche Intelligenz« zu nutzen, erklären wir im Artikel **OpenAI mit VBA**

(www.vbentwickler.de/355). Das Ergebnis eines solchen Aufrufs sieht wie in Listing 1 aus.

Ein solches Ergebnis können wir in eine **String**-Variable einfügen und dann mithilfe von Zeichenketten-Funktionen die gewünschten Informationen heraus-

```
{
  "id": "cmp1-6kCY6I3xWbEU1jecYvr9nKV5GsY3s",
  "object": "text_completion",
  "created": 1676469346,
  "model": "text-davinci-003",
  "choices": [
    {
      "text": "\n\nLeider gibt es keine native OpenAI-Integration in Microsoft Office. Wenn Sie OpenAI jedoch in Microsoft Office verwenden möchten, können Sie versuchen, eine Lösung zu implementieren, die OpenAI-APIs verwendet, um die Anwendungen in Ihrer Office-Suite zu erweitern. Solche Lösungen beinhalten normalerweise die Entwicklung eigener Apps und Add-Ins, die OpenAI-APIs verwenden, um bestimmte Funktionen in Microsoft Office bereitzustellen.",
      "index": 0,
      "logprobs": null,
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 14,
    "completion_tokens": 146,
    "total_tokens": 160
  }
}
```

Listing 1: Beispiel für eine JSON-Datei

OpenAI mit VBA

Schlagwörter wie OpenAI oder ChatGPT beherrschen die Schlagzeilen der Welt. Wir wollen nicht entscheiden, ob es gut oder schlecht ist, ob es Dir den Job wegnimmt oder ob es eine super Unterstützung ist, sondern wir zeigen in diesem Artikel einfach, wie Du es per VBA steuern und in eigenen Anwendungen nutzen kannst. Dabei greifen wir auf die Rest API von OpenAI zu und ermöglichen, dass Du die Antworten auf die mit der Anwendung eingegebene Frage nutzen kannst.

Ob ChatGPT, OpenAI et cetera nun nützlich für uns Entwickler sind oder nicht: Wir können per Rest API darauf zugreifen, also ist es auf jeden Fall interessant! Den Einstieg findest Du unter folgendem Link:

<https://openai.com/api/>

Entweder Du hast schon einen Zugang, dann meldest Du Dich dort unter **Log In** an, oder Du registrierst Dich auf der Seite mit dem Button **Sign up**. Dazu benötigst Du einen Google- oder Microsoft-Account oder Du registrierst Dich neu mit Deiner E-Mail-Adresse. Im letzteren Fall klickst Du noch den Bestätigungslink in der an Dich versendeten E-Mail an, um die Registrierung abzuschließen. Schließlich fragt OpenAI noch einige Informationen wie Name, Telefonnummer et cetera ab. Die Telefonnummer muss mit dem an diese Nummer gesendeten Code bestätigt werden.

Auf der Plattform findest Du anschließend oben rechts die Möglichkeit, auf Deinen Account zuzugreifen und dort finden wir auch den Befehl **View API Keys**. (siehe Bild 1). Während wir Abfragen über die Webseite ohne Weiteres absen-

den können, weil wir dort angemeldet sind, benötigen wir für den Zugriff über die Rest API einen API Key, um uns zu authentifizieren.

Warum sollen wir uns überhaupt anmelden und können OpenAI nicht anonym nutzen? Weil es kostenpflichtig ist. Die aktuellen Preise für die Nutzung der verschiedenen APIs finden wir auf der folgenden Seite:

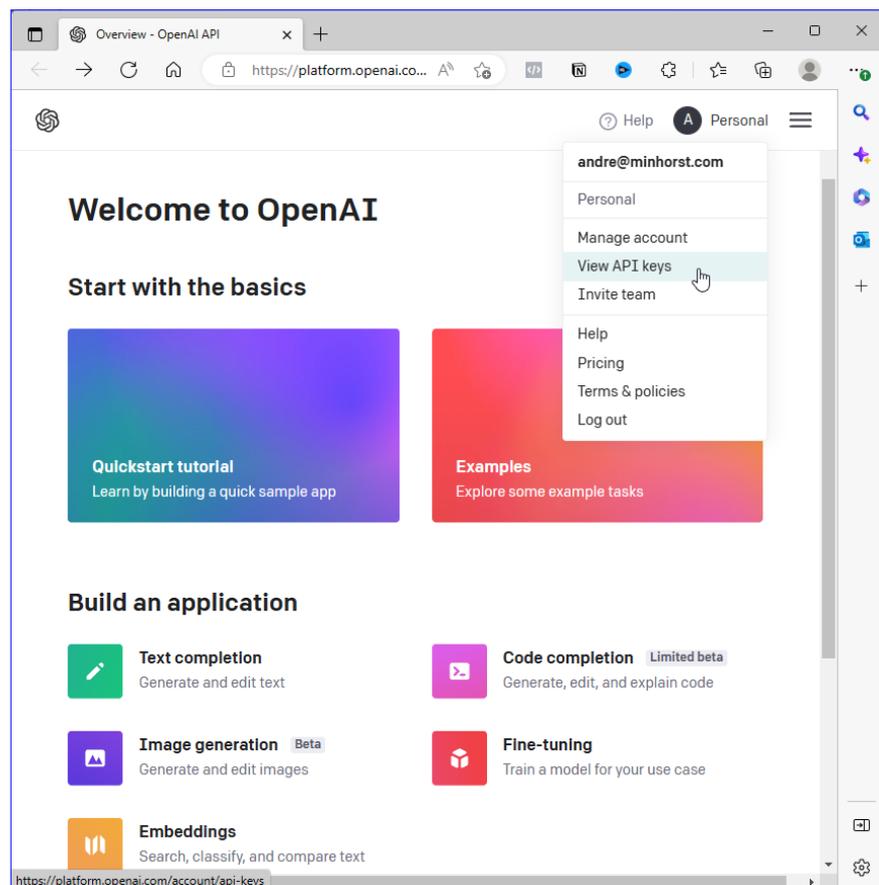


Bild 1: Von OpenAI zum API Key

PDFs aufteilen und zusammenfügen mit .NET

Gelegentlich möchte man PDF-Dateien bearbeiten – und das am besten ohne eigenes Zutun, sondern komplett programmgesteuert. Ein Kunde ist beispielsweise an mich herangetreten und wollte ein riesiges PDF-Dokument, das eine Menge Rechnungen in einer einzigen Datei enthielt, aufteilen. Als Ergebnis wünscht er sich einzelne Dateien, die jeweils eine Rechnung enthielten. Grund genug, einmal zu schauen, wie man mit .NET an dieses Problem herangehen kann. Genügend NuGet-Pakete mit Funktionen zum Bearbeiten von PDF-Dokumenten sind verfügbar, sodass wir die Qual der Wahl haben. Für diese Aufgabe haben wir das NuGet-Paket PDFsharp herangezogen, das verspricht, mit PDF-Seiten umgehen zu können – egal, ob es um das Entfernen, Hinzufügen, Extrahieren oder Zusammenstellen neuer Dokumente geht.

Anforderungen an ein NuGet-Paket zum Analysieren und Bearbeiten von PDF-Dokumenten

Speziell auf die oben angegebene Aufgabenstellung bezogen können wir zwei Anforderungen definieren:

- Wir benötigen eine Funktion, mit der wir ein PDF-Dokument seitenweise durchlaufen und im Text der Seiten nach bestimmten Ausdrücken suchen können. Diese Aufgabe erledigt PDFsharp nicht zufriedenstellend, daher schauen wir uns dazu in einem weiteren Artikel ein besser geeignetes Paket an.
- Außerdem benötigen wir eine Funktion, mit der wir einzelne Seiten eines PDFs-Dokuments extrahieren und als eigenständige Dokumente speichern können.

Beides würde theoretisch auch Adobe Acrobat Pro bieten, aber dies ist kostenpflichtig und wir sollen sehen, ob wir auch mit kostenloser Software weiterkommen.

NuGet-Paket suchen

Also starten wir Visual Studio und legen ein neues Projekt namens **PDFTools** an. Das Projekt soll den Typ **WPF-App (.NET Framework)** erhalten (siehe Bild 1), damit wir die Funktionen über eine Benutzeroberfläche steuern können – beispielsweise, um den Namen der zu bearbeitenden PDF-Datei anzugeben und die gewünschten Funktionen aufzurufen.

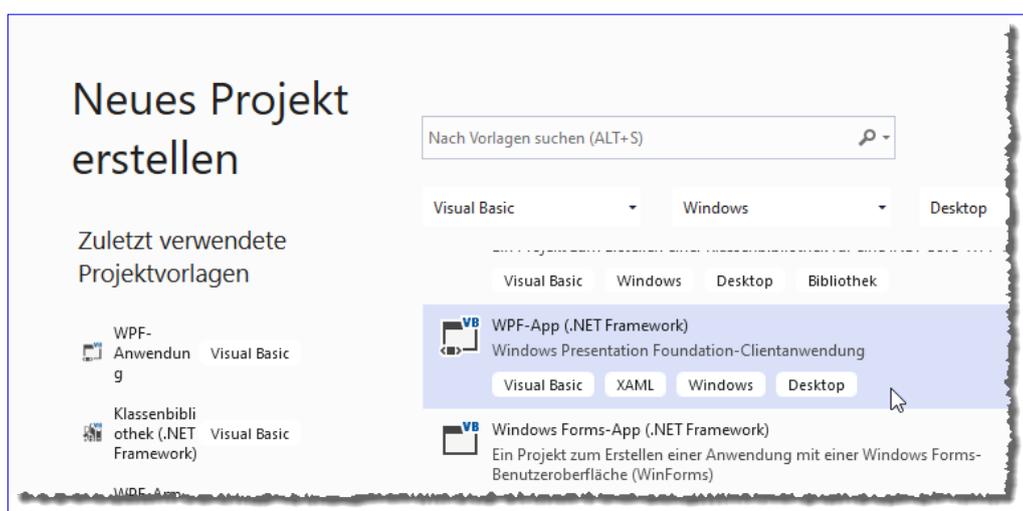


Bild 1: Anlegen einer neuen WPF-Anwendung

QR-Codes per .NET-Anwendung erstellen

QR-Codes sind praktische Helfer in der heutigen Zeit der Smartphones und ihren Kameras. Man kann damit wichtige Informationen so codieren, dass man diese mit der Kamera erkennen und weiterverarbeiten kann – standardmäßig so, dass der Code eine URL zu einer Webseite erhält und man diese dann im Browser des Smartphones aufrufen und die Inhalte konsumieren kann. Dieser Artikel zeigt, wie Du QR-Codes mit .NET-Anwendung erstellen kannst. Dreh- und Angelpunkt ist dabei ein NuGet-Paket, das die Funktionen für die Erstellung von QR-Codes auf Basis verschiedener Informationen bereitstellt. Der Artikel zeigt, wie wir dieses Paket in ein .NET-Projekt einbinden und es dann nutzen können.

QR-Codes und ihre Anwendung

Kennst Du eigentlich die Bedeutung der Abkürzung QR? Ich habe sie lustigerweise zum ersten Mal bewusst wahrgenommen, als ich für diesen Artikel recherchiert habe. QR steht für Quick Response.

Für QR-Codes gibt es mittlerweile sehr viele Anwendungsfälle. Hier sind nur einige von ihnen:

- **Produktinformationen:** QR-Codes können auf Produktverpackungen platziert werden, um Kunden schnell und einfach auf weitere Informationen wie Produktbeschreibungen, Kundenbewertungen oder Anleitungen zu verweisen.
- **Marketing und Werbung:** QR-Codes können in Marketing- und Werbekampagnen eingesetzt werden, um Kunden auf spezielle Angebote, Wettbewerbe oder Veranstaltungen aufmerksam zu machen. Sie können auch verwendet werden, um Kunden auf eine Landingpage oder eine mobile App zu leiten.
- **Eventmanagement:** QR-Codes können auf Eintrittskarten oder Veranstaltungsbroschüren platziert werden, um Besuchern den Zugang zu zusätzlichen Informationen, Updates und Planungstools zu ermöglichen.

- **Mobile Zahlungen:** QR-Codes können als Zahlungsoption für mobile Zahlungsanwendungen verschiedener Banken verwendet werden.
- **Identifikation und Authentifizierung:** QR-Codes können für die Identifikation von Personen oder als Bestätigung für den Zugang zu bestimmten Orten oder Veranstaltungen verwendet werden.
- **Gesundheitswesen:** QR-Codes können für die schnelle Übertragung von Patientendaten und medizinischen Informationen verwendet werden.
- **Tourismus und Reisen:** QR-Codes können für die Bereitstellung von Informationen zu touristischen Attraktionen, Transportmöglichkeiten und Unterkünften verwendet werden.

QR-Codes in .NET

Im Beispiel wollen wir eine WPF-Anwendung nutzen, um Texte eingeben und daraus QR-Codes zu erzeugen.

Dazu erstellen wir eine neue Anwendung auf Basis der Vorlage **WPF-App (.NET Framework)** – siehe Bild 1.

Nach dem Erstellen des Projekts erscheint das Hauptfenster von Visual Studio .NET und zeigt die WPF-Seite im Entwurf und mit dem Beschreibungscode im

Standalone-Apps mit .NET programmieren

Nicht immer möchte man Erweiterungen gezielt für eine Office-Anwendung programmieren. Gelegentlich fallen Aufgaben an, die man zwar mit einer der Office-Anwendungen erledigen könnte, aber dazu benötigt man auch immer die jeweilige Office-Anwendung und ein passendes Dokument wie ein Excel-Workbook oder eine Access-Datenbank. Und gerade bei Access ist eine der meist gestellten Fragen: Wie kann ich die Datenbank in eine .exe-Datei umwandeln? Die Antwort lautet: Gar nicht. Wenn es aber allein darum geht, Aufgaben zu erledigen, die nicht unbedingt mit Office zusammenhängen, dann könnte man auch schnell eine .NET-App programmieren. Die hat den Vorteil, dass man erstens viel mehr Steuerelemente nutzen kann, zweitens viel mehr Bibliotheken zur Verfügung hat und drittens eine .exe erstellen kann, die man sogar noch weitergeben kann. In diesem Artikel schauen wir uns die Grundlagen für die Erstellung einer einfachen .exe-Datei mit .NET an.

Aufgaben für eine .NET-App

Aufgaben für solche Anwendungen gibt es wie Sand am Meer. Damit der Aufwand zum Erstellen einer solchen Anwendung gerechtfertigt ist, sollte die Aufgabe nicht nur einmal vorkommen, sonst könnte man diese auch von Hand erledigen. Es sollte sich also um eine Aufgabe handeln, die entweder sehr umfangreich ist und dennoch sich wiederholende Schritte enthält oder eine, die man regelmäßig erledigen muss und für die man die Anwendung immer wieder nutzen kann.

Hier sind ein paar Beispiele:

- Dokumente umbenennen und kopieren oder verschieben, zum Beispiel um regelmäßige Eingänge in den Download-Ordner in andere Ordner zu übertragen
- E-Mails von Outlook auslesen und in bestimmten Verzeichnissen speichern
- Seiten aus PDF-Dokumenten extrahieren oder zu neuen Dokumenten zusammensetzen – Beispiel siehe **PDFs aufteilen und zusammenfügen mit .NET** (www.vbentwickler.de/356)

- PDF-Dokumente analysieren (Beispiel siehe **PDF-Dokumente analysieren mit VB.NET** (www.vbentwickler.de/357))
- Alle möglichen Aufgaben, die mit dem Abrufen oder Hochladen von Daten an REST APIs beziehungsweise Webservices zu tun haben – .NET bietet hier für alle möglichen Dienste Bibliotheken in sogenannten NuGet-Paketen an
- Daten aus verschiedenen Quellen einlesen und weiterverarbeiten
- Automation von Abläufen wie Sichern von Dateien, Senden von E-Mails oder Ausführen von Skripten

Dir fallen sicher noch weitere Anwendungsfälle ein. Diese kannst Du gern als Thema für weitere Artikel vorschlagen – schreibe dazu einfach eine E-Mail an andre@minhorst.com.

Werkzeuge

Wir stellen in diesem Artikel die Vorgehensweise und einige Grundlagen zum Erstellen von .exe-Programmen mit Benutzeroberfläche vor. Die Benutzerober-

Anwendungsdaten speichern per VB.NET

Wenn man wie im Artikel »Standalone-Apps mit .NET programmieren« beschrieben kleine Hilfsprogramme erstellt, kommt es vor, dass man dort Daten eingibt, die auch nach dem Schließen und dem erneuten Öffnen der Anwendung noch erhalten sein sollen. Wenn Du in einer solchen Anwendung beispielsweise immer wieder Daten aus dem gleichen Verzeichnis verarbeiten möchtest, willst Du das Verzeichnis nicht jedes Mal erneut auswählen. Man kann damit auch andere Daten wie Verbindungszeichenfolgen, Benutzernamen für Logins oder ganz allgemein Optionen speichern. All dies könnte man auch in eine Datenbank schreiben, aber wenn die Menge der Daten überschaubar ist, erhalten wir mit der in diesem Artikel vorgestellten Lösung eine wesentlich leichtgewichtige Alternative.

Als Entwickler, der sich viel mit Microsoft Access beschäftigt, würde ich mir wünschen, dass diese Anwendung sich zum Beispiel beim Auswählen von Dateien für den Import das Verzeichnis merken würde, das ich beim letzten Mal verwendet habe. Leider ist das nicht der Fall, und dieses Verhalten trifft man leider immer wieder an verschiedenen Stellen an.

Wenn ich selbst Beispiele oder Lösungen programmiere, bei denen Dateien für den Import oder Export von Daten festgelegt werden müssen, baue ich daher allein aus Faulheit rechtzeitig eine Möglichkeit ein, damit ein einmal ausgewähltes Verzeichnis gespeichert und beim nächsten Öffnen der Anwendung nicht nochmals ausgewählt werden muss.

Während dies in Access logischerweise mit einer Optionentabelle geschieht, die man schnell selbst anlegt, ist dies bei einer .NET-Standalone-Anwendung wie in **Standalone-Apps mit .NET programmieren** (www.vbentwickler.de/358) beschrieben recht aufwendig – zumindest, wenn diese Anwendung nicht ohnehin eine Datenbank verwendet. Allerdings gibt es eine Alternative, nämlich die Datei **App.config**. Diese dient speziell zum Speichern von Konfigurationsdateien, und welche Daten man darin speichert, ist dem Entwickler selbst überlassen. Dieser Artikel zeigt, wie Du Daten aus Textfeldern beim Schließen der Anwendung

speicherst und diese beim erneuten Öffnen wiederherstellst.

Dabei gehen wir wie folgt vor:

- Erstellen einer Anwendung mit Optionen
- Definieren der Einstellungen
- Erstellen einer Methode zum Speichern der Einstellung
- Erstellen einer Methode zum Wiederherstellen der Einstellung

Erstellen einer Anwendung mit Optionen

Als Erstes legen wir ein neues VB.NET-Projekt wie in **Standalone-Apps mit .NET programmieren** (www.vbentwickler.de/358) beschrieben an. Diese enthält Steuerelemente zum Eingeben von Daten – zum Beispiel Textfelder zur Eingabe von Texten und Zahlen, ein Kontrollkästchen und ein Auswahlfeld. Der Entwurf sieht wie in Bild 1 aus. Außerdem finden wir zwei Schaltflächen vor, mit denen wir die Daten speichern und wiederherstellen können.

Die Definition der Steuerelemente lautet in gekürzter Form wie folgt:

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

ANLAGEN SPEICHERN PER KONTEXTMENÜ

Mit unserer Lösung speicherst Du Anlagen direkt in das gewünschte Verzeichnis, statt Dich durch den Dateidialog zu wühlen.

SEITE 54

WORD-VORLAGE

Eine Word-Vorlage kann viel Arbeit sparen, wenn man gelegentlich Anschreiben versenden möchte. Wir liefern die Basics für eine praktische Dokumentvorlage.

SEITE 11

OUTLOOK-RIBBONS UND KONTEXTMENÜS

Zwei Artikel zeigen, wie Du das Ribbon und das Kontextmenü von Outlook per COM-Add-In anpassen kannst.

SEITE 34



André Minhorst Verlag

Excel: Bilder in Worksheets einfügen

Excel ist ein leistungsstarkes und vielseitiges Tool für die Datenanalyse und -visualisierung. Es ermöglicht es den Benutzern, große Datenmengen effektiv zu verwalten und in verschiedenen Formen darzustellen. Bilder sind dabei oft eine wichtige Ergänzung, um die Analyse von Daten oder Informationen zu unterstützen. Das Einfügen von Bilddateien in Excel-Worksheets per VBA ist eine solche Aufgabe, die es Benutzern ermöglicht, Bilder in ihre Arbeitsmappen einzufügen, um die Datenanalyse und -präsentation zu verbessern. In diesem Artikel werden wir uns damit beschäftigen, wie man Bilddateien in Excel-Worksheets per VBA einfügen kann und welche Vorteile und Anwendungsfälle es gibt.

Wozu Bilder in Excel einfügen?

Für das Einfügen von Bildern in Excel-Dokumenten gibt es verschiedene Anwendungszwecke. Hier sind ein paar davon:

- Einfügen von Firmenlogos oder Produktbildern in Rechnungen oder Angebote, um sie ansprechender zu gestalten.
- Hinzufügen von Fotos von Mitarbeitern oder Teams in Organigrammen, um sie persönlicher und einladender zu gestalten.
- Einfügen von Diagrammen oder Grafiken, die auf Daten basieren, um die Visualisierung von Ergebnissen oder Trends zu verbessern.
- Hinzufügen von Screenshots oder Bildern von Webseiten, um die Analyse von Daten oder Informationen zu unterstützen.
- Einfügen von Bildern, um Anleitungen oder Arbeitsanweisungen in Excel-Tabellen zu ergänzen, um den Prozess oder das Ergebnis besser zu erklären.

Bild einfügen per Benutzeroberfläche

Wenn wir ein Bild in eine Zelle einfügen wollen, gelingt dies am einfachsten über die Benutzeroberfläche.

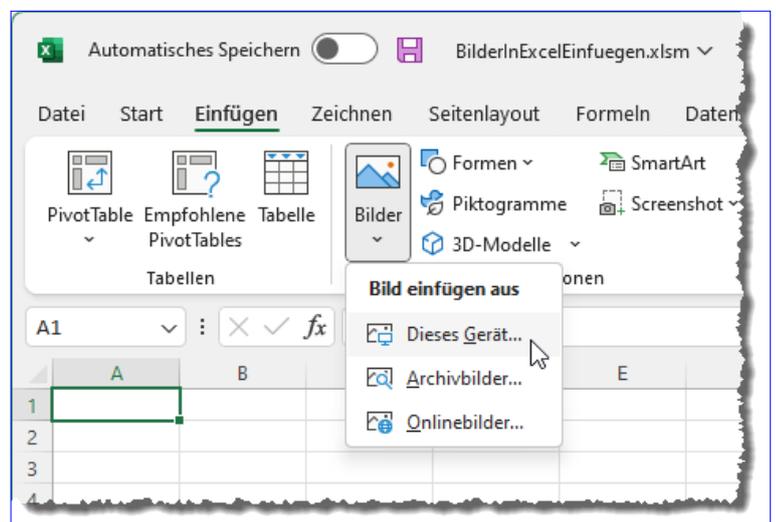


Bild 1: Bild einfügen per Benutzeroberfläche

Dazu markieren wir die Zielzelle und wählen aus dem Ribbon den Eintrag **Einfügen|Illustrationen|Bilder** aus und dann die Schaltfläche **Dieses Gerät...** (siehe Bild 1).

Damit aktivieren wir einen **Datei öffnen**-Dialog, mit dem wir die einzufügende Bilddatei auswählen.

Diese erscheint anschließend in der aktuell markierten Zelle (siehe Bild 2).

Klicken wir mit der rechten Maustaste auf das neu hinzugefügte Bild, finden wir im Kontextmenü beispiels-

Outlook: Kontextmenüs anpassen

In den Office-Anwendungen Word, Excel oder Access passen wir vorhandene Kontextmenüs per VBA über das Objektmodell von Office an. Auch das Hinzufügen und die Anzeige benutzerdefinierter Kontextmenüs erledigen wir auf diese Weise. Unter Outlook sieht die Situation anders aus: Hier wurde die Definition von Kontextmenüs bereits in die Ribbondefinition integriert. Wir haben dort einen eigenen Abschnitt namens »contextMenu« mit dem wir vorhandene Kontextmenüs anpassen und erweitern können. In diesem Artikel schauen wir uns an, wie wir solche Anpassungen vornehmen und welche Möglichkeiten sich daraus ergeben.

Kontextmenüs in Outlook

Outlook bietet genau wie die übrigen Office-Anwendungen eine Reihe von Kontextmenüs, die für die verschiedensten Elemente oder Bereiche aufgerufen werden können. In Bild 1 sehen wir beispielsweise das Kontextmenü für eine E-Mail.

Technik zum Anpassen

Die schlechte Nachricht ist: Kontextmenüs unter Outlook lassen sich nur per COM-Add-In anpassen. Die gute ist: Die Basis dafür haben wir bereits in einem weiteren Artikel namens **Outlook: Ribbon per COM-Add-In anpassen** (www.vbentwickler.de/376) gelegt. Dort haben wir mit dem Tool **twinBASIC** bereits ein COM-Add-In erstellt, mit dem wir die normalen Ribbonelemente anpassen können, also die **tab**-, **group**- oder **button**-Elemente im Ribbon von Outlook. Dort haben wir auch bereits die Besonderheit hervorgehoben, dass Outlook als einzige Office-Anwendung mehrere Fenster mit jeweils einem eigenen Ribbon hat. Auch dies ist beim Anpassen von Kontextmenüs zu berücksichtigen.

Anpassen oder auch neue Kontextmenüs definieren?

Die nächste Frage, die wir uns stellen müssen: Können wir nur bestehende Kontextmenüs erweitern oder auch

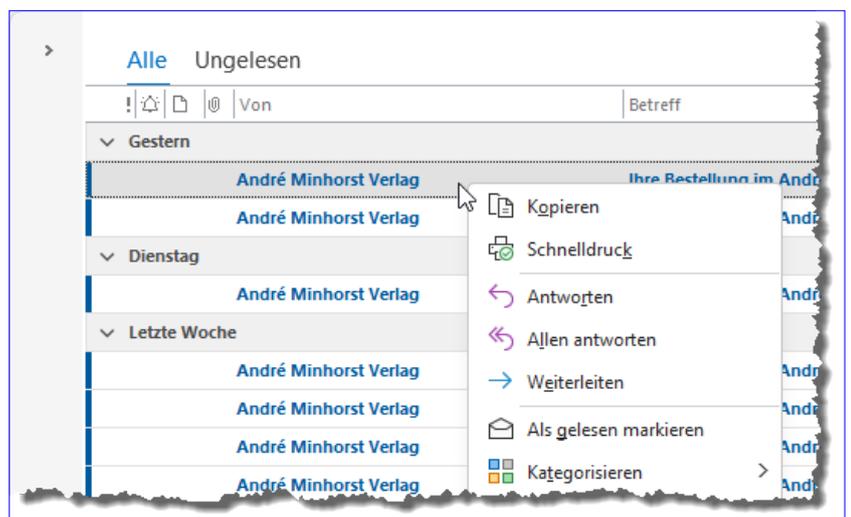


Bild 1: Ein Kontextmenü in Outlook

neue Kontextmenüs erstellen? Unter Access beispielsweise kann man auf das Betätigen der rechten Maustaste per Ereignisprozedur reagieren und per VBA ein neues Kontextmenü zusammenstellen und dieses auch anzeigen. Probieren wir also aus, ob wir auch in Outlook benutzerdefinierte Kontextmenüs per VBA erstellen können. Dazu fügen wir die folgende Prozedur in ein neues Modul im VBA-Editor von Outlook ein und führen sie aus:

```
Public Sub KontextmenueAnlegen()
    Dim cbr As CommandBar
    Dim cbb As CommandBarButton
    Set cbr = Application.CommandBars.Add( _
        "Benutzerdefiniert", msoBarPopup, , True)
```

Kontextmenüs per VBA programmieren

Vor sehr langer Zeit (2003) gab es für die Office-Anwendungen noch die Möglichkeit, Menüleisten, Symbolleisten und Kontextmenüs über die Benutzeroberfläche anzupassen. Die Menüleisten und Symbolleisten sind seit der Version 2007 Geschichte, und damit ist auch der Editor für die Gestaltung dieser Elemente verschwunden. Allerdings gibt es noch die Kontextmenüs, und diese lassen sich auch noch anpassen – zum Beispiel, um eigene Befehle hinzuzufügen. Diese könnten beispielsweise Teil von Add-Ins, COM-Add-Ins oder auch von Dokumenten sein. In diesem Artikel zeigen wir, wie man vorhandene Kontextmenüs bearbeitet oder eigene Kontextmenüs anlegt und diese bei Bedarf aufruft.

Unterschiede zwischen Outlook und den übrigen Office-Anwendungen

Zwischen Outlook und den übrigen Office-Anwendungen gibt es einen entscheidenden Unterschied für die individuelle Anpassungen von Kontextmenüs: Während wir unter Word, Excel, Access oder PowerPoint das Objektmodell aus der Bibliothek **Microsoft Office x.0 Object Library** nutzen können, um die gewünschten Anpassungen vorzunehmen, ist dies in Outlook nicht möglich. Hier sind die Kontextmenüs bereits Bestandteil des Ribbons. Änderungen der Kontextmenüs müssen wir in Outlook daher über die Ribbondefinition vornehmen, und zwar über die Einträge im Unterelement **contextMenus**. Dieses Thema behandeln wir im vorliegenden Artikel nicht, aber wir gehen im Detail darauf in einem weiteren Artikel namens **Kontextmenüs in Outlook anpassen** (www.vbentwickler.de/369) ein.

Beispieldateien

Da die CommandBars sich in den verschiedenen Office-Anwendungen Access, Excel und Word leicht unterschiedlich verhalten, haben wir jeweils eine Beispieldatei für jede Anwendung bereitgestellt:

- Access: **KontextmenusPerVBA.accdb**
- Excel: **KontextmenusPerVBA.xlsm**

- Word: **KontextmenusPerVBA.docm**

Objektmodell verfügbar machen

Um die Kontextmenüs anzupassen, benötigen wir Elemente der Bibliothek **Microsoft Office x.0 Object Library**. Diese ist beispielsweise in den VBA-Projekten von Word und Excel bereits referenziert, nicht jedoch unter Access oder in anderen Projekten auf Basis von Visual Studio oder twinBASIC. Damit wir die Bibliothek etwa in Access unter Einsatz von IntelliSense nutzen können, fügen wir einen Verweis auf diese Bibliothek hinzu. Dazu öffnen wir zunächst den

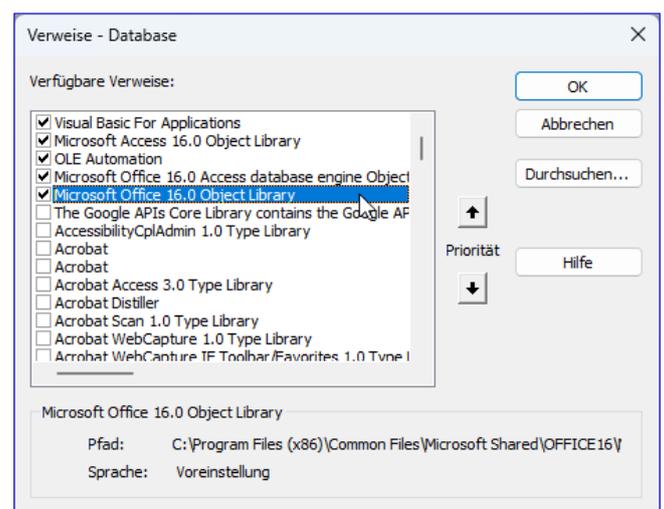


Bild 1: Aktivieren des Verweises auf die Bibliothek **Microsoft Office x.0 Object Library**

Kontextmenüs per VBA programmieren

Vor sehr langer Zeit (2003) gab es für die Office-Anwendungen noch die Möglichkeit, Menüleisten, Symbolleisten und Kontextmenüs über die Benutzeroberfläche anzupassen. Die Menüleisten und Symbolleisten sind seit der Version 2007 Geschichte, und damit ist auch der Editor für die Gestaltung dieser Elemente verschwunden. Allerdings gibt es noch die Kontextmenüs, und diese lassen sich auch noch anpassen – zum Beispiel, um eigene Befehle hinzuzufügen. Diese könnten beispielsweise Teil von Add-Ins, COM-Add-Ins oder auch von Dokumenten sein. In diesem Artikel zeigen wir, wie man vorhandene Kontextmenüs bearbeitet oder eigene Kontextmenüs anlegt und diese bei Bedarf aufruft.

Unterschiede zwischen Outlook und den übrigen Office-Anwendungen

Zwischen Outlook und den übrigen Office-Anwendungen gibt es einen entscheidenden Unterschied für die individuelle Anpassungen von Kontextmenüs: Während wir unter Word, Excel, Access oder PowerPoint das Objektmodell aus der Bibliothek **Microsoft Office x.0 Object Library** nutzen können, um die gewünschten Anpassungen vorzunehmen, ist dies in Outlook nicht möglich. Hier sind die Kontextmenüs bereits Bestandteil des Ribbons. Änderungen der Kontextmenüs müssen wir in Outlook daher über die Ribbondefinition vornehmen, und zwar über die Einträge im Unterelement **contextMenus**. Dieses Thema behandeln wir im vorliegenden Artikel nicht, aber wir gehen im Detail darauf in einem weiteren Artikel namens **Kontextmenüs in Outlook anpassen** (www.vbentwickler.de/369) ein.

Objektmodell verfügbar machen

Um die Kontextmenüs anzupassen, benötigen wir Elemente der Bibliothek **Microsoft Office x.0 Object Library**. Diese ist in beispielsweise in den VBA-Projekten von Word und Excel bereits referenziert, nicht jedoch unter Access oder in anderen Projekten auf Basis von Visual Studio oder twinBASIC. Damit wir die Bibliothek etwa in Access unter Einsatz von IntelliSense nutzen können, fügen wir einen Verweis auf

diese Bibliothek hinzu. Dazu öffnen wir zunächst den VBA-Editor der jeweiligen Anwendung, beispielsweise mit der Tastenkombination **Alt + F11**. Im VBA-Editor zeigen wir mit dem Menübefehl **Extras|Verweise** den **Verweise**-Dialog an. Hier selektieren Sie den Eintrag aus Bild 1 und fügen diese so hinzu.

Das Objektmodell zum Anpassen von Kontextmenüs

Zeigen wir nun mit **F2** den Objektkatalog an, können wir dort die Bibliothek **Office** auswählen und finden dort beispielsweise die Auflistung **CommandBars** vor

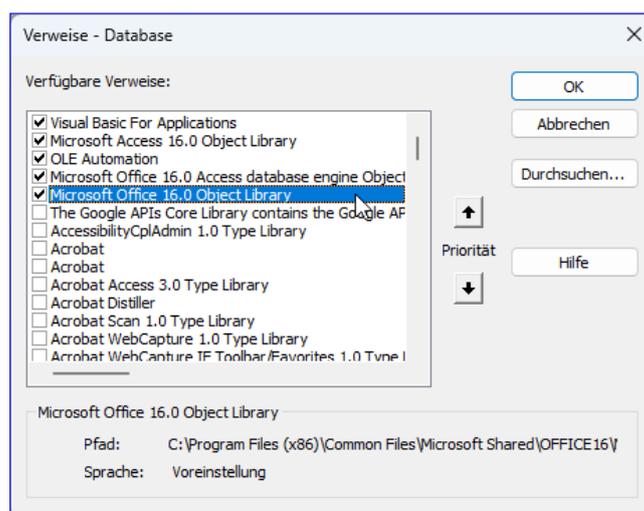


Bild 1: Aktivieren des Verweises auf die Bibliothek **Microsoft Office x.0 Object Library**

Outlook: Anhang speichern per Kontextmenü

Das Speichern von Anhängen in E-Mails gelingt in Outlook recht einfach: Man öffnet die E-Mail, klickt mit der rechten Maustaste auf den Anhang und wählt aus dem Kontextmenü den Eintrag »Speichern unter« aus. Danach allerdings fragt Outlook den Speicherort für den Anhang ab und hier startet man immer im gleichen Verzeichnis – in der Regel das Dokumente-Verzeichnis des aktuellen Benutzers. Dieser Artikel zeigt, wie wir dieses Verzeichnis auf ein anderes Verzeichnis einstellen können, aber das reicht in vielen Fällen nicht aus: Rechnungen sollen in ein bestimmtes Verzeichnis gespeichert werden, Anfragen von Kunden in einem bestimmten Verzeichnis für den jeweiligen Kunden et cetera. Diese Aufgaben werden wir mit einem Tool vereinfachen, das gleich im Kontextmenü die Möglichkeit zum Speichern in verschiedenen Verzeichnissen bietet. Und noch mehr: Wir wollen das Tool so programmieren, dass der Benutzer selbst eintragen kann, welche Kontextmenü-Einträge zum Speichern in verschiedenen Verzeichnissen genutzt werden können.

