AUSGABE 01/2024

VISUAL BASIC ENTWICKLER

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

IN DIESEM HEFT:

André Minhorst Verlag

Von Termin zu Termin

Termine verwalten ist ein wichtiger Aspekt von Tools wie Outlook. Es gibt jedoch heutzutage viele verschiedene Möglichkeiten, Termine zu verwalten. Und es gibt ebenfalls eine Reihe von Endgeräten, mit denen man die Termine einträgt, sich erinnern lässt oder Termine organisiert. Den guten, alten Kalender in Papierform klammern wir bewusst aus, da er sich nicht für eine einfache Synchronisierung anbietet. Für alle anderen ist es jedoch sinnvoll, Termine zwischen den verschiedenen Plattformen zu synchronisieren. Während wir in der Ausgabe 6/2023 einen ausführlichen Blick auf den Google Calender geworfen haben, vertiefen wir dieses Thema in der aktuellen Ausgabe und holen noch Outlook als weiteren Kalender hinzu.

Dabei starten wir im Artikel **Outlook: Kalender und Termine programmieren** ab Seite 4 mit den Grundlagen rund um die Programmierung von Kalendern und Terminen in Outlook. Hier schauen wir uns an, wie wir auf einen Outlook-Kalender und die enthaltenen Termine zugreifen und Termine anlegen, bearbeiten, löschen oder verschieben. Wir betrachten die verschiedenen Eigenschaften von Terminen und wie diese mit den in der Benutzeroberfläche von Outlook angezeigten Terminen korrelieren.

Termine unter Outlook stellen, wie die meisten Objekte in Microsoft Office, eigene Ereignisse bereit. Diese schauen wir uns im Artikel **Outlook: Ereignisse für Termine implementieren** ab Seite 15 an. Dabei sind nicht nur die beiden Ereignisse interessant, die beim Öffnen und Schließen von Terminen ausgelöst werden. Spannender für die Verwaltung von Terminen sind solche Ereignisse, wie sie von den Auflistungen ausgelöst werden, in denen die Termin-Elemente enthalten sind. Damit können wir Prozeduren implementieren, die beim Anlegen, Löschen oder Ändern von Terminen ausgelöst werden.

Bevor wir uns den Artikel zuwenden, die Outlook und den Google Calendar zusammenbringen, werfen wir unter dem Titel **Google-Authentifizierung mit OAuth2, Update** ab Seite 22 einen Blick auf einige Erweiterungen zum Artikel **Google-Authentifizierung mit OAuth2**, den wir im vorherigen Heft veröffentlicht haben. Hier geht es vor allem darum, wie wir das Token für den Zugriff auf den Google Calendar aktualisieren können, wenn es einmal abgelaufen ist.

Darauf aufbauend stellen wir im Artikel **Google Calendar per Rest-API programmieren, Teil 2** ab Seite 31 weitere Techniken für den Zugriff per VBA auf die Termine im Google Calendar vor. Wir zeigen, wie wir Termine anlegen, bestehende Termine abrufen, Termine aktualisieren und Termine absagen. Außerdem rufen wir komplette Terminlisten ab.

Damit kommen wir zur Verschmelzung des Outlook-Kalenders und des Google Calendars. Im Artikel **Termine von Outlook zum Google Calendar exportieren** ab Seite 42 zeigen wir, wie wir Termine aus Outlook in den Google Calendar schreiben kannst. Aber nicht nur das: Wir zeigen auch, wie Änderungen oder Löschvorgänge an Terminen in Outlook in den Google Calendar übertragen werden.

Das Know-how aus den vorangegangenen Artikel führen wir unter dem Titel **Outlook: Termine per COM-Add-In nach Google** ab Seite 49 zusammen. Hier erstellen wir ein COM-Add-In, mit dem wir Outlook um Ribboneinträge und Kontextmenü-Befehle erweitern, mit denen wir Termine in den Google Calendar tragen können. Außerdem sorge dieses Add-In nach dem Installieren dafür, dass neue, geänderte oder gelöschte Termine automatisch nach Google übettragen werden.

Nun viel Spaß beim Lesen!

Dein André Minhorst

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

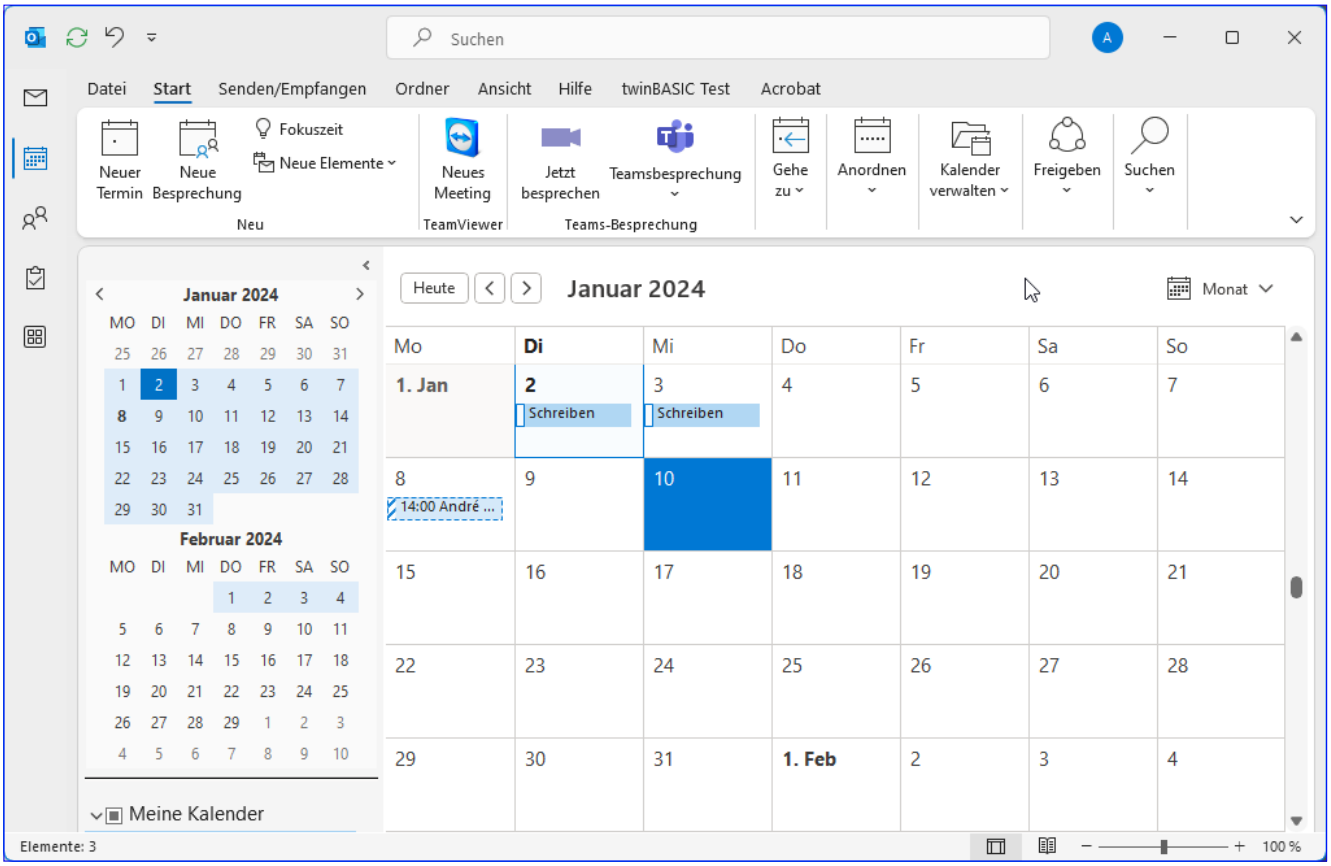

Bild 1: Der Outlook-Kalender in der Monatsansicht

enthaltenen Termine zugreifen. Dabei haben wir folgende Aktionen im Blick:

- Referenzieren von Kalendern
- Funktion zum Holen des Standardkalenders
- Durchlaufen und Referenzieren von Terminen
- Aktuell markierten Termin referenzieren
- Wichtige Eigenschaften von Terminen
- Eigenschaften für den aktuellen Termin ausgeben
- Details zu Erinnerungen
- Methoden von Terminen
- Ereignisse von Terminen
- Anlegen eines neuen Termins
- Auslesen bestimmter Termine
- Löschen vorhandener Termine
- Bearbeiten vorhandener Termine
- Details zu Zeitzonen

Wo programmieren wir Outlook?