Ausgangssituation: Viel Arbeit mit dem Dateidialog

Standardmäßig bietet Outlook zwei Aufrufmöglichkeiten zum Speichern von Anhängen an:

- beim Rechtsklick auf einen der Anhänge in der Mail (siehe Bild 1) oder
- beim Markieren des Anhangs im Ribbon (siehe Bild 2).

Beim Anklicken dieser Einträge erscheint ein **Anlage speichern**-Dialog zur Auswahl des Speicherortes, der standardmäßig das Dokumente-Verzeichnis des aktuellen Benutzers anzeigt. Für viele Anwendungen wäre es schon ein Gewinn, wenn sich dieser Dialog das zuletzt verwendete Verzeichnis merken würde, aber das ist nicht der Fall. Für eine einfache Lösung schauen wir uns daher im ersten Schritt an, wie man dieses Verzeichnis dauerhaft umstellen kann.

Zielverzeichnis per Registry einstellen

Dazu öffnen wir die Registry von Windows (in der Window-Suche **RegEdit**

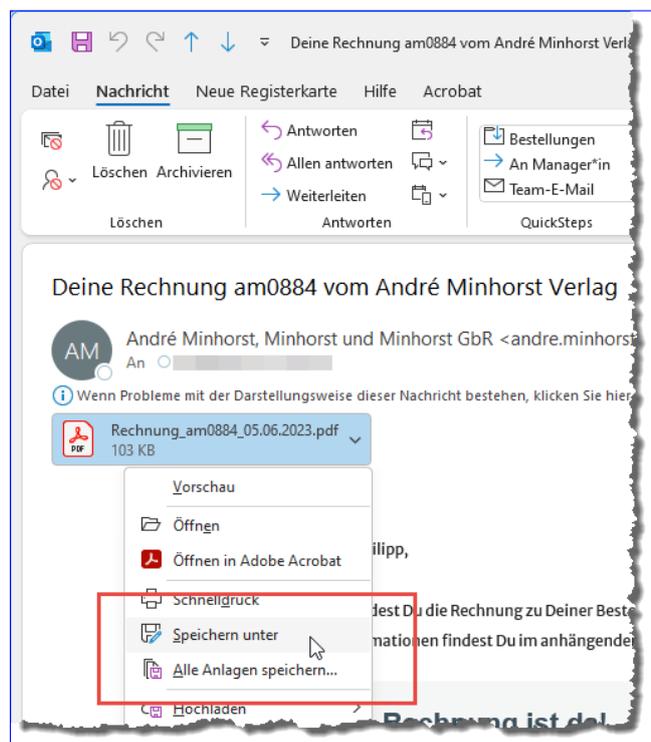


Bild 1: Kontextmenü-Einträge zum Speichern von Anhängen

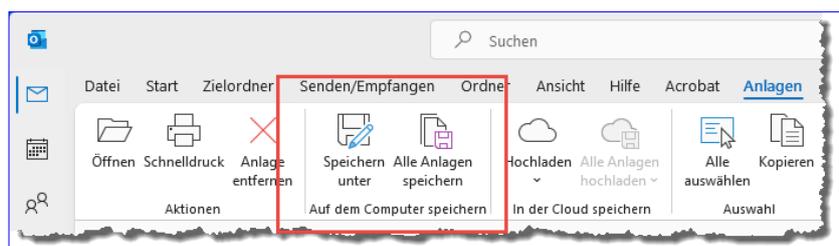


Bild 2: Ribbon-Einträge zum Speichern von Anhängen

Outlook: Ribbon per COM-Add-In anpassen

Wenn wir Outlook um eigene Funktionen erweitern wollen, stellt sich eine Frage: Wie wollen wir diese auslösen? Es gibt einige Ereignisse, die wir bereits im Artikel [Outlook: Explorer automatisieren \(www.vbentwickler.de/307\)](http://www.vbentwickler.de/307) erläutert haben. Diese werden beispielsweise durch Benutzeraktionen wie das Verschieben einer E-Mail in einen anderen Ordner ausgelöst. Aber wie können wir eigene Funktionen über die Benutzeroberfläche starten? Dazu bietet sich das Ribbon an. Hier können wir eigene Bereiche definieren, in denen wir unsere Funktionsaufrufe unterbringen. Der vorliegende Artikel erläutert, wie wir das Ribbon unter Outlook anpassen. Dabei sind einige Dinge zu berücksichtigen – zum Beispiel, dass es nicht wie bei den übrigen Office-Anwendungen nur ein Fenster gibt, das eine eigene Ribbondefinition verwendet.

Ribbon-Elemente im Überblick

Wir beginnen direkt mit dem eingangs erwähnten Umstand, dass Outlook nicht nur eine Ribbondefinition verwendet wie die übrigen Office-Anwendungen.

Das wird offensichtlich, wenn wir beispielsweise den Ribbonbefehl **Start|Neu|Neue E-Mail** aufrufen. Nicht nur das Outlook-Hauptfenster, sondern auch das Fenster zum Erstellen einer neuen E-Mail weist ein Ribbon auf (siehe Bild 1). Das macht die Sache im Gegensatz zu den übrigen Office-Anwendungen komplizierter. Woher wissen wir, für welches Fenster wir gerade das Ribbon anpassen? Und wie passen wir es in Outlook überhaupt an?

Ribbon per Benutzeroberfläche anpassen

Genau wie die übrigen Office-Anwendungen bietet auch Outlook die Möglichkeit, das Ribbon über die

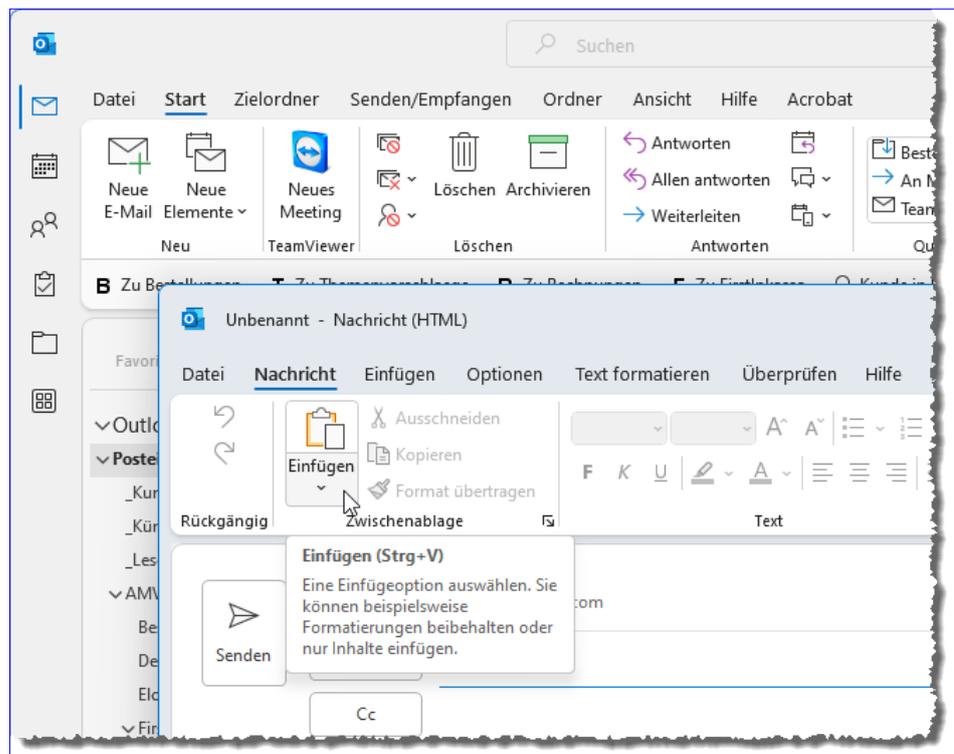


Bild 1: Outlook hat verschiedene Fenster mit Ribbons

Benutzeroberfläche anzupassen. Dazu öffnen wir die Outlook-Optionen und wechseln zum Bereich **Menüband anpassen** (siehe Bild 2). Hier finden wir in der rechten Liste einige Einträge mit den Hauptregisterkarten. Je nachdem, welcher Objekttyp gerade im Explorer angezeigt wird, also beispielsweise E-Mail oder Termin, erscheint ein anderes **Start**-Tab. Diese werden

Word-Vorlage für Anschreiben erstellen

Manchmal gibt es Situationen, da benötigt man das gute, alte Anschreiben. Adresse eintragen, Betreff hinzufügen, Ort und Datum festlegen und schließlich noch den Inhalt schreiben. Wenn man das nur hin und wieder machen muss, wie es bei mir der Fall ist, kann es schon mal sein, dass man nicht mehr weiß, wo man die Vorlage gespeichert hat, die man beim vorherigen Mal genutzt hat. Oder wo überhaupt ein Word-Dokument ist, das man bereits erstellt hat und dass man kopieren, anpassen und ausdrucken kann. Auf diese Weise braucht man nicht bei Adam und Eva anzufangen und den Briefkopf neu zu erstellen und die gute alte DIN-Norm herauszusuchen, die Informationen über die Positionen der einzelnen Elemente enthält. In diesem Artikel schauen wir uns an, wie wir eine passende Vorlage erstellen; in einem weiteren, wie wir diese aus einem Formular heraus mit den gewünschten Informationen füllen.

In diesem Artikel erledigen wir die folgenden Aufgaben:

- Erstellen einer Word-Vorlage, welche die wichtigsten Informationen wie Absenderadresse, Feld für die Empfängeradresse, Betreffzeile und Inhalt enthält
- Erstellen eines Formulars, das automatisch beim Erstellen eines neuen Dokuments auf Basis dieser Vorlage erscheint und die Informationen abfragt, die in das Dokument eingetragen werden sollen.

Warum sollten wir ein Formular bemühen und nicht direkt die notwendigen Texte in das Dokument eintragen? Weil wir eine Dokumentvorlage erstellen wollen, die auch von ungeübten Benutzern einfach eingesetzt werden kann.

Und da ist es erheblich einfacher, die Texte in ein Formular einzutragen als in die einzelnen Bereiche des Word-Dokuments, die leicht verschoben oder gelöscht werden können.

Und ganz ehrlich: Word ist wegen seiner Komplexität und der vielen Funktionen nicht gerade dafür ausgelegt, das Einsteiger damit schnell einfache Dokumen-

te erstellen. Da passiert es schnell, dass jemand lieber schnell einen Brief mit Excel schreibt, weil er da zumindest die Position bestimmter Elemente über die Spaltenbreiten und -höhen definieren kann.

Briefelemente erstellen

Ein Brief soll verschiedene Elemente enthalten. Dazu gehören:

- Briefkopf mit Absenderadresse und weiteren Informationen sowie gegebenenfalls einem Logo
- Adressfeld mit der Empfängeradresse und, falls nicht auf dem Umschlag abgedruckt, mit der Absenderadresse als Einzeiler über der Empfängeradresse
- Zeile mit dem Betreff
- Eigentlicher Inhalt des Briefs
- Gegebenenfalls eine Fußzeile für geschäftliche Informationen wie Bankverbindung, Steuernummern et cetera

Wir wollen alle Elemente mit Ausnahme des Inhalts über ein Formular erfassen. Den Inhalt selbst kann der Benutzer dann direkt im Dokument schreiben.

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

SETUPS ERSTELLEN MIT INNO SETUP

Gleich drei Artikel zeigen, wie man kostenlos Setups für die in diesem Magazin vorgestellten Toole erstellen kann.

SEITE 30

PDF-EXPORT MIT WORD

Word bietet keine einfache Möglichkeit zum Export von Inhalten in PDF-Dokumente. Wir liefern eine Lösung für den Export per Mausklick über das Ribbon.

SEITE 7/58

.NET-COM-DLLS UNTER VBA NUTZEN

Zwei Artikel zeigen, wie man die .NET-Techniken auch unter Office-VBA nutzen kann – inklusive Debugging.

SEITE 15



André Minhorst Verlag

COM-DLLs und Add-Ins unter VB.NET debuggen

Im Artikel »Office-VBA per COM-DLL mit VB.NET erweitern« und in einigen weiteren Artikel haben wir bereits gezeigt, wie man Visual Studio zum Erstellen von COM-DLLs und COM-Add-Ins nutzen kann. Im vorliegenden Artikel ergänzen wir die dort vorstellten Techniken zum Erstellen von COM-DLLs und COM-Add-Ins noch um die Vorgehensweise zum Debuggen des Codes dieser Projekte. Während WPF-Projekte oder Konsolenanwendungen sich von Visual Studio .NET starten und auch debuggen lassen, gelingt das bei COM-DLLs und COM-Add-Ins nicht so leicht. Es handelt sich dabei nicht um ausführbare Dateien, also müssen wir einen kleinen Umweg gehen – und diesen beschreiben wir im vorliegenden Artikel.

Wir beziehen uns in diesem Artikel zunächst auf die Beispiellösung aus dem Artikel **Office-VBA per COM-DLL mit VB.NET erweitern** (www.vbentwickler.de/378). Später schauen wir uns noch an, wie das Debuggen mit einem COM-Add-In aussieht.

Wenn wir eine COM-DLL wie die aus dem genannten Artikel in Visual Studio .NET öffnen und sie starten wollen, indem wir auf die **Starten**-Schaltfläche klicken oder die Taste **F5** betätigen, erhalten wir die Meldung aus Bild 1. Wir müssen also einen anderen Weg finden, um unseren Code zu debuggen.

Die Debuggen-Eigenschaften

Auch wenn die Anzahl der Programmierer, die COM-DLLs und COM-Add-Ins vergleichsweise überschaubar sein dürfte, so bietet Visual Studio auch für diese eine passende Lösung an. Diese finden wir in den Projekteigenschaften, die wir mit einem Doppelklick auf

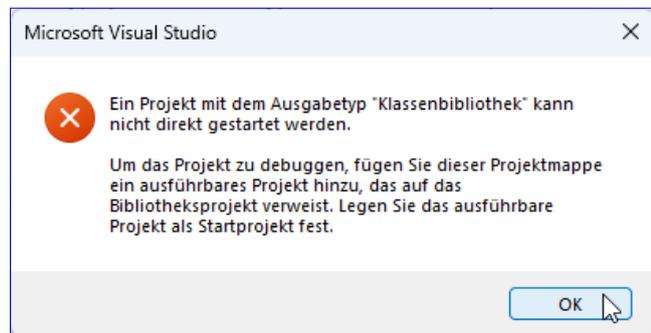


Bild 1: Fehlermeldung beim Versuch, ein nicht ausführbares Projekt zu starten

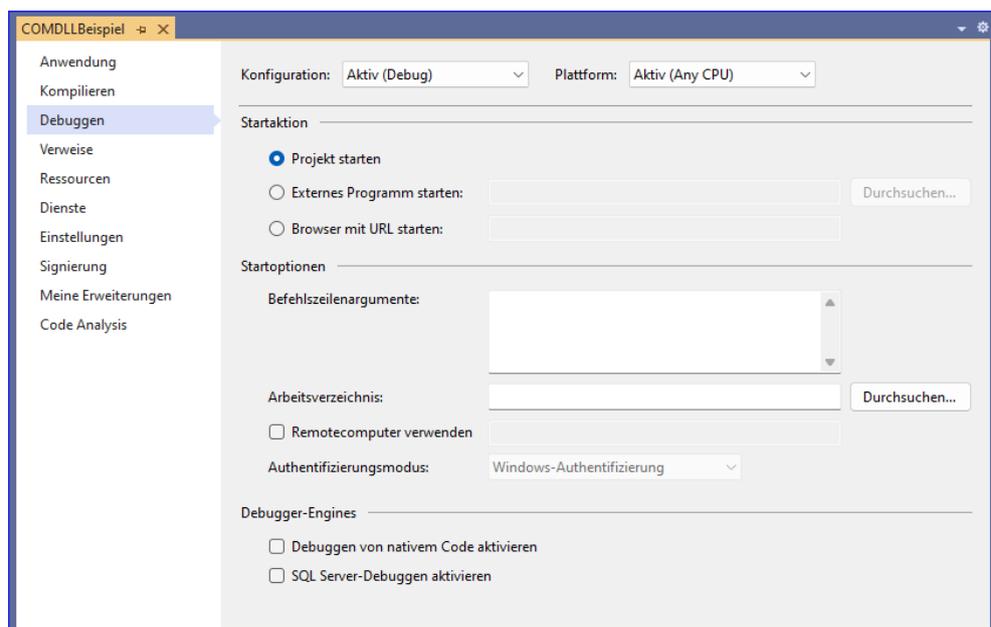


Bild 2: Der Bereich **Debuggen** in den Projekteigenschaften

Installation mit Inno Setup: Bilder und Images

Wie heißt es so schön: Das Auge isst mit. Das gilt nicht nur bei der Nahrungsaufnahme, sondern auch beim Konsum der Optik eines Setups. Mit Inno Setup können wir an verschiedenen Stellen Icons und Images unterbringen. Wie das gelingt, welche Tricks man dabei anwenden kann und wie wir eventuelle Stolpersteine umgehen, schauen wir uns im vorliegenden Artikel an. Dabei stoßen wir an unerwarteter Stelle an die Grenzen, die uns der Windows Defender auferlegt. Doch eins nach dem anderen: Erst einmal schauen wir, wo wir in einen mit Inno Setup erstellten Setup Icons und Bilder unterbringen können.

Voraussetzungen

In diesem Artikel zeigen wir, wie man beim Erstellen von Setup-Routinen mit dem kostenlosen Tool Inno Setup Installationsroutinen mit Bildern versehen kann. Die Grundlagen zu Inno Setup stellen wir im Artikel **Installation mit Inno Setup: Die Basics** (www.vbentwickler.de/382) vor.

Bitmaps und Icons

Eines vorweg: Wenn wir ein Setup wirklich mit hübschen Bildern versehen wollen, die auch noch einheitlich daherkommen, benötigen wir entweder ein Set von passenden Bildern oder wir investieren Zeit und Mühe in das Erstellen der verschiedenen Formate.

Auch wenn es dazu ausreichend Tools gibt – wir bevorzugen fertige Icons und Bilder.

Im konkreten Fall nutzen wir ein Produkt von www.iconexperience.com, das wir vor vielen Jahren erworben haben und das uns nach wie vor gute Dienste leistet. Hier finden wir sowohl **.ico**-Dateien, die jeweils verschiedene Größen enthalten, sowie **.png**-Dateien in den Größen 16x16, 24x24, 32x32, 48x48, 64x64, 128x128, 256x256 und 512x512. Im Falle verschiedener Bilder für ein mit Inno Setup erstelltes Setup be-

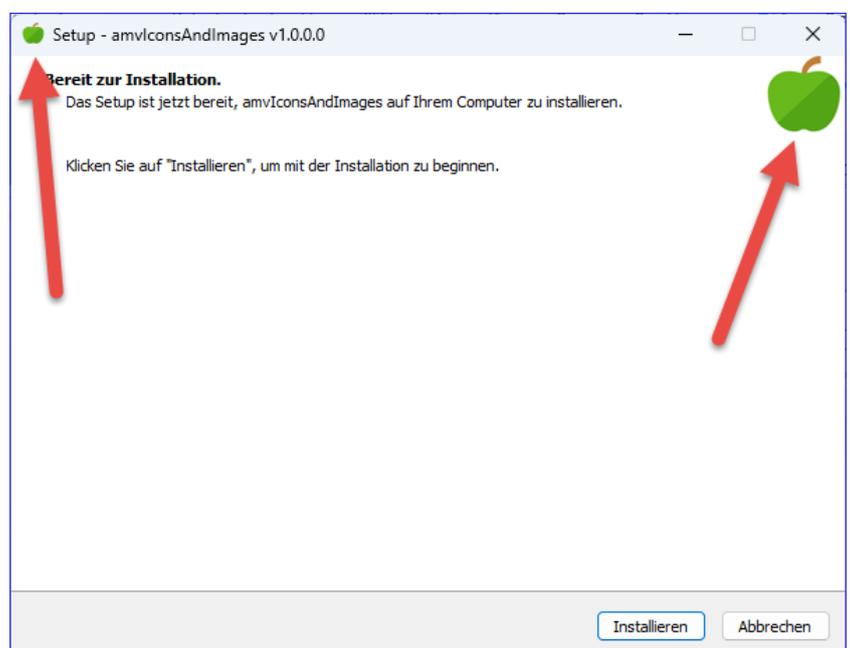


Bild 1: Bilder im Setup

nötigen wir allerdings **.bmp**-Dateien. Auch das ist mit dem angesprochenen Produkt kein Problem, denn es kommt mit einem Tool, mit dem wir die **.png**-Dateien in andere Formate wie **.bmp** oder **.gif** umwandeln können.

Im Setup angezeigte Bilder

Im Setup sehen wir drei verschiedene Bilder. Eine für die Eigenschaft **SetupIconFile** angegebene **.ico**-Datei wird als Icon links oben in der Titelleiste eingeblendet. Diese erscheint auf jeder Seite des Setups. Ein zweites

Installation mit Inno Setup: Die Basics

Inno Setup ist ein zu verlässiges Tool zum Erstellen von Setups für verschiedene Anwendungen. Es funktioniert, wenn Du einfach nur eine Datenbank, eine Excel-Arbeitsmappe oder ein Word-Dokument in das gewünschte Verzeichnis beim Benutzer kopieren möchtest. Genauso kann es kompliziertere Vorgänge abbilden, die das Anlegen von Registry-Einträgen enthalten. Dieser Artikel zeigt, wie Du Inno Setup installierst und welche Möglichkeiten es gibt, die verschiedenen Einstellungen vorzunehmen. Neben dem textbasierten Editor gibt es nämlich auch noch eine professionelle Benutzeroberfläche. In weiteren Teilen schauen wir uns dann Setups und Einstellungen für verschiedene Zwecke an.

Warum Setups? Die folgende Auflistung zeigt, welche Aufgaben wir damit erledigen können:

- **Dateien kopieren:** Das Setup kopiert die erforderlichen Dateien von den Installationsquellen auf den Zielcomputer. Dies kann Programmdateien, Konfigurationsdateien, Bibliotheken, Grafiken, Dokumentationen und so weiter umfassen.
- **Verzeichnisse erstellen:** Das Setup kann Verzeichnisse auf dem Zielsystem erstellen, um die benötigten Dateien zu organisieren. Dies können Installationsverzeichnisse, Konfigurationsordner, temporäre Ordner et cetera sein.
- **Registry-Einträge anlegen:** Setups können Registrierungseinträge in der Windows-Registrierung erstellen, um Informationen über die installierte Anwendung zu speichern. Dies können Einstellungen, Dateiverknüpfungen, DLL-Pfade und andere Informationen sein, die von der Anwendung verwendet werden.
- **Verknüpfungen erstellen:** Setups können Verknüpfungen im Startmenü, auf dem Desktop oder in anderen Ordnern erstellen, um den Benutzern den einfachen Zugriff auf die installierte Anwendung zu ermöglichen.
- **Systemkonfiguration:** Einrichten von Systemkonfigurationen oder -einstellungen, die für die ordnungsgemäße Funktion der Anwendung erforderlich sind. Dies könnte die Konfiguration von Diensten, Umgebungsvariablen oder anderen Systemkomponenten beinhalten.
- **Abhängigkeiten verwalten:** Wenn die Anwendung von bestimmten Bibliotheken oder Komponenten abhängt, können diese Abhängigkeiten während der Installation eingerichtet werden. Dies umfasst beispielsweise das Installieren von Laufzeitbibliotheken oder anderen notwendigen Komponenten.
- **Benutzerinteraktion:** Einrichtung von Dialogfeldern, um den Benutzer während der Installation zu führen. Das können Lizenzvereinbarungen, Auswahl von Installationskomponenten, Festlegen von Installationspfaden und so weiter sein.
- **Deinstallation:** Neben der Installation kann das Setup auch Mechanismen zum Deinstallieren der Anwendung bereitstellen. Das beinhaltet das Entfernen von Dateien, Verzeichnissen, Registry-Einträgen und Verknüpfungen.
- **Updates und Patches:** Einrichtung von Mechanismen zur Aktualisierung oder Patchen der Anwen-

Office-VBA per COM-DLL mit VB.NET erweitern

VBA ist, formulieren wir es einmal freundlich, seit einiger Zeit nicht mehr aktualisiert worden. Hier und da gibt es kleine Anpassungen in den Objektbibliotheken, aber der Sprachumfang an sich hat keine großen Schritte gemacht. Auch VB.NET bringt nicht täglich neue Sprachkonstrukte hervor. Aber dafür gibt es beispielsweise zahllose Erweiterungen in Form von NuGet-Paketen, die man leicht in einem VB.NET-Projekt verfügbar machen kann. Aus VB.NET-Projekten kann man aber auch eine COM-DLL erzeugen, die wir wiederum von einem VBA-Projekt aus referenzieren und nutzen können. Und somit können wir auch den Funktionsumfang von Word, Excel, Outlook, Access und Co. erweitern. Dieser Artikel zeigt die Grundlagen zur Erstellung eines COM-Add-Ins, das wir von unseren Office-Anwendungen aus nutzen können.

Voraussetzung: Visual Studio .NET

Die einzige Voraussetzung, die wir neben der zu erweiternden Office-Anwendung brauchen, ist eine Version von Visual Studio .NET. Microsoft bietet eine für viele Fälle kostenlos nutzbare Version an, die Du beispielsweise hier findest:

<https://visualstudio.microsoft.com/de/vs/community/>

Visual Studio als Administrator öffnen

Wenn wir die geplante DLL erstellen und testen wollen, benötigen wir in der Regel Administrator-Rechte. Dazu öffnen wir Visual Studio direkt als Administrator. Dazu klicken wir mit der rechten Maustaste auf das Visual Studio-Icon und wählen dort den Eintrag **Als Administrator ausführen** aus. Die anschließende Meldung bestätigen wir.

DLL-Projekt erstellen

Anschließend erstellen wir ein DLL-Projekt. Aus dem Startbildschirm von Visual Studio wählen wir dazu den Eintrag **Neues Projekt erstellen** aus. Im Dialog Neues Projekt erstellen finden wir einige Auswahlfelder, mit denen wir die Auswahl aus Bild 1 treffen (**Visual Basic**, **Windows**, **Bibliothek**) und dann den Eintrag **Klassenbibliothek (.NET Framework)** selektieren.

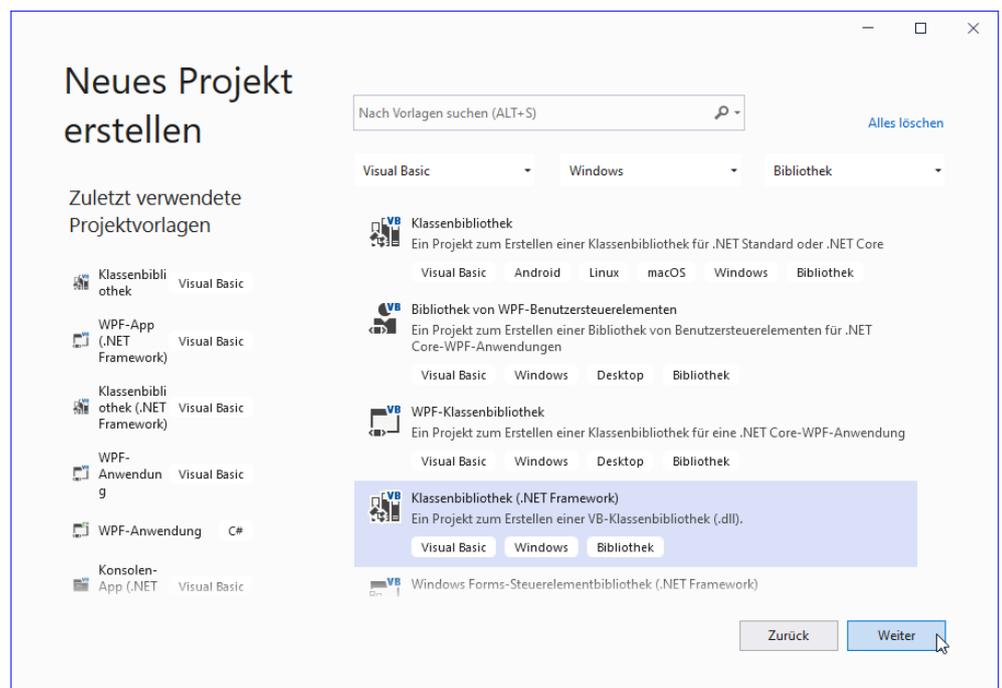


Bild 1: Auswahl des Typs für das neue Projekt

Outlook: Alternatives VBA-Projekt verwenden

Wenn Du eine neue VBA-Anwendung für den Einsatz in Outlook programmierst, möchtest Du vielleicht in einem frischen, leeren VBA-Projekt arbeiten. Bei mir ist das standardmäßig verwendete VBA-Projekt in Outlook recht voll mit allen möglichen produktiven und zum Testen verwendeten Ereignisprozeduren und weiteren Routinen. Eine neue Funktion so zu testen, als wenn diese auf dem Rechner eines Nutzers mit einem leeren Outlook-VBA-Projekt landet, ist so kaum möglich. Es gibt jedoch gute Nachrichten: Wir können recht fix ein neues VBA-Projekt für Outlook erstellen, das wir alternativ zu dem produktiv genutzten VBA-Projekt nutzen. Noch besser: Es gibt die Möglichkeit, beim Start anzugeben, welches VBA-Projekt wir aktuell nutzen möchten.

Frische und umfangreiche VBA-Projekte

Das VBA-Projekt von Outlook öffnen wir mit der Tastenkombination **Alt + F11**. Wenn Du noch keine Änderungen an diesem VBA-Projekt vorgenommen hast, sieht dieses wie in Bild 1 aus. Das ist recht aufgeräumt – es gibt lediglich das Klassenmodul `ThisOutlookSession`.

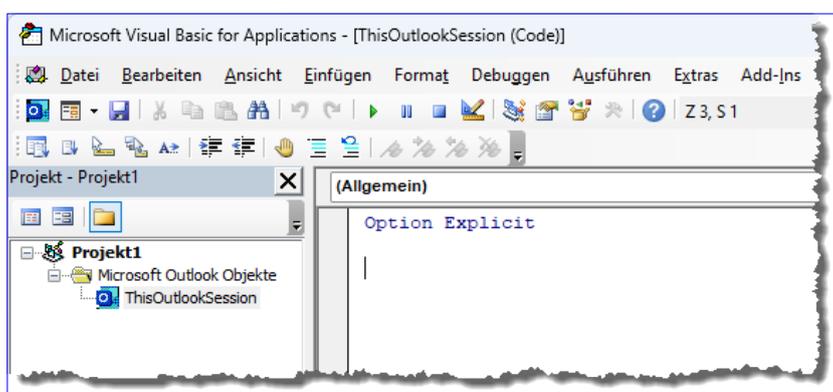


Bild 1: Ein unbeflecktes VBA-Projekt unter Outlook

Mein VBA-Projekt sieht etwas anders aus – allein der Projekttexplorer ist bereits recht voll (siehe Bild 2). Hier befinden sich einige tausend Zeilen Code, die teilweise produktiv im Einsatz sind und teilweise Überreste von Experimenten und Beispielen für Artikel sind.

VBA-Projekt »zurücksetzen«

Wenn Du bei Dir eine ähnliche Situation im VBA-Projekt von Outlook vorfindest wie ich beim mir, wünschst Du Dir

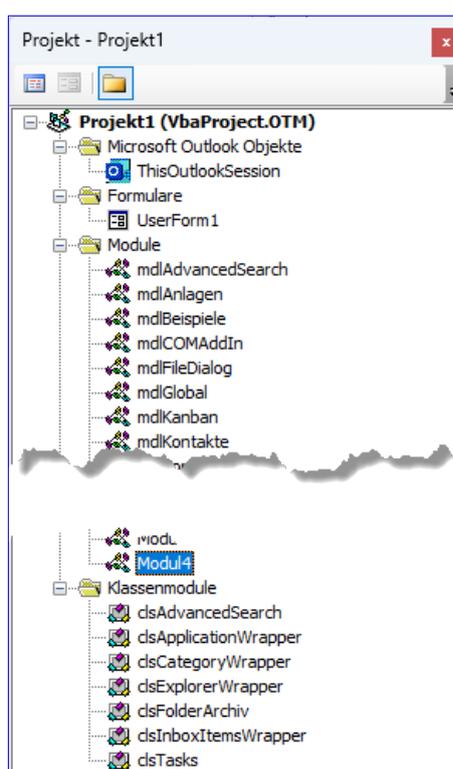


Bild 2: Das VBA-Projekt des Autors

vielleicht zu manchen Gelegenheiten, einmal ein leeres VBA-Projekt nutzen zu können. Das ist in wenigen Schritten erledigt:

- Wir benennen das bestehende VBA-Projekt um.
- Wir öffnen Outlook, welches das VBA-Projekt nicht mehr findet und dieses neu erstellt.

Fertig! Nun können wir mit einem neuen Outlook-VBA-Projekt arbeiten. Nun wollen wir dieses allerdings nur gelegentlich zum Experimentieren nutzen und standardmäßig unser

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

APPS ENTWICKELN MIT TWINBASIC

Wir starten in eine neue Artikelreihe rund um die Entwicklung von Windows-Anwendungen.

SEITE 4

STEUERELEMENTE UND EIGENSCHAFTEN IN TWINBASIC

Ohne Steuerelemente geht nichts – also schauen wir uns diese an!

SEITE 17

COM-ADD-INS REGISTRIEREN

Tipps und Tricks rund um das Registrieren von COM-Add-Ins, wenn das Setup nicht funktioniert.

SEITE 15



André Minhorst Verlag

COM-Add-Ins registrieren

Aus verschiedenen Gründen kann die Installation eines COM-Add-Ins für eine Office-Anwendung fehlschlagen – oder zumindest erreicht man damit nicht das gewünschte Ergebnis. Zum Beispiel könnte ein Benutzer das Setup mit dem COM-Add-In, das normalerweise für den aktuellen Benutzer installiert wird, im Kontext eines anderen Benutzers installieren – zum Beispiel als Administrator. Dann erscheint das COM-Add-In für den Benutzer jedoch nicht. In diesem Artikel zeigen wir zwei schnelle Wege, wie die Installation dennoch gelingt, sofern die DLL-Datei mit dem COM-Add-In vorliegt. Außerdem schauen wir uns an, wo in der Registry der Eintrag für ein COM-Add-In landet und was die verschiedenen Einträge bedeuten.

Setups haben meist den Zweck, die Installation einer Software für den Benutzer so angenehm wie möglich zu machen. Manchmal klappt es allerdings einfach nicht wie gewünscht. Das hängt zum Beispiel damit zusammen, dass die Registry des Benutzers anders aufgebaut ist, als es das Setup erwartet. Dann werden die für den reibungslosen Betrieb erforderlichen Einträge in der Registry aus Sicht des Setups zwar an der richtigen Stelle vorgenommen, aber sie werden vom System nicht berücksichtigt.

Es gibt nämlich bestimmte Registry-Bereiche, die Office-Anwendungen beim Start untersuchen, um zu erfahren, ob COM-Add-Ins oder andere Anwendungen vorliegen, die mit der Office-Anwendung gestartet werden sollen.

In diesem Magazin haben wir bereits einige COM-Add-Ins vorgestellt, die selbst festlegen, an welche Stellen der Registry die notwendigen Einträge geschrieben werden sollen. In dem COM-Add-In aus dem Artikel **COM-Add-In für Word: PDF-Export** (www.vbentwickler.de/383) legen wir das beispielsweise wie in Bild 1 fest. Die Konstante **RootRegistryFolder_WORD** nimmt den Basispfad auf, die Funktion **DllRegisterServer** legt die Einträge beim Registrieren der Anwendung in der Registry an.

Wie landen die Einträge in der Registry?

Wenn wir wie in dem Artikel gezeigt mit twinBASIC entwickeln und die Anwendung kompilieren, startet twinBASIC automatisch die Prozedur **DllRegisterServer** und trägt so die Einträge in die Registry ein.

```
Const RootRegistryFolder_WORD As String = "HKCU\SOFTWARE\Microsoft\Office\Word\Addins\" & AddinQualifiedClassName & "\"  
  
► run DllRegistration.DllRegisterServer  
Public Function DllRegisterServer() As Boolean  
    On Error GoTo RegError  
    Dim wscript As Object = CreateObject("wscript.shell")  
    wscript.RegWrite RootRegistryFolder_WORD & "FriendlyName", AddinProjectName, "REG_SZ"  
    wscript.RegWrite RootRegistryFolder_WORD & "Description", AddinProjectName, "REG_SZ"  
    wscript.RegWrite RootRegistryFolder_WORD & "LoadBehavior", 3, "REG_DWORD"  
    Return True  
RegError:
```

Bild 1: Anweisungen zum Anlegen der Registry-Einträge

twinBASIC: Daten von Form zu Form

Wenn man mit SDI-Forms arbeiten, also mit solchen Formularen, die als einzelne Fenster geöffnet werden, möchte man manchmal eine der folgenden beiden Aufgaben erledigen – oder auch beide: Das aufrufende Formular soll Daten an das aufgerufene Formular übergeben, beispielsweise um den Primärschlüssel eines im Detailformular anzuzeigenden Datensatzes zu übergeben. Oder man möchte ein Formular zum Abfragen von Daten öffnen und diese dann vom aufrufenden Formular aus aufrufen. Wie beides gelingt und welche unterschiedlichen Wege es dazu gibt, schauen wir uns in diesem Artikel an. Außerdem betrachten wir, welche Arten von Daten man grundsätzlich übertragen können sollte.