Wir führen die nachfolgend vorgestellten Beispiele im VBA-Projekt von Outlook aus. Das heißt, dass wir Outlook öffnen und mit der Tastenkombination **Alt + F11** zum VBA-Editor wechseln. Hier legen wir für unsere Beispiele ein neues Modul an, beispielsweise namens **mdlKalenderUndTermine**. Mehr zum Ausführen von VBA-Prozeduren in Outlook haben wir im Artikel **Outlook: Codebeispiele ausprobieren** (**www.vbentwickler.de/313**) beschrieben.

Referenzieren von Kalendern

Unter Outlook finden wir den Standardkalender und gegebenenfalls noch weitere, freigegebene Kalender. Wir wollen uns in diesem Artikel auf den Standardkalender beziehen.

Dazu nutzen wir die Prozedur aus Listing 1. Hier deklarieren wir zunächst Objektvariablen für die Outlook-Anwendung selbst, für den MAPI-Namespace, der alle Informationen wie E-Mails, Kontakte, Termine et cetera enthält und eine **Folder**-Variable namens **objCalendar**. Warum der Datentyp **Folder**? Weil alle Elemente von Outlook in Foldern gespeichert sind. Es gibt kein spezielles Objekt etwa mit dem Typ **Calendar**.

```
Public Sub ReferenceCalendar()
    Dim objOutlook As Outlook.Application
    Dim objMAPI As Outlook.NameSpace
    Dim objCalendar As Outlook.Folder
    Set objOutlook = New Outlook.Application
    Set objMAPI = objOutlook.GetNamespace("MAPI")
    Set objCalendar = objMAPI.GetDefaultFolder(olFolderCalendar)
    Debug.Print "Calender name: " & objCalendar.Name
    Debug.Print "Appointments: " & objCalendar.Items.count
End Sub
```

Listing 1: Referenzieren des Standardkalenders

Danach referenzieren wir zuerst die Outlook-Anwendung. Mit **New Outlook.Application** greifen wir entweder auf die laufende Outlook-Anwendung zu oder erzeugen eine neue Instanz.

In der aktuellen Umgebung, also im VBA-Editor mit dem VBA-Projekt von Outlook, sollten wir allerdings mit **Application** auf die aktuelle Outlook-Instanz zugreifen – hier könnte die Verwendung von **New Outlook.**

Application zu Problemen führen. Aber damit wir den Code ohne Umschweife in andere VBA-Projekte beispielsweise von Excel oder Access übertragen oder diesen in twinBASIC oder ähnlichen Entwicklungsumgebungen nutzen können, greifen wir hier im Text explizit auf die jeweiligen Objekte zu.

Die nächste Anweisung referenziert den MAPI-Namespace mit der Variablen **objMAPI**. Dann greifen wir mit der **GetDefaultFolder** und dem Parameter **olFolderCalendar** auf den **Kalender**-Ordner von Outlook zu und speichern den Verweis in **objCalendar**. Dann geben wir den Namen des Folders aus, der in der deutschen Version Kalender heißt, und ermitteln noch die Anzahl der im Kalenderordner enthaltenen Elemente.

Funktion zum Holen des Standardkalenders

Diese Schritte sind immer nötig, wenn wir auf Elemente des Kalender-Ordners zugreifen wollen. Damit wir nicht immer die gleichen Anweisungen in die folgenden Beispielprozeduren und -funktionen schreiben müssen, programmieren wir die Prozedur **ReferenceCalendar** in eine Funktion um, die wir **GetDefaultCalendar** nennen. Diese Funktion soll einen Rückgabewert des Datentyps **Folder** liefern. Sie enthält weitestgehend die gleichen Anweisungen wie die Prozedur, von der sie abgeleitet wurde – am Ende gibt sie jedoch noch den Verweis auf den Kalender als Funktionswert zurück:

```
Public Function GetDefaultCalendar() As Folder
    Dim objOutlook As Outlook.Application
    Dim objMAPI As Outlook.NameSpace
    Dim objCalendar As Outlook.Folder
    Set objOutlook = New Outlook.Application
    Set objMAPI = objOutlook.GetNamespace("MAPI")
    Set objCalendar = _
        objMAPI.GetDefaultFolder(olFolderCalendar)
    Set GetDefaultCalendar = objCalendar
End Function
```

Durchlaufen und Referenzieren von Terminen

In der folgenden Prozedur durchlaufen wir alle Termine des aktuellen Kalenders. Das können, wenn Du den Kalender intensiv nutzt, viele Einträge sein. Du kannst vorsichtshalber innerhalb der Schleife die Anweisung **DoEvents** einbauen, damit Du zwischenzeitlich abbrechen kannst, wenn es zu lange dauert.

Die Prozedur **ListAppointments** deklariert zwei Objekte zum Referenzieren der Kalender-Einträge. Eigentlich wollen wir auf die **AppointmentItem**-Elemente zugreifen, aber der Kalender-Ordner kann auch Elemente anderer Typen enthalten. Deshalb referenzieren wir jedes Element erst einmal mit der Variablen **objItem** mit dem allgemeinen Datentyp **Object**.

Zuvor lesen wir allerdings den Kalender-Ordner mit der zuvor beschriebenen Funktion **GetDefaultCalendar** ein. Danach durchlaufen wir in einer **For...Each**-Schleife alle Elemente der Auflistung **objCalendar.Items**. In der Schleife untersuchen wir in einer **Select Case**-Bedingung den Wert von **TypeName(objItem)**, was beispielsweise die Zeichenkette **AppointmentItem** liefert. Genau diesen Fall wollen wir untersuchen und weisen den Inhalt von **objItem** der Variablen **objAppointmentItem** zu. Dann geben wir beispielhaft den Betreff des Termins aus:

```
Public Sub ListAppointments()
    Dim objCalendar As Outlook.Folder
    Dim objAppointmentItem As Outlook.AppointmentItem
    Dim objItem As Object
    Set objCalendar = GetDefaultCalendar
    For Each objItem In objCalendar.Items
        Select Case TypeName(objItem)
            Case "AppointmentItem"
                Set objAppointmentItem = objItem
                With objAppointmentItem
                    Debug.Print .Subject
                End With
```

```
        End Select
    Next objItem
End Sub
```

Aktuell markierten Termin referenzieren

Gerade beim Experimentieren mit den Eigenschaften von Kalenderelementen kann es hilfreich sein, den aktuell markierten Eintrag im Kalender zu referenzieren.

Das erledigen wir mit der folgenden Funktion, die ein Objekt des Typs **Outlook.AppointmentItem** zurückliefern soll. Sie referenziert erst die Outlook-Anwendung und dann den aktiven Explorer (sprich: das Hauptfenster).

Schließlich prüft sie, ob der Wert der Eigenschaft **Count** der Auflistung **Selection** ungleich **0** ist. In diesem Fall referenziert sie das erste markierte Element mit der Variablen **objAppointmentItem**, das sie dann auch als Funktionswert zurückliefert:

```
Public Function GetSelectedAppointmentItem() _
        As Outlook.AppointmentItem
    Dim objOutlook As Outlook.Application
    Dim objExplorer As Outlook.Explorer
    Dim objAppointmentItem As AppointmentItem
    Set objOutlook = New Outlook.Application
    Set objExplorer = objOutlook.ActiveExplorer
    If Not objExplorer.Selection.count = 0 Then
        Set objAppointmentItem = _
            objExplorer.Selection.Item(1)
    End If
    Set GetSelectedAppointmentItem = objAppointmentItem
End Function
```

Wichtige Eigenschaften von Terminen

Bevor wir nun genauer auf die verschiedenen Aktionen eingehen, die wir mit Terminen erledigen können, sehen wir uns einige wichtige Eigenschaften von Terminen an. In Bild 2 haben wir markiert, wo wir einige der Eigenschaften in einem Outlook-Termin in der Benutzeroberfläche finden.

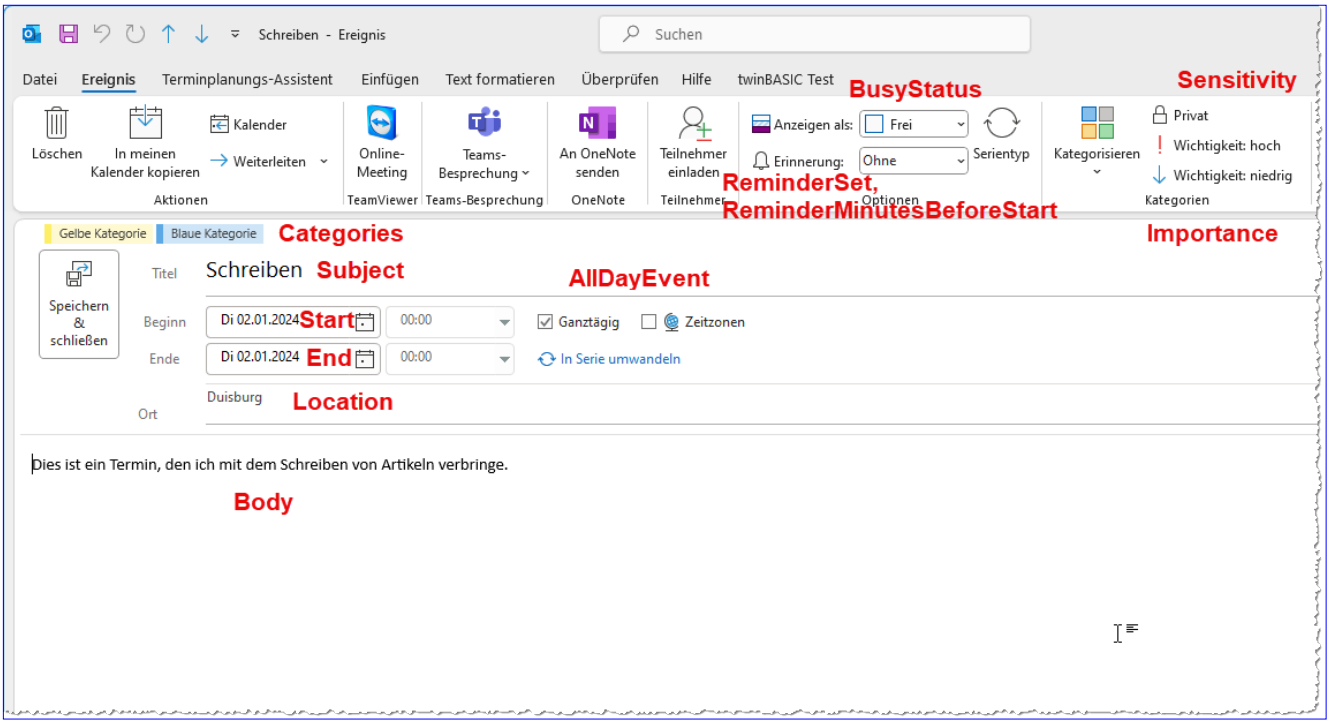

Bild 2: Ein Termin im Inspektor.

Und hier sind die Eigenschaften:

- **AllDayEvent**: Gibt an, ob es sich um einen ganztätigen Termin handelt.
- **BillingInformation**: Normalerweise nicht genutzte Eigenschaft, die man mit benutzerdefinierten Werten füllen kann.
- **Body**: Inhalt des Termins.
- **BodyFormat**: Format des Inhalts, kann die Werte **olFormatHTML**, **olFormatPlain** oder **olFormatRichText** annehmen.
- **BusyStatus**: Zeigt an, welchen Status der Bericht anzeigen soll – zum Beispiel **Frei** (**0 – olFree**), **An anderem Ort tätig** (**4 – olWorkingElsewhere**), **Mit Vorbehalt** (**1 – olTentative**), **Gebucht** (**2 – olBusy**) oder **Außer Haus** (**3 – olOutOfOffice**) (siehe Bild 3)
- **Categories**: Kategorien des Termins. Liefert eine Liste der Kategorien, denen der Termin zugeordnet ist.
- **Duration**: Dauer des Termins in Minuten
- **IsRecurring**: Gibt an, ob es sich um einen Serientermin handelt.
- **GetOrganizer**: Liefert den Organisator des Termins.
- **Location**: Gibt den Ort des Termins an.
- **ReminderMinutesBeforeStart**: Gibt an, wie viele Minuten vor einem Termin eine Erinnerung erfolgen soll.
- **ReminderOverrideDefault**: Gibt an, ob die Standardeinstellungen des Benutzers für eine Erinnerung überschrieben werden sollen.
- **ReminderPlaySound**: Gibt an, ob mit der Erinnerung ein Ton abgespielt werden soll.
- **ReminderSet**: Gibt an, ob eine Erinnerung ausgegeben werden soll.
- **ReminderSoundFile**: Gibt den Pfad zu einer Sounddatei an, die bei Eintreten der Erinnerung abgespielt werden soll.
- **RTFBody**: Inhalt des Termins im RTF-Format

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 Calender 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.

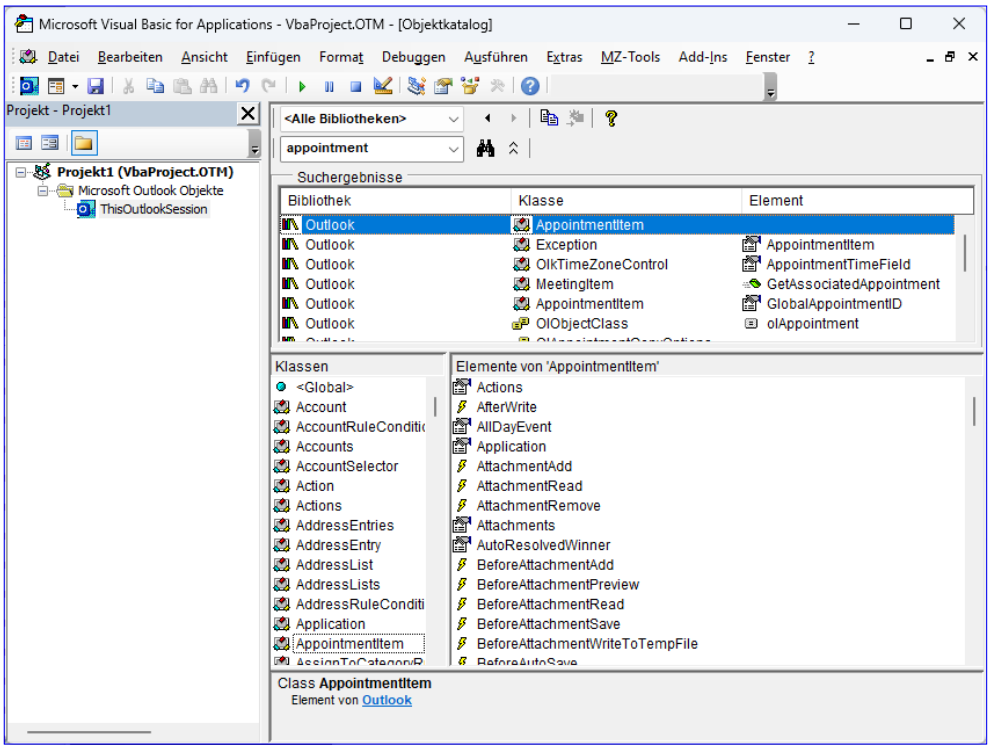

Bild 1: Ereignisse des Termins im Objektkatalog

Ereignisse ausprobieren

Zum Annähern an brauchbare Ereignisse bietet es sich an, diese einmal zu implementieren und sie mit Haltepunkten oder der **Stop**-Anweisung zu versehen. Dann sehen wir beim Arbeiten mit einem **AppointmentItem**-Element direkt, wann welche Ereignisse ausgelöst werden.