Es wird immer wieder Anlässe geben, bei denen wir einen Button auf einer Form verwenden, um eine andere Form damit zu öffnen. In manchen Fällen reicht dies aus – wir verwenden die andere Form dann für einen Zweck, der in sich abgeschlossen ist. In vielen Fällen wird es jedoch so sein, dass wir Daten von einem zum anderen Fenster übergeben wollen – und vielleicht auch wieder zurück. Ein Beispiel sind Forms, die an Daten gebunden sind. Wenn wir etwa ein Form-Objekt nutzen, um darauf eine Liste von Produkten anzuzeigen, wollen wir meist auch ein weiteres Form verwenden, um die Details eines einzelnen Produkts zu liefern – beispielsweise um diese bearbeiten zu können. Wenn wir dann einen der Einträge in der Liste markieren und auf eine Schaltfläche klicken, um die Form mit den Details anzuzeigen, müssen wir dieser Form auf irgendeine Weise die Information übergeben, zu welchem Produkt es die Detaildaten anzeigen soll.

Andersherum wollen wir nach dem Bearbeiten des Produkts oder auch nach dem Anlegen eines neuen Produkts und dem Schließen des Detailfensters Informationen an das aufrufende Fenster zurückgeben – zum Beispiel, weil wir den soeben bearbeiteten oder angelegten Datensatz in der Liste markieren wollen oder weil die Änderungen in den Daten direkt in den dort angezeigten Eintrag übernommen werden sollen.

Es gibt auch noch einfachere Beispiele, die es bereits gibt – zum Beispiel das mit der Funktion **MsgBox** an-

gezeigte Meldungsfenster oder das mit **InputBox** angezeigte Eingabefenster für einen einfachen Text.

Wenn wir einmal Daten vom Benutzer abfragen wollen, die zu umfangreich sind als jene, die wir mit **MsgBox** und **InputBox** ermitteln können, müssen wir eigene Forms dafür entwickeln – und die dort ermittelten Daten wollen wir auf irgendeine Weise für die Weiterverarbeitung auslesen.

Weitere Beispiele für das Übertragen von Daten

Beispiele für das Übertragen von Daten zwischen zwei Form-Objekten sind die folgenden:

- Anmeldeformular
- Suchformular
- Einstellungen und Konfigurationen
- Übertragung der ID eines ausgewählten Datensatzes von einer Hauptform zur Bearbeitungsform.
- Workflows über mehrere Forms

Möglichkeiten zur Übertragung von Daten an die zu öffnende Form

Im Folgenden schauen wir uns verschiedene Möglichkeiten an, wie wir Daten vom aufrufenden **Form-Ob-**

twinBASIC: Fenster öffnen, schließen und mehr

Form-Objekte waren in VB6 und sind in twinBASIC das Element der Wahl, wenn es um die Abbildung von Benutzeroberflächen geht. In diesem Artikel zeigen wir die grundlegenden Techniken rund um die Verwendung und Programmierung von Forms. Dabei zeigen wir, wie Du ein Form-Element öffnest, es wieder schließt und welche Öffnungsmodi es gibt. Dabei beschreiben wir auch, wie Du die Position eines Formulars nach dem Öffnen festlegen kannst. Außerdem schauen wir uns schon einige der grundlegenden Ereignisseigenschaften von Forms an und zeigen, wie wir diese in Form von Ereignisprozeduren implementieren können.

Fenster oder Form?

In den folgenden Abschnitten werden wir synonym von Fenstern und Forms sprechen. Allgemein ist zwar nicht jedes Fenster ein **Form**-Objekt, aber im Kontext des vorliegenden Artikels schon.

Forms öffnen

Wenn wir mit einer neuen twinBASIC-Anwendung starten, haben wir zwei Möglichkeiten, das erste Fenster zu öffnen:

- Durch Angabe des Fensters für die Eigenschaft **Project: Startup Object** oder
- durch Aufrufen der Prozedur **Main**, der wir den Code zum Öffnen eines Fensters zuweisen.

Form als Startfenster öffnen

Wir schauen uns zuerst die erste Möglichkeit an, wo wir ein Fenster als Startfenster festlegen. Damit wir in unserem neuen Projekt namens **Forms_OpenCloseAndCo** nicht die ganze Zeit mit einem Startform namens **Form1** arbeiten müssen, löschen wir dieses und fügen über den Kontextmenü-Eintrag **Add|Add Form** ein neues **Form**-Element hinzu, das wir gleich nach dem Anlegen **frmMain** nennen.

Achtung: Nur wenn wir dies erledigen, solange der Bearbeitungsmodus des neuen **Form**-Elements im

Projektexplorer aktiviert ist, werden alle Elemente des Form-Objekts umbenannt – also auch die Eigenschaft **Name** des **Form**-Elements, der Name der Code behind-Klassendatei und auch der Name der Klasse selbst. Hast Du diesen Zeitpunkt verpasst, kein Problem – lösche das Element einfach wieder und lege es erneut an.

Haben wir dies erledigt, brauchen wir nur noch die Optionen des Projekts aufzurufen und hier den Eintrag **frmMain** auszuwählen (siehe Bild 1).

Starten wir das Projekt nun, wird die Form **frmMain** angezeigt.

Startform über die Prozedur Main öffnen

Die zweite Möglichkeit ist das Öffnen des Fensters per Code. Wenn wir das Startformular **frmMain** per Code öffnen wollen, statt über die Eigenschaft, entfernen wir

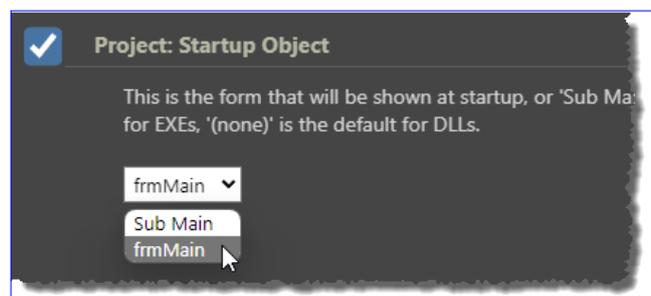


Bild 1: Einstellen der Startform **frmMain**

twinBASIC: Ereigniseigenschaften von Fenstern

Im Artikel **twinBASIC: Forms öffnen, schließen und mehr** (www.vbentwickler.de/389) haben wir bereits gezeigt, wie wir Fenster beziehungsweise Form-Elemente mit twinBASIC-Befehlen öffnen und schließen können und welche Varianten es dabei gibt. Über ein wichtiges Feature von Form-Elementen haben wir dabei noch nicht gesprochen: Die sogenannten Ereigniseigenschaften. Für diese können wir Prozeduren hinterlegen, die beim Auslösen bestimmter Ereignisse des Fensters ausgelöst werden – beispielsweise beim Öffnen, Schließen, Aktivieren, Deaktivieren oder auch beim Ändern der Größe oder der Position eines Fensters. Und auch wenn der Benutzer das Fenster an einer Stelle anklickt, die keine Steuerelemente enthält, wird ein Ereignis ausgelöst, auf das wir reagieren können. Wie wir solche Ereigniseigenschaften nutzen können und wir Du diese anlegst, zeigen wir in diesem Artikel.

Ereignisprozeduren anlegen

Im Artikel **twinBASIC: Forms öffnen, schließen und mehr** (www.vbentwickler.de/389) haben wir gelernt, wie wir **Form**-Elemente öffnen und schließen können, aber eine weitere, mächtige Möglichkeit haben wir uns noch nicht angesehen. Dabei handelt es sich um die Ereigniseigenschaften der **Form**-Klasse.

Was aber sind Ereignisse, Ereigniseigenschaften und Ereignisprozeduren überhaupt?

Ereignisse existieren erst einmal unabhängig von Dingen wie Ereigniseigenschaften oder Ereignisprozeduren. Ereignisse geschehen bei der Benutzung – ein **Form**-Element wird geöffnet, eine Schaltfläche wird angeklickt, ein Form-Element verliert den Fokus, weil ein anderes **Form**-Element diesen erhält, der Benutzer klickt mit rechten Maustaste auf einen Bereich, um ein Kontextmenü anzuzeigen und so weiter.

Hier setzen wir mit Ereigniseigenschaften und Ereignisprozeduren an, um uns diese Ereignisse zu Nutze zu machen. Dabei legen wir im Prinzip fest, dass beim Auftreten eines Ereignisses eine bestimmte Prozedur ausgelöst werden soll. Das geschieht durch das Erstellen einer Ereignisprozedur und durch das Einstellen

einer Ereigniseigenschaft auf den Namen dieser Prozedur (zumindest in twinBASIC – in anderen Entwicklungsumgebungen gibt es andere Vorgehensweisen).

Dann geschieht Folgendes:

- Der Benutzer führt eine Aktion durch, für die es eine Ereigniseigenschaft gibt, beispielsweise das Anklicken des **Form**-Elements.
- Die Anwendung schaut, ob für die entsprechende Ereigniseigenschaft eine Ereignisprozedur hinterlegt ist.
- Falls ja, wird diese Ereignisprozedur aufgerufen.

Da sich dies sehr theoretisch anhört, schauen wir uns nun in der Praxis an, wie es funktioniert.

Beispiel für eine Ereignisprozedur

Wir schauen uns eines der ersten Ereignisse an, das beim Öffnen eines Formulars ausgelöst wird, und das durch die **Load**-Eigniseigenschaft repräsentiert wird.

Um eine Ereignisprozedur anzulegen, die durch dieses Ereignis ausgelöst wird, wechseln wir in der Entwurfs-

twinBASIC: Grundlagen zur App-Entwicklung

VB6-Anwendungen – braucht das noch jemand? Oh ja! Gerade wer mal eben schnell eine Windows-Anwendung mit einer Benutzeroberfläche (oder auch ohne) programmieren will, kann immer noch gut auf seine guten, alten Visual Basic 6-Kenntnisse zurückgreifen – und zwar mit dem Nachfolger twinBASIC. In diesem Magazin haben wir schon die eine oder andere Lösung damit programmiert, die wir als COM-Add-In oder als COM-DLL für die Integration in eine der Office-Anwendungen oder auch zur Erweiterung von VBA genutzt haben. Und in diesem Artikel gehen wir noch einen Schritt weiter: Wir zeigen die grundlegenden Werkzeuge von twinBASIC zur Entwicklung von ausführbare twinBASIC-Anwendung mit eigener Benutzeroberfläche. Mit twinBASIC können wir unsere .exe-Anwendung in Zukunft vielleicht sogar auf Nicht-Windows-Rechnern laufen lassen. Zumindest aber ist die Kompatibilität mit 64-Bit-Systemen gesichert.

VB6 im modernen Zeitalter: Ein Relikt mit Wert

Visual Basic 6 (VB6) mag auf den ersten Blick wie ein Überbleibsel aus der Technologie-Vergangenheit erscheinen, doch die Realität ist, dass diese robuste Programmiersprache immer noch ihren festen Platz in der Software-Entwicklung hat. Erstens ist die Migration von alten VB6-Anwendungen auf neuere Technologien oft ein zeitaufwändiger und kostspieliger Prozess, den viele Unternehmen scheuen. Daher bleiben VB6-Programme in Betrieb und erfordern weiterhin Wartung und Support.

Zweitens bietet VB6 eine vertraute Umgebung für Entwickler, die über Jahrzehnte hinweg mit der Sprache gearbeitet haben, und ermöglicht eine schnelle und effiziente Anwendungsentwicklung. Auch wenn VB6 nicht die neuesten Funktionen oder Sprachkonstrukte bietet, bleibt seine Einfachheit und Effizienz in vielen Kontexten unübertroffen. Es ist ein klares Beispiel dafür, dass es nicht immer das Neueste und Beste sein muss, um relevant und wertvoll in der IT-Landschaft zu bleiben.

twinBASIC als Entwicklungsumgebung

Neu ist der Compiler und die Entwicklungsumgebung, die wir in diesem Fall nutzen. Leser unseres Magazins kennen twinBASIC bereits aus verschiedenen Veröffentlichungen, daher empfiehlt sich für den Einstieg

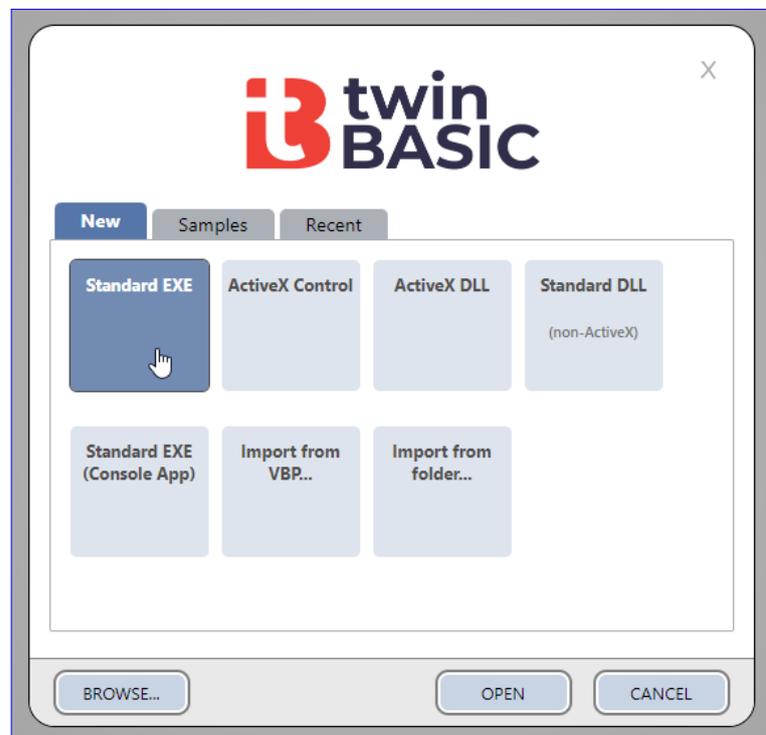


Bild 1: Erstellen einer Standard-Exe

twinBASIC: Menüs erstellen

Während man sich beim Programmieren von Office-Anwendungen wie Word, Excel, Outlook oder Access mit XML-Definitionen für das Ribbon beschäftigen muss, kann man unter twinBASIC zumindest für die Fenstermenüs auf eine einfache und praktische Benutzeroberfläche zurückgreifen. Okay, für Kontextmenüs ist dann doch wieder VB-Code gefragt, aber das ist auch in den meisten Office-Anwendungen noch die gängige Vorgehensweise (nicht in Outlook, dort sind die Kontextmenüs bereits in die Ribbondefinition integriert). In diesem Artikel schauen wir uns erst einmal die Möglichkeiten an, twinBASIC-Anwendungen über die Benutzeroberfläche mit Menüs auszustatten und ihre Eigenschaften per Code anzupassen.

Hauptmenü anlegen

Bereits wenn wir ein Formular in twinBASIC öffnen, ist die Möglichkeit zum Hinzufügen eines Menüs nicht zu übersehen (siehe Bild 1).

Klicken wir diese Schaltfläche an, erscheint direkt der neue Menüeintrag und bietet die Möglichkeit, den Namen anzupassen (siehe Bild 2).

Schon jetzt können wir das Projekt starten und den Menüpunkt sehen (siehe Bild 3). Durch wiederholtes Betätigen der Schaltfläche können wir weitere Hauptmenüpunkte hinzufügen.

Menüpunkte hinzufügen

Klicken wir nun auf das ersten angelegten Menü, sehen wir eine weitere Schaltfläche (siehe Bild 4). Mit einem Klick fügen wir einen Menübefehl hinzu.

Dieser bietet auch direkt die Möglichkeit zum Umbenennen an (siehe Bild 5).

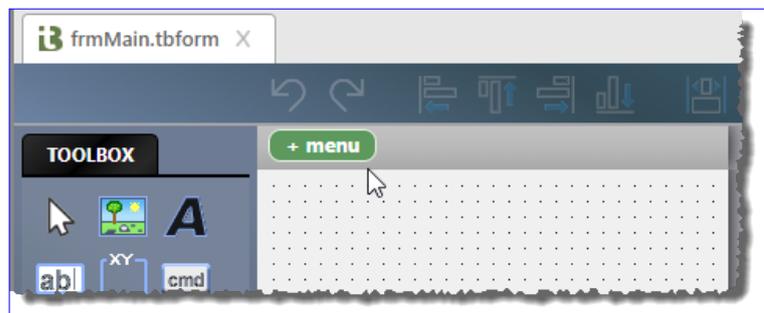


Bild 1: Button zum Anlegen eines Menüs

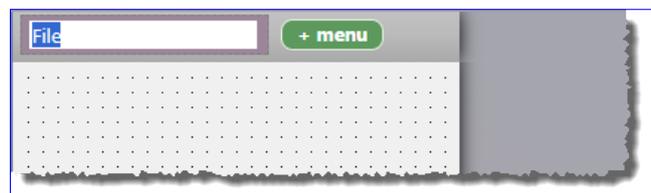


Bild 2: Ändern der Menübeschriftung

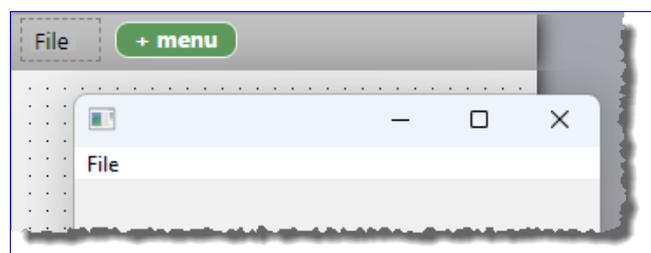


Bild 3: Der erste Menüpunkt im Einsatz

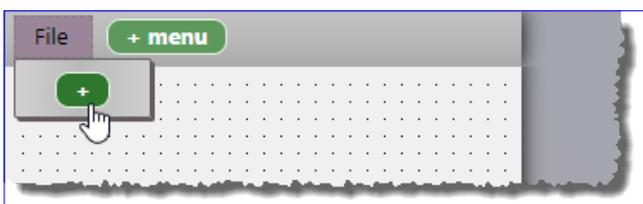


Bild 4: Hinzufügen eines Menüpunktes

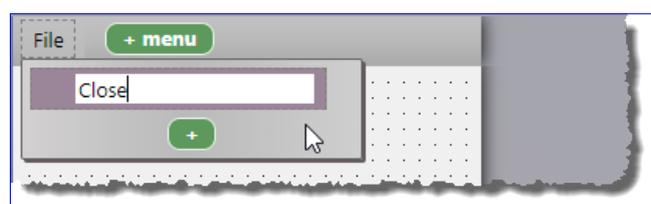


Bild 5: Der neue Menüpunkt

twinBASIC: Überblick der Controls und Eigenschaften

twinBASIC bietet genau wie Visual Basic 6 einige Steuerelemente an – neben den vielfach verwendeten Elementen wie Button, TextBox und Label haben wir die Auswahl zwischen vielen weiteren Steuerelementen, mit denen sich die wesentlichen Anforderungen an Desktop-Anwendungen umsetzen lassen. In diesem Artikel schauen wir uns die Steuerelemente einmal im Überblick an und betrachten einige der Eigenschaften, welche die meisten Steuerelemente gemeinsam haben. In weiteren Artikeln gehen wir dann im Detail auf die einzelnen Steuerelemente ein und zeigen, wie wir diese programmieren und nutzen können.

Steuerelemente unter twinBASIC

twinBASIC bietet soweit die gleichen Steuerelemente, die wir auch in VB6 finden. Sobald wir eine Form in der Entwurfsansicht öffnen, erscheint links daneben der Bereich **TOOLBOX** mit den Steuerelementen (siehe Bild 1). Im oberen Bereich finden wir die eingebauten Steuerelemente von twinBASIC beziehungsweise VB6. Darunter werden weitere Steuerelemente angezeigt, die wir uns ebenfalls ansehen werden.

Hier sind die Standard-Steuerelemente:

- **PictureBox:** Ein Steuerelement, das dazu dient, Grafiken, Bilder oder sogar Zeichnungen anzuzeigen. Es kann auch als Container für andere Steuerelemente dienen.
- **Label:** Ein Steuerelement, das dem Benutzer Text anzeigt. Es wird häufig verwendet, um anderen Steuerelementen auf einem Formular einen Kontext oder eine Beschriftung zu geben. Das **Label**-Steuerelement kann keinen Fokus erhalten und der Benutzer kann seinen Inhalt nicht direkt ändern.
- **TextBox:** Ein Steuerelement, das dem Benutzer ermöglicht, Text einzugeben oder anzuzeigen. Es kann auch dazu verwendet

werden, Passwörter oder andere verdeckte Eingaben zu akzeptieren.

- **Frame:** Ein Container-Steuerelement, das dazu dient, andere Steuerelemente zu gruppieren und zu organisieren, oft in Kombination mit **OptionButton**-Elementen, um eine Gruppe zu bilden.

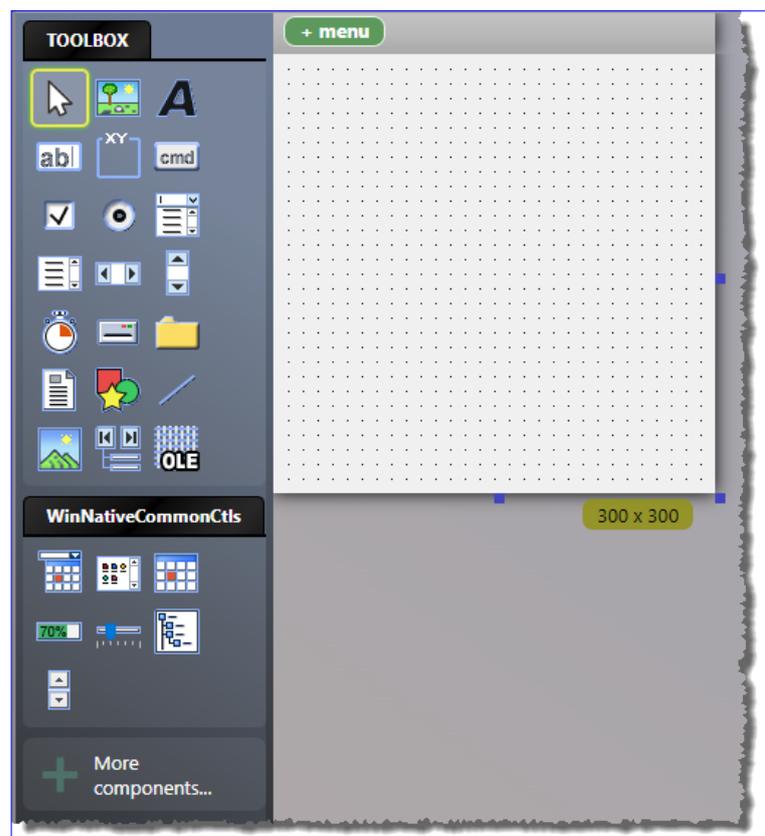


Bild 1: Die Toolbox mit den Steuerelementen

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

DATENAUSTAUSCH MIT GOOGLE CALENDAR

Im Schwerpunkt stellen wir den Zugriff auf den Google Calendar per Rest API vor.

SEITE 46

ANWENDUNGSDATEN IN DER REGISTRY

Optionen und andere Anwendungsdaten kann mit speziellen Befehlen leicht in der Registry speichern und wieder einlesen.

SEITE 4

JSON-DOKUMENTE PER VBA ZUSAMMENSTELLEN

Die kompliziert aufgebauten JSON-Dokumente stellen wir jetzt einfach per VBA zusammen..

SEITE 10



André Minhorst Verlag

Anwendungsdaten in der Registry

Wenn man es gewohnt ist, mit Access zu arbeiten, liegt es nahe, Anwendungsdaten wie Optionen et cetera in einer Tabelle der Datenbankdatei zu speichern. Unter Excel, Word, Outlook oder auch für twinBASIC-Anwendungen ist das nicht so einfach. Wir könnten zwar eine Datenbank zu diesem Zweck heranziehen, aber je nach Anwendungsfall gibt es praktischere Lösungen – zum Beispiel Textdateien, XML-Dateien oder auch die Registry. Letztere schauen wir uns in diesem Artikel an. Wie können wir dort Einstellungen sichern und wieder abrufen? Wo in der Registry landen diese dann? Können wir überhaupt per VB, VBA und twinBASIC darauf zugreifen? All dies klären wir auf den folgenden Seiten.

Anwendungsdaten verwalten

Wie schon in der Einleitung geschrieben, gibt es viele Orte, an denen man Anwendungsdaten speichern kann. In einer Access-Datenbank würde man direkt eine Tabelle der verwendeten Datenbank nutzen und in allen Anwendungen wäre es möglich, mit wenigen Zeilen Code Daten in Text- oder XML-Dateien zu schreiben und diese auszulesen. Eine weitere Option ist die Registry.

Welche der Optionen man nutzt, hängt von der Art der zu speichernden Daten ab. Wir gehen in diesem Artikel davon aus, dass es sich nur um einige wenige Optionen handelt, die gespeichert werden sollen.

Anwendungsdaten in der Registry

Es gibt einige API-Funktionen, mit denen man an beliebige Stellen der Registry schreiben kann. Je nachdem, wohin man schreiben möchte, benötigt man aber auch wieder Administrator-Rechte.

Grundsätzlich ist es auch keine besonders gute Idee, die Registry an nicht dafür vorgesehenen Stellen zu verändern – selbst wenn man nur ein paar Datensätze hinzufügt.

Neben diesen API-Funktionen gibt es aber auch noch spezielle VBA-Funktionen, mit denen wir bestimmte Bereiche der Registry für uns nutzen können. Dazu

brauchen wir keine Admin-Rechte, denn diese Einträge liegen innerhalb des Bereichs des jeweiligen Benutzers.

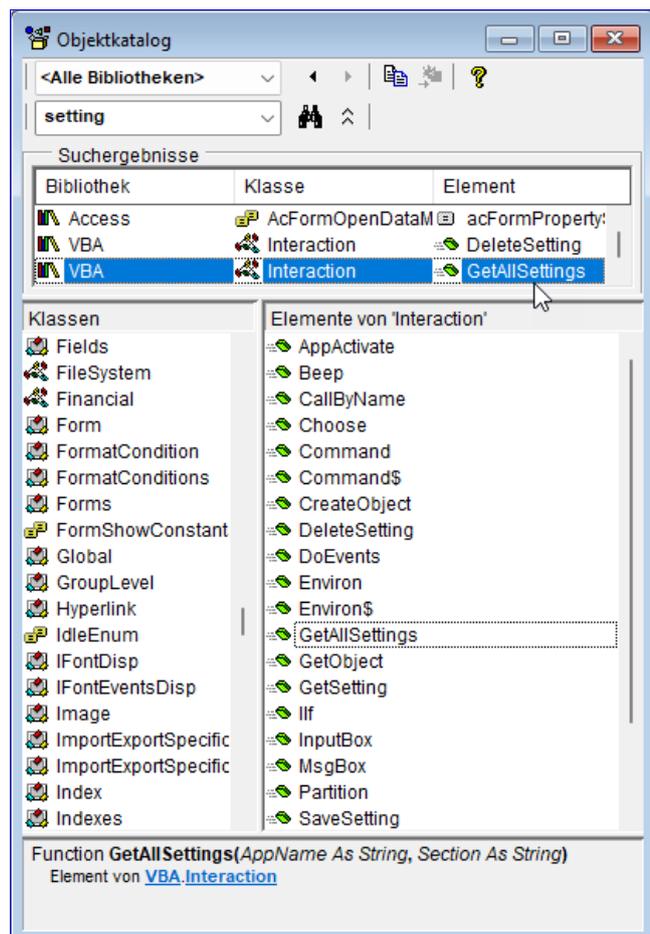


Bild 1: Elemente der Interaction-Klasse

Google Calendar per Rest-API programmieren

Google-Kalender sind praktisch: Sie sind von überall erreichbar, können mit endlos vielen Schnittstellen und Diensten verbunden werden und bieten weitere Vorteile. Noch schöner wäre es natürlich, wenn wir auch per VBA auf diese Kalender zugreifen könnten. Die Vorbereitungen haben wir bereits in zwei weiteren Artikeln erledigt – damit haben wir eine App bei Google angelegt und eine COM-DLL programmiert, mit der wir ein Token für die Authentifizierung bei Google generieren können. Damit folgt nun die Kür: Wir erzeugen VBA-Prozeduren, um auf die verschiedenen Informationen der Google Calendar API zuzugreifen. Dazu gehören Kalender, Termine und vieles mehr. Welche Möglichkeiten sich zum Lesen, Schreiben, Ändern und Löschen von Terminen bieten, zeigen wir auf den folgenden Seiten.

Keine Rest-API ohne JSON

Die Kommunikation mit der Rest-API des Google Calendars erfolgt wie bei den meisten Rest-APIs über JSON. Wir haben dazu bereits eine Bibliothek vorgestellt, die uns den Zugriff auf die in JSON-Dateien enthaltenen Informationen unter VBA stark vereinfacht. Alles Weitere dazu liest Du im Artikel **Mit JSON arbeiten** (www.vbentwickler.de/361).

Vorbereitungen

Wie schon in der Einleitung erwähnt, sind zwei wichtige Schritte nötig, bevor wir mit der eigentlichen Programmierung des Rest-API-Zugriffs beginnen können. Jeder wird in einem eigenen Artikel abgehandelt:

- **Google Calendar programmieren: Vorbereitungen** (www.vbentwickler.de/408): Hier liest Du, wie Du eine App bei Google anlegst und die Client-ID und das Client-Secret ermittelst, das für das Ermitteln eines Tokens für den Zugriff auf die Rest-API erforderlich ist.
- **Google-Token per DLL holen** (www.vbentwickler.de/409): Hier zeigen wir, wie Du eine COM-DLL programmierst, mit

der Du auf Basis der Client-ID und des Client-Secret ein Access-Token für den Zugriff auf den Kalender eines Benutzers sowie ein Refresh-Token für das schnelle Aktualisieren des Access-Tokens generieren kannst.

Außerdem fügen wir der Anwendung, egal, ob es sich um eine Excel-, Outlook-, Access- oder twinBASIC-Anwendung handelt, neben dem Verweis auf die COM-DLL **amvGoogleOAuth2** noch zwei weitere wichtige Verweise hinzu. Dabei handelt es sich um Mi-

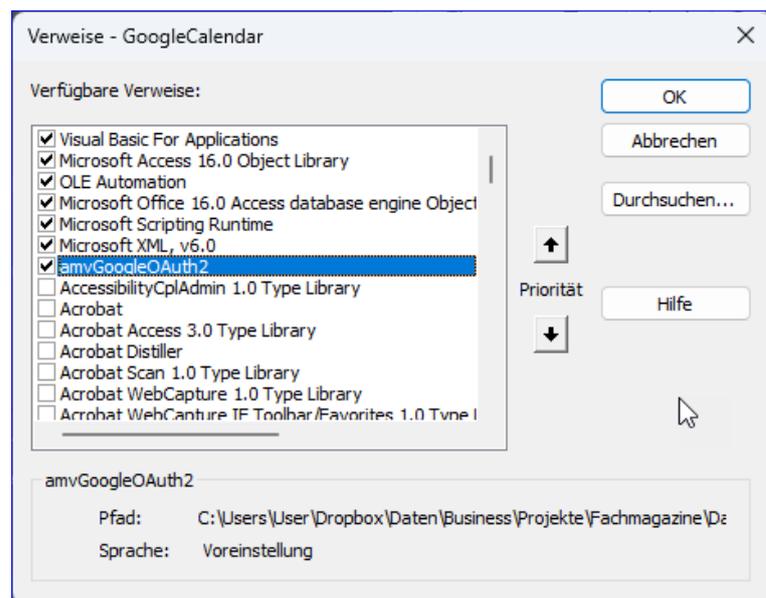


Bild 1: Verweise für die Lösung

Google-Token per DLL holen

Wenn wir Daten eines Google-Kontos wie beispielsweise Google Calendar programmieren wollen, um diese abzurufen oder zu bearbeiten, benötigen wir verschiedene Dinge. Das erste ist eine Anwendung, die wir wie im Artikel »Google Calendar programmieren: Vorbereitungen« erläutern. Das erste Resultat ist die Definition einer Anwendung. Das zweite, wichtige Ergebnis sind zwei für den Zugang notwendige Daten, nämlich ClientID und ClientSecret. Mit diesen können wir dann online ein OAuth2-Token holen, das wir wiederum für den Zugriff auf die Rest-API von Google benötigen. Im vorliegenden Artikel zeigen wir, wie man ClientID und ClientSecret nutzt, um das begehrte Token auszulesen. Dazu erstellen wir eine COM-DLL, die wir von Office-Anwendungen, VB6 oder twinBASIC aus nutzen können, um über die Rest-API von Google beispielsweise auf Kalenderdaten zugreifen zu können.

OAuth2-Token – woher nehmen?

Das Objekt unserer Begierde für den Zugriff auf die Rest-API beispielsweise des Google Calendars ist ein sogenanntes Token. Um dieses zu erhalten, sind einige Voraussetzungen zu erfüllen.

Als Erstes müssen wir ein Projekt erstellen, in dessen Kontext wir auf die Rest-API zugreifen wollen. Ist dieses Projekt angelegt, erhalten wir von Google zwei Informationen: die **ClientID** und das **ClientSecret**. Wie wir an diese Informationen kommen, beschreiben wir im Artikel **Google Calendar programmieren: Vorbereitungen** (www.vbentwickler.de/408).

Nun wollen wir die Daten des Google Calendars für einen bestimmten Benutzer auslesen oder bearbeiten. Beziehungsweise möchten wir dem Benutzer eine Anwendung zur Verfügung stellen, mit der er auf seinen Google Calendar zugreifen kann. Damit dies gelingt, benötigt er ein Token. Dieses Token wird auf Basis von **ClientID** und **ClientSecret** unseres Projekts und einer Anmeldung des Benutzers mit seinen Benutzerdaten erzeugt. Diese Anmeldung erfordert die Anzeige eines Web-Dialogs von Google, wo der Benutzer seine Daten eingibt. Danach können wir das Token auslesen. All dies erledigen wir mit einer COM-DLL, wie wir sie in diesem Artikel beschreiben. Wir übergeben dieser

die **ClientID** und das **ClientSecret** und ermitteln das Token. Beim ersten Aufruf muss der Benutzer seine Daten eingeben, danach erst wieder nach einer bestimmten Zeit – in der Zwischenzeit kann das einmal generierte Token verwendet werden.

Warum mit Visual Studio?

Warum erledigen wir diese Aufgabe mit einer COM-DLL auf Basis von .NET – können wir dies nicht mit VB, VBA oder twinBASIC programmieren? Das ist vielleicht möglich, aber unter .NET finden wir fertige Pakete, die das Ermitteln des Tokens wesentlich vereinfachen. Wir benötigen dazu lediglich ein NuGet-Paket, das wir dem Projekt hinzufügen und anschließend mit wenigen Codezeilen nutzen können.

Visual Studio als Administrator starten

Die benötigte COM-DLL erstellen wir also mit Visual Studio .NET. Damit wir es gleich von Visual Studio aus registrieren können, müssen wir dieses als Administrator starten – dabei hilft ein Rechtsklick auf den Eintrag **Visual Studio 2022** (oder andere Version) und Auswahl des Befehls **Als Administrator öffnen**.

Neues Projekt erstellen

Als Erstes erstellen wir ein neues Projekt namens **amvGoogleOAuth2** mit dem Typ **Klassenbibliothek**

Google Calendar programmieren – Vorbereitungen

Einer der kompliziertesten Vorgänge beim Zugriff auf Rest APIs und ähnliche Dienste ist das Ermitteln des für die Authentifizierung notwendigen Tokens. Bevor das überhaupt möglich ist, müssen wir jedoch eine App bei Google anlegen, die uns den Zugriff im Kontext des jeweiligen Benutzers erlaubt. Das einfache Anlegen einer App reicht dazu nicht aus – für das Abfragen des zum Anmelden notwendigen Tokens benötigen wir zwei Daten namens ClientID und ClientSecret. Wie wir das alles organisieren, zeigen wir im vorliegenden Artikel. Voraussetzung ist, dass bereits ein Google-Konto vorhanden ist, mit dem wir die App anlegen können. In weiteren Artikeln zeigen wir dann, wie wir ClientID und ClientSecret für die Abfrage des Tokens nutzen können wie wir damit schließlich auf die Rest-API von Google zugreifen können, um Daten des Google Calendars zu lesen, zu erstellen, zu bearbeiten und zu löschen.

Google-Konto erforderlich

Wenn Du noch kein Google-Konto hast, kannst Du kostenlos eines anlegen. Die Anmeldung bei Google mit einem Konto ist jedenfalls Voraussetzung für das Anlegen eines Projekts, das wir später für den Zugriff auf die Rest-API von Google nutzen können.

Wir starten auf folgender Adresse:

<https://console.cloud.google.com/>

Hier erscheint, wenn Du nicht bereits mit Deinem Google-Konto angemeldet sein solltest, der Dialog aus Bild 1. Hier kannst Du entweder die Daten eines be-

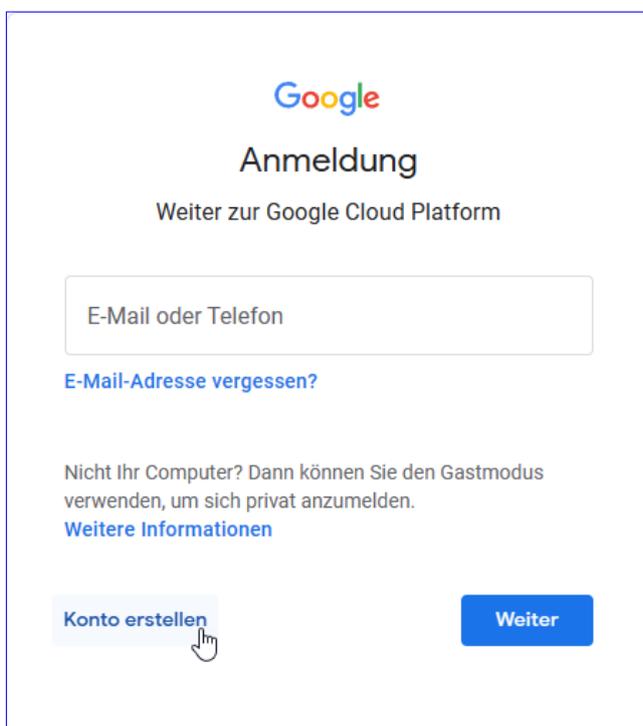


Bild 1: Anmelden oder neues Konto erstellen



Bild 2: Auswahl des Kontotyps

OAuth2-Token für Google per .NET-App holen

Einer der wenigen Schritte, die wir nicht mit klassischem Visual Basic oder per VBA abgebildet haben, ist das Ermitteln eines OAuth2-Tokens für Rest-APIs wie die von Google oder anderen Anbietern. Die Aufgabe ist, mit den online einmalig ermittelten Daten Client-ID und Client-Secret ein Access-Token oder noch besser ein Refresh-Token zu holen. Das Access-Token ist in der Regel zeitlich begrenzt, das Refresh-Token ist haltbarer und ermöglicht es uns, neue Access-Tokens zu holen – dies übrigens mit einer reinen VB6/VBA-Prozedur. In diesem Artikel zeigen wir, wie wir eine kleine Anwendung mit Benutzeroberfläche auf Basis von WPF/VB.NET in Visual Studio erstellen. Diese soll die Eingabe von Client-ID und Client-Secret erlauben und dafür die Werte eines Refresh- und eines Access-Tokens zurückliefern.

Projekt erstellen

Nach dem Starten von Visual Studio, in diesem Fall in der Version 2022, legen wir ein neues Projekt des Typs **WPF-Anwendung** an. Um dieses schnell anzuzeigen, wählen wir als Filter **Visual Basic**, **Windows** und **Desktop** aus (siehe Bild 1).

Damit landen wir beim zweiten Schritt, wo wir das Projekt konfigurieren. Hier legen wir den Projektna-

men fest und den Namen der Projektmappe (in diesem Fall identisch) und wählen den Speicherort für das Projekt aus (siehe Bild 2). Im nächsten Schritt behalten wir die Voreinstellung bei und erstellen das Projekt.

Benutzeroberfläche definieren

Danach finden wir den Entwurf des Startfensters der Anwendung namens **MainWindow.xaml** vor. Hier können wir nun die Steuerelemente anlegen, die wir

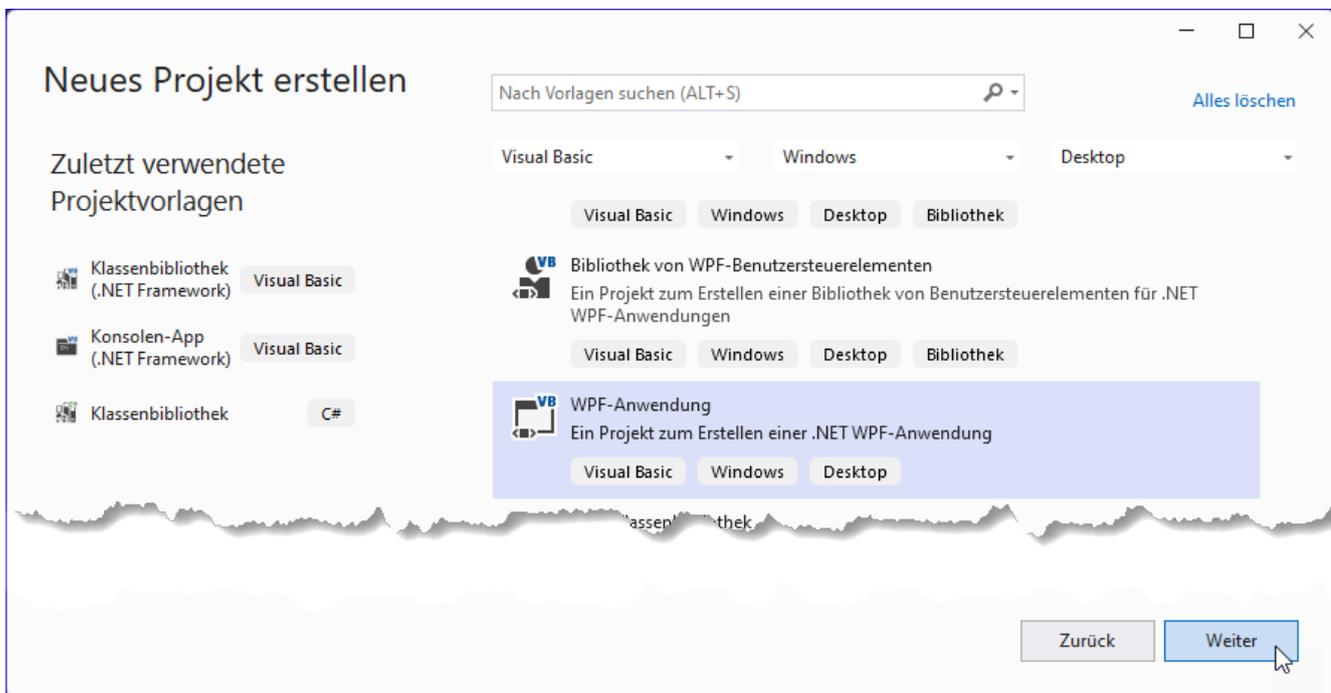


Bild 1: Anlegen einer WPF-Anwendung

JSON-Dokumente per Objektmodell zusammenstellen

Der Umgang mit JSON-Dokumenten wird immer wichtiger, da immer mehr Rest-APIs dieses Format nutzen, um Daten auszutauschen. In einem weiteren Artikel namens [Mit JSON arbeiten](http://www.vbentwickler.de/361) (www.vbentwickler.de/361) haben wir bereits gezeigt, wie wir JSON-Dokumente möglichst einfach per VBA lesen können. Das ist schon die halbe Miete, aber die Kommunikation mit Rest-APIs ist keine Einbahnstraße: Wir müssen auch immer wieder mal Daten im JSON-Format an eine Rest-API senden. Wir nutzen dazu wiederum die Bibliothek von Tim Hall, die uns eine Funktion namens `ConvertToJSON` zur Verfügung stellt. Dieser müssen wir nun nur noch die Daten in einem entsprechenden Format übergeben. Wie das gelingt, zeigen wir im vorliegenden Artikel.

Im Artikel [Google Calendar per Rest-API programmieren](http://www.vbentwickler.de/410) (www.vbentwickler.de/410) hatten wir die Aufgabe, ein JSON-Dokument mit den Daten eines in Google Calendar anzulegenden Ereignisses zu erstellen. Das kann man auf die klassische Art machen, indem man dieses einfach zeilenweise als String in einer Variablen erfasst.

Dafür brauchen wir, wie in Listing 1 zu sehen, keine weiteren Hilfsmittel – der JSON-Ausdruck wird anschließend beispielsweise per `Debug.Print` wie folgt ausgegeben und kann so auch an die Rest-API geschickt werden:

```
{
  "start": {
    "dateTime": "2023-11-22T17:00:00",
    "timeZone": "Europe/Berlin"
  },
  "end": {
    "dateTime": "2023-11-22T18:30:00",
    "timeZone": "Europe/Berlin"
  },
  "summary": "Summary",
  "description": "description"
}
```

```
Public Sub JSONPerString()
    Dim strJSON As String
    strJSON = "{" & vbCrLf
    strJSON = strJSON & "  ""start"": {" & vbCrLf
    strJSON = strJSON & "    ""dateTime"": ""2023-11-22T17:00:00"," & vbCrLf
    strJSON = strJSON & "    ""timeZone"": ""Europe/Berlin"" & vbCrLf
    strJSON = strJSON & "  }," & vbCrLf
    strJSON = strJSON & "  ""end"": {" & vbCrLf
    strJSON = strJSON & "    ""dateTime"": ""2023-11-22T18:30:00"," & vbCrLf
    strJSON = strJSON & "    ""timeZone"": ""Europe/Berlin"" & vbCrLf
    strJSON = strJSON & "  }," & vbCrLf
    strJSON = strJSON & "  ""summary"": ""Summary"," & vbCrLf
    strJSON = strJSON & "  ""description"": ""description"" & vbCrLf
    strJSON = strJSON & "}" & vbCrLf
    Debug.Print strJSON
End Sub
```

Listing 1: Beispiel für das Erfassen eines JSON-Dokuments per `String`-Variable

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

OUTLOOK: KALENDER UND TERMINE

Im Schwerpunkt programmieren wir Kalender und Termine in Outlook.

SEITE 4

OUTLOOK: EREIGNISSE FÜR TERMINE

Anlegen, Bearbeiten und Löschen von Terminen lösen Ereignisse aus, die wir für viele Zwecke nutzen können.

SEITE 15

TERMINE VON OUTLOOK ZUM GOOGLE CALENDAR

Mit dieser Synchronisation machen wir Termine aus Outlook überall verfügbar.

SEITE 42



André Minhorst Verlag

Google-Authentifizierung mit OAuth2, Update

In den beiden Artikeln »OAuth2-Token für Google per .NET-App holen« (www.vbentwickler.de/413) und »Google-Token per DLL holen« (www.vbentwickler.de/409) haben wir Techniken beschrieben, mit denen wir ein Google OAuth2-Token ermitteln können, das wir für den Zugriff auf die Google Rest API per VBA benötigen. Dazu haben wir das NuGet-Paket `Google.Apis.Calendar.v3` verwendet. Leider funktionierte das Ermitteln des Access-Tokens mit dem Refresh-Token nicht wie gewünscht. Also stellen wir in diesem Artikel eine Erweiterung der Projekte aus den vorgenannten Artikeln vor, mit denen wir den Zugriff immer erneuern können – wenn auch jeweils auf Kosten einer erneuten Anmeldung über den Webbrowser.

Wie erhielten, wenn wir per VBA die mit den oben genannten Lösungen ermittelten Access-Token für den Zugriff auf die Google Calendar-API genutzt haben, in vielen Fällen die Meldung, dass Token nicht mehr gültig sei (siehe Bild 1).

Andererseits erlaubten uns die in den beiden Projekten verwendeten Versuche der Authentisierung über die Methode `AuthorizeAsync` der Klasse `GoogleWebAuthorizationBroker` nicht, das Token zu erneuern oder dieses erneut freizuschalten.

Nach einer Weile intensiven Suchens haben wir im Internet herausgefunden, dass diese Klasse eine Datei im folgenden Verzeichnis anlegt:

```
C:\Users\[Benutzername]\AppData\Roaming\
Google.Apis.Auth
```

Diese Datei heißt `Google.Apis.Auth.OAuth2.Responses.TokenResponse-user` und sieht wie in Bild 2 aus.

Erst, wenn wir diese löschen, erhalten wir die Möglichkeit, uns mit dem Web-

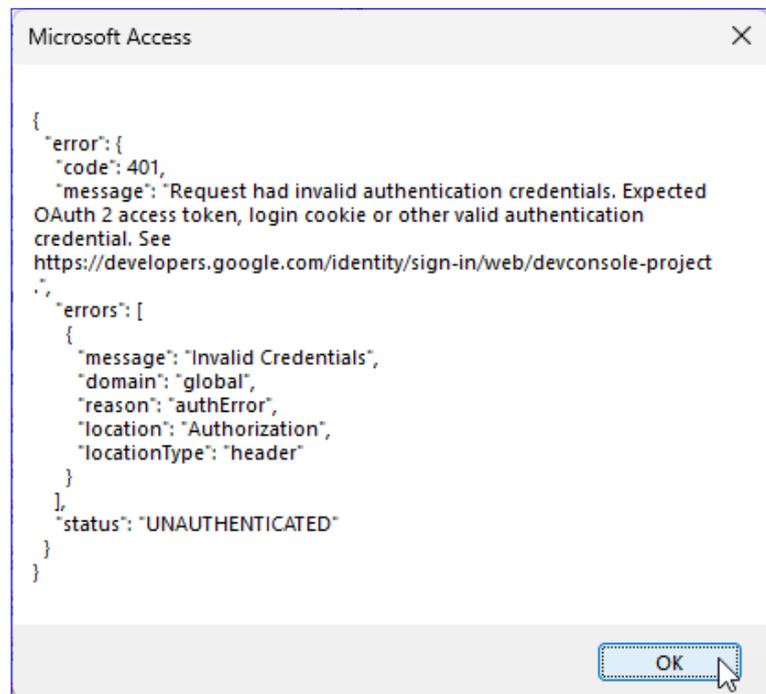


Bild 1: Fehlermeldung bei fehlgeschlagener Authentifizierung

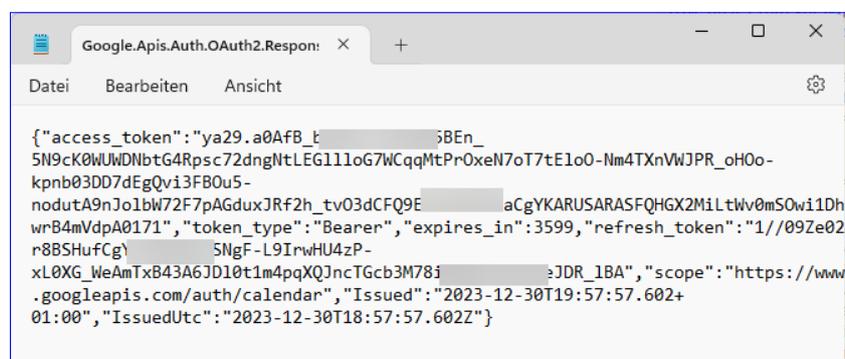


Bild 2: Die Datei mit den aktuellen Authentifizierungsdaten

Google-Authentifizierung mit OAuth2, Update

In den beiden Artikeln »OAuth2-Token für Google per .NET-App holen« (www.vbentwickler.de/413) und »Google-Token per DLL holen« (www.vbentwickler.de/409) haben wir Techniken beschrieben, mit denen wir ein Google OAuth2-Token ermitteln können, das wir für den Zugriff auf die Google Rest API per VBA benötigen. Dazu haben wir das NuGet-Paket `Google.Apis.Calendar.v3` verwendet. Leider funktionierte das Ermitteln des Access-Tokens mit dem Refresh-Token nicht wie gewünscht. Also stellen wir in diesem Artikel eine Erweiterung der Projekte aus den vorgenannten Artikeln vor, mit denen wir den Zugriff immer erneuern können – wenn auch jeweils auf Kosten einer erneuten Anmeldung über den Webbrowser.

Wie erhielten, wenn wir per VBA die mit den oben genannten Lösungen ermittelten Access-Token für den Zugriff auf die Google Calendar-API genutzt haben, in vielen Fällen die Meldung, dass Token nicht mehr gültig sei (siehe Bild 1).

Andererseits erlaubten uns die in den beiden Projekten verwendeten Versuche der Authentifizierung über die Methode `AuthorizeAsync` der Klasse `GoogleWebAuthorizationBroker` nicht, das Token zu erneuern oder dieses erneut freizuschalten.

Nach einer Weile intensiven Suchens haben wir im Internet herausgefunden, dass diese Klasse eine Datei im folgenden Verzeichnis anlegt:

C:\Users\[Benutzername]\AppData\Roaming\
Google.Apis.Auth

Diese Datei heißt `Google.Apis.Auth.OAuth2.Responses.TokenResponse-user` und sieht wie in Bild 2 aus.

Erst, wenn wir diese löschen, erhalten wir die Möglichkeit, uns mit dem Web-

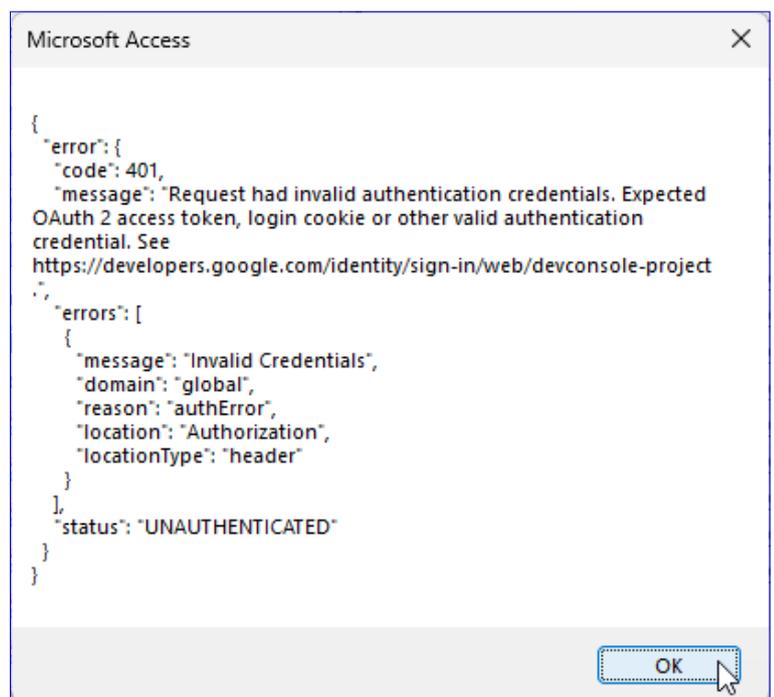


Bild 1: Fehlermeldung bei fehlgeschlagener Authentifizierung

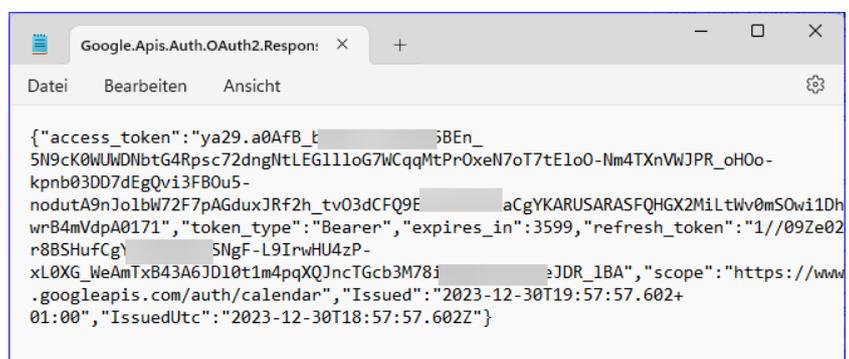


Bild 2: Die Datei mit den aktuellen Authentifizierungsdaten

Google Calendar per Rest-API programmieren, Teil 2

Im ersten Teil unserer Artikelserie haben wir grundlegende Funktionen vorgestellt, die es ermöglichen, Termine zwischen Outlook und Google Calendar zu synchronisieren. Jetzt gehen wir einen Schritt weiter und optimieren diese Funktionen an, um die Aktualisierung von Terminen effektiver zu gestalten. Zum Beispiel ist es von Vorteil, die EventID eines im Google Calendar erstellten Termins im entsprechenden Outlook-Termin zu speichern, um Änderungen zwischen beiden Kalendern synchron zu halten. Außerdem fügen wir neue Funktionen zur Aktualisierung von Kalendereinträgen und zum Abrufen von Terminen anhand ihrer EventID hinzu. Darüber hinaus diskutieren wir Herausforderungen, auf die wir gestoßen sind, und präsentieren Lösungen sowie Erweiterungen, die wir implementiert haben.

Anpassungen der bestehenden Funktionen

Die Funktionen aus dem ersten Teil dieser Artikelreihe, **Google Calendar per Rest-API programmieren** (www.vbentwickler.de/410), haben wir noch ein wenig angepasst, damit wir besser für Exporte und anschließend notwendige Aktualisierungen vorbereitet sind.

Wenn wir also einen Termin aus Outlook zum Google Calendar exportieren wollen, ist es sinnvoll, dass wir die **EventID** für den im Google Calendar angelegten Termin im Outlook-Termin zu speichern. Auf diese Weise können wir Änderungen an diesem Termin in Outlook auf den Termin im Google Calendar übertragen.

Und auch wenn der Termin in Outlook gelöscht wird, wollen wir dies gegebenenfalls im Google Calendar abbilden – also passen wir die Funktion, mit der wir einen Termin im Google Calendar anlegen, so an, dass diese die **EventID** des neu angelegten Termins zurückgibt.

Diese schreiben wir in der Funktion **InsertEvent** in den Parameter **strEventID** (siehe Listing 1). Außerdem haben wir noch weitere Parameter hinzugefügt, mit denen man die Zeitzone für das Start- und das Enddatum übergeben kann. Diese beiden Parameter heißen **strStartTimeZone** und **strEndTimeZone** und

erhalten den Standardwert **Europe/Berlin**. Und mit dem Parameter **bolAllDayEvent** können wir angeben, ob es sich bei dem anzulegenden Termin um einen Ganztagestermin handelt.

Für den Parameter **intColor** haben wir als Standardwert den Wert **collLavendel** vorgemerkt.

Die Zeitzonen übergeben wir in der Funktion jeweils für das Element **timeZone**.

Für die Verarbeitung des Parameters **bolAllDayEvent** mussten wir eine **If...Then**-Bedingung zur Funktion hinzufügen. Wenn **bolAllDayEvent** den Wert **False** enthält, werden die Werte für Start- und Enddatum wie zuvor hinzugefügt. Im **Else**-Teil der Bedingung behandeln wir den Fall eines ganztägigen Termins.

Hier ermitteln wir als Startzeit das Datum der mit **datStart** übergebenen Startzeit und stellen die Uhrzeit auf **00:00** ein, indem wir die Nachkommastellen von **datStart** mit der **Int**-Funktion entfernen.

Als Enddatum legen wir in **datEnd** das Datum des folgenden Tages fest – wieder mit der Uhrzeit **00:00**. Die Rest-API von Google Calendar sieht keine spezielle Eigenschaft für einen ganztägigen Termin vor, sodass wir diese Notation verwenden.

Outlook: Ereignisse für Termine implementieren

Wenn wir Aufgaben erledigen wollen, die in Zusammenhang mit dem Anlegen, Bearbeiten oder Löschen von Terminen zu tun haben, kommen wir nicht um die Programmierung der Ereignisse von Terminen herum. Die Ereignisse eines Termins selbst zu implementieren ist halbwegs intuitiv, aber wo finden wir zum Beispiel die Ereignisprozedur, die ausgelöst, wenn wir einen neuen Termin anlegen? Der Termin selbst kann dieses Ereignis noch nicht enthalten, denn es gibt ihn ja zu diesem Zeitpunkt noch nicht. Tatsächlich wollen wir auch erst auf das Speichern des neuen Termins reagieren. Dazu müssen wir diesen aber dennoch erst einmal mit einer geeigneten Objektvariablen referenzieren. Wie das gelingt und wie wir alle notwendigen Ereignisse bei der Nutzung eines Termins implementieren können, zeigen wir in diesem Artikel.

Outlook bietet die Möglichkeit, auf verschiedene Ereignisse zu reagieren, die mit dem Erstellen, Verschieben, Löschen oder Ändern von Terminen zu tun haben. Das ist immer dann interessant, wenn wir auf irgendeine Weise auf eines dieser Ereignisse reagieren müssen – beispielsweise, wenn wir die Termine mit einem Google Calendar oder einem anderen Kalender synchronisieren wollen, wie wir es in **Termine von Outlook zum Google Calendar exportieren** (www.vbentwickler.de/418) beschreiben.

Um diese Funktionen optimal zu nutzen, müssen wir jede Änderung an einem Termin von Outlook nach Google übertragen – und dabei handelt es sich um das Anlegen, Ändern und Löschen eines Termins.

Die erste Aufgabe hierbei ist, überhaupt herauszufinden, welche Ereignisse wir dazu nutzen können.

Die erste Anlaufstelle für solche Informationen ist immer der Objektkatalog des VBA-Editors (zu öffnen mit F2).

Hier suchen wir nach dem **AppointmentItem**-Element und erhalten direkt einige Ereignisse – das sind die mit dem Blitz-Symbol versehenen Einträge in Bild 1.

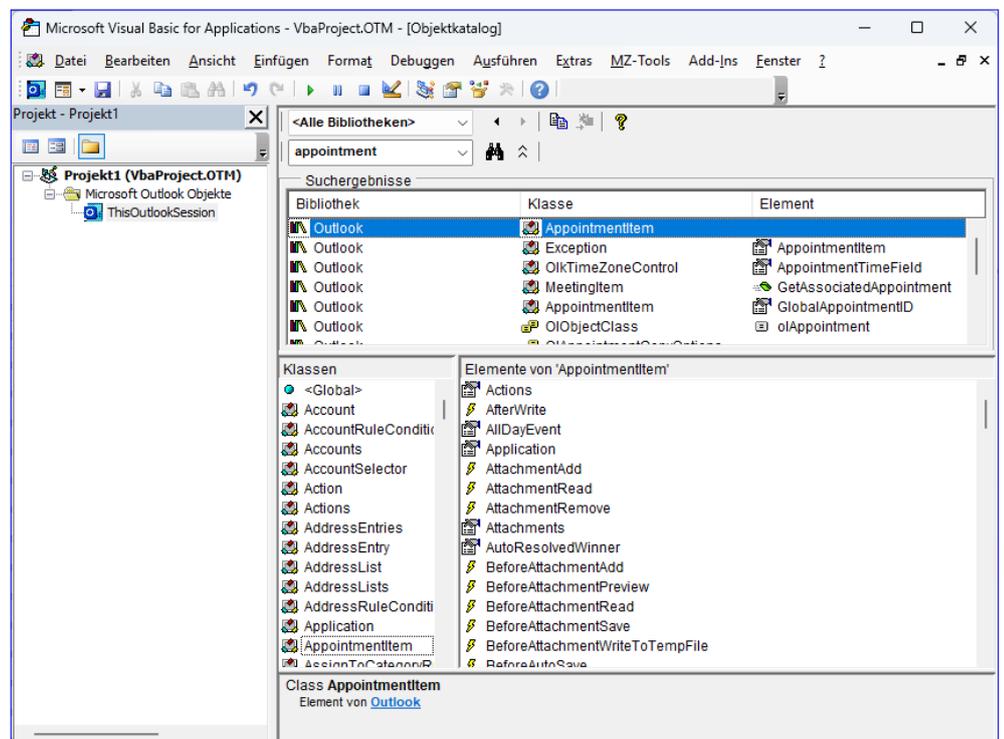


Bild 1: Ereignisse des Termins im Objektkatalog

Outlook: Kalender und Termine programmieren

Outlook-Kalender und -Termine sind neben den E-Mails und Kontakten weitere wichtige Elemente. In diesem Artikel schauen wir uns an, wie wir per VBA auf die einzelnen Kalender und die darin enthaltenen Termine zugreifen können. Dabei durchlaufen wir Kalender und Termine, um diese auszulesen, legen neue Termine an, löschen Termine und bearbeiten vorhandene Termine. Wozu das alles? Damit wir wissen, welche Elemente und welche Eigenschaften wir per VBA referenzieren müssen, um verschiedene Aufgaben erfüllen zu können: Zugriff von anderen Anwendungen, um Termine anzulegen, Termine zu lesen oder auch um Termine aus Outlook heraus in andere Kalenderanwendungen wie beispielsweise Google Calendar zu exportieren.

Den Outlook-Kalender dürfte mittlerweile jeder kennen: Er bietet auf der linken Seite die Monatsübersicht, auf der rechten Seite sehen wir die jeweils aktive Ansicht der Termine. In Bild 1 wird beispielsweise die

Monatsübersicht dargestellt. Daneben gibt es auch noch Tages- und verschiedene Wochenansichten. Die Ansichten interessieren uns in diesem Artikel jedoch nicht, denn wir wollen ausschließlich per VBA auf die

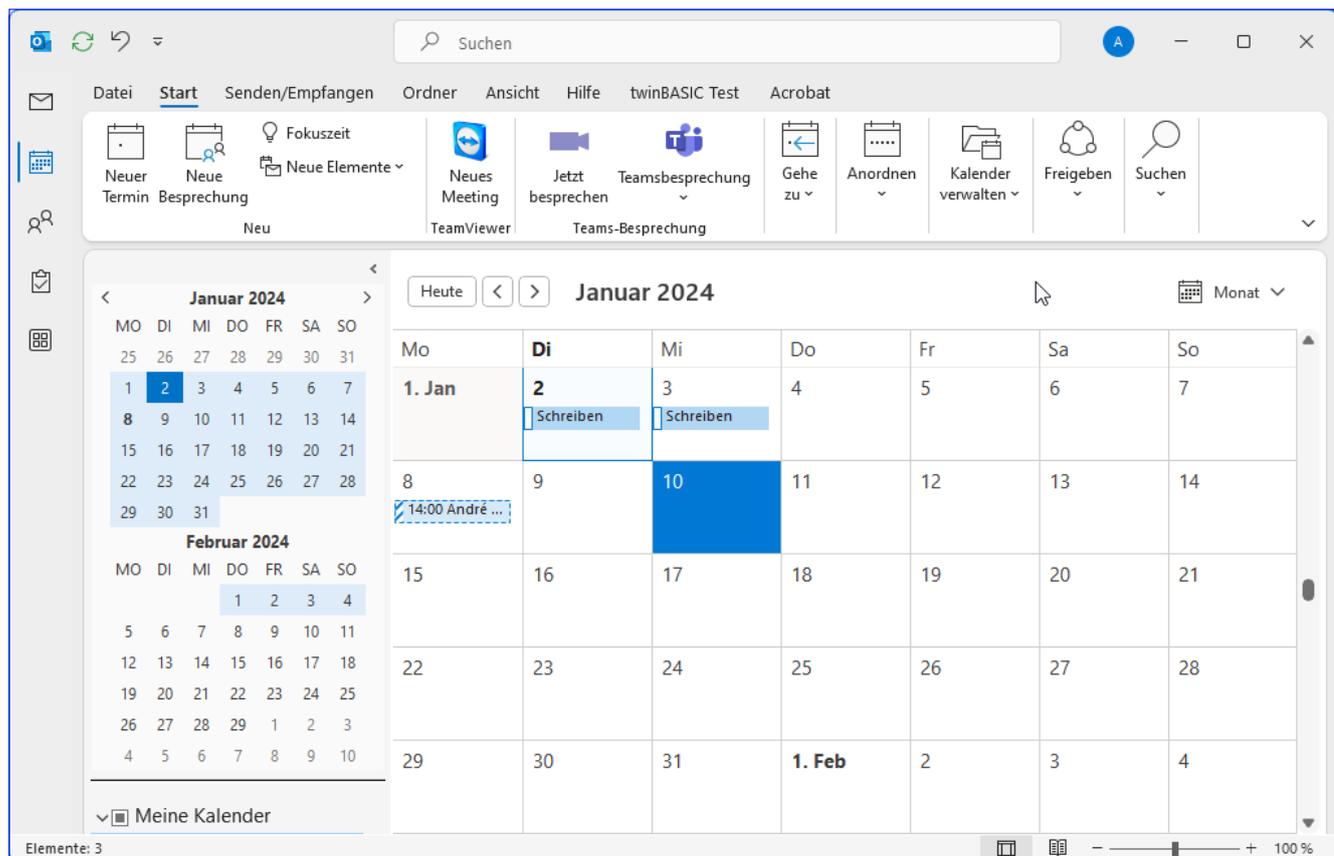


Bild 1: Der Outlook-Kalender in der Monatsansicht

Termine von Outlook zum Google Calendar exportieren

Wenn wir Outlook und Google Calendar nutzen, wollen wir vielleicht auch Termine von Outlook aus zum Google Calendar exportieren. In diesem Artikel zeigen wir, wie das für einfache Termine mit den grundlegenden Eigenschaften gelingen kann. Später schauen wir uns an, wie wir die hier geschaffenen Prozeduren von verschiedenen Stellen in Outlook aufrufen können – beispielsweise über das Ribbon oder das Kontextmenü eines Termins.

Ziel dieses Artikels

Wenn Du diesen Artikel bis zum Ende gelesen hast, verfügst Du über Kenntnisse, um einen Termin aus Outlook per VBA in den Google Calendar zu übertragen (siehe Bild 1). Zusätzlich erfährst Du, wie Änderungen an diesem Termin in Outlook automatisch in den Termin im Google Calendar übertragen werden.

Übertragen von Terminen mit Basisdaten

Termine von Outlook und von Google Calendar können sehr viele unterschiedliche Eigenschaften erfordern – gerade, wenn es um Serientermine geht. Deshalb wollen wir es in diesem Artikel erst einmal einfach halten und nur Termine mit Betreff, Inhalt und den wichtigsten Informationen wie Startdatum und

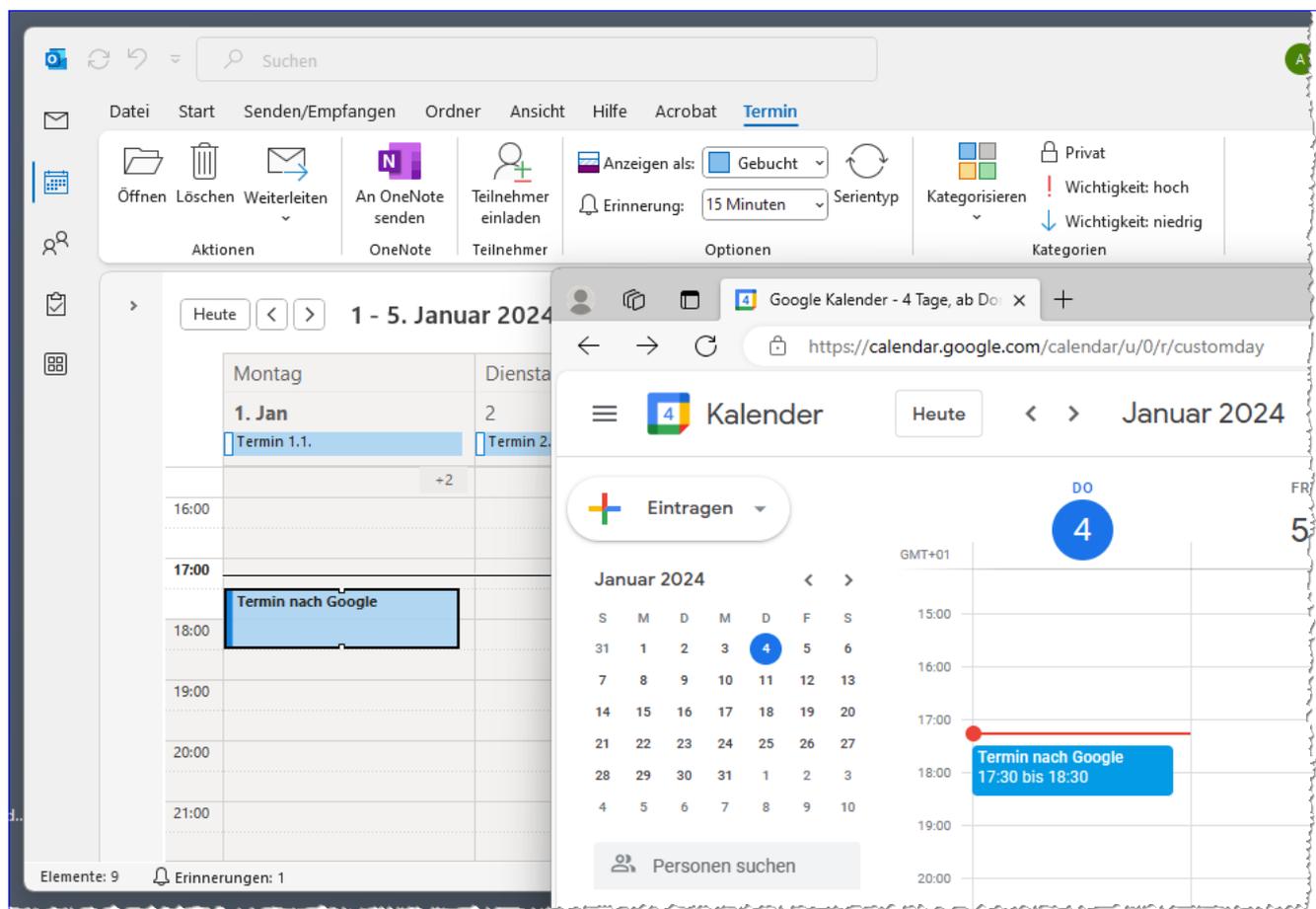


Bild 1: Ein von Outlook in den Google Calendar übertragener Termin

Outlook: Termine per COM-Add-In nach Google

In verschiedenen anderen Artikeln haben wir die Techniken vorgestellt, mit denen per VBA wir Termine in den Google Calendar eintragen können und auf Ereignisse wie das Anlegen, Ändern oder Löschen von Terminen in Outlook reagieren können. Wir wollen dies nun alles in einem COM-Add-In zusammenbringen. Das COM-Add-In stellt Ribbon- und Kontextmenü-Einträge bereit, mit denen man Termine per Mausklick nach Google übertragen kann und hält Automatismen bereit, die dafür sorgen, dass Änderungen an Terminen wie das Anlegen, Bearbeiten oder Löschen automatisch in den Google Calendar übertragen werden. Damit ist nur noch eine kurze Installation nötig, um diese Funktionen in Outlook bereitzustellen.