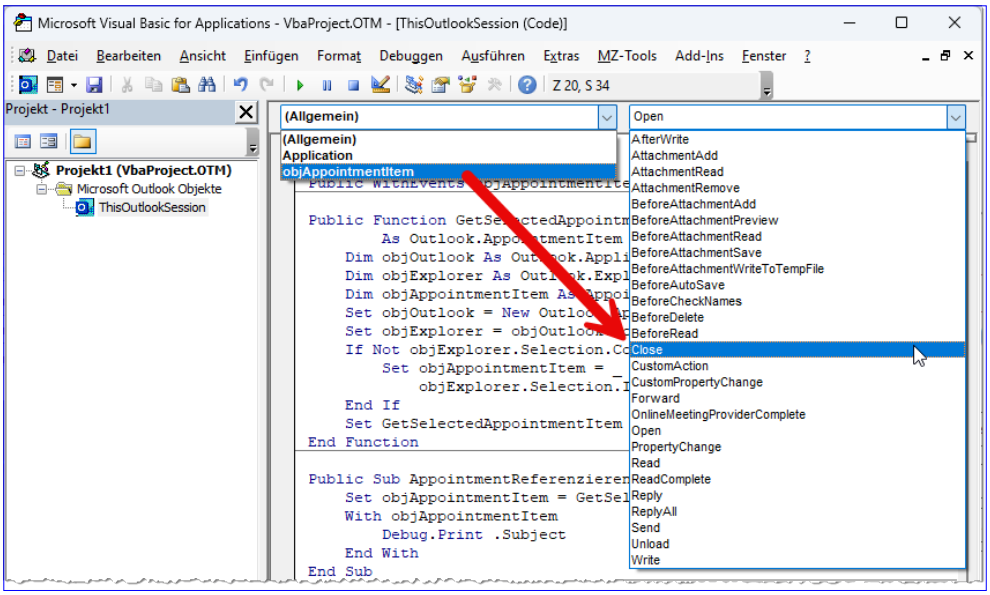

Bild 2: Implementieren einer Ereignisprozedur

Um die Ereignisse zu implementieren, benötigen wir allerdings erst einmal eine Objektvariable, die wir mit einem Verweis auf das **AppointmentItem**-Element füllen und für die wir die Ereignisse implementieren können. Diese deklarieren wir im Modul **ThisOutlookSession** wie folgt:

```
Public WithEvents objAppointmentItem _
    As Outlook.AppointmentItem
```

Außerdem verwenden wir die im Artikel **Outlook: Kalender und Termine programmieren** (**www.vbentwickler.de/415**) vorgestellte Funktion **GetSelectedAppointmentItem**, mit der wir den aktuellen Termin referenzieren.

Die folgende Funktion weist der Variablen **objAppointmentItem** den aktuell markierten Termin zu:

```
Public Sub ReferenceSelectedAppointmentItem()
    Set objAppointmentItem = GetSelectedAppointmentItem
End Sub
```

Danach können wir im VBA-Editor im linken Kombinationsfeld die Variable **objAppointmentItem** auswählen und erhalten im rechten Kombinationsfeld alle verfügbaren Ereignisse (siehe Bild 2). Klicken wir eines davon an, wird die entsprechende Ereignisprozedur zum Modul hinzugefügt.

Wir haben zunächst einfach jeder dieser Ereignisprozeduren die **Stop**-Anweisung hinzugefügt (siehe Bild 3). Beim Öffnen und beim Schließen werden diese beiden Ereignisprozeduren ausgelöst:

- **objAppointmentItem_Open**
- **objAppointmentItem_Close**

Bei anderen Aktionen traten jedoch gleich mehrere Ereignisse auf, daher sind wir von der **Stop**-Anweisung abgegangen und haben **Debug.Print**-Anweisungen hinzufügt, mit denen wir teilweise noch die Parameter ausgeben – zum Beispiel so:

```
Private Sub objAppointmentItem_PropertyChange(ByVal _
        Name As String)
    Debug.Print "objAppointmentItem_PropertyChange", Name
End Sub
```

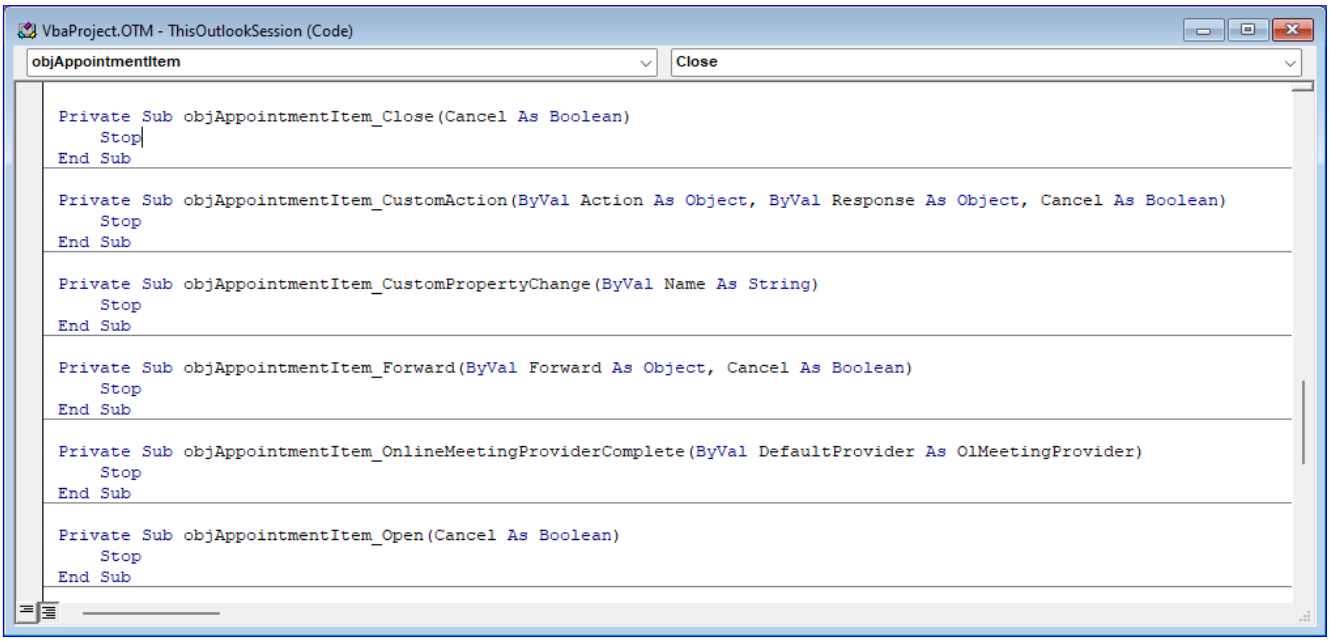

Bild 3: Implementierte Ereignisprozeduren eines **AppointmentItem**-Elements

Damit erhalten wir allein beim Ändern der Startzeit eines geöffneten Termins von 9:00 Uhr auf 9:30 Uhr wie in Bild 4 gleich sechs Ereignisse. Diese interessieren uns allerdings nicht, wenn wir beispielsweise die Änderungen an einen Google Calendar übertragen wollen. Dann benötigen wir nur zu einem bestimmten mindestens einer Eigenschaft geändert haben. Zum Glück gibt es noch andere Möglichkeiten, die noch einen weiteren Vorteil bieten: Wir haben oben nun einen Termin referenziert und seine Ereignisse imple-

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).

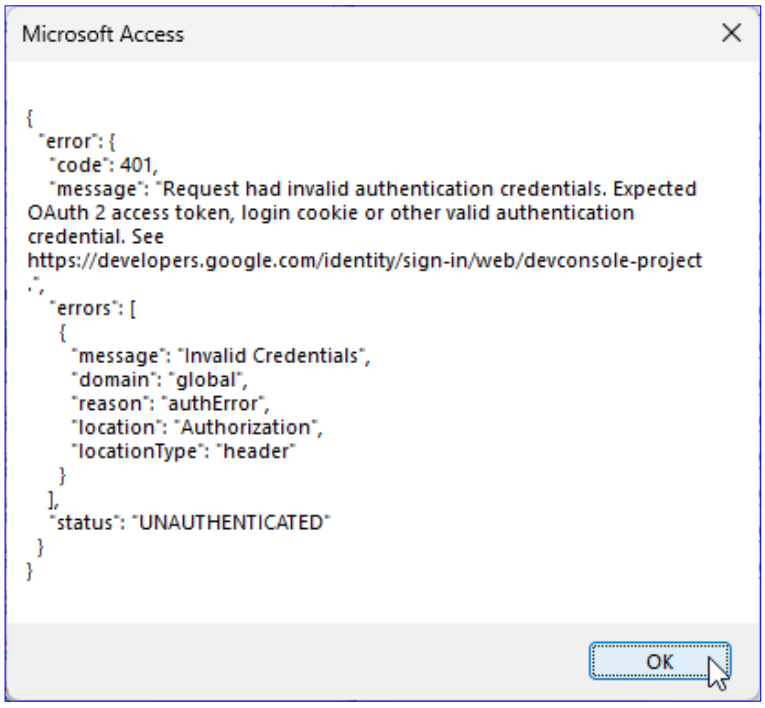

Bild 1: Fehlermeldung bei fehlgeschlagener Authentifizierung

Andererseits erlaubten uns die in den beiden Projekten verwendeten Versuche der Autho-risierung 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.

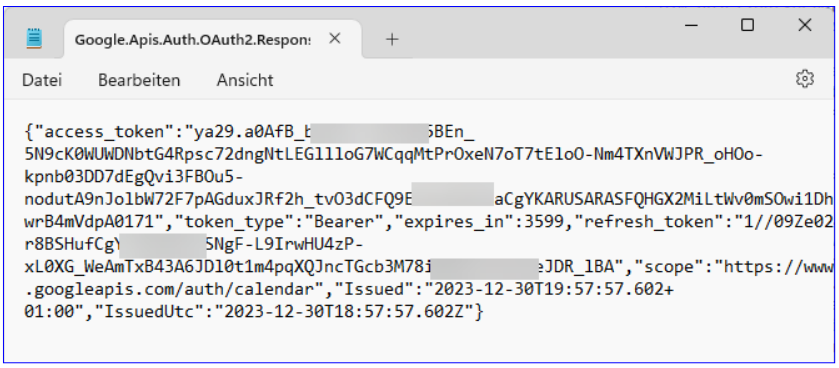

Bild 2: Die Datei mit den aktuellen Authentifizierungsdaten

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

dialog aus Bild 3 erneut zu authentifizieren.

Danach können wir wieder für eine Stunde mit diesen Zugangsdaten arbeiten, danach müssen wir uns erneut authentifizieren.

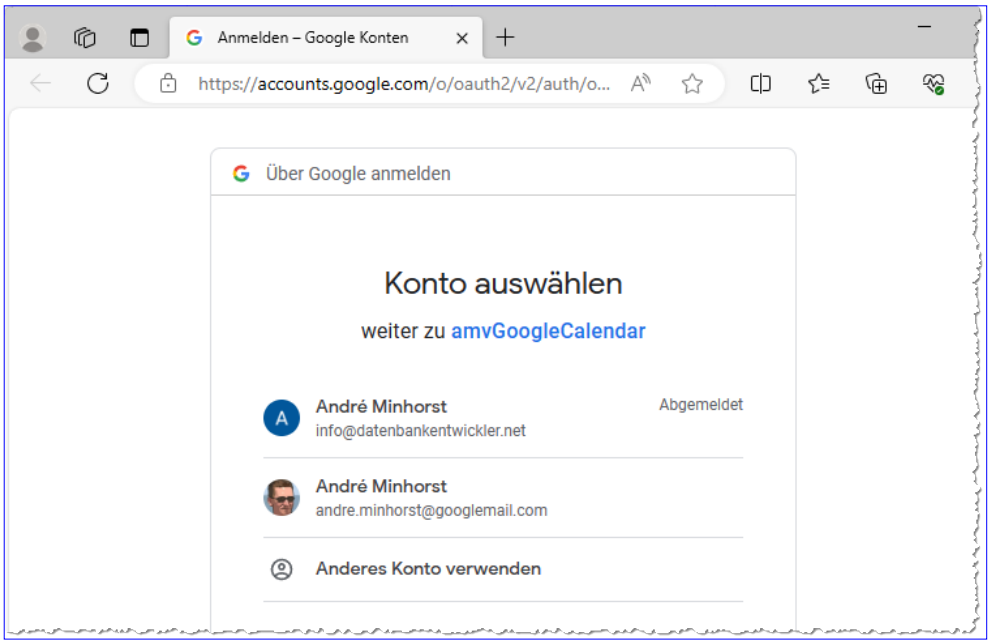

Bild 3: Erneute Authentifizierung nach dem Löschen der Datei

Herausfinden, ob das Konto noch verbunden ist

Wenn wir das Konto so ausgewählt und den Zugriff gestattet haben, können wir mit den VBA-Routinen, die wir zum Beispiel im Artikel **Google Calendar per Rest-API programmieren** (**www.vbentwickler.de/410**) vorgestellt haben, auf den Kalender des Benutzers zugreifen, in dessen Kontext wir den Zugriff freigeschaltet haben.

Manchmal gelingt dies nicht, und wir können dann die Verbindung für dieses Konto zuerst löschen und dann wieder herstellen. Dazu müssen wir erst einmal wissen, wo man einsieht, ob für dieses Konto überhaupt noch eine Verbindung besteht.