Voraussetzungen und Vorbereitungen

Die erste Voraussetzung ist, dass Du eine App bei Google erstellt hast. Wie das gelingt, zeigen wir im Artikel **Google Calendar programmieren – Vorbereitungen** (www.vbentwickler.de/408). Außerdem benötigst Du die .NET-DLL, mit der wir das Access-Token für den Zugriff auf den Google Calendar holen: **Google-Authentifizierung mit OAuth2, Update** (www.vbentwickler.de/414). Grundlegende Techniken zum automatischen Reagieren auf das Hinzufügen, Ändern und Löschen von Outlook-Terminen beschreiben wir in **Outlook: Ereignisse für Termine implementieren** (www.vbentwickler.de/417). Außerdem liefert der Artikel **Termine von Outlook zum Google Calendar exportieren** (www.vbentwickler.de/418) Informationen darüber, wie wir Termine von Outlook zum Google Calendar schicken. Schließlich lernst Du Einiges über das Programmieren der Rest-API von Google in den Artikeln **Google Calendar per Rest-API programmieren** (www.vbentwickler.de/410) und **Google Calendar per Rest-API programmieren, Teil 2** (www.vbentwickler.de/416)

Lösung ausprobieren

Bevor wir beschreiben, wie die Lösung funktioniert, zeigen wir erst einmal, wie Du sie auf Deinem Rechner installieren und ausprobieren kannst. Dazu führen wir folgende Schritte aus:

- .NET-DLL installieren
- COM-Add-In für Outlook installieren
- Google-App für den Zugriff auf den Kalender erstellen
- Outlook starten und die beim Erstellen der Google-App erhaltenen Daten (Client-ID, Client-Secret) in die Optionen eintragen
- Authentifizieren
- Loslegen!

Doch eins nach dem anderen.

.NET-DLL installieren

Die .NET-DLL aus dem Artikel **Google-Token per DLL holen** (www.vbentwickler.de/409) findest Du in den Verzeichnissen **x64** und **x86**. Je nachdem, welche Office-Version Du installiert hast, verwendest Du die aus dem Verzeichnis **x64** (64-Bit) oder **x86** (32-Bit). Du benötigst das komplette Verzeichnis auf Deinem Rechner. Wo Du dieses speicherst, spielt keine Rolle.

Um die jeweilige Version zu installieren, gehst Du wie folgt vor:

Outlook: Termine per COM-Add-In nach Google

In verschiedenen anderen Artikeln haben wir die Techniken vorgestellt, mit denen per VBA wir Termine in den Google Calendar eintragen können und auf Ereignisse wie das Anlegen, Ändern oder Löschen von Terminen in Outlook reagieren können. Wir wollen dies nun alles in einem COM-Add-In zusammenbringen. Das COM-Add-In stellt Ribbon- und Kontextmenü-Einträge bereit, mit denen man Termine per Mausklick nach Google übertragen kann und hält Automatismen bereit, die dafür sorgen, dass Änderungen an Terminen wie das Anlegen, Bearbeiten oder Löschen automatisch in den Google Calendar übertragen werden. Damit ist nur noch eine kurze Installation nötig, um diese Funktionen in Outlook bereitzustellen.

Voraussetzungen und Vorbereitungen

Die erste Voraussetzung ist, dass Du eine App bei Google erstellt hast. Wie das gelingt, zeigen wir im Artikel **Google Calendar programmieren – Vorbereitungen** (www.vbentwickler.de/408). Außerdem benötigst Du die .NET-DLL, mit der wir das Access-Token für den Zugriff auf den Google Calendar holen: **Google-Authentifizierung mit OAuth2, Update** (www.vbentwickler.de/414). Grundlegende Techniken zum automatischen Reagieren auf das Hinzufügen, Ändern und Löschen von Outlook-Terminen beschreiben wir in **Outlook: Ereignisse für Termine implementieren** (www.vbentwickler.de/417). Außerdem liefert der Artikel **Termine von Outlook zum Google Calendar exportieren** (www.vbentwickler.de/418) Informationen darüber, wie wir Termine von Outlook zum Google Calendar schicken. Schließlich lernst Du Einiges über das Programmieren der Rest-API von Google in den Artikeln **Google Calendar per Rest-API programmieren** (www.vbentwickler.de/410) und **Google Calendar per Rest-API programmieren, Teil 2** (www.vbentwickler.de/416)

Lösung ausprobieren

Bevor wir beschreiben, wie die Lösung funktioniert, zeigen wir erst einmal, wie Du sie auf Deinem Rechner installieren und ausprobieren kannst. Dazu führen wir folgende Schritte aus:

- .NET-DLL installieren
- COM-Add-In für Outlook installieren
- Google-App für den Zugriff auf den Kalender erstellen
- Outlook starten und die beim Erstellen der Google-App erhaltenen Daten (Client-ID, Client-Secret) in die Optionen eintragen
- Authentifizieren
- Loslegen!

Doch eins nach dem anderen.

.NET-DLL installieren

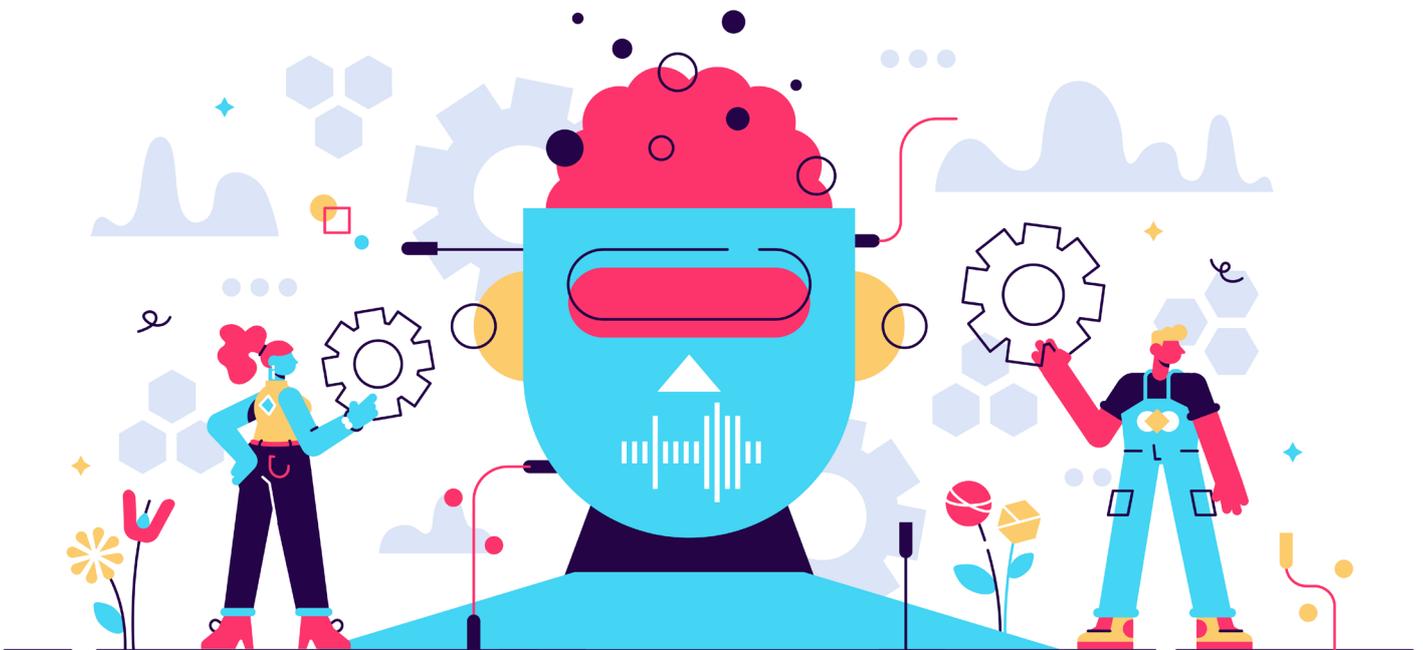
Die .NET-DLL aus dem Artikel **Google-Token per DLL holen** (www.vbentwickler.de/409) findest Du in den Verzeichnissen **x64** und **x86**. Je nachdem, welche Office-Version Du installiert hast, verwendest Du die aus dem Verzeichnis **x64** (64-Bit) oder **x86** (32-Bit). Du benötigst das komplette Verzeichnis auf Deinem Rechner. Wo Du dieses speicherst, spielt keine Rolle.

Um die jeweilige Version zu installieren, gehst Du wie folgt vor:

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

KLASSEN MIT VBA PROGRAMMIEREN

Im Schwerpunkt zeigen wir, wozu Klassen gut sind und wie Du sie programmierst – mit Eigenschaften, Methoden und Ereignissen.

SEITE 4

VBA-EDITOR PROGRAMMIEREN

Du musst nicht jede Codezeile selbst schreiben. Wir zeigen Dir, wie Du das per VBA automatisieren kannst.

SEITE 25

DATEIAUSWAHLDIALOGE PER ASSISTENT

Keine Lust mehr, immer die Dokumentation nachzuschlagen? Wir zeigen, wie Du mit wenigen Klicks alle Dateidialoge baust.

SEITE 51



André Minhorst Verlag

Dateiauswahl-Dialog-Assistent programmieren

Dateidialoge benötigt man immer wieder. Ob man nun Dateien zum Öffnen oder Bearbeiten auswählen möchte, ob man einen Pfad zum Speichern einer Datei braucht oder ob man ein Verzeichnis selektieren will – am einfachsten geht das mit den praktischen Dateidialogen. Die Office-Bibliothek bietet sogar alle benötigten Varianten über die `FileDialog`-Klasse an. Dumm ist nur, dass man nicht ständig Filedialoge programmiert, sondern nur alle paar Wochen, Monate oder sogar Jahre. Dann muss man sich immer wieder einarbeiten, um die verschiedenen Parameter – Titel, Schaltflächenbeschriftungen, Standardverzeichnis und `-dateiname`, Filter, Dateierweiterungen und so weiter zu definieren. Wie schön wäre es doch, wenn wir solche Dateidialoge mit einem kleinen Assistenten zusammenstellen könnten. Also machen wir uns ans Werk und schaffen einen solchen Wizard!

Wann, wie und wo nutzen wir den Dateidialog-Assistenten?

Die erste Frage, die sich stellt, ist: Wo und wie wollen wir diesen Assistenten aufrufen und was genau soll dieser eigentlich produzieren? Als Erstes fällt uns ein, dass er eine Funktion liefern soll, die den Dateidialog aufruft und uns die gewünschte Datei oder das Verzeichnis zurückliefert.

Wir könnten uns also darauf beschränken, diesen Assistenten für das Zusammenstellen des benötigten Codes zu nutzen und diesen dann beispielsweise an der aktuellen Stelle im Codefenster des VBA-Editors einzufügen oder diesen in die Zwischenablage zu schreiben, damit wir diesen selbst einfügen können.

In vielen Fällen mag das ausreichen: Dann ruft man den Dateidialog aus einer anderen Prozedur heraus auf und verwendet die gelieferten Informationen in den folgenden Codezeilen für den gewünschten Zweck.

In anderen Fällen spielt die Benutzeroberfläche eine Rolle: Dann möchte man vielleicht in einem Access-Formular ein Verzeichnis etwa zum Exportieren von Daten anzeigen und auswählen. Dazu fügt man diesem ein Textfeld oder ein Bezeichnungsfeld zur Anzeige

des Pfades hinzu und legt daneben eine Schaltfläche an, mit der man den Dateiauswahl-Dialog aufruft.

Letztlich würde auch hier erst einmal der Assistent zum Zusammenstellen von Funktionen zum Anzeigen von Dateidialogen ausreichen.

Man könnte die benötigten Steuerelemente und die Programmlogik zum Aufruf der Funktionen dann selbst hinzufügen – immerhin ist das der wesentlich weniger aufwendige Teil dieser Aufgabe.

Wir werden uns also zunächst darauf beschränken, den Code für die Anzeige von Dateidialogen abhängig von den gewünschten Parametern zu produzieren.

Wo wollen wir diese Funktionalität bereitstellen? Sinnvoll ist ein COM-Add-In für den VBA-Editor, denn dort werden wir den zu erstellenden Code auch nutzen.

Wo dort wollen wir die Funktion aufrufen? Hier gibt es drei mögliche Stellen:

- einen Menüeintrag, beispielsweise unterhalb des Eintrags **Add-Ins** oder in einem eigenen Hauptmenü

Dateiauswahl-Dialog-Assistent programmieren

Dateidialoge benötigt man immer wieder. Ob man nun Dateien zum Öffnen oder Bearbeiten auswählen möchte, ob man einen Pfad zum Speichern einer Datei braucht oder ob man ein Verzeichnis selektieren will – am einfachsten geht das mit den praktischen Dateidialogen. Die Office-Bibliothek bietet sogar alle benötigten Varianten über die `FileDialog`-Klasse an. Dumm ist nur, dass man nicht ständig Filedialoge programmiert, sondern nur alle paar Wochen, Monate oder sogar Jahre. Dann muss man sich immer wieder einarbeiten, um die verschiedenen Parameter – Titel, Schaltflächenbeschriftungen, Standardverzeichnis und `-dateiname`, Filter, Dateierweiterungen und so weiter zu definieren. Wie schön wäre es doch, wenn wir solche Dateidialoge mit einem kleinen Assistenten zusammenstellen könnten. Also machen wir uns ans Werk und schaffen einen solchen Wizard!

Wann, wie und wo nutzen wir den Dateidialog-Assistenten?

Die erste Frage, die sich stellt, ist: Wo und wie wollen wir diesen Assistenten aufrufen und was genau soll dieser eigentlich produzieren? Als Erstes fällt uns ein, dass er eine Funktion liefern soll, die den Dateidialog aufruft und uns die gewünschte Datei oder das Verzeichnis zurückliefert.

Wir könnten uns also darauf beschränken, diesen Assistenten für das Zusammenstellen des benötigten Codes zu nutzen und diesen dann beispielsweise an der aktuellen Stelle im Codefenster des VBA-Editors einzufügen oder diesen in die Zwischenablage zu schreiben, damit wir diesen selbst einfügen können.

In vielen Fällen mag das ausreichen: Dann ruft man den Dateidialog aus einer anderen Prozedur heraus auf und verwendet die gelieferten Informationen in den folgenden Codezeilen für den gewünschten Zweck.

In anderen Fällen spielt die Benutzeroberfläche eine Rolle: Dann möchte man vielleicht in einem Access-Formular ein Verzeichnis etwa zum Exportieren von Daten anzeigen und auswählen. Dazu fügt man diesem ein Textfeld oder ein Bezeichnungsfeld zur Anzeige

des Pfades hinzu und legt daneben eine Schaltfläche an, mit der man den Dateiauswahl-Dialog aufruft.

Letztlich würde auch hier erst einmal der Assistent zum Zusammenstellen von Funktionen zum Anzeigen von Dateidialogen ausreichen.

Man könnte die benötigten Steuerelemente und die Programmlogik zum Aufruf der Funktionen dann selbst hinzufügen – immerhin ist das der wesentlich weniger aufwendige Teil dieser Aufgabe.

Wir werden uns also zunächst darauf beschränken, den Code für die Anzeige von Dateidialogen abhängig von den gewünschten Parametern zu produzieren.

Wo wollen wir diese Funktionalität bereitstellen? Sinnvoll ist ein COM-Add-In für den VBA-Editor, denn dort werden wir den zu erstellenden Code auch nutzen.

Wo dort wollen wir die Funktion aufrufen? Hier gibt es drei mögliche Stellen:

- einen Menüeintrag, beispielsweise unterhalb des Eintrags **Add-Ins** oder in einem eigenen Hauptmenü

Ereignisse in Klassen programmieren

Eines der wichtigsten Features vieler eingebauter Klassen in den Office-Anwendungen sind die Ereignisse. Damit können wir Ereignisprozeduren implementieren, mit denen wir beispielsweise auf das Anklicken von Schaltflächen, dem Öffnen oder Schließen von Dokumenten oder dem Wechseln eines Tabellenblatts in einem Excel-Arbeitsblatt reagieren können. Wenn man selbst Klassen programmiert, findet sich früher oder später ein Anlass, dieser ein eigenes Ereignis hinzuzufügen, dass durch eine bestimmte Aktion ausgelöst wird. Dieser Artikel zeigt, wie wir solche Ereignisse programmieren und auslösen und wie wir diese in Form von Ereignisprozeduren in den Klassen implementieren, welche das entsprechende Objekt instanziiert haben.

Ereignisse in Klassenmodulen

Wer schon einmal eine Ereignisprozedur beispielsweise für das Anklicken einer Schaltfläche, das Öffnen eines Word-Dokuments, das Markieren eines Ranges in Excel et cetera programmiert hat, kennt zumindest schon einmal die Verbraucherseite von Ereignissen.

Hier können wir zum Beispiel in Excel in der Klasse **Tabelle1** eines Excel-Dokuments im Codefenster im linken Kombinationsfeld den Eintrag **Worksheet** auswählen, was im rechten Kombinationsfeld automatisch die Standardmethode **SelectionChange** selektiert. Dadurch wird automatisch die Ereignisprozedur **Worksheet_SelectionChange** angelegt, mit der wir in diesem Fall die Adresse des neu ausgewählten Bereichs im Direktbereich des VBA-Editors ausgeben (siehe Bild 1).

Nun schauen wir uns die andere Seite an, auf der wir Ereignisse definieren, die wir dann wiederum als Verbraucher nutzen können. Wozu aber überhaupt Ereignisse? Weil wir benutzerdefiniert auf bestimmte Ereignisse reagieren können wollen. Stellen wir uns beispielsweise vor, wir wollen den Benutzer unserer

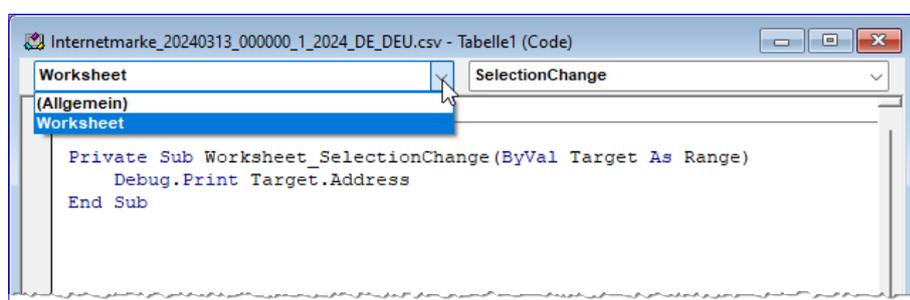


Bild 1: Ereignisprozedur beim Ändern der Markierung in einem Excel-Arbeitsblatt

Klasse **clsAdresse** darüber informieren, wenn die Methode **Leeren** erfolgreich aufgerufen wurde. Dann können wir das einfach erledigen, indem wir der Methode **Leeren** eine entsprechende Meldung hinzufügen:

```
Public Sub Leeren()
    ...
    m_Land = ""
    MsgBox "Die Eigenschaften der Adresse wurden geleert.", _
        vbOKOnly + vbExclamation, "Adresse geleert"
End Sub
```

Rufen wir die Methode nun von einer anderen Prozedur aus auf, erscheint diese Meldung (siehe Bild 2).

Nun stellen wir uns vor, wir wollen die Meldung dieser Klasse nicht innerhalb der Klasse verdrahten, sondern dem Entwickler, der diese Klasse in seinem Code ver-

Klassen programmieren unter VBA

Neben den Standardmodulen gibt es in VBA-Projekten auch noch einen weiteren Typ von Modulen, nämlich die Klassenmodule. Diese kommen wiederum in zwei Ausführungen: Es gibt allein-stehende Klassenmodule und eingebaute Klassenmodule, die den Code zu bestimmten Objekten enthalten – wie beispielsweise die Klassenmodule zu Formularen und Berichten in Access, zu Word-Dokumenten für das Dokument oder in Excel für Worksheet und Workbook. Auch in Outlook gibt es Klassenmodule. Wir wollen uns an dieser Stelle jedoch auf die allein stehenden Klassenmodule konzentrieren, also auf solche, die wir selbst anlegen müssen. Hier schauen wir uns an, warum man diese überhaupt nutzen sollte, welche Anwendungszwecke es gibt und welche Best Practices sich für uns etabliert haben.

Unterschied zwischen Standardmodulen, eingebauten Klassenmodulen und benutzerdefinierten Klassenmodulen

Zunächst einmal wollen wir den Unterschied zwischen den verschiedenen Typen von Modulen erläutern.

Standardmodule

Wir beginnen mit den Standardmodulen:

- Standardmodule sind einfache Container für VBA-Code, die in einer VBA-Projektdatei enthalten sind und die selbst hinzufügen müssen.
- Sie enthalten normalerweise Funktionen, Prozeduren und Variablen, die in verschiedenen Teilen des Projekts verwendet werden können.
- Ein Standardmodul kann Funktionen und Prozeduren enthalten, die unabhängig voneinander sind und direkt vom Rest des Codes aufgerufen werden können.
- Standardmodule werden oft verwendet, um allgemeine Funktionen und Hilfs-

routinen zu speichern, die von verschiedenen Teilen des Projekts benötigt werden.

Klassenmodule allgemein

Wir schauen uns erst einmal die allgemeinen Eigenschaften an, die allen Klassenmodulen gemein sind:

- Klassenmodule ermöglichen die Definition benutzerdefinierter Datenstrukturen, die als Objekte bezeichnet werden.
- Sie dienen dazu, spezifische Arten von Objekten zu definieren, die Eigenschaften, Methoden und Ereignisse haben können.

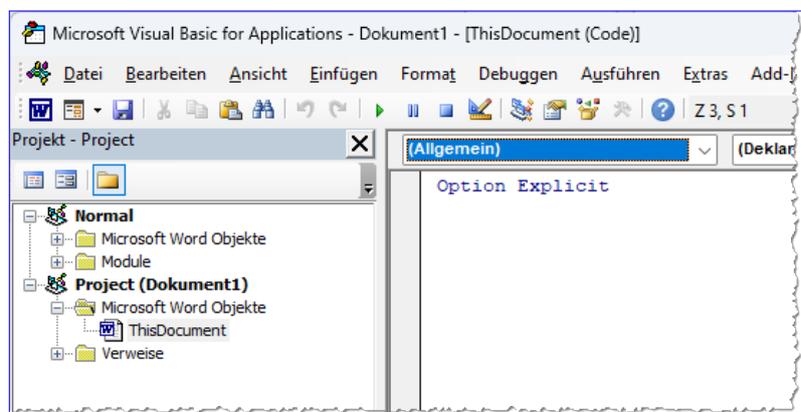


Bild 1: Ein eingebautes Klassenmodul, hier in Word.

twinBASIC: COM-Add-In für den VBA-Editor

Wer den VBA-Editor mit eigenen Tools erweitern möchte, kommt um die Programmierung von COM-Add-Ins kaum herum – zumindest nicht, wenn er eine schicke Benutzeroberfläche und die Integration in Menü, Symbolleisten und Kontextmenüs wünscht. In diesem Artikel zeigen wir daher, wie wir die Basis für ein solches COM-Add-In mit twinBASIC programmieren. Ausgehend davon kannst Du direkt loslegen und Deine gewünschten Funktionen einbauen – es sind nur jeweils wenige Anpassung notwendig. Wir erklären Schritt für Schritt, wie die Basis des COM-Add-Ins aufgebaut ist und welche Anpassungen Du vornehmen musst, um ein COM-Add-In für Deine eigenen Anwendungen zu bauen.

Start mit Vorlage

Für den Start kannst Du das Beispielprojekt **VBECOMAddInBasis.twinproj** aus dem Download zu diesem Artikel verwenden. Diese Datei öffnest Du in der Entwicklungsumgebung twinBASIC, deren jeweils aktuelle Fassung Du unter dem folgenden Link findest:

<https://github.com/twinbasic/twinbasic/releases>

Dort klickst Du auf Assets und kannst dann wie in Bild 1 die **.zip**-Datei mit der Entwicklungsumgebung herunterladen. Eine Installation ist nicht notwendig, es reicht das Entpacken im gewünschten Verzeichnis.

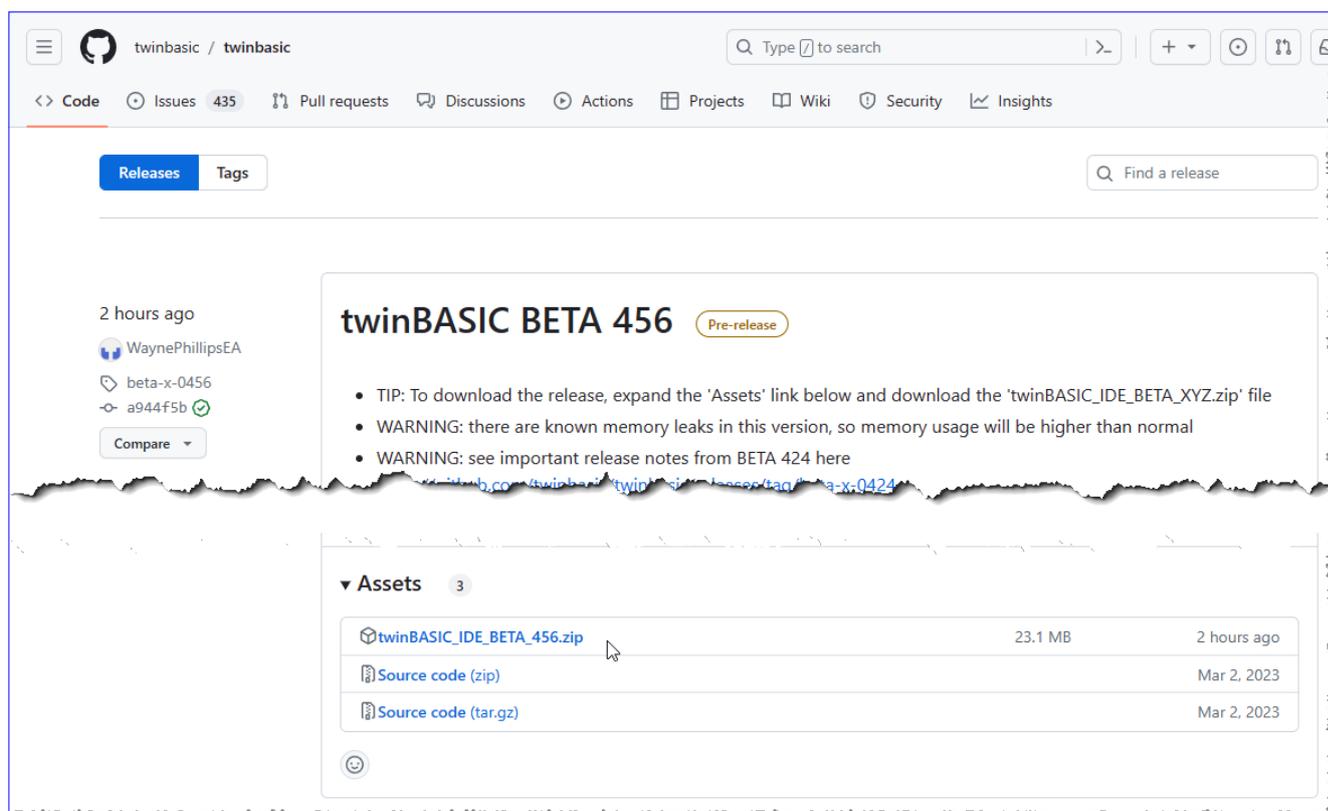


Bild 1: Download der neuesten twinBASIC-Version

VBA-Editor: Quellcode-Bearbeitung automatisieren

Der VBA-Editor ist die Gemeinsamkeit der Office-Anwendungen wie Access, Excel, Outlook, PowerPoint und Word. Wer eine oder mehrere dieser Anwendungen programmiert, um dem Benutzer die Arbeit damit zu erleichtern, kennt sich mehr oder weniger mit VBA aus. In der Regel wird im Arbeitsalltag eines Entwicklers jede Codezeile von Hand neu programmiert. Gegebenenfalls kopiert man bestehenden Code und passt diesen an den jeweiligen Anwendungszweck an. Aber warum nicht einen Schritt weitergehen und VBA-Code zur Erstellung von VBA-Code selbst nutzen? Visual Basic bietet eigens zum Zweck der Programmierung der Elemente und des Codes im VBA-Editor eine eigene Bibliothek namens Microsoft Visual Basic for Applications Extensibility 5.3. Welche Möglichkeiten diese allgemein bietet und welche Elemente, Eigenschaften und Methoden sie bereitstellt, schauen wir uns in diesem Artikel an.

Der VBA-Editor ist die Entwicklungsumgebung für die Programmierung der verschiedenen Office-Anwendungen und der darin enthaltenen Dokumente und Elemente. Er zeigt im Projekt-Explorer die programmierbaren Elemente wie Standardmodule und Klassenmodule an und erlaubt es, den Code dieser Elemente im Code-Editor anzuzeigen und zu bearbeiten.

Die Codeeingabe selbst ist bis auf wenige Hilfestellungen durch IntelliSense Aufgabe des Entwicklers – hier ist viel Tipparbeit angezeigt oder gegebenenfalls noch das Kopieren und Anpassen bereits vorhandenen Codes.

IntelliSense unterstützt uns allerdings auch nur dabei, den Datentyp für Variablen auszuwählen oder die Eigenschaften oder Methoden von Objektvariablen zu selektieren. Wenn wir mehr aus der Entwicklungsumgebung herausholen wollen, müssen wir uns also selbst helfen.

Dabei können wir bereits mit den üblichen Mitteln von VBA Einiges erreichen: So können wir die Zeichenkettenfunktionen in Kombination mit Schleifen

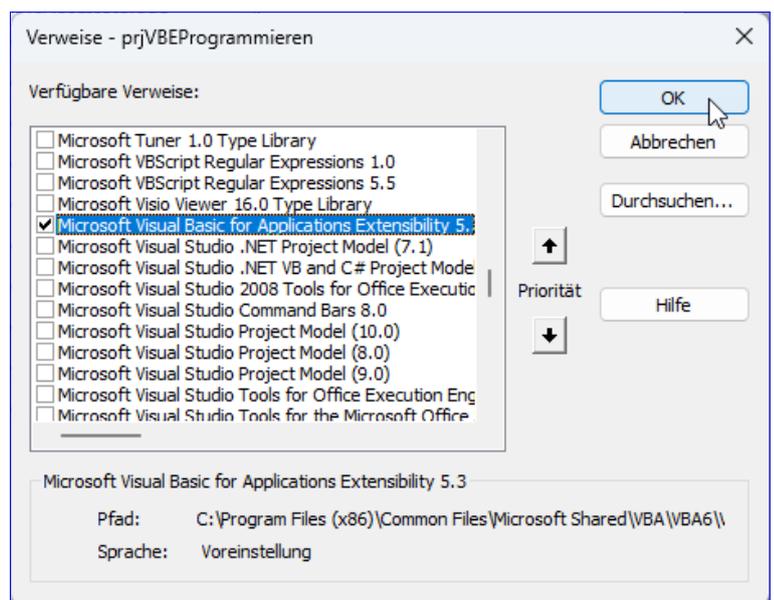


Bild 1: Setzen eines Verweises auf die Bibliothek Microsoft Visual Basic for Applications Extensibility 5.3

nutzen, um bestimmte Abfolgen von Anweisungen im Direktbereich des VBA-Editors auszugeben, die wir dann kopieren und in unseren Code einfügen.

Wir können jedoch noch viel mehr erreichen, wenn wir eine Bibliothek namens **Microsoft Visual Basic for Applications Extensibility 5.3** einbinden. Diese fügen wir über den **Verweise**-Dialog, den wir mit dem

Wozu mit Klassen programmieren?

Wer die Office-Anwendungen Access, Excel, Outlook, PowerPoint oder Word mit VBA programmiert, kann eine Menge erreichen, ohne jemals eine eigene Klasse zu programmieren. Wer sich die gesamten Möglichkeiten eröffnen will, kommt jedoch irgendwann nicht mehr um die Programmierung benutzerdefinierter Klassen herum. Dieser Artikel zeigt verschiedene Szenarien auf, wann man benutzerdefinierte Klassen programmieren und darauf basierende Objekte instanziiieren kann und soll. Dabei geht es um Begriffe wie Eigenschaften, Methoden, Ereignisse und darum, wo und wie man Klassen einsetzt.

Wer in den VBA-Projekten von Outlook oder von Dokumenten auf Basis von Access, Excel, PowerPoint oder Word programmiert, ahnt es vielleicht manchmal nicht, aber tatsächlich programmiert man fast zwangsläufig innerhalb von Klassenmodulen. Das Modul **Projekt1**, das angezeigt wird, wenn man das VBA-Projekt von Outlook öffnet, das Modul **ThisDocument** in Word, die Module **DieseArbeitsmappe** oder **Tabelle1** in Excel oder auch die Module, die man in Access beim Anlegen von Ereignisprozeduren für Formulare, Steuerelemente oder Berichte verwendet, sind allesamt Klassenmodule.

Diese sind jedoch mit der jeweiligen Anwendung (Outlook) beziehungsweise mit dem jeweiligen Dokument oder dem Element verknüpft, in deren Kontext sie geöffnet werden. Dadurch ergeben sich einige Vorteile: Wir können nämlich in diesen direkt auf einige Elemente des jeweiligen Objekts zugreifen. Im Klassenmodul **ThisOutlookSession** von Outlook greifen wir beispielsweise direkt auf die Eigenschaften, Methoden und Ereignisse des **Application**-Objekts zu.

Im Klassenmodul eines Access-Formulars greifen wir direkt auf die Eigenschaften, Methoden und Ereignisse des jeweiligen Formulars und der enthaltenen Steuerelemente zu. Im Klassenmodul **ThisDocument** eines Word-Dokuments stehen die entsprechenden Elemente des **Document**-Objekts zur Verfügung und in Excel können wir in der Klasse **DieseArbeitsmappe** über die

Variable **Workbook** auf die enthaltenen Eigenschaften, Methoden und Ereignisse zugreifen sowie in den Klassen für die einzelnen Tabellen auf das jeweilige **Worksheet**-Element.

Wenn wir bereits eine solche Vielfalt von eingebauten Elementen haben, warum sollten wir noch benutzerdefinierte Klassen entwickeln? Dazu gibt es verschiedene Gründe, die wir in den folgenden Abschnitten beschreiben. Außerdem sehen wir uns an, warum es in diesen Fällen ein Klassenmodul sein muss und warum es nicht reicht, ein Standardmodul zu verwenden.