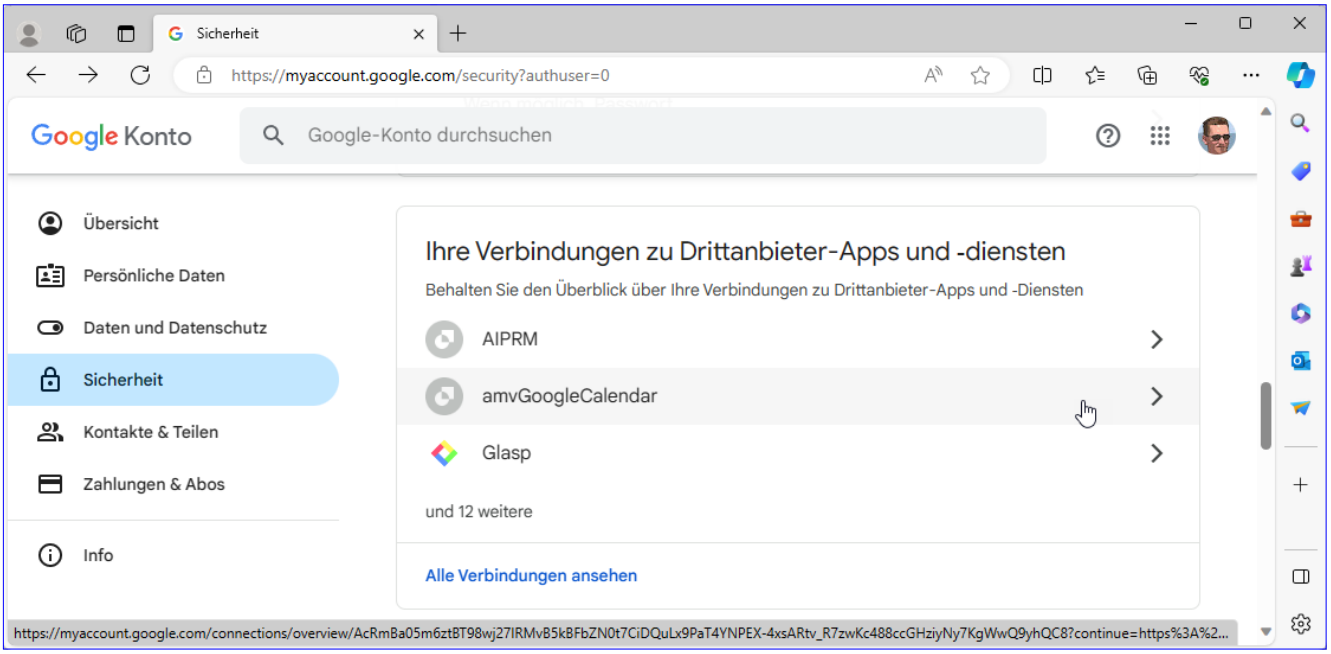

Bild 4: Der Bereich **Sicherheit** im Google-Benutzerkonto

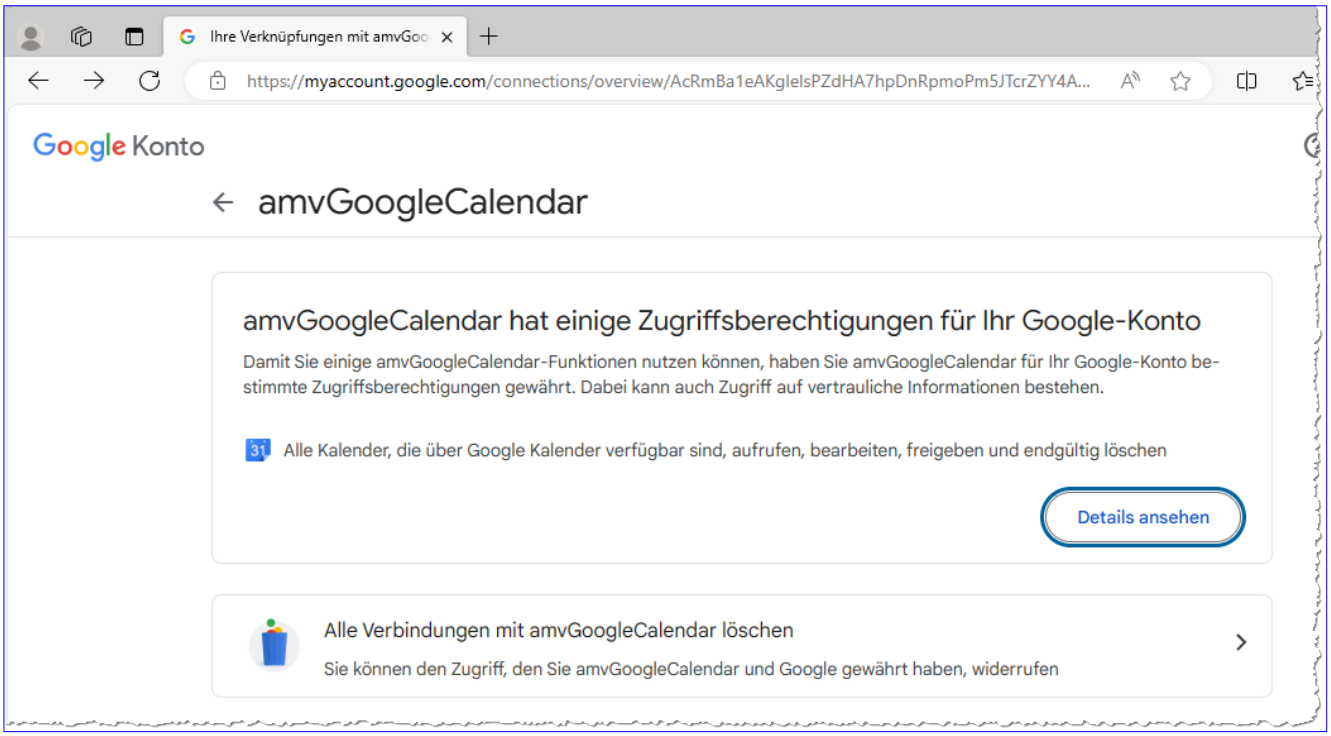

Bild 5: Details zur App **amvGoogleCalendar**

Dazu geben wir im Browser die folgende URL ein:

```
https://myaccount.google.com/
```

Auf der nun erscheinenden Seite wählen wir links den Bereich **Sicherheit** aus und scrollen dann nach unten zu **Ihre Verbindungen zu Drittanbieter-Apps und -diensten** (siehe Bild 4).

Hier erscheint bereits der Eintrag namens **amvGoogleCalender**, den wir untersuchen wollen. Sollte dieser nicht direkt sichtbar sein, klicken wir unten auf **Alle Verbindungen ansehen**.

Danach klicken wir auf den Eintrag **amvGoogleCalendar**. Es erscheint die Anzeige aus Bild 5. Hier sehen wir zwei Möglichkeiten:

- Anzeige von Details über die Zugriffsberechtigungen
- Löschen der Verbindungen mit der App **amvGoogleCalendar**

Wenn wir die erste Möglichkeit nutzen und auf die Schaltfläche **Details ansehen** klicken, finden wir die Seite aus Bild 6 vor. Hier sehen wir wichtige Informationen, zum Beispiel wann der Zugriff gewährt wurde – also wann das Access-Token freigeschaltet wurde. Wir können noch einen Bereich aufklappen, der uns anzeigt, welche Berechtigungen wir der App genau freigeben. Wir können aber auch hier auf Zugriff entfernen klicken, was dazu führt, dass eine Meldung mit den Konsequenzen angezeigt wird. In diesem Fall werden alle Berechtigungen der App für unseren Kalender entzogen. Wir können dies nur wiederherstellen, indem wir das Access-Token erneut generieren.

Entfernen wir die Zugriffsrechte für die App und versuchen anschließend, mit den VBA-Prozeduren auf den Google-Kalender zuzugreifen, erhalten wir wieder

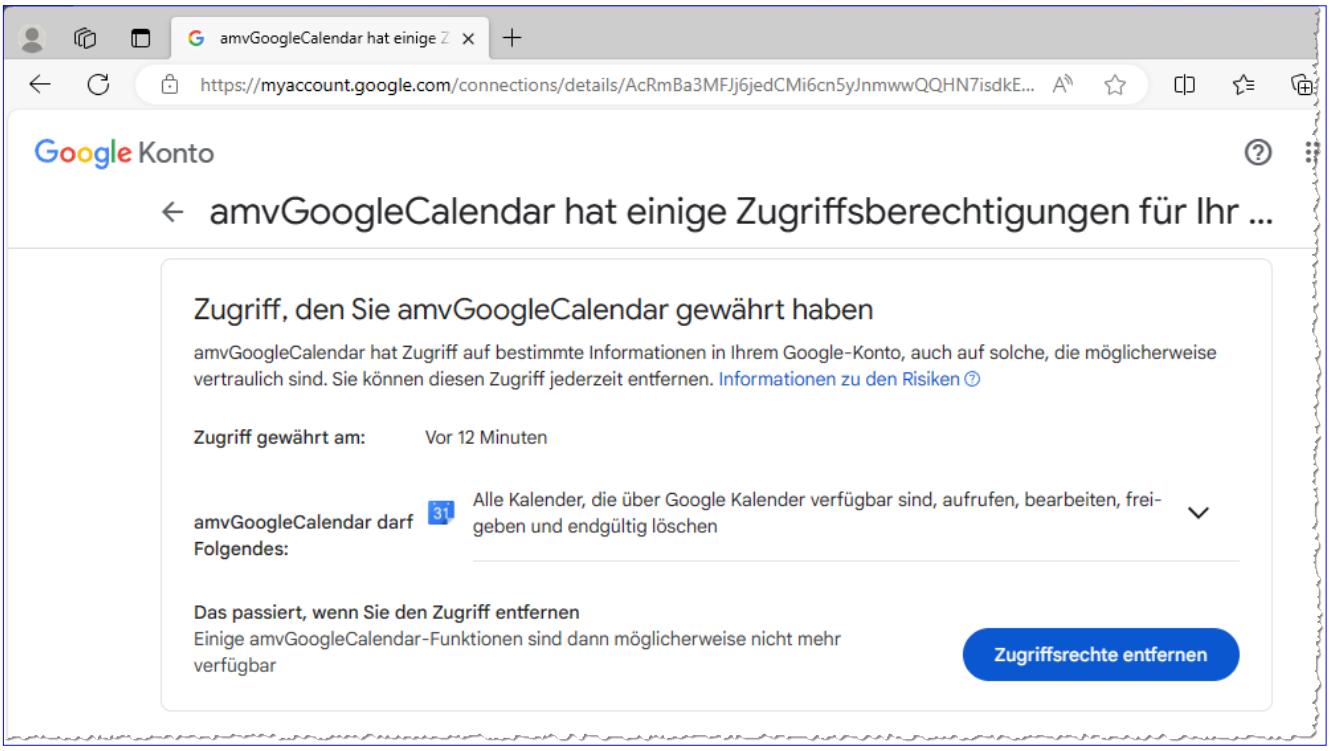

Bild 6: Details über die Zugriffsberechtigungen

die Antwort, dass wir mit ungültigen Credentials zugegriffen haben.

Erneut anmelden

Rufen wir nun beispielsweise die Funktion **GetTokens** aus der VBA-Lösung auf, läuft

Apis.Auth.OAuth2.Responses.TokenResponse-user noch im Ordner **C:\Users\User\AppData\Roaming\Google.Apis.Auth** gespeichert ist, nimmt die DLL an, das dies noch gültige Zugangsdaten sind.