Grund für Klassen: Eigenschaften, Methoden und Ereignisse zusammenfassen

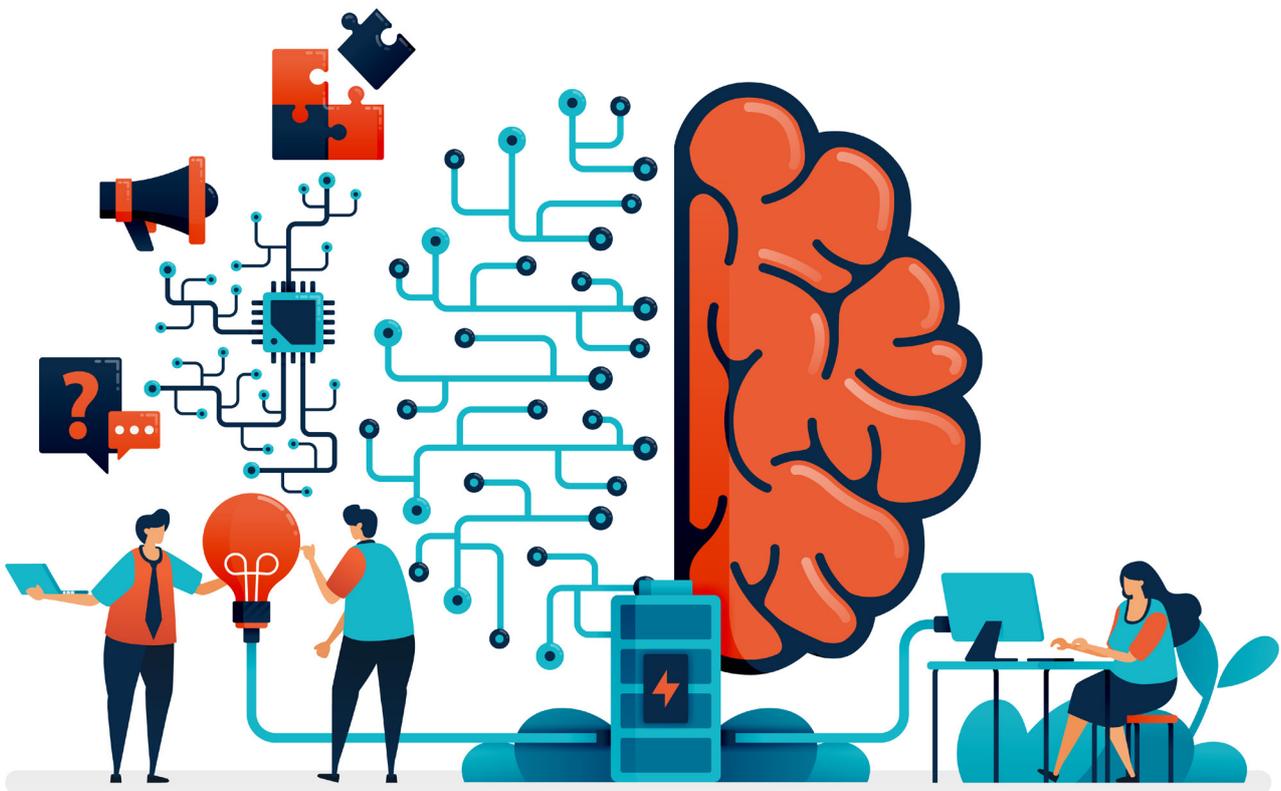
Ein wichtiger Anwendungszweck für Klassen ist schlicht und einfach das Zusammenfassen von Elementen in einer Code-Einheit. Wann immer wir im Code allein Informationen zusammenstellen, die zusammengehören, können wir dies mit einer Klasse erledigen.

Für diese legen wir dann Eigenschaften fest, mit denen wir die Informationen erfassen können. Sobald wir ein Objekt auf Basis der Klasse erstellt haben, können wir diesem Informationen über die Eigenschaften zuweisen und die Eigenschaften wieder auslesen. Wir können auch dafür sorgen, dass das Objekt seine Eigen-

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

AUFGABEN VERWALTEN MIT MICROSOFT TO DO

Verwalte Deine Aufgaben mit der kostenlosen Aufgabenverwaltung von Microsoft.

SEITE 4

TO DO PROGRAMMIEREN MIT POWER AUTOMATE

Lerne Power Automate kennen am Beispiel der Programmierung von Microsoft To Do über VBA.

SEITE 33

MENÜS ANPASSEN IM VBA-EDITOR

Entdecke coole Features in den Menüs des VBA-Editors und passe sie nach Deinen Wünschen an.

SEITE 13



André Minhorst Verlag

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

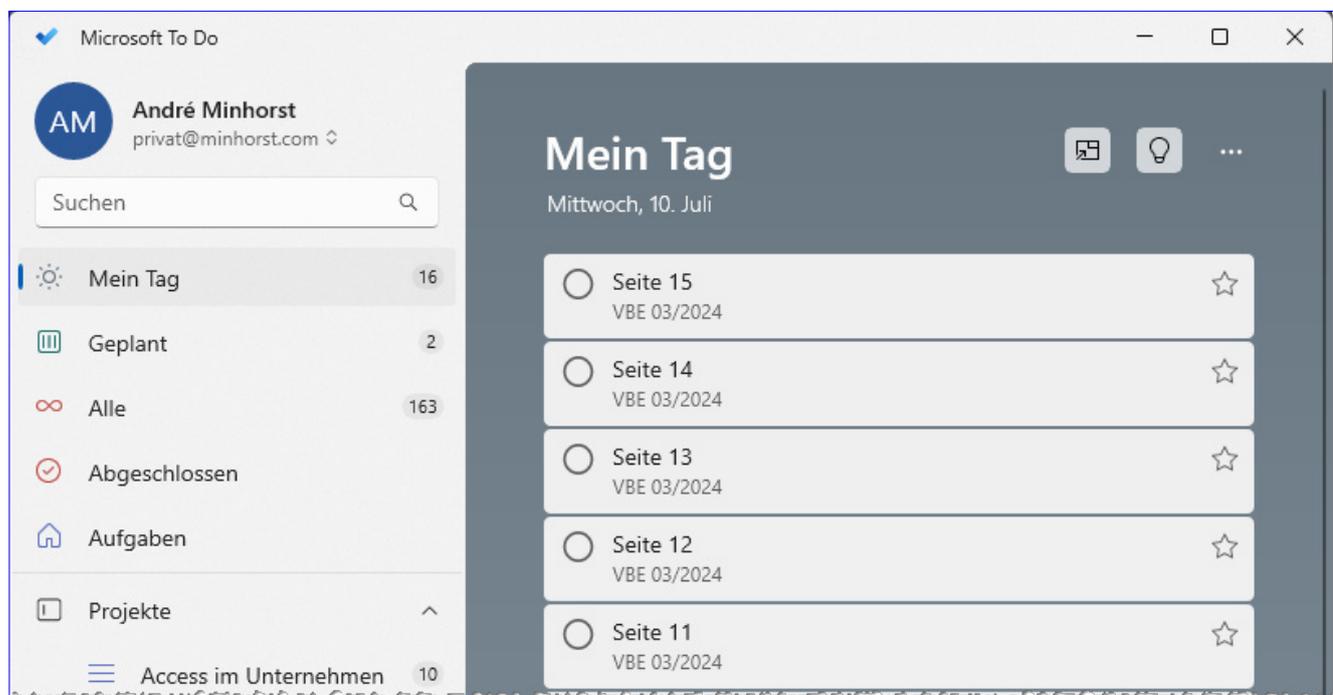


Bild 1: Der Bereich **Mein Tag** in Microsoft To Do

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.

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).

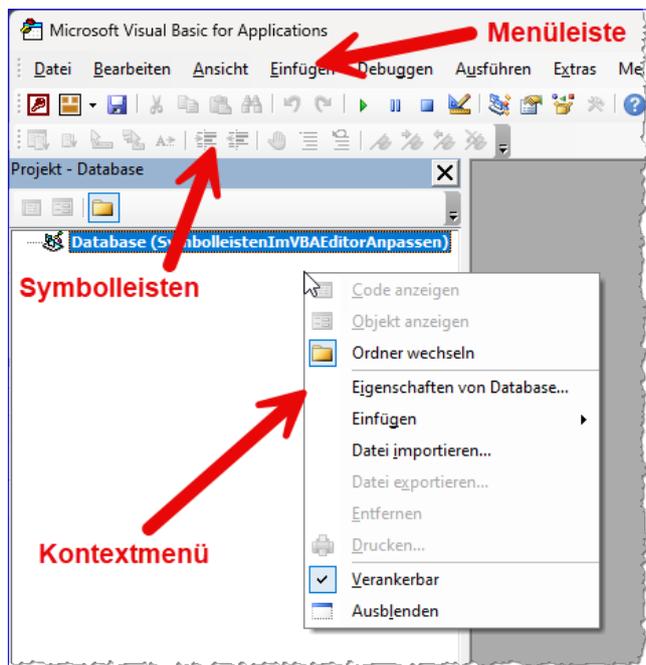


Bild 1: Alle drei Arten von Menüs

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.

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

technischen 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.

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 Klammerspaars 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
```

VBA-Editor: Klasseneigenschaften per Mausclick

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 Mausclicks 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:

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**.

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

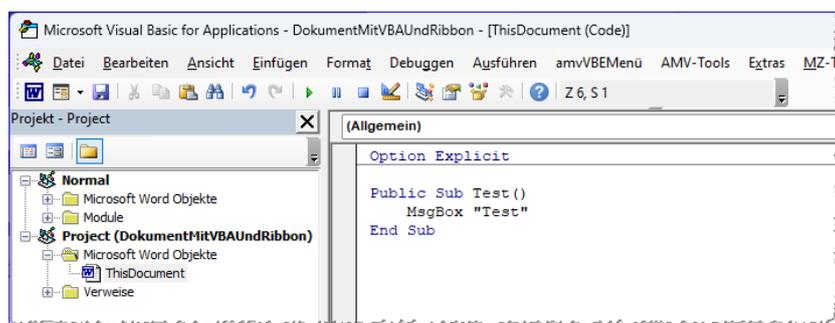


Bild 1: Anlegen einer VBA-Prozedur für das aktuelle Dokument

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

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

DHL VERSANDETICKETEN PER VBA ERSTELLEN

Nutze die neue Rest-API für DHL Geschäftskunden zum Erstellen von Versandetiketten.

SEITE 50

POWERPOINT-FOLIEN ÜBERSETZEN

Übersetze Slides, ohne Animationen et cetera zu löschen oder das Dokument anderweitig zu manipulieren.

SEITE 22

VBA-EDITOR-MENÜS PER VBA-CODE ERWEITERN

Steuere und erweitere die Menüs des VBA-Editors nach Deinen eigenen Wünschen.

SEITE 4



André Minhorst Verlag

DHL-Versandetiketten erstellen per VBA

Wer Kunden und Bestellungen mit einer Access-Datenbank verwaltet oder gegebenenfalls auch mit einer Excel-Tabelle, möchte vielleicht Zeit sparen und die Etiketten für den Versand von Lieferungen an seine Kunden automatisieren. Das gelingt mit den Webservices von DHL. Wir haben bereits einmal eine solche Lösung vorgestellt, aber DHL hat seine Schnittstellen für die Erstellung von Versandetiketten aktualisiert. In diesem Artikel schauen wir uns an, wie die neuen Schnittstellen funktionieren: Welche Daten benötige ich? Welche URL muss für den Zugriff verwendet werden? In welcher Form übergebe ich beispielsweise die Adresdaten an den Webservice?

Voraussetzung für die Verwendung der Lösung

Wenn wir die Webservices von DHL für die Erstellung von Versandetiketten nutzen wollen, sind einige Voraussetzungen zu erfüllen. Als Erstes benötigen wir ein Konto als Geschäftskunde bei DHL. Wie Du dieses anlegst, wollen wir hier nicht im Detail beschreiben. Wir wollen Dir aber zumindest den Link zum Geschäftskundenportal mit auf den Weg geben:

<https://www.dhl.de/de/geschaeftskunden.html>

Nach der Anmeldung benötigen wir später Deine Zugangsdaten für Dein Kundenkonto, also Benutzername und Kennwort.

Als Zweites benötigst Du ein Entwicklerkonto bei DHL. Dazu besuchst Du die folgende Adresse und erstellst einen Account:

<https://developer.dhl.com/>

Damit können wir bereits starten.

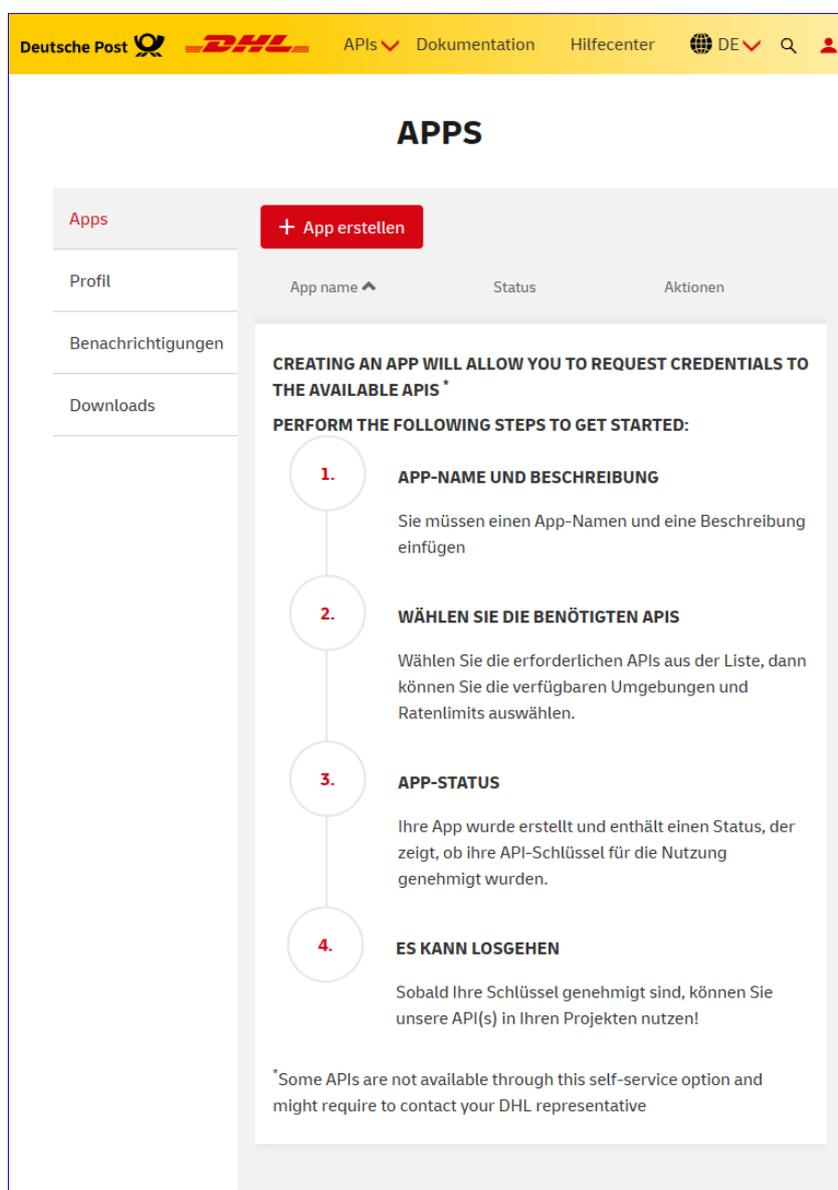


Bild 1: App anlegen für weitere Daten

Klassenprogrammierung mit COM-Add-In vereinfachen

Im Artikel »VBA-Editor: Klasseneigenschaften per Mausklick« (www.vbentwickler.de/422) haben wir eine Prozedur vorgestellt, mit denen man aus einer einfachen Variablendeklaration innerhalb eines Klassenmoduls eine private Membervariable und jeweils eine Property Get-Methode und eine Property Set/Let-Methode erzeugen kann. Der einzige Haken bei dieser Lösung ist, dass man diese bisher noch über den Direktbereich aufrufen musste. Da das nicht sonderlich komfortabel ist, stellen wir in diesem Artikel eine Lösung vor, bei der wir mit twinBASIC ein COM-Add-In für den VBA-Editor erstellen, mit dem wir die Funktion aus dem oben genannten Artikel über die Menüleiste, die Symbolleiste oder auch über das Kontextmenü des VBA-Editors aufrufen kann.

Wir bauen hier auf den Vorbereitungen auf, die wir im Artikel **twinBASIC: COM-Add-In für den VBA-Editor** (www.vbentwickler.de/421) vorgestellt haben. Hier haben wir ein Basis-COM-Add-In für den VBA-Editor vorgestellt, das einfach nur ein paar Dummy-Einträge im Menü und im Kontextmenü anzeigt,

die Meldungsfenster liefern. Den ersten sehen wir in Bild 1.

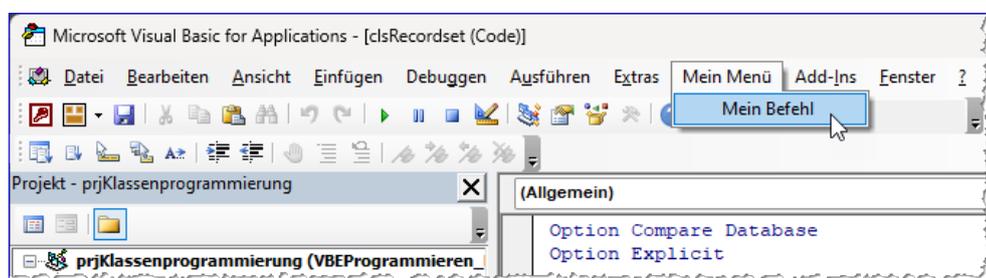


Bild 1: Menüeintrag unseres COM-Add-Ins

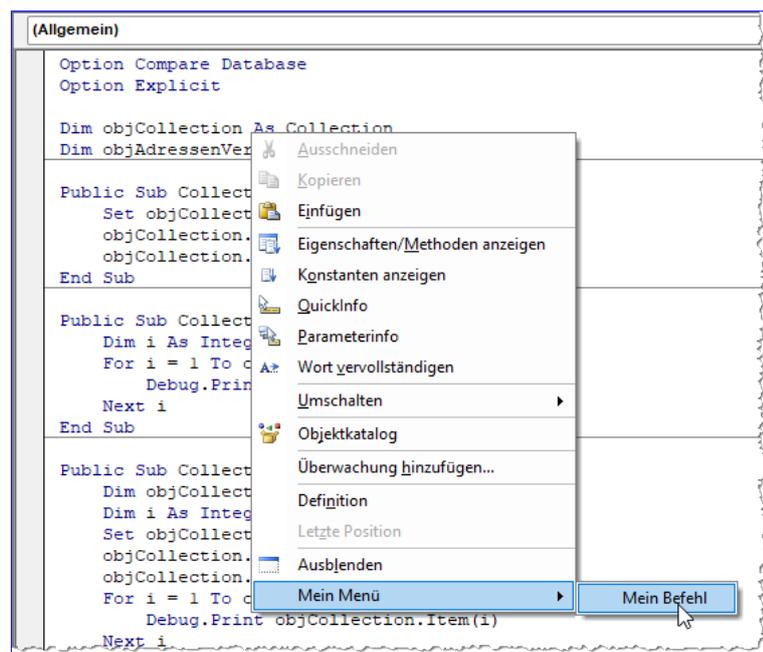


Bild 2: Menüeintrag im Kontextmenü

Den zweiten Eintrag, den wir im Kontextmenü untergebracht haben, finden wir im Kontextmenü in Bild 2 ganz unten.

In der Symbolleiste haben wir noch keinen Eintrag untergebracht. Auch das wollen wir gegebenenfalls noch erledigen. Erst einmal schauen wir uns jedoch die zu programmierende Funktion an – wo im Menü wir diese optimal unterbringen.

Umwandeln von Variablen in Property Get/Let/Set

Die Aufgabe lautet, die Prozedur, die wir im oben genannten Artikel entwickelt haben, über ein COM-Add-In für den VBA-Editor

Menüs per VBA programmieren

Das Menüsystem des VBA-Editors lässt sich über die Benutzeroberfläche bereits einfach anpassen. Das haben wir im Artikel »Menüsystem im VBA-Editor anpassen« (www.vbentwickler.de/434) gezeigt. Außerdem haben wir uns im Artikel »Kontextmenüs per VBA programmieren« (www.vbentwickler.de/368) bereits angeschaut, wie wir per VBA Kontextmenüs erstellen und anpassen können. Es fehlen also noch die Informationen, wie wir die eigentlichen Menüleisten und Symbolleisten mit VBA programmieren können. Wie das gelingt, schauen wir uns im vorliegenden Artikel an. Wir zeigen, wie vorhandene Menüs angepasst und wie neue Menüs erstellt werden können.

Warum haben wir eigentlich einen eigenen Artikel zum Thema Kontextmenüs per VBA programmieren veröffentlicht und nicht direkt etwas weiter ausgeholt und die Menü- und Symbolleisten ebenfalls beschrieben? Weil es damals darum ging, die Benutzeroberfläche der Office-Anwendungen zu erweitern.

Und da diese bekanntlich seit Office 2007 keine Menü- und Symbolleisten mehr verwenden, sondern das Ribbon, haben wir uns auf die Kontextmenüs beschränkt.

Das wirft nun die Frage auf, warum wir uns dann jetzt mit den Menü- und Symbolleisten beschäftigen wollen. Nun: Es gibt noch das gallische Dorf unter den Microsoft-Anwendungen, in denen noch Menü- und Symbolleisten verwendet werden.

Dabei handelt es sich um die Anwendung, mit der wir täglich arbeiten: den VBA-Editor. Dieser verwendet noch das gute, alte Menü- und Symbolleistensystem, das jüngere VBA-Entwickler möglicherweise von Office selbst gar nicht mehr kennen.

Und da wir beispielsweise mit twinBASIC den VBA-Editor erweitern wollen und das Menüsystem nun einmal der Hauptort zum Unterbringen von Steuerelementen zum Aufrufen der Funktionen der ent-

sprechenden COM-Add-Ins ist, schauen wir uns nun doch noch einmal die Grundlagen zu den Menü- und Symbolleisten an.

Office-Verweis sinnvoll

Dazu ist es sinnvoll, dem VBA-Projekt erst einmal einen Verweis auf die Bibliothek **Microsoft Office 16.0 Object Library** hinzuzufügen.

Das erledigen wir über den Verweise-Dialog, den wir über den Menüeintrag **Extras|Verweise** des VBA-Editors öffnen (siehe Bild 1).

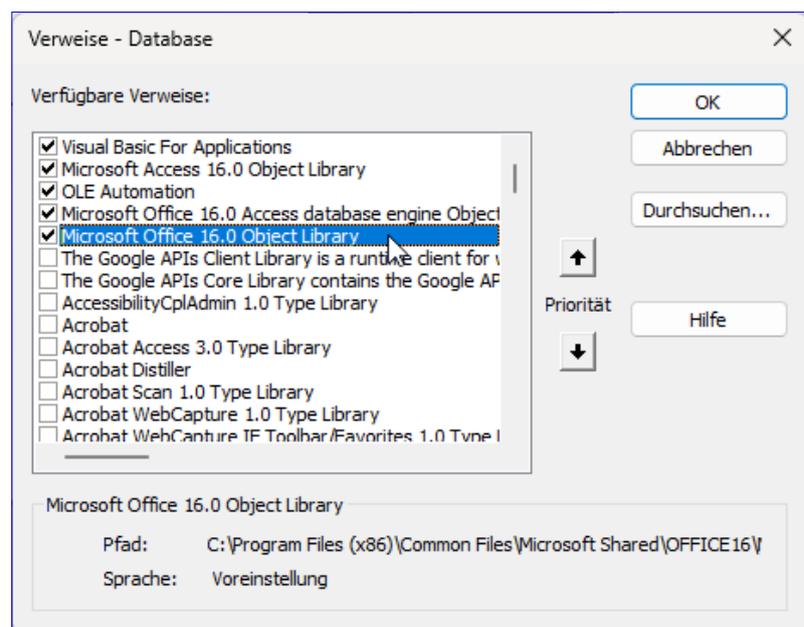


Bild 1: Hinzufügen eines Verweises auf die Office-Bibliothek

Menü-Steuererelemente per VBA programmieren

Im Artikel »Menüs per VBA programmieren« (www.vbentwickler.de/435) haben wir uns bereits angesehen, wie wir die Menüstruktur selbst im VBA-Editor per VBA programmieren können. Damit wissen wir nun, wie wir Hauptmenüleisten, Symbolleisten und Kontextmenüs erstellen und anzeigen können. Es fehlt allerdings noch das Salz in der Suppe, nämlich die Steuererelemente auf diesen Menüs. Welche es gibt und wie man diese hinzufügt und mit Aktionen versieht, schauen wir uns in diesem Artikel an.

Wer den VBA-Editor mit eigenen Funktionen erweitern will und das beispielsweise mit COM-Add-Ins auf Basis von twinBASIC realisiert, möchte die hinzugefügten Funktionen auch auf irgendeine Weise für den Benutzer zugänglich machen. Dazu bietet sich das Menüsystem an. Damit können wir in der Menüleiste, den Symbolleisten und in Kontextmenüs Befehle zum Aufrufen der benutzerdefinierten Funktionen hinzufügen. Allerdings brauchen wir hier nicht nur die entsprechenden Elemente, sondern wir müssen auch noch Steuererelemente hinzufügen. Dies werden meist einfache Schaltflächen sein. Wir schauen uns allerdings in diesem Artikel alle der wenigen verfügbaren Steuererelemente für die Menüs des VBA-Editors an.

Steuererelemente des CommandBar-Elements

Dem **CommandBar**-Element können wir die folgenden Steuererelemente hinzufügen:

- Schaltfläche (**msoControlButton**)
- Auswahlfeld (**msoControlComboBox**)
- Untermenü (**msoControlPopup**)

Um ein solches Steuererelement hinzuzufügen, verwenden wir die **Add**-Methode der **Controls**-Auflistung des **CommandBar**-Objekts. Das folgende, einfache Beispiel fügt einfach jeweils ein Element des jeweiligen Typs zu einer neu erstellten Symbolleiste hinzu:

```
Public Sub AddControls()  
    Dim cbr As CommandBar  
    Dim cbb As CommandBarButton  
    Dim cbc As CommandBarComboBox  
    Dim cbp As CommandBarPopup  
    On Error Resume Next  
    VBE.CommandBars("Symbolleiste mit Controls").Delete  
    On Error GoTo 0  
    Set cbr = VBE.CommandBars.Add( _  
        "Symbolleiste mit Controls")  
    With cbr  
        Set cbb = cbr.Controls.Add(msoControlButton)  
        With cbb  
            .Caption = "Button"  
            .Style = msoButtonCaption  
        End With  
        Set cbc = cbr.Controls.Add(msoControlComboBox)  
        With cbc  
            .Caption = "ComboBox"  
        End With  
        Set cbp = cbr.Controls.Add(msoControlPopup)  
        With cbp  
            .Caption = "Popup"  
        End With  
        .Visible = True  
    End With  
End Sub
```

Die Prozedur löscht zunächst eine gegebenenfalls bereits von einem vorherigen Test vorhandene Symbolleiste namens **Symbolleiste mit Controls** aus der

PowerPoint: Texte automatisiert übersetzen

Neulich war es mal wieder so weit: Eine PowerPoint-Präsentation musste her. Und das auch noch auf Englisch. Okay, das Schul-Englisch ist zum Verstehen und schriftliche Kommunikation ausreichend, aber eine PowerPoint-Präsentation für englischsprachiges Fachpublikum sollte schon annähernd perfekt sein. Wozu gibt es Übersetzungsdienste? Also habe ich meine Texte auf Deutsch zurechtgelegt und diese von der KI übersetzen lassen. Dann habe ich alles in die PowerPoint-Präsentation eingefügt und noch die Animationen hinzugefügt, damit beispielsweise Stichpunkte Schritt für Schritt eingeblendet werden können. All das hat so gut geklappt, dass ich die Präsentation anschließend auch noch für ein Video aufbereiten wollte – diesmal jedoch auf Deutsch. Also habe ich erstmal eine komplette Seite kopiert, übersetzen lassen und wieder zurückgeschrieben. Das habe ich für einige Folien gemacht und dann schnell festgestellt, dass so alle Animationen verloren gehen. Der nächste Ansatz dann: Absatz für Absatz in die Zwischenablage, übersetzen lassen, wieder zurückschreiben. So blieben die Animationen erhalten, aber es war zu viel Handarbeit. Wozu beherrsche ich – im Gegensatz zu Englisch – eigentlich perfekt VBA? Also habe ich mich an die Programmierung der Übersetzung der enthaltenen Texte begeben. Das Ergebnis siehst Du in diesem Artikel!

PowerPoint ist in diesem Magazin bisher noch gar nicht vorgekommen – höchstens als Randnotiz in Zusammenhang mit der Office-Programmierung. In diesem Artikel wollen wir jedoch direkt einmal eine praktische Lösung liefern.

Ausgangspunkt ist ein Dokument mit Folien wie der aus Bild 1. Wir haben hier verschiedene Absätze mit englischem Text, die wir gern übersetzen würden. Dabei gibt es verschiedene Vorgehensweisen:

- Die erste ist, einfach manuell in das Dokument zu gehen und die

Texte Wort für Wort zu übersetzen. Das ist recht viel Arbeit und kostet entsprechend Zeit.

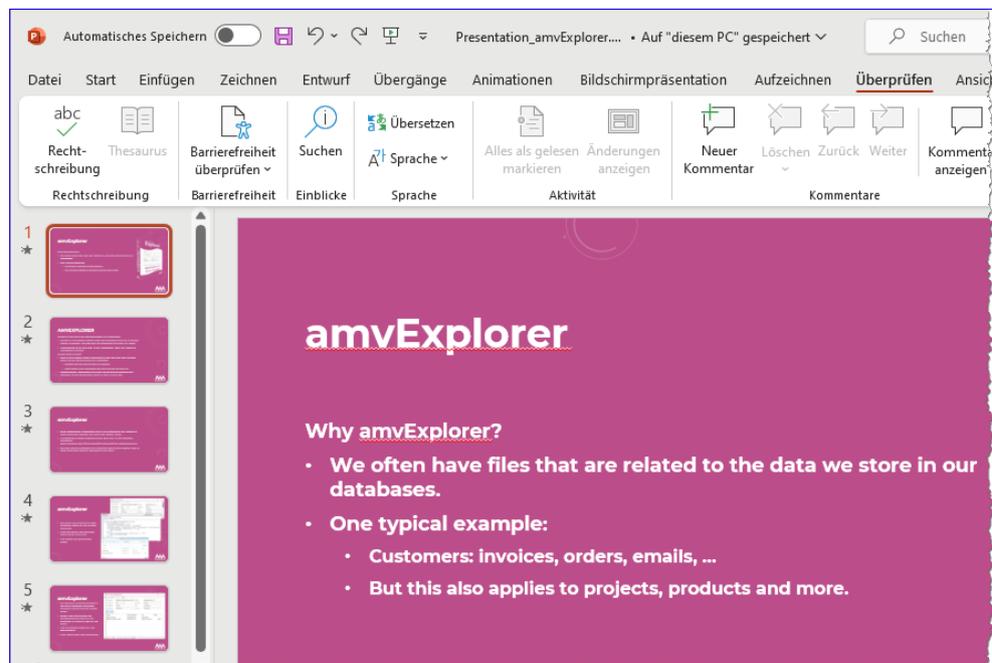


Bild 1: Ein zu übersetzendes PowerPoint-Dokument

PowerPoint-Übersetzung per COM-Add-In

Im Artikel »PowerPoint: Texte automatisiert übersetzen« (www.vbentwickler.de/437) haben wir VBA-Code produziert, mit dem wir alle Absätze aller Folien in einem PowerPoint-Dokument automatisch übersetzen können. Dabei nutzen wir den Dienst DeepL. Leider müssen wir, um diesen Code in einem PowerPoint-Dokument verwenden zu können, das Modul erst in das jeweilige Dokument integrieren. Wenn man oft PowerPoint-Folien übersetzen muss, ist das recht aufwändig. Da nehmen wir lieber den Aufwand in Kauf, einmal ein COM-Add-In für diesen Zweck zu programmieren, dass wir dann auch noch an Dich weitergeben können, damit Du es für Dich und Deine Mitarbeiter und/oder Kunden einsetzen kannst.

Das Werkzeug für COM-Add-Ins: twinBASIC

Wie üblich nutzen wir das in der 32-Bit-Version sogar kostenlose Produkt twinBASIC für die Erstellung des COM-Add-Ins. Im Download findest Du, auch wenn Du Dir noch nicht die kostenpflichtige Version von twinBASIC gekauft hast, aber auch eine 64-Bit-Version der DLL, die das COM-Add-In enthält. Da wir keine DLL-Funktionen nutzen, ist der Code für beide Varianten allerdings identisch.

twinBASIC findest Du beispielsweise unter dem folgenden Link:

<https://github.com/twinbasic/twinbasic/releases/tag/beta-x-0585>

Nach dem Start (es ist keine Installation nötig) erscheint der Dialog aus Bild 1. Hier wählen wir unter Samples den Eintrag **Sample 5. MyCOMAddin** aus.

Anschließend erscheint der Dialog aus Bild 2, wo wir den Namen des Projekts festlegen – hier auf **amvPowerPointTranslator**. Danach finden wir bereits die Entwicklungsumgebung vor, in der wir die ersten grundlegenden Änderungen durchführen.

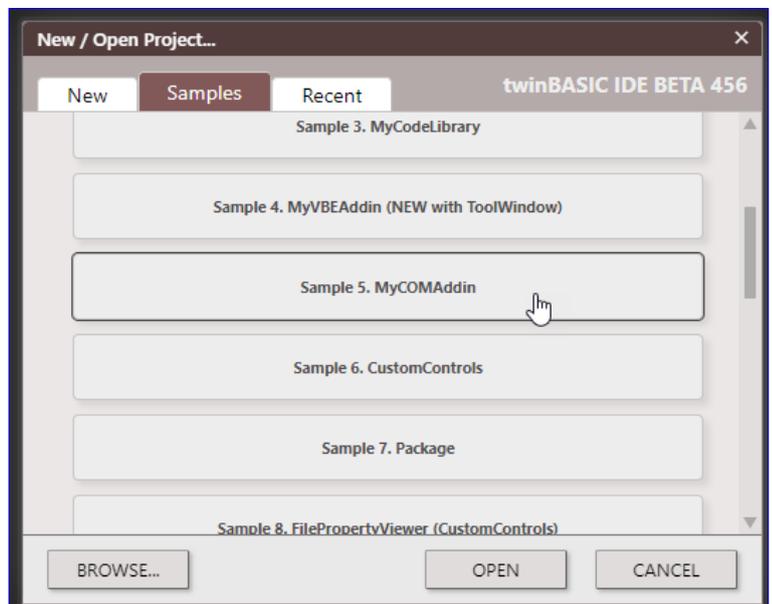


Bild 1: Auswahl des Projekttyps

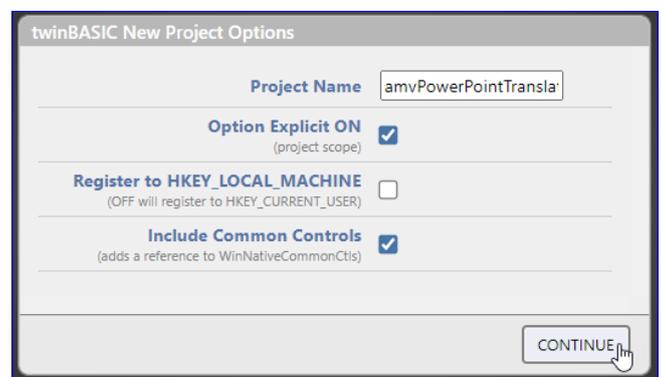


Bild 2: Projektname für das neue Projekt

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

PROJEKTE VERWALTEN MIT TRELLO

Lerne das bekannte Projektmanagement-Tool Trello kennen.

SEITE 4

TRELLO PER VBA STEUERN

Erfahre, wie Du Projekte, Listen und Aufgaben per VBA in Trello anlegst und verwaltest.

SEITE 18

E-MAILS ERNEUT VERSENDEN

Lerne, E-Mails per VBA erneut zu senden statt nur über die Benutzeroberfläche.

SEITE 54



André Minhorst Verlag

Outlook: E-Mails erneut senden

Manchmal kommen E-Mails in Postfächern an, wo man diese nicht optimal weiterverarbeiten kann. Ein Beispiel sind E-Mails von Kunden, die Fragen zur Buchhaltung haben, aber ihre E-Mails an die allgemeine info@-Adresse geschickt haben. Dann können wir diese E-Mail zwar an die Buchhaltungs-Adresse weiterleiten, aber wenn die Buchhaltung dann auf diese E-Mail antworten soll, enthält diese bereits die beim Weiterleiten automatisch eingefügten Elemente – das wirkt auf den Kunden nicht besonders professionell. Oder wir haben eine der vielen »Software As A Service«-Anwendung, die ein eigenes Postfach haben, über das wir Informationen direkt dorthin schicken können. Ein Beispiel ist die Projektverwaltung Trello, der man E-Mails an eine spezielle E-Mail-Adresse zusenden kann, die dann dort automatisch als Aufgaben angelegt werden. In diesem Artikel zeigen wir, wie wir mit wenigen Anweisungen eine Kopie dieser E-Mail an eine weitere Adresse schicken können. Das erledigen wir zunächst durch den Aufruf einer VBA-Prozedur. Anschließend schauen wir uns an, wie wir das auch über die Benutzeroberfläche erledigen können.

Es gibt verschiedene Gründe, warum man eine E-Mail nicht an eine andere Adresse weiterleiten sollte, sondern diese so dorthin schicken möchte, dass man nicht erkennt, dass es sich um eine Weiterleitung handelt. Wenn wir nämlich die Weiterleiten-Schaltfläche für eine E-Mail betätigen, sehen wir nicht nur das Kürzel **WG:** im Betreff, sondern finden auch noch die Kopfdaten der ursprünglichen E-Mail über dem eigentlichen Inhalt der E-Mail (siehe Bild 1).

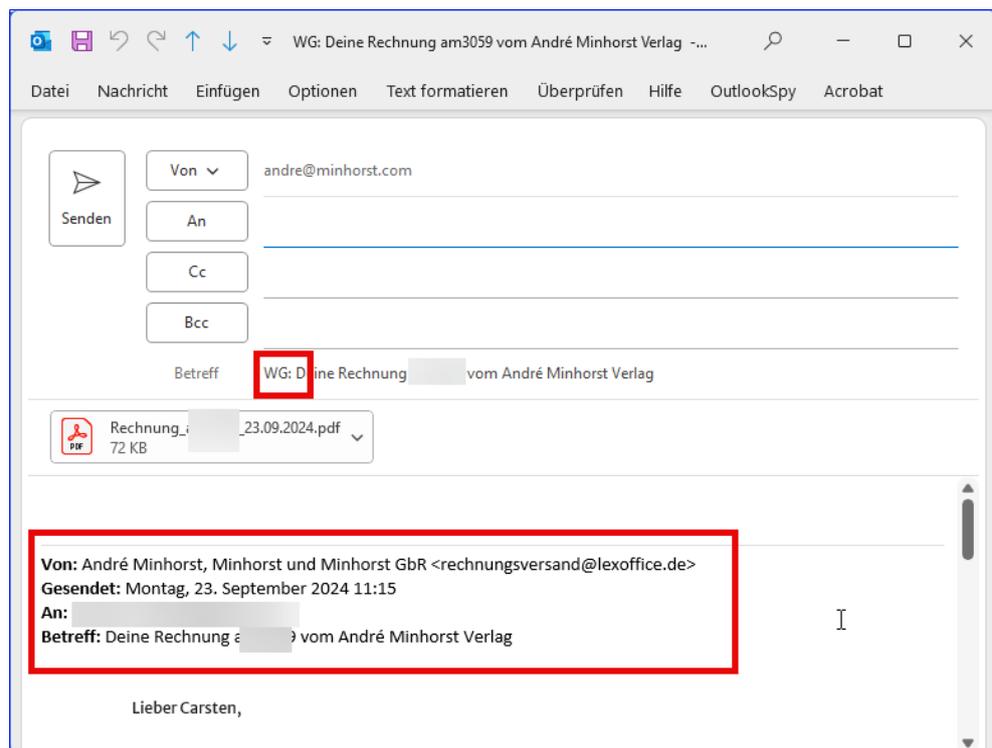


Bild 1: Eine zum Weiterleiten vorbereitete E-Mail

Wenn man selbst nicht der eigentliche Empfänger einer E-Mail ist und diese so an diesen weiterleiten möchte, dass er diese im Originalzustand erhält (abgesehen von der Empfängerad-

resse natürlich), gibt es in Outlook nur einen einzigen weg, der außerdem nicht durch einen entsprechenden VBA-Befehl abbildbar ist.

Outlook: Gesendete Mails per COM-Add-In verschieben

In einem anderen Artikel namens »Outlook: Mails nach dem Senden per VBA verschieben« (www.vbentwickler.de/440) haben wir die grundlegenden Techniken gezeigt, mit denen wir E-Mails, die wir selbst versendet haben, nicht in den Ordner Gesendete Elemente verschieben, sondern in einen Ordner unserer Wahl. Damit können wir einfach E-Mails in bestimmten Kontexten direkt in einen Ordner verschieben, wo auch die übrigen E-Mails zu diesem Thema landen. Eigentlich läuft diese Funktion automatisch und bedarf keiner Benutzer-Interaktion, aber wir benötigen eine Möglichkeit, die Regeln für das Verschieben der E-Mails zu definieren. Dazu haben wir eine Textdatei benutzt, die wir für den Benutzer auf einfachem Wege zugänglich machen wollen, zum Beispiel durch das Anklicken eines Ribbonbuttons. Dazu verwenden wir ein COM-Add-In, dem wir auch noch die eigentliche Funktion hinzufügen – und noch ein paar Extras. In diesem Artikel beschreiben wir, wie Du ein solches COM-Add-In mit twinBASIC ganz einfach selbst bauen kannst.

Das fertige COM-Add-In findest Du in Versionen für 32-Bit und 64-Bit im Download zu diesem Artikel. Wir beschreiben in diesem Artikel die interessantesten Funktionen des COM-Add-Ins.

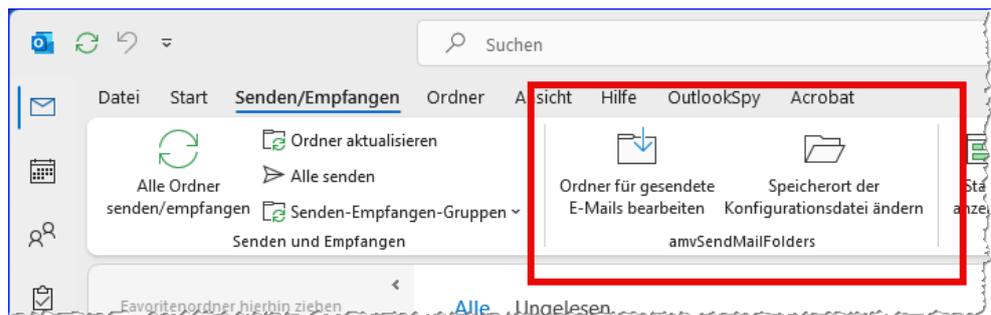


Bild 1: Integration des COM-Add-Ins in die Benutzeroberfläche

Funktion des COM-Add-Ins

Das COM-Add-In soll Outlook um Funktionen erweitern, mit denen wir festlegen können, dass gesendete E-Mails mit bestimmten Empfänger-E-Mail-Adressen in vorgegebene Outlook-Ordner verschoben werden.

Das ist eine Funktion, die automatisiert ablaufen sollte. Allerdings ist dazu die Konfiguration der E-Mails und der gewünschten Zielordner erforderlich. Um diese zu realisieren, fügen wir dem Ribbon von Outlook mit dem COM-Add-In zwei Einträge hin einer eigenen Gruppe hinzu.

Diese soll wie in Bild 1 im Ribbon-Tab **Senden/Empfangen** erscheinen. Hier finden wir zwei neue Einträge:

- **Ordner für gesendete E-Mails bearbeiten:** Klicken wir diesen Eintrag an, erscheint zunächst die Meldung aus Bild 2. Danach wird die Textdatei mit der Konfiguration in der als Text-Editor voreingestellt-

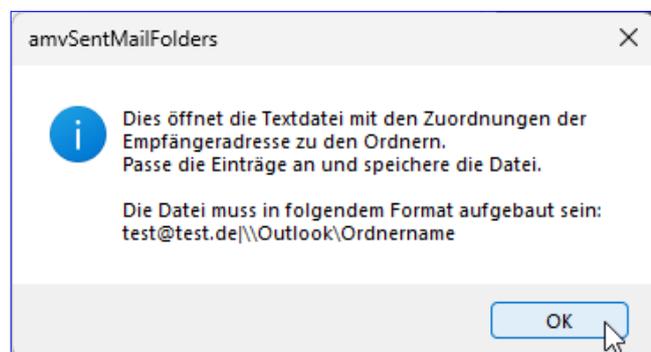


Bild 2: Meldung mit Anweisungen für die Konfigurationsdatei

Outlook: Mails nach dem Senden per VBA verschieben

Für eingehende E-Mails gibt es unter Outlook die Regel-Funktion, mit der man einstellen kann, unter welchen Umständen eine E-Mail, die im Posteingang landet, in einen anderen Ordner verschoben werden soll. Das kann man beispielsweise nutzen, um immer wiederkehrende Rechnungen direkt in den entsprechenden Ordner zu verschieben. Wünschenswert wäre, wenn es eine solche Funktion auch für versendete E-Mails gäbe. Da Outlook hier aber keine eingebaute Funktion bereitstellt, schauen wir uns das Thema einmal genauer an und entwickeln VBA-Code, mit dem wir diese Aufgabe selbst steuern können. Dabei wollen wir sowohl über die Benutzeroberfläche gesendete E-Mails erfassen als auch E-Mails, die wir per VBA absenden.

Wenn wir in Outlook mit der rechten Maustaste auf eine E-Mail in einem beliebigen Ordner klicken, erscheint das Kontextmenü und offeriert uns den Eintrag **Regeln|Regel erstellen ...** (siehe Bild 1).

Damit öffnen wir einen Dialog namens Regel erstellen, mit dem wir genauer festlegen können, wann die Regel ausgelöst werden soll und was wir damit bewirken

wollen (siehe Bild 2). Wir können hier zum Beispiel angeben, dass nur E-Mails mit einem bestimmten Betreff, für einen bestimmten Empfänger oder von einem bestimmten Absender verarbeitet werden sollen. Außerdem geben wir an, was mit der E-Mail geschehen soll. Hier reichen die Möglichkeiten von einer Benachrichtigung über die Ausgabe eines Sounds bis hin zum Verschieben in einen bestimmten Ordner.

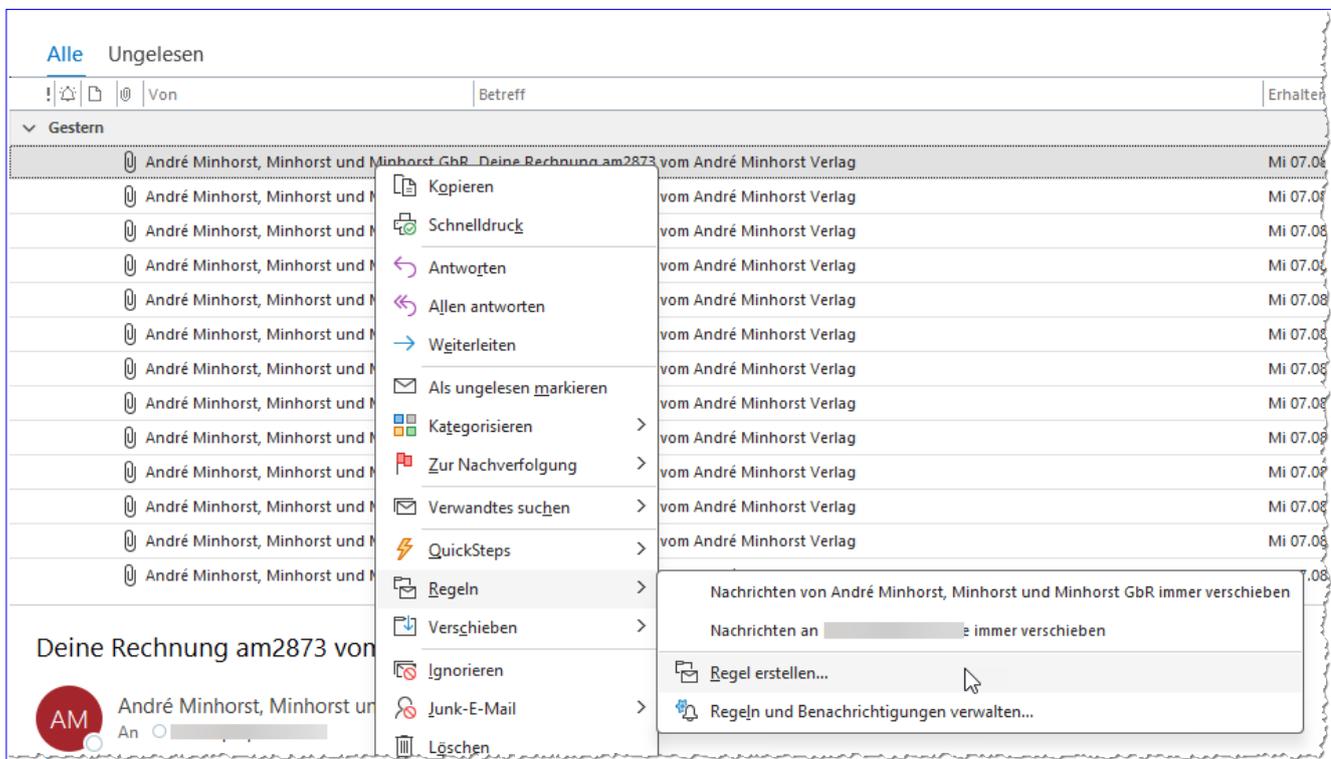


Bild 1: Anlegen einer neuen Regel für eingehende Nachrichten

Projekte verwalten mit Trello

Mittlerweile gibt es einige Online-Tools, mit denen man Projekte verwalten kann. Der Vorteil solcher Tools in der heutigen Zeit ist, dass wir diese von überall aus nutzen können. Egal, ob Mitarbeiter vor Ort arbeiten, beim Kunden sind oder auch im Homeoffice sitzen – die Projektdaten sind immer zugriffsbereit und können jederzeit aktualisiert werden. Das könnte auch für uns als Access-, Office- oder VBA-Entwickler interessant sein. Die Zeiten sollten vorbei sein, an denen man sich als Access- oder Excel-Entwickler in seiner Ehre verletzt fühlt, wenn man Projekte nicht mit seiner Lieblings-Officeanwendung verwalten kann. Da diese Tools nur bedingt onlinetauglich sind, wir aber dennoch immer den Überblick über den Stand verschiedener Projekte haben wollen, sollten wir uns für alternative Möglichkeiten öffnen. Aber nicht nur deshalb schauen wir uns in diesem Artikel das Projektmanagement-Tool Trello an, sondern auch, weil es eine für uns sehr wichtige Eigenschaft erfüllt: Es liefert eine Rest-API-Schnittstelle, über die wir die Daten des Tools auslesen und auch bearbeiten können. Wie das gelingt, betrachten wir in einem weiteren Artikel.

Die grundlegende Struktur von Trello sieht wie folgt aus:

- Ein **Arbeitsbereich** ist repräsentiert die höchste Ebene in Trello. Er dient dazu, einem oder mehreren Benutzern wie einer Abteilung oder einem Projektteam Informationen bereitzustellen. Dazu fassen diese Arbeitsbereiche ein oder mehrere Boards zusammen.
- Ein **Board** ist eine Ablage, auf der man Projekte oder Arbeitsprozesse unterbringen kann. Jedes Board enthält eine oder mehrere Listen, die mit Aufgaben gefüllt werden. Die Listen entsprechen dabei aber nicht solchen Listen wie etwa in To-Do-Listen, sondern stehen für einen Status – genau so wie es in Kanban gehandhabt wird. Man kann die einzelnen Aufgaben dann von einer Liste wie **Zu erledigen** zu **In Bearbeitung** und dann zu **Erledigt** verschieben.
- **Karten** entsprechen den Aufgaben. Jede Karte kann eine Aufgabe aufnehmen mit Informationen wie Titel, Beschreibung, Fälligkeitsdatum, zugeordnete

Mitglieder, Labels, Datum, Anhänge, Kommentare et cetera bis hin zu benutzerdefinierten Feldern.

- Die bereits erwähnten Listen wie **Zu erledigen**, **In Bearbeitung** oder **Erledigt** sind jeweils einem Board untergeordnet und können eine oder mehrere Karten mit Aufgaben aufnehmen.
- Karten nehmen zwar bereits Aufgaben auf, wir können jeder Karte jedoch auch noch **Checklisten** unterordnen. Diese enthalten Unteraufgaben, die wiederum einem Mitarbeiter zugeordnet werden oder einen Fälligkeitstermin erhalten können.
- **Labels** sind farbige Tags, die Karten zugewiesen werden können, um sie nach Kategorie, Priorität oder Thema zu organisieren.
- **Mitglieder** sind Benutzer, die zu Karten zugewiesen werden und für die Ausführung dieser Aufgaben verantwortlich sind.

Schauen wir uns nun an, wie wir diese Elemente verwenden können.

Projekte verwalten mit Trello

Mittlerweile gibt es einige Online-Tools, mit denen man Projekte verwalten kann. Der Vorteil solcher Tools in der heutigen Zeit ist, dass wir diese von überall nutzen können. Egal, ob Mitarbeiter vor Ort arbeiten, beim Kunden sind oder auch im Homeoffice sitzen – die Projektdaten sind immer zugriffsbereit und können jederzeit aktualisiert werden. Das könnte auch für uns als Access-, Office- oder VBA-Entwickler interessant sein. Die Zeiten sollten vorbei sein, an denen man sich als Access- oder Excel-Entwickler in seiner Ehre verletzt fühlt, wenn man Projekte nicht mit seiner Lieblings-Officeanwendung verwalten kann. Da diese Tools nur bedingt onlinetauglich sind, wir aber dennoch immer den Überblick über den Stand verschiedener Projekte haben wollen, sollten wir uns für alternative Möglichkeiten öffnen. Aber nicht nur deshalb schauen wir uns in diesem Artikel das Projektmanagement-Tool Trello an, sondern auch, weil es eine für uns sehr wichtige Eigenschaft erfüllt: Es liefert eine Rest-API-Schnittstelle, über die wir die Daten des Tools auslesen und auch bearbeiten können. Wie das gelingt, betrachten wir in einem weiteren Artikel.

Die grundlegende Struktur von Trello sieht wie folgt aus:

- Ein **Arbeitsbereich** ist repräsentiert die höchste Ebene in Trello. Er dient dazu, einem oder mehreren Benutzern wie einer Abteilung oder einem Projektteam Informationen bereitzustellen. Dazu fassen diese Arbeitsbereiche ein oder mehrere Boards zusammen.
- Ein **Board** ist eine Ablage, auf der man Projekte oder Arbeitsprozesse unterbringen kann. Jedes Board enthält eine oder mehrere Listen, die mit Aufgaben gefüllt werden. Die Listen entsprechen dabei aber nicht solchen Listen wie etwa in To-Do-Listen, sondern stehen für einen Status – genau so wie es in Kanban gehandhabt wird. Man kann die einzelnen Aufgaben dann von einer Liste wie **Zu erledigen** zu **In Bearbeitung** und dann zu **Erledigt** verschieben.
- **Karten** entsprechen den Aufgaben. Jede Karte kann eine Aufgabe aufnehmen mit Informationen wie Titel, Beschreibung, Fälligkeitsdatum, zugeordnete

Mitglieder, Labels, Datum, Anhänge, Kommentare et cetera bis hin zu benutzerdefinierten Feldern.

- Die bereits erwähnten Listen wie **Zu erledigen**, **In Bearbeitung** oder **Erledigt** sind jeweils einem Board untergeordnet und können eine oder mehrere Karten mit Aufgaben aufnehmen.
- Karten nehmen zwar bereits Aufgaben auf, wir können jeder Karte jedoch auch noch **Checklisten** unterordnen. Diese enthalten Unteraufgaben, die wiederum einem Mitarbeiter zugeordnet werden oder einen Fälligkeitstermin erhalten können.
- **Labels** sind farbige Tags, die Karten zugewiesen werden können, um sie nach Kategorie, Priorität oder Thema zu organisieren.
- **Mitglieder** sind Benutzer, die zu Karten zugewiesen werden und für die Ausführung dieser Aufgaben verantwortlich sind.

Schauen wir uns nun an, wie wir diese Elemente verwenden können.

Trello per Rest-API steuern

Trello ist eines der bekanntesten Projektmanagement-Tools. Damit lassen sich Projekte, Aufgaben und ToDos im Kanban-Style verwalten. Die Basisfunktionen sind bereits mächtig und dabei aber nicht unübersichtlich und man kann über sogenannte PowerUps viele zusätzliche Funktionen integrieren. Nicht zu reden von den Automationen – viele sind interne Automationen, die innerhalb von Trello genutzt werden können und natürlich gibt es auch einige Möglichkeiten, Trello und die enthaltenen Elemente über Werkzeuge wie Zapier oder Make von anderen Tools zu steuern und umgekehrt. Aber wir wären keine VB/VBA-Entwickler, wenn wir nicht per Code auf die in Trello angelegten Boards, Karten und Eigenschaften zugreifen wollten, um diese zu lesen oder zu schreiben. Die Weboberflächen vieler noch so guter Tools bieten letztlich nicht immer alle nötigen Funktionen, die man aber abbilden kann, wenn das Tool eine Rest-API anbietet und man sich mit der Steuerung solcher APIs unter VB/VBA auskennt. Dann können wir beispielsweise schnell die Projekte oder Aufgaben aus einer Excel- oder Access-Tabelle in Trello anlegen oder auch auf die in Trello enthaltenen Daten zugreifen, um diese mit den Daten in unserer Datenbankwendung abzugleichen. In diesem Artikel zeigen wir, wie dies gelingt.

Voraussetzungen für die Nutzung der Trello-Rest-API

Voraussetzung ist, dass wir einen Trello-Account angelegt haben.

Im Gegensatz zu verschiedenen anderen Anbietern gibt es bei Trello keine grundsätzlichen Einschränkungen in Abhängigkeit vom Zahlungsplan, den man gewählt hat – die Rest-API können wir immer nutzen.

Allerdings sind bestimmte Funktionen erst mit den höheren, kostenpflichtigen Plänen verfügbar. So können wir beispielsweise mit den höheren Plänen eine höhere Anzahl von Aufrufen pro Zeiteinheit absetzen oder auch erweiterte Funktionen nutzen. Benutzerdefinierte Felder beispielsweise können wir mit dem kostenlosen Plan noch nicht bearbeiten oder nutzen.

Erste Schritte für die Trello-Rest-API

Wann immer wir eine Rest-API verwenden wollen, benötigen wir einen Key, mit dem wir uns bei der Rest-API für unseren Account referenzieren können.

Das bedeutet, dass wir irgendwo in unserem Trello-Kundenkonto eine solche Zeichenfolge finden oder anlegen können sollten. Die grundlegenden Informationen finden wir unter dieser URL:

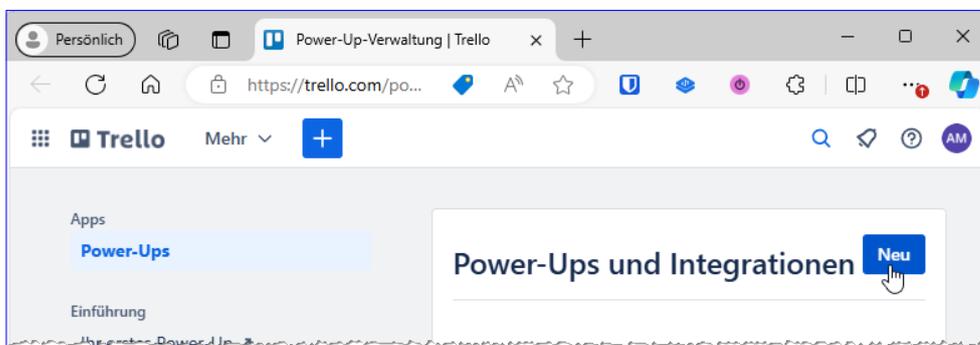


Bild 1: Seite zum Anlegen eines Power-Ups oder einer Integration

Trello per Rest-API steuern

Trello ist eines der bekanntesten Projektmanagement-Tools. Damit lassen sich Projekte, Aufgaben und Todos im Kanban-Style verwalten. Die Basisfunktionen sind bereits mächtig und dabei aber nicht unübersichtlich und man kann über sogenannte PowerUps viele zusätzliche Funktionen integrieren. Nicht zu reden von den Automationen – viele sind interne Automationen, die innerhalb von Trello genutzt werden können und natürlich gibt es auch einige Möglichkeiten, Trello und die enthaltenen Elemente über Werkzeuge wie Zapier oder Make von anderen Tools zu steuern und umgekehrt. Aber wir wären keine VB/VBA-Entwickler, wenn wir nicht per Code auf die in Trello angelegten Boards, Karten und Eigenschaften zugreifen wollten, um diese zu lesen oder zu schreiben. Die Weboberflächen vieler noch so guter Tools bieten letztlich nicht immer alle nötigen Funktionen, die man aber abbilden kann, wenn das Tool eine Rest-API anbietet und man sich mit der Steuerung solcher APIs unter VB/VBA auskennt. Dann können wir beispielsweise schnell die Projekte oder Aufgaben aus einer Excel- oder Access-Tabelle in Trello anlegen oder auch auf die in Trello enthaltenen Daten zugreifen, um diese mit den Daten in unserer Datenbankwendung abzugleichen. In diesem Artikel zeigen wir, wie dies gelingt.

Voraussetzungen für die Nutzung der Trello-Rest-API

Voraussetzung ist, dass wir einen Trello-Account angelegt haben.

Im Gegensatz zu verschiedenen anderen Anbietern gibt es bei Trello keine grundsätzlichen Einschränkungen in Abhängigkeit vom Zahlungsplan, den man gewählt hat – die Rest-API können wir immer nutzen.

Allerdings sind bestimmte Funktionen erst mit den höheren, kostenpflichtigen Plänen verfügbar. So können wir beispielsweise mit den höheren Plänen eine höhere Anzahl von Aufrufen pro Zeiteinheit absetzen oder auch erweiterte Funktionen nutzen. Benutzerdefinierte Felder beispielsweise können wir mit dem kostenlosen Plan noch nicht bearbeiten oder nutzen.

Erste Schritte für die Trello-Rest-API

Wann immer wir eine Rest-API verwenden wollen, benötigen wir einen Key, mit dem wir uns bei der Rest-API für unseren Account referenzieren können.

Das bedeutet, dass wir irgendwo in unserem Trello-Kundenkonto eine solche Zeichenfolge finden oder anlegen können sollten. Die grundlegenden Informationen finden wir unter dieser URL:

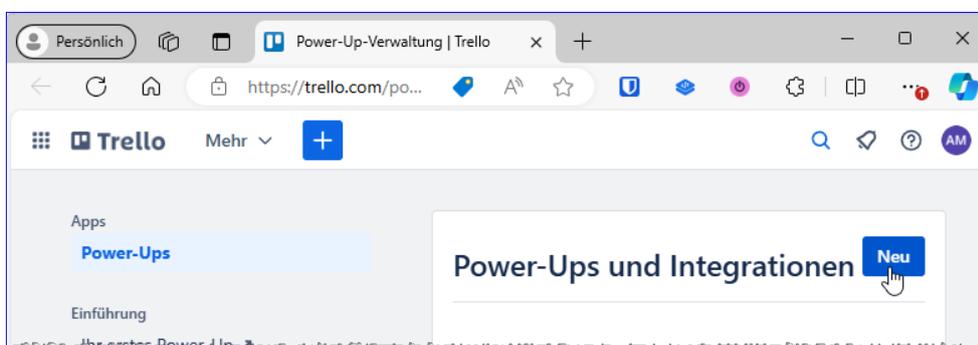


Bild 1: Seite zum Anlegen eines Power-Ups oder einer Integration

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

DATEIZUGRIFF MIT VBA

Lerne verschiedene Techniken für den Zugriff auf Dateien und Verzeichnisse kennen..

SEITE 7

AUTOMATION MIT ZAPIER, MAKE UND CO.

Erfahre, wie Du externe Softwaretools einmal ohne VBA-Einsatz automatisieren kannst.

SEITE 27

BASICS DER ADODB-PROGRAMMIERUNG

Nutze ADODB für den Zugriff auf die Daten von SQL Server- und anderen Datenbanken.

SEITE 44



André Minhorst Verlag

ADODB: Connections und Connectionstrings

Wer auf die Inhalte von Datenbanken wie Access, SQL Server und anderen zugreifen möchte, benötigt eine spezielle Datenzugriffstechnologie. Unter VB6, VBA und twinBASIC verwendet man dazu in der Regel die DAO-Bibliothek, in Office »Microsoft Office 16.0 Access database engine Object Library« genannt, oder die ADODB-Bibliothek (»Microsoft ActiveX Data Objects 6.1 Library«). Geschichtlich wurde mal die eine, mal die andere von Microsoft als die zu bevorzugende Datenzugriffstechnik bezeichnet. Derzeit verwendet man meist DAO, vor allem in Verbindung mit Access-Datenbanken, aber beim Zugriff auf SQL Server-Datenbanken bietet ADODB einige Features, die wir mit DAO nicht nutzen können. In diesem Artikel steigen wir in die Programmierung von Datenbankzugriffen mit ADODB ein. Dabei schauen wir uns als Erstes die Connection-Klasse an, mit der erst einmal eine Verbindung zur Datenbank aufgebaut werden kann.

ADODB wurde mit der Version Access 2000 erstmalig eingeführt. Zuvor war DAO die meistverwendete Datenzugriffstechnik (Bibliothek **Microsoft DAO 3.6 Object Library**) in Zusammenhang mit dem Zugriff über die JET-Datenbank-Engine. Im Laufe der Jahre hat Microsoft jedoch entschieden, dass DAO wieder die wichtigste Technik für den Zugriff auf Datenbanken sein soll. Damit einher ging mit Access 2007 die Einführung einer neuen Variante der JET-Engine namens **ACE** (Access Database Engine) und damit eine neue Bibliothek mit dem Titel **Microsoft Office 16.0 Access database engine Object Library**.

Wie bereits erwähnt, gibt es jedoch einige Einsatzzwecke, bei denen ADODB Vorteile gegenüber DAO hat, weil es verschiedene Funktionen zur Verfügung stellt, die es in DAO nicht gibt. Dazu gehören die folgenden:

- **Bessere Unterstützung für SQL Server:** ADODB ist für client-server-basierte Datenbanken wie SQL Server optimiert. DAO ist primär für Microsoft Access und Jet-Datenbanken konzipiert und weniger effizient bei der Arbeit mit SQL Server.
- **Unterstützung für OLE DB:** ADODB verwendet OLE DB oder ODBC, um sich mit SQL Server zu

verbinden, was eine performante und flexible Verbindung ermöglicht. DAO ist auf die Jet-Engine beschränkt, die für SQL Server nicht optimiert ist. Mit DAO können wir nur in direkt über Tabellenverknüpfungen oder Pass-Through-Abfragen auf SQL Server-Tabellen zugreifen.

- **Bessere Performance bei großen Datenmengen:** ADODB kann große Datenmengen effizient über serverseitige Cursor oder Forward-Only-Recordsets abrufen. DAO lädt oft ganze Datensatzgruppen in den Speicher, was zu Performance-Problemen führt.
- **Mehr Flexibilität bei der Abfrageverarbeitung:** ADODB erlaubt den direkten Aufruf von gespeicherten Prozeduren und somit parametrisierten Abfragen, was die Sicherheit und Performance erhöht. DAO unterstützt keine direkten Prozeduraufrufe in SQL Server.
- **Transaktionsunterstützung für SQL Server:** ADODB bietet eine bessere Transaktionskontrolle mit **BeginTrans**, **CommitTrans** und **RollbackTrans**. DAO bietet zwar Transaktionsunterstützung, aber diese ist für Jet-Datenbanken optimiert und nicht für SQL Server.

ADODB: SQL-Befehle schnell ausführen mit Execute

ADODB bietet verschiedene Techniken, um SQL-Befehle zum Abfragen oder Ändern von Daten auszuführen. Die bekannteste zum Abfragen von Daten dürfte die `OpenRecordset`-Methode sein. Für das Ausführen von Anweisungen kann man für vollen Komfort am besten die `Command`-Klasse mit der `Execute`-Methode verwenden. Aber auch die `Connection`-Klasse bietet bereits eine `Execute`-Methode an. Mit dieser lässt sich schnell und flexibel Einiges erledigen. Welche Möglichkeiten sie bietet und wie wir diese für den Datenzugriff nutzen können, zeigen wir in diesem Artikel.

Die `Execute`-Methode des `Connection`-Objekts ist die einfachste Möglichkeit, eine Abfrage zum Ändern der Daten einer SQL Server-Datenbank auszuführen. Damit können wir jedoch nicht nur Daten ändern, in dem wir eine der folgenden Aktualisierungsabfrage nutzen:

- **DELETE:** Löschen von Datensätzen
- **UPDATE:** Aktualisieren von Datensätzen
- **INSERT INTO:** Einfügen von Datensätzen

Wir können durch das Ausführen der `Execute`-Methode auch solche SQL-Anweisungen ausführen, mit denen wir den Entwurf des Datenmodells selbst ändern – zum Anlegen, Ändern oder Löschen von Tabellen, Feldern, Indizes und vielem mehr.

Erstellen einer Tabelle mit Execute

Im ersten Beispiel erstellen wir erst einmal eine Tabelle, mit der wir danach weitere Aktionen durchführen. Dieses Beispiel finden wir in Listing 1. Hier deklarieren wir zunächst ein `Connection`-Objekt sowie eine Zeichenkette zum Speichern der SQL-Anweisung.