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 Aktualisieren 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 **colLavendel** 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 ganztätigen Termin vor, sodass wir diese Notation verwenden.

In der Benutzeroberfläche sehen wir anschließend allerdings einen Ganztagestermin (siehe Bild 1). Wenn die Funktion **HTTPRequest** den Wert **True** zurückliefert, lesen wir aus dem mit **strResponse** gelieferten JSON-Dokument den Wert **id** aus und tragen diesen für den Parameter **strEventID** ein. Diesen kann man dann nach dem Aufruf verarbeiten, wenn dies nötig ist.

```
Public Function InsertEvent(strToken As String, strCalendarID As String, strSummary As String, strDescription _
        As String, datStart As Date, datEnd As Date, Optional bolAllDayEvent As Boolean, Optional strStartTimeZone _
        As String = "Europe/Berlin", Optional strEndTimeZone As String = "Europe/Berlin", Optional intColor _
        As eColor = colLavendel, Optional strResponse As String, Optional ByRef strEventID As String) As Boolean
    Dim strURL As String, strMethod As String, strAuthorization As String, objJSON As Object
    Dim strJSON As String, dictMain As Dictionary, dictStart As Dictionary, dictEnd As Dictionary
    Set dictMain = New Dictionary
    Set dictStart = New Dictionary
    Set dictEnd = New Dictionary
    dictMain.Add "start", dictStart
    dictMain.Add "end", dictEnd
    If Not bolAllDayEvent Then
        dictStart.Add "dateTime", Format(datStart, "yyyy-mm-ddThh:nn:ss")
        dictEnd.Add "dateTime", Format(datEnd, "yyyy-mm-ddThh:nn:ss")
    Else
        datStart = Int(datStart)
        datEnd = datStart + 1
        dictStart.Add "dateTime", Format(datStart, "yyyy-mm-ddThh:nn:ss")
        dictEnd.Add "dateTime", Format(datEnd, "yyyy-mm-ddThh:nn:ss")
    End If
    dictStart.Add "timeZone", strStartTimeZone
    dictEnd.Add "timeZone", strEndTimeZone
    dictMain.Add "summary", strSummary
    dictMain.Add "description", strDescription
    dictMain.Add "colorId", intColor
    strJSON = ConvertToJson(dictMain)
    strURL = cStrURLBase & "/calendars/" & strCalendarID & "/events"
    strMethod = "POST"
    strAuthorization = "Bearer " & strToken
    If RefreshAccessToken("InsertEvent") Then
        Select Case HTTPRequest(strURL, strAuthorization, strMethod, , strJSON, strResponse)
            Case 200
                InsertEvent = True
                Set objJSON = ParseJson(strResponse)
                strEventID = objJSON.Item("id")
            Case Else
                MsgBox strResponse
        End Select
    Else
        MsgBox "Access-Token ungültig oder nicht aktualisierbar."
    End If
End Function
```

Listing 1: Neue Version der Funktion **InsertEvent**

Rest-API-Funktion zum Abrufen eines Termins per EventID

Bei den im ersten Teil dieses Artikels beschriebenen Funktionen können wir die kompletten Informationen zu einem Termin abrufen. In den Beispielen speichern wir diese Daten in einer Access-Tabelle, darunter auch die **EventID**, ein eindeutiges Merkmal eines Google Calendar-Termins. Damit können wir wiederum gezielt erneut auf einen solchen Termin zugreifen – beispielsweise um den aktuellen Stand abzufragen, denn ein Termin kann ja auch durch den Benutzer in Google geändert werden.

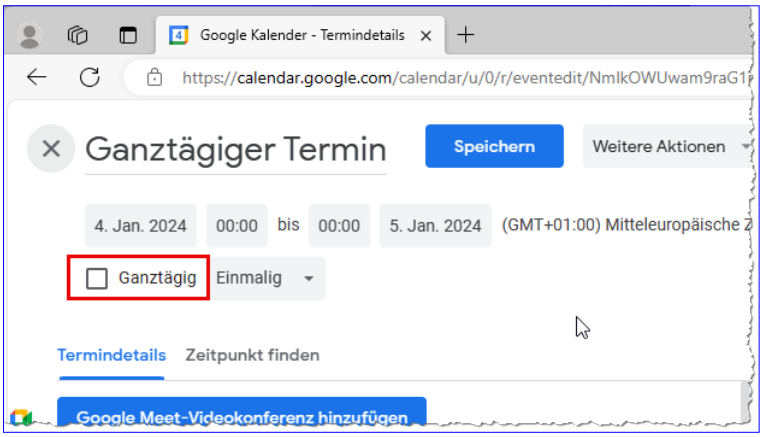

Bild 1: Ein ganztägiger Termin im Google Calendar

Dies erledigen wir mit der Funktion **GetEventByID** (siehe Listing 2). Diese erwartet folgende Parameter:

- **strToken**: Access-Token für den Zugriff auf den Google Calendar
- **strCalendarID**: Bezeichnung des Kalenders, in der Regel die Google-E-Mail-Adresse
- **strEventID**: ID des zu ermittelnden Termins
- **strJSON**: Parameter, der das Ergebnis der Abfrage entgegennimmt

```
Public Function GetEventByID(strToken As String, strCalendarID As String, strEventID As String, Optional strJSON _
        As String) As Boolean
    Dim strURL As String
    Dim strMethod As String
    Dim strResponse As String
    Dim strAuthorization As String
    strURL = cStrURLBase & "/calendars/" & strCalendarID & "/events/" & strEventID
    strMethod = "GET"
    strAuthorization = "Bearer " & strToken
    If RefreshAccessToken("GetEventByID") Then
        Select Case HTTPRequest(strURL, strAuthorization, strMethod, , , strResponse)
            Case 200
                strJSON = strResponse
                GetEventByID = True
            Case Else
                GetEventByID = False
        End Select
    Else
        MsgBox "Access-Token ungültig oder nicht aktualisierbar."
    End If
End Function
```

Listing 2: Funktion zum Einlesen eines Termins von Google

Die Funktion stellt die URL für den Zugriff auf die Rest-API zusammen, die beispielsweise so aussieht:

```
https://www.googleapis.com/calendar/v3/calendars/andre.
minhorst@googlemail.com/events/36kis3n1kt1sc5ch82a54k4u5k
```

Als Methode verwenden wir **GET**, und mit der Variablen **strAuthorization** übergeben wir das Token an die Funktion, die den eigentlichen Request ausführt. Zuvor rufen wir noch die Prozedur **CheckToken** aus, welche die Gültigkeit des Tokens prüft. Ist dieses ungültig, erfolgt erst gar kein Zugriff auf die Rest-API und es erscheint eine Meldung. Danach erfolgt der Aufruf der Funktion **HTTPRequest**. Das Ergebnis ist ein Status. Lautet dieser **200**, war der Zugriff erfolgreich und wir geben die Antwort über den Rückgabeparameter **strJSON** zurück. Außerdem stellen wir den Rückgabewert der Funktion auf **True** ein.

Diese Funktion rufen wir beispielsweise wie folgt auf:

```
Public Sub Test_GetEventByID()
    Dim strToken As String
    Dim strCalendarID As String
    Dim strEventID As String
    Dim strJSON As String
    Dim objJSON As Object
    strToken = GetAppSetting("AccessToken")
    strCalendarID = GetAppSetting("CalendarID")
    strEventID = "36kis3n1kt1sc5ch82a54k4u5k"
    If GetEventByID(strToken, strCalendarID, strEventID, _
            strJSON) = True Then
        Set objJSON = ParseJson(strJSON)
        Debug.Print GetJSONDOM(strJSON, True)
    End If
End Sub
```

Die beiden Parameter **strToken** und **strCalendarID** lesen wir aus der Registry aus, die **EventID** haben wir hier fest angegeben. War der Aufruf erfolgreich, lassen wir uns das Zugriffsmodell für das JSON-Dokument im Direktbereich ausgeben. Hier können wir uns dann die Elemente heraussuchen, die wir in der aufrufenden Prozedur weiterverarbeiten wollen (siehe Bild 2).