```
Public Sub TabelleErstellen()  
    Dim cnn As ADODB.Connection  
    Dim strSQL As String  
  
    Set cnn = New ADODB.Connection  
    cnn.ConnectionString = "Provider=MSOLEDBSQL;Data Source=amvDesktop2023;Initial Catalog=Anlagen;" _  
        & "Integrated Security=SSPI;"  
    cnn.Open  
  
    strSQL = "CREATE TABLE tblKategorien (" & vbCrLf  
    strSQL = strSQL & "    KategorieID INT IDENTITY(1,1) PRIMARY KEY," & vbCrLf  
    strSQL = strSQL & "    Kategoriebezeichnung VARCHAR(255) NOT NULL" & vbCrLf  
    strSQL = strSQL & ");"  
  
    cnn.Execute strSQL  
  
    cnn.Close  
    Set cnn = Nothing  
End Sub
```

Listing 1: Prozedur zum Erstellen einer Tabelle

Dateien mit VBA-Bordmitteln verwalten

Die Verwaltung von Dateien per Code ist öfter gefragt, als man denkt. Damit lassen sich Verzeichnisse erstellen, auslesen und entfernen, Dateien erstellen, löschen und bearbeiten und vieles mehr. Leider sind die Bordmitteln von VBA hier teilweise ein wenig sperrig. Allerdings sind sie für einfache Aufgaben durchaus ausreichend und erfordern nicht den Einsatz einer zusätzlichen Bibliothek wie der Scripting Runtime, wie es beispielsweise beim FileSystemObject der Fall ist. Also schauen wir uns in diesem Artikel einmal die Elemente von VBA an, mit denen wir auf die Elemente des Dateisystems zugreifen können und zeigen, was sich damit alles anstellen lässt. Schließlich schauen wir aber auch noch kritisch auf mögliche Nachteile.

Definitionen

In diesem Artikel werden wir die verschiedenen Elemente des Dateisystems benennen. Dabei wollen wir

folgende Elemente definieren: Ein Pfad ist der gesamte Pfad von der Wurzel bis zum Dateinamen inklusive Dateiendung, also zum Beispiel `c:\Verzeichnis1\`

`Datei1.txt`, oder bis zu einem Verzeichnis (`c:\Verzeichnis1`). Ein Verzeichnis ist das darin enthaltene `Verzeichnis1`. Eine Datei oder Dateiname ist `Datei1.txt`.

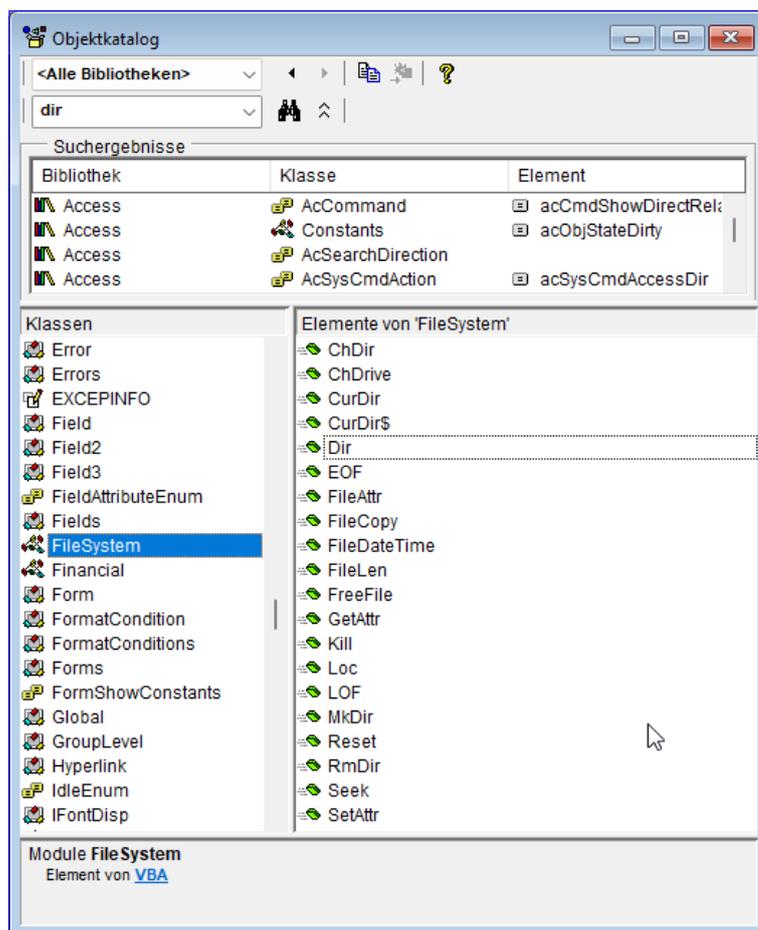


Bild 1: Elemente der Klasse FileSystem im Objektkatalog

Die FileSystem-Klasse und Alternativen

Eine der bekanntesten Anweisungen zum Arbeiten mit dem Dateisystem ist vermutlich die `Dir()`-Funktion. Suchen wir im Objektkatalog nach dieser, finden wir schnell heraus, dass sie zu einer Klasse namens `FileSystem` gehört (siehe Bild 1).

Diese enthält noch einige weitere Elemente, von denen man vermutlich nicht alle auf dem Schirm hat.

In diesem Artikel werden wir uns diese im Detail ansehen und verschiedene praktische Lösungen zeigen, die sich damit realisieren lassen – ganz ohne weitere Bibliotheken.

Allerdings wollen wir alternative Möglichkeiten, die überdies in vielen Fällen leichter

Automation mit Zapier, Make und Co.

Es gibt immer mehr Möglichkeiten, sogenannte SaaS (Software as a Service) durch Automationen zu verknüpfen. SaaS sind zum Beispiel Calendly zur Terminplanung, Online-Buchhaltungsprogramme wie lexoffice, Projektmanagement-Tools wie Trello, E-Mail-Marketing-Tools, E-Commerce-Anwendungen, CRM-Systeme, Business Intelligence Tools wie Power BI oder täglich verwendete Tools wie Microsoft Teams, Zoom oder die Google Suite. Diese Tools haben wichtige Gemeinsamkeiten: Sie sind allesamt online verfügbar und somit über den Browser steuerbar. Und sie bieten Automations-Schnittstellen an, mit denen sie einerseits Trigger auslösen können und damit andererseits verschiedene Aktionen anstoßen. Da Du als Leser dieses Magazins zweifelsohne früher oder später mit Tools wie diesen in Verbindung kommen wirst, wollen wir Dir zeigen, wie Du Automationen zwischen diesen Tools erstellen kannst. Dabei wollen wir in diesem Einführungsartikel zwei der bekanntesten Tools ansehen, und zwar Zapier und Make.com. Microsoft Power Automate haben wir bereits früher einmal in diesem Magazin vorgestellt.

Wozu überhaupt Automation?

Diese Frage wirst Du Dir vermutlich nicht stellen, denn mit zum Beispiel mit VBA-Code automatisierst Du ja selbst auch bereits Deine Access-Anwendungen. Du definierst Ereignisprozeduren, die Du für Ereigniseigenschaften von Formularen oder Steuerelementen hinterlegst und führst bestimmte Aktionen aus, wenn der Benutzer diese betätigt.

Vielleicht verwendest Du sogar anwendungsübergreifende Automationen, wo Du beispielsweise von Access aus andere Anwendungen wie Excel, Word oder Outlook so programmierst, dass Du von Access aus auf Aktionen in diesen Anwendungen reagieren kannst.

Software as a Service (SaaS)

Seit einigen Jahren gibt es immer mehr Anwendungen, die nicht wie Microsoft oder andere Desktop-Anwendungen komplett auf Deinem Rechner ablaufen, sondern die auf einem Server liegen und die Du über den Internetbrowser bedienen kannst. Dazu gehören Anwendungen aus den folgenden Bereichen:

- Buchhaltung und Finanzen (Lexware Office, Datev, Easybill, ...)
- Vertrieb und Kundenmanagement (CRM-Systeme)
- Projektmanagement (Trello, Asana, Notion, ...)
- Schulungsplattformen/E-Learning
- E-Mail-Marketing-Tools zum Versenden von Newslettern
- Online-Shops (Shopware, Shopify, Magenta, ...)
- Business Intelligence (Power BI)
- Kundenbetreuung (Zendesk, Intercom)
- Kommunikation und Videokonferenzen (Microsoft Teams, Slack, ...)

Vielleicht bist Du schon seit Ewigkeiten Access- oder VB-Entwickler und stehst auf dem Standpunkt, dass Du bis zu Deinem Ruhestand auch noch mit den al-

SQL Server: Fehlerbehandlung in DAO und ADODB

Die Fehlerbehandlung in VBA ist relativ einfach zu handhaben, sobald man einmal die grundlegenden Techniken kennt. Nicht viel anders sollte es eigentlich sein, wenn man die Daten nicht mehr aus Access-Tabellen bezieht, sondern aus SQL Server-Tabellen. Es gibt jedoch einige Unterschiede, die man kennen sollte. In diesem Artikel zeigen wir, welche Besonderheiten auftreten, wenn man nicht mehr auf Access-Tabellen zugreift, sondern auf verknüpfte SQL Server-Tabellen und dabei die Datenzugriffstechniken DAO- und ADODB nutzt. Im letzteren Fall gibt es sogar noch Unterschiede bezüglich der verwendeten Treiber/Provider.

Wenn wir mit VBA programmieren und es wird ein Laufzeitfehler ausgelöst, liefert dieser immer genau ein Fehlerobjekt namens **Err** zurück, das uns mit Eigenschaften wie **Number** oder **Description** Informationen über den Fehler liefert.

Wenn wir mit DAO auf die Daten eines SQL Servers oder einer anderen per ODBC-Tabellenverknüpfung eingebundenen Datenbanksystems zugreifen, erhalten wir ebenfalls eine Fehlermeldung mit **Err**.

Allerdings lautet diese immer gleich – nämlich den Fehler **ODBC-Aufruf fehlgeschlagen** mit der Nummer **3146**.

Damit kann man eigentlich nichts anfangen – da dieser Fehler immer in Zusammenhang mit dem Datenzugriff auftritt, wissen wir zwar grob, worum es geht, aber nicht genau, was der eigentliche Fehler ist.

Man kann zwar nun die auslösende SQL-Anweisung kopieren, sie im SQL Server Management Studio ausführen und erhält dann dort die eigentliche Fehlermeldung. Das hilft uns allerdings nicht, wenn wir im VBA-Code darauf reagieren wollen.

In diesem Artikel stellen wir Dir eine Möglichkeit vor, wie Du dennoch auf die Fehlerinformationen zugreifen kannst. Dazu gibt es in DAO und ADODB verschiedene Möglichkeiten.

Unterscheidung der verschiedenen Fehlerklassen

Zuerst jedoch ein kleiner Blick auf das »normale« **Err**-Objekt. Dieses hat den Typ **ErrObject** und bietet die folgenden Eigenschaften und Methoden.

Hier zunächst die Eigenschaften:

- **Description:** Enthält die Fehlerbeschreibung als String.
- **HelpContext:** Gibt die ID des Hilfethemas zurück, das mit dem Fehler verbunden ist.
- **HelpFile:** Gibt den Pfad zur zugehörigen Hilfedatei zurück (falls vorhanden).
- **LastDLLError:** Gibt den letzten Windows-API-Fehlercode zurück (nur bei API-Aufrufen relevant).
- **Number:** Gibt die Fehlernummer zurück. **0** bedeutet, dass kein Fehler aufgetreten ist. Bei eingebauten Elementen sind die Fehlernummern positiv und ein- bis fünfstellig. Es gibt aber auch Fehlermeldungen mit großen negativen Fehlernummern.
- **Source:** Gibt die Quelle des Fehlers zurück (zum Beispiel **VBAProject**).

Und hier sind die beiden Methoden:

Ist die 32- oder 64-Bit-Version von Office installiert?

Diese Frage, ob Office in der 32-Bit- oder in der 64-Bit-Version vorliegt, ist für viele Aufgaben interessant. Seit Access 2019 wird Office standardmäßig in der 64-Bit-Version installiert. Es gibt jedoch auch immer alternativ die 32-Bit-Version. Vor Access 2019 war die 32-Bit-Installation Standard. Wozu aber benötigen wir diese Information überhaupt? Wenn wir das VBA-Projekt einer Access-, Excel-, Word- oder PowerPoint-Datei programmieren oder das Outlook-Objektmodell und dabei weder ActiveX-Steuerelemente noch Integrationen wie COM-DLLs oder COM-Add-Ins oder API-Funktionen nutzen, spielt es keine Rolle, ob wir mit 32-Bit- oder 64-Bit-Office arbeiten. Sobald jedoch eines der genannten Elemente auftaucht, müssen wir genau prüfen, ob dieses unter beiden Versionen arbeitet. Wir schauen uns kurz an, wo besonderes Augenmerk gefragt ist und wie wir es dem Benutzer mitteilen können, wenn seine Office-Version und unsere Erweiterungen nicht kompatibel sind.

Wie finde ich heraus, ob die 32- oder 64-Bit-Version von Access/Office installiert ist? Die Information, ob auf einem Rechner die 32-Bit- oder 64-Bit-Version von Microsoft Access oder Office installiert ist, ist in verschiedenen Szenarien wichtig. Beispielsweise spielt dies bei der Entwicklung von Access-Anwendungen oder bei der Verwendung von VBA (Visual Basic for Applications) eine Rolle, da die verwendeten Bibliotheken und API-Deklarationen an die Bit-Version angepasst werden müssen.

Das ist vor allem bei den folgenden Situationen wichtig:

- Wenn wir API-Funktionen verwenden. Diese müssen unter 64-Bit anders deklariert werden als unter 32-Bit. In den meisten Fällen erhalten wir bereits beim Kompilieren eines VBA-Projekts in der 64-Bit-Version Fehlermeldungen, wenn die Deklaration der API-Funktionen nicht kompatibel ist. Der Teufel steckt aber im Detail, denn es können auch Korrekturen der Datentypen etwa von Parametern erforderlich sein, da es sonst zu Laufzeitfehlern kommt.

- Wenn wir ActiveX-Steuerelemente von Drittherstellern nutzen, sind diese oft nur für die 32-Bit-Versionen von Office verfügbar. Gerade bei der Verwendung älterer Steuerelemente, die nicht mehr weiterentwickelt werden, ist oft das Ersetzen des vollständigen Steuerelements durch eine Alternative notwendig.
- Wenn wir COM-DLLs oder COM-Add-Ins nutzen, gelten die gleichen Regeln. Diese sind entweder für 32-Bit oder für 64-Bit entwickelt worden und funktionieren nur mit der jeweiligen Office-Version zusammen. Wer selbst solche Elemente entwickelt, beispielsweise mit twinBASIC, VB6 oder Visual Studio .NET, hat immerhin die Möglichkeit, die COM-DLLs oder COM-Add-Ins für die entsprechende Version zu kompilieren.

Möglichkeiten zur Prüfung auf 32-Bit oder 64-Bit

In den folgenden Abschnitten schauen wir uns verschiedene Möglichkeiten an, um herauszufinden, ob Office in der 32-Bit- oder in der 64-Bit-Version installiert ist:

VBA und Textdateien: Alles über Open, Close und Co.

VBA bietet einige Möglichkeiten für das Erstellen, Füllen, Ändern und Auslesen von Textdateien. Mit diesen kann man zwar nicht alle Aufgaben bewältigen, aber für einfache Zwecke lohnt sich ein Blick auf diese Sammlung. Wir meinen damit Anweisungen wie **Open**, **Close**, **Print**, **Input** und einige weitere, die spannenderweise im Objektkatalog noch nicht einmal erwähnt werden. Wir werden uns in diesem Artikel eingehend mit diesen Anweisungen beschäftigen und in verschiedenen Beispielen zeigen, wie sich damit immer wiederkehrende Aufgaben wie das Schreiben oder Lesen von Textdateien leicht bewältigen lässt.

Im Artikel **Dateien mit VBA-Bordmitteln verwalten** (www.vbentwickler.de/****) haben wir uns angesehen, welche Elemente die Klasse **FileSystem** für uns bereithält.

Hier finden wir einige Anweisungen, mit denen wir Verzeichnisse erstellen und löschen, Dateien und Verzeichnisse durchlaufen und weitere Dinge erledigen können.

Die Klasse **FileSystem** enthält jedoch auch einige Elemente, die wir in diesem Zusammenhang gar nicht benötigen. Sie sind vielmehr für den Einsatz mit einigen Anweisungen vorgesehen, die wir nicht in der Klasse **FileSystem** und sogar noch nicht einmal überhaupt im Objektkatalog des VBA-Editors finden.

Dabei geht es um solche Anweisungen wie **Open**, **Close**, **Write** et cetera:

- **Close**: Schließt eine oder mehrere zuvor mit **Open** geöffnete Dateien.
- **Get**: Dient dem Binärzugriff auf Dateien und wird in diesem Artikel nicht erläutert.
- **Input #**: Erlaubt das Einlesen der
- **Line Input #**: Erlaubt das zeilenweise Einlesen von Inhalten einer Textdatei.

- **Open**: Öffnet eine Datei für den lesenden oder schreibenden Zugriff oder erstellt eine neue Datei für den schreibenden Zugriff.
- **Print #**: Schreibt Freitext in Dateien.
- **Put**: Dient dem Binärzugriff auf Dateien und wird in diesem Artikel nicht erläutert.
- **Seek**: Ruft die aktuelle Position beim Binärzugriff ab oder setzt diese. Wird in diesem Artikel nicht erläutert.
- **Width**: Erlaubt die Begrenzung der Zeilenlänge beim Schreiben mit **Print**.
- **Write #**: Schreibt strukturierte Daten in Dateien.

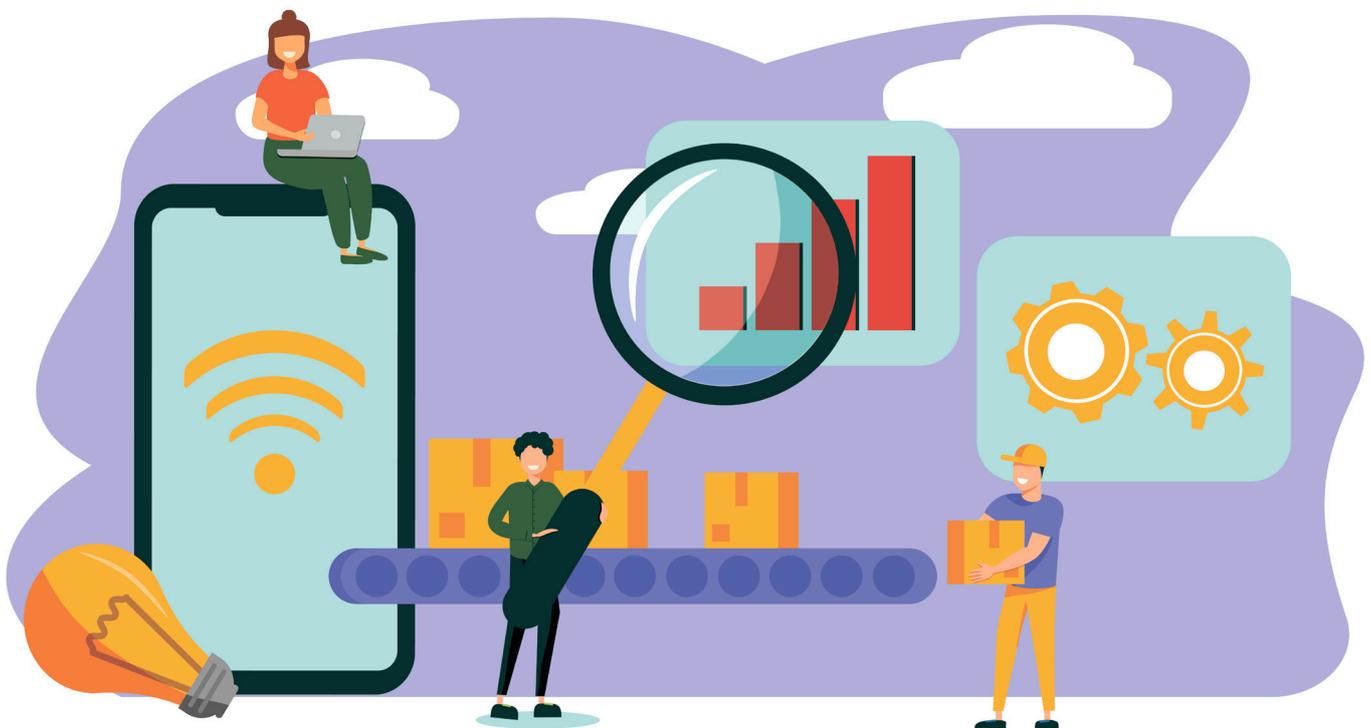
Neben diesen Anweisungen erläutern wir in diesem Artikel auch noch diese Elemente, die Teil der Klasse **FileSystem** sind:

- **EOF**: Abkürzung für **End Of File**. Erwartet die Angabe einer Dateinummer einer mit **Open** geöffneten Datei. Für diese Datei wird angegeben, ob beim Lesen das Ende der Datei erreicht wurde.
- **FileLen**: Gibt die Größe einer Datei oder eines Verzeichnisses (immer 0) in Bytes zurück und erwartet den Pfad zu des Elements als Parameter.

VISUAL BASIC

ENTWICKLER

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



IN DIESEM HEFT:

AUTOMATION MIT ZAPIER IN DER PRAXIS

Steige tiefer in den Aufruf von Zapier-Automationen per VBA ein.

SEITE 4

EBAY PER VBA STEUERN

Lerne, wie Du per VBA auf die eBay-API zugreifen kannst und hole Dir per VBA die notwendigen Token.

SEITE 38

ZWISCHENABLAGE PROGRAMMIEREN

Lese die Zwischenablage mit VBA aus und fülle sie per VBA mit beliebigen Zeichenketten.

SEITE 58



André Minhorst Verlag

Automationen mit Zapier per VBA starten

Zapier ist ein Tool, mit dem wir SaaS-Anwendungen für verschiedene Bereiche automatisieren können. Das geschieht nach dem Prinzip, dass eine solche Anwendung eine Automation triggert und dadurch Aktionen mit der gleichen oder auch mit anderen Anwendungen angestoßen werden. Wie zum Beispiel das Eintragen einer Aufgabe in eine Aufgabenliste, wenn eine neue E-Mail eingegangen ist. Wie haben in weiteren Artikel bereits gezeigt, wie wir solche Automationen mit Tools wie Zapier realisieren können. Aber welchen Bezug gibt es zum eigentlichen Thema dieses Magazins, nämlich dem Programmieren mit Sprachen wie VBA, VB6, twinBASIC et cetera – abgesehen davon, dass wir diese Automationen als Selbstständiger, Freiberufler, aber auch als Angestellter für unser Unternehmen gewinnbringend einsetzen können? Genau: Wie können diese Automationen natürlich auch per Code triggern. Wie das im Falle von Zapier funktioniert, erklären wir im vorliegenden Artikel.

Wenn Du Dich bereits mit Zapier zur Automation von SaaS-Anwendungen beschäftigt hast, kannst Du direkt in diesen Artikel einsteigen. Falls nicht, empfehlen wir zuvor die Lektüre der folgenden Artikel:

- **Automation mit Zapier, Make und Co.** (www.vbentwickler.de/448)
- **Automation mit Zapier in der Praxis** (www.vbentwickler.de/449)

Hier findest Du einige Grundlagen allgemein zum Thema Automatisierung und ein erstes Beispiel für eine Automation auf Basis von Microsoft Outlook und Microsoft To Do.

Automation per VBA nutzen

Wie Du in den obigen Artikeln erfährst, gibt es verschiedene Möglichkeiten, wie Du Automationen mit den verschiedenen Tools von VBA aus starten kannst. Für das Tool Zapier sind die Möglichkeiten ein klein wenig eingeschränkter, denn wir können damit keine Ergebnisse über die Erfolgsmeldung hinaus als Rückgabewert erhalten. Bei dem anderen Tool Make.com sieht dies anders aus – hier können wir explizit fest-

legen, welche Informationen nach dem Durchlaufen einer Automation an den aufrufenden VBA-Code zurückgeliefert werden sollen. Dies schauen wir uns in weiteren Artikeln an.

Dafür liefert Zapier wieder mehr Möglichkeiten bei der Zusammenarbeit mit dem SQL Server und anderen Datenbanksystemen, denn hier können wir mit dem sogenannten Polling auch auf Änderungen im Datenbestand reagieren.

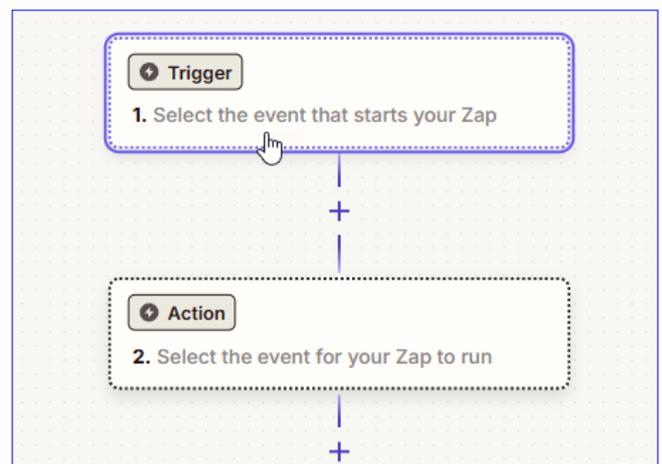


Bild 1: Diesen Trigger wollen wir per VBA ansteuern.

Automation mit Zapier in der Praxis

Zapier ist neben Make.com oder Microsoft Power Automate ein weit verbreitetes Tool, mit dem wir Automationen definieren können. Mit diesen können wir sowohl verschiedenen SaaS-Produkte untereinander so verbinden, dass diese Daten austauschen und automatische Prozesse angestoßen werden. Wir können uns aber auch von unserer Anwendung aus in diese Prozesse einklicken, indem wir diese per VBA anstoßen. In diesem Artikel schauen wir uns jedoch erst einmal an, was Zapier ist, wie wir ein Konto bei Zapier anlegen und wir erste, einfache Automationen einrichten.

Zapier ist wie die Produkte, die wir damit automatisieren können, eine SaaS-Lösung – also Software as a Service. Diese zeichnen sich dadurch aus, dass sie in der Cloud betrieben werden. Dadurch entstehen viele Vorteile. Wir können über das Internet auf diese zugreifen, wodurch wie die Installation auf unserem Rechner sparen. Damit geht einher, dass wir von überall auf diese Dienste zugreifen können. Die SaaS-Produkte werden automatisch aktualisiert, sind skalierbar und wir zahlen die Nutzung entsprechend der geplanten Nutzung.

Die grundlegende Erläuterung von Automationstools wie Zapier oder Make.com haben wir bereits im Artikel **Automation mit Zapier, Make und Co.** (www.vbentwickler.de/448) erläutert.

Nun schauen wir uns am Beispiel von Zapier an, wie die ersten Schritte damit funktionieren.

Dazu rufen wir die Webseite <https://zapier.com> auf und finden direkt ein Beispiel dafür vor, wie die Automationen grafisch

abgebildet werden (siehe Bild 1). Hier können wir direkt mit einem Klick auf den Link **Sign up** beginnen.

Vorab eine Information: In einem gewissen Rahmen können wir Zapier sogar kostenlos nutzen. Dabei sind wir allerdings eingeschränkt auf Automationen, die maximal aus zwei Schritten bestehen. Außerdem können wir nur 100 Aufrufe im Monat nutzen. Wir können jedoch bereits so viele verschiedene Automationen programmieren, wie wir möchten.

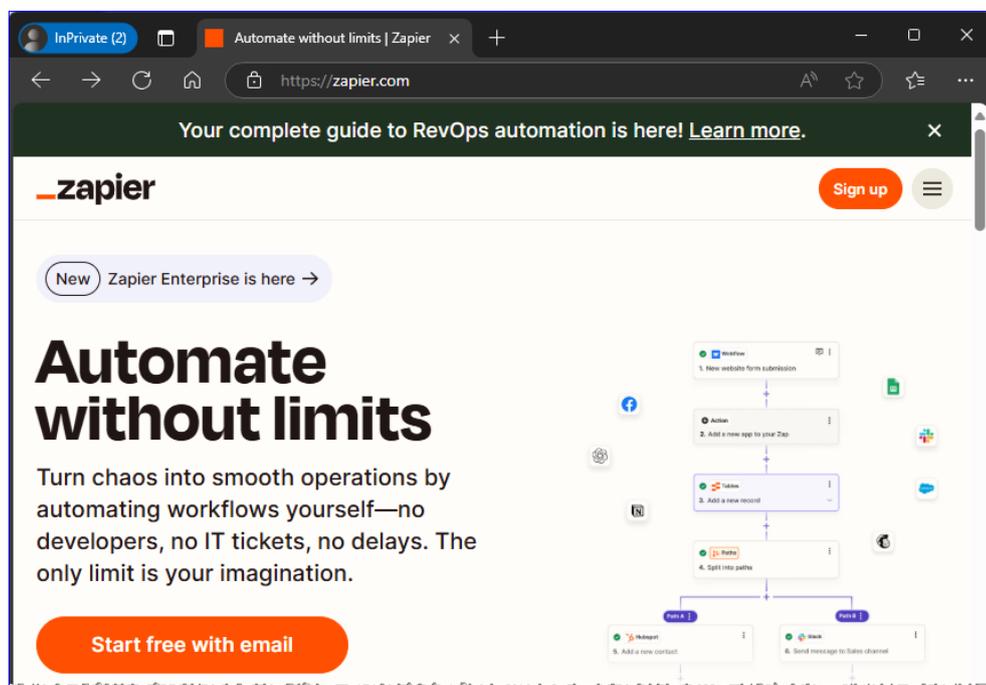


Bild 1: Startseite von Zapier

eBay per VBA steuern: Rest-API-Zugriff

Im ersten Teil der Artikelreihe zum Thema Steuerung von eBay mit VBA haben wir gezeigt, wie man einen Entwickler-Account anlegt, eine neue Anwendung bei eBay erstellt, grundlegende Zugriffsdaten holt und die für die Authentifizierung im Kontext eines bestimmten Benutzerkontos notwendigen Informationen holt – hier speziell das Authentifizierungstoken. Damit haben wir die Basis geschaffen, um per VBA auf die Rest API von eBay zuzugreifen. Damit fahren wir in diesem Teil fort: Wir wollen den grundlegenden Zugriff auf die Rest-API von eBay herstellen und verwenden dazu einfache Beispiele. Wie sich zeigt, gibt es zwei verschiedene Kontexte, in denen wir hier arbeiten, den Anwendungs- und den Benutzerkontext. Beide beschreiben wir in diesem Artikel.

Anwendungs- und Benutzerkontext

Wie im ersten Teil der Artikelreihe namens **eBay per VBA steuern: Zugangsdaten holen** (www.vbentwickler.de/****) beschrieben, gibt es zwei verschiedene Kontexte, in denen wir auf die Rest-API von eBay zugreifen können:

- **Anwendungskontext:** Dies erlaubt uns die Nutzung allgemein zugänglicher Informationen – also beispielsweise solche, die man auch auf der Webseite **ebay.com** als nicht angemeldeter Benutzer einsehen kann. Hier für reicht das im oben genannten Artikel ermittelte Anwendungstoken zur Authentifizierung aus.
- **Benutzerkontext:** Hiermit können wir solche Aktionen durchführen, die man auch als angemeldeter Benutzer erledigen kann. Dazu benötigen wir das Benutzertoken aus dem vorherigen Artikel. Damit wir die Benutzerfunktionen nutzen können, muss das Benutzertoken auch im Kontext der Anmeldung des entsprechenden Benutzers geholt werden.

Zu beachten ist, dass man je nach der verwendeten API und Funktion keine Authentifizierung, die Anwendungsauthentifizierung oder die Benutzerauthentifizierung benötigt. Wann welche Authentifizierung

benötigt wird, erfährt man am einfachsten in der API-Dokumentation beziehungsweise im API-Explorer.

Zurechtfinden im API-Dschungel

eBay hat einen umfangreichen Bestand an API-Funktionen. Einen ersten Überblick findet man auf der folgenden Webseite:

<https://developer.ebay.com/develop>

Von hier aus geht es weiter zu den absoluten Grundlagen, zum Entwickeln von Anwendungen zum Verkaufen oder Kaufen oder zu weiteren Themen.

Im Bereich **Kaufen** interessiert uns beispielsweise, wie wir eine Suche nach Artikeln per VBA realisieren können. Im Bereich **Verkaufen** wäre es spannend, eigene Artikel voll automatisiert per VBA bei eBay einstellen zu können. Oder auch herauszufinden, welche Artikel wir gerade bei eBay anbieten und in welcher Anzahl, um hier gegebenenfalls aufzustocken. Beide Bereiche bieten zahlreiche Funktionen bereit, wir wollen uns jedoch auf den Bereich des Verkaufens konzentrieren.

Über diesen können wir uns in den weiterführenden Artikeln des oben genannten Links einen ersten Überblick verschaffen. Diesen finden wir zum Zeitpunkt der Erstellung dieses Artikels unter dem folgenden Link:

eBay per VBA steuern: Zugangsdaten holen

Im ersten Teil einer Artikelreihe zum Thema Steuerung von eBay mit VBA zeigen wir, wie man einen Entwickler-Account anlegt, eine neue Anwendung bei eBay erstellt, grundlegende Zugriffsdaten holt und die für die Authentifizierung im Kontext eines bestimmten Benutzerkontos notwendigen Informationen holt – hier speziell das Authentifizierungstoken. Damit schaffen wir die Basis, um per VBA auf die Rest API von eBay zuzugreifen. In weiteren Artikeln beschreiben wir den Zugriff und wie wir verschiedene Operationen wie das Anlegen von Angeboten realisieren können.

Um eBay mit VBA zu steuern, brauchen wir als Erstes ein Konto im **ebay developers program**. Um dorthin zu gelangen, reicht eine Websuche nach den Begriffen **ebay developer**.

Für die Registrierung werden die folgenden Informationen benötigt, die wir in ein Formular wie in Bild 1 eintragen: Benutzername, Kennwort, E-Mail-Adresse und Telefonnummer. Nachdem wir diese eingetragen und die Schaltfläche **Join** betätigt haben, landen wir auf der nächsten Seite im Registrierungsprozess. Hier erhalten wir den Hinweis, dass eBay uns eine E-Mail zugesendet hat. Diese enthält einen Bestätigungslink, mit dem wir den Vorgang fortsetzen können.

Nachdem wir diesen betätigt haben, landen wir auf der Willkommen-Seite des Entwicklerprogramms und können uns dort mit unseren Daten anmelden (siehe Bild 2).

Die Freude währt in der Regel nur kurz, denn gegebenenfalls bekommen wir hier die Meldung, dass die Registrierung nun überprüft wird und in ca. 24 Stunden freigeschaltet wird.

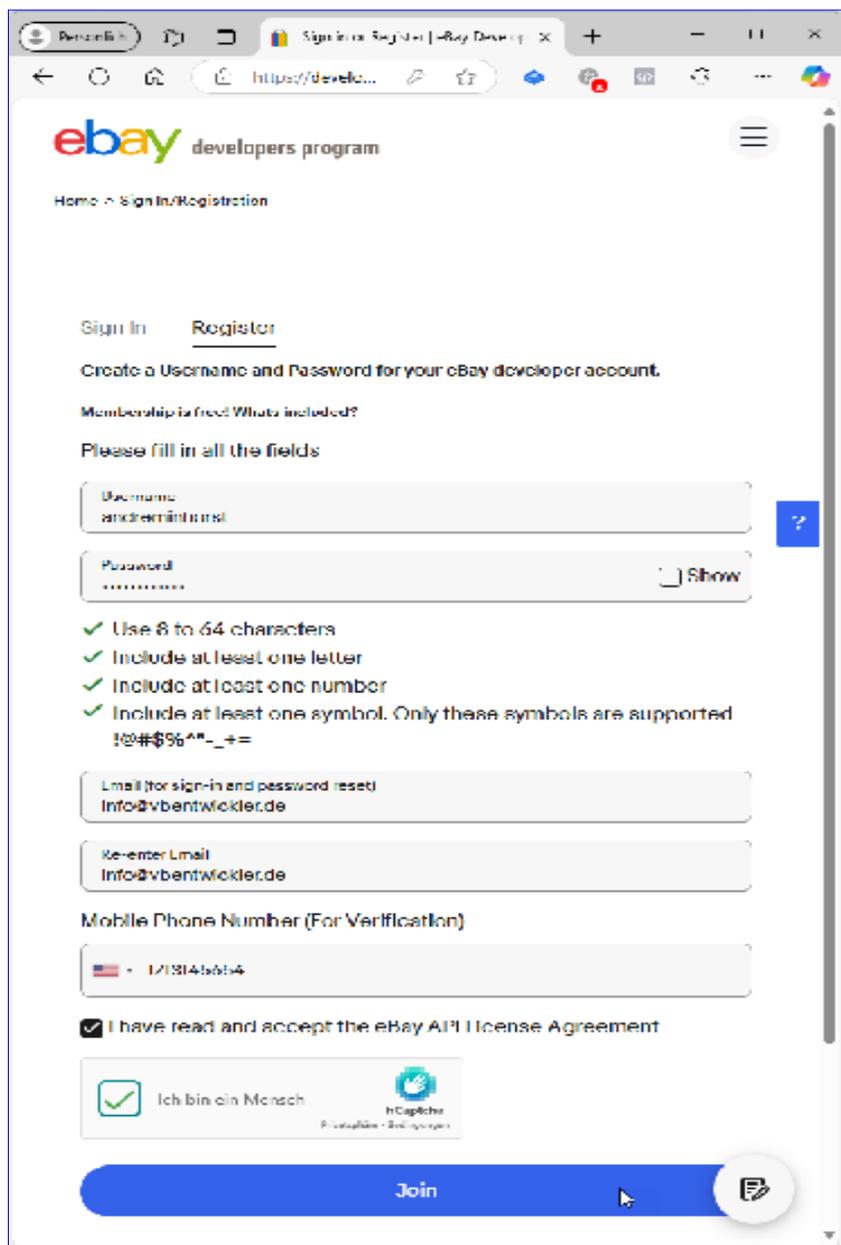


Bild 1: Registrieren für das Developer-Program von eBay

Office: Eingebaute Kontextmenübefehle recyceln

Wir haben bereits in einigen weiteren Artikel beschrieben, wie wir unseren Anwendungen in Word, Excel oder Access benutzerdefinierte Kontextmenü-Einträge oder sogar vollständige Kontextmenüs hinzufügen können. Was wird dort noch nicht berücksichtigt haben: Manchmal möchte man vielleicht ein eingebautes Element in einem benutzerdefinierten Kontextmenü oder gar ein vollständiges Menü nachbauen, um dann selbst genauer zu steuern, wann dieses angezeigt werden soll. Ein gutes Beispiel sind die Befehle für die Verwendung der Zwischenablage, also Ausschneiden, Kopieren und Einfügen. Es kann sinnvoll sein, diese einem benutzerdefinierten Kontextmenü hinzuzufügen. In diesem Artikel beschreiben wir, wie Du herausfindest, welche Informationen über das eingebaute Kontextmenü-Element benötigst und wie Du dieses in einem benutzerdefinierten Kontextmenü abbildest.

In den vorherigen Ausgaben haben wir bereits in einigen Artikeln die Programmierung von Kontextmenüs beschrieben:

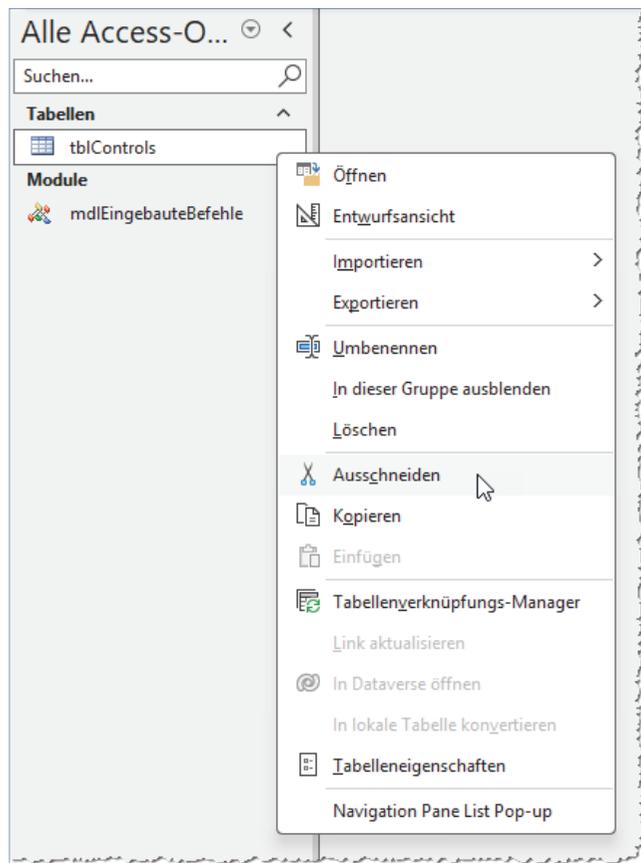


Bild 1: Beispiel für ein eingebautes Kontextmenü

- Menüs im VBA-Editor anpassen (www.vbentwickler.de/434)
- Menüs per VBA programmieren (www.vbentwickler.de/435)
- Kontextmenüs per VBA programmieren (www.vbentwickler.de/368)

Auch auf die Anpassung von Kontextmenüs in Outlook sind wir bereits eingegangen – siehe **Outlook: Kontextmenüs anpassen** (www.vbentwickler.de/369).

In diesem Artikel nun schauen wir uns im Detail an, wie wir die eingebauten Kontextmenü-Befehle in benutzerdefinierte Kontextmenüs einbauen können – zum Beispiel die aus Bild 1.

Dazu sind die folgenden Schritte nötig:

- Ermitteln von Informationen, mit denen wir die eingebauten Kontextmenü-Einträge identifizieren können
- Einbau der Einträge in ein benutzerdefiniertes Kontextmenü