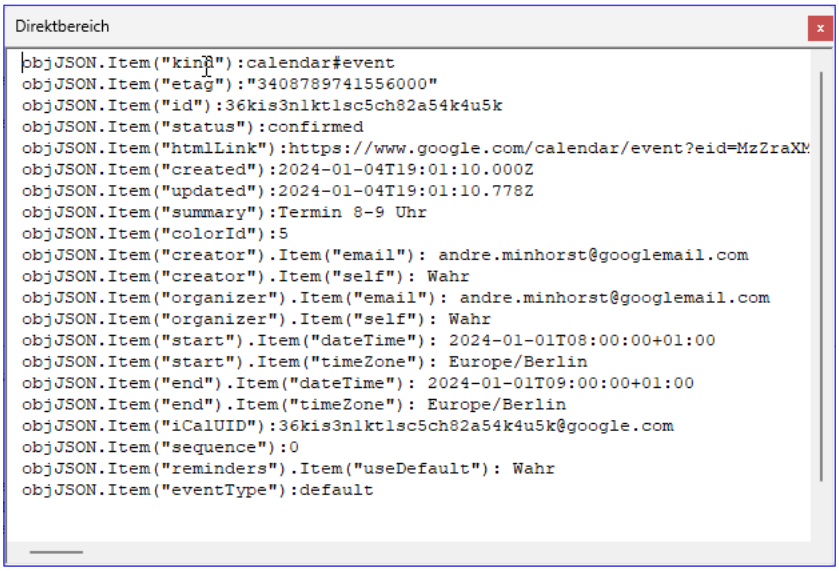

Bild 2: Ausdrücke zum Auslesen von Termineigenschaften

Rest-API-Funktion zum Aktualisieren eines Kalendereintrags

Wenn wir einen vorhandenen Termin im Google Calendar aktualisieren wollen, haben wir zwei Möglichkeiten. Die erste ist die Patch-Methode. Damit können wir gezielt einzelne Eigenschaften anpassen. Mehr dazu liest Du hier:

```
https://developers.google.com/calendar/
api/v3/reference/events/patch?hl=en
```

Bei dieser Methode werden nur die Eigenschaften des Termins aktualisiert, die wir im Aufruf der Rest-API an Google übergeben. Die übrigen Eigenschaftswerte werden beibehalten. Die Alternative ist das Update. Damit aktualisieren wir den kompletten Termin:

```
https://developers.google.com/calendar/
api/v3/reference/events/update?hl=en
```

```
Public Function UpdateEvent(strToken As String, strCalendarID As String, strSummary As String, strDescription _
        As String, datStart As Date, datEnd As Date, bolAllDayEvent, Optional strStartTimeZone _
        As String = "Europe/Berlin", Optional strEndTimeZone As String = "Europe/Berlin", Optional _
        intColor As eColor = colLavendel, Optional strResponse As String, Optional ByRef strEventID As String) _
        As Boolean
    '... Identisch mit InsertEvent
        strURL = cStrURLBase & "/calendars/" & strCalendarID & "/events/" & strEventID
        strMethod = "PUT"
        strAuthorization = "Bearer " & strToken
        If RefreshAccessToken("UpdateEvent") Then
            Select Case HTTPRequest(strURL, strAuthorization, strMethod, , strJSON, strResponse)
                Case 200
                    UpdateEvent = True
                    Set objJSON = ParseJson(strResponse)
                    strEventID = objJSON.Item("id")
                Case 404
                    MsgBox "Der Termin mit der EventID '" & strEventID & "' konnte nicht gefunden werden."
                Case Else
                    MsgBox strResponse
            End Select
        Else
            MsgBox "Access-Token ungültig oder nicht aktualisierbar."
        End If
End Function
```

Listing 3: Funktion zum Aktualisieren eines Termins

Damit werden alle Eigenschaften aktualisiert. Wir machen es uns einfach und aktualisieren einfach den kompletten Termin. Die dazu verwendete Funktion **UpdateEvent** ist der Funktion **InsertEvent** sehr ähn- EventID angeben, die es nicht gibt, und gibt eine entsprechende Meldung aus.

»Löschen« von Terminen im Google

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 Kontextmenüs 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 Calender 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

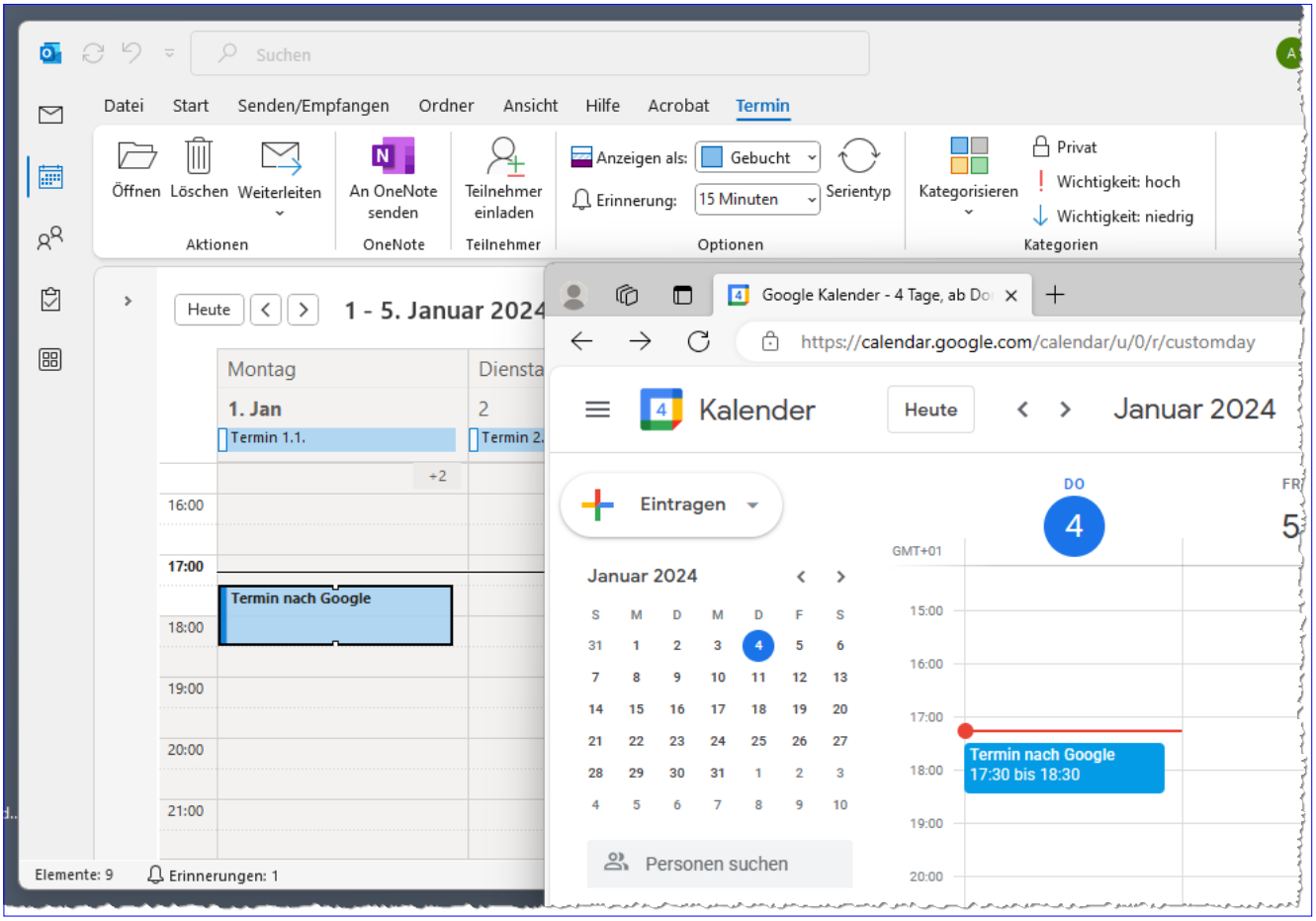

Bild 1: Ein von Outlook in den Google Calender übertragener Termin

-zeit sowie Enddatum und -zeit übermitteln. Auch um ganztägige Termine wollen wir uns kümmern.

Als Basis für die Prozeduren dieses Artikels können wir die im Artikel **Google Calendar per Rest-API programmieren** (**www.vbentwickler.de/410**) vorgestellten Techniken nurtzen, die wir im Artikel **Google Calendar per Rest-API programmieren, Teil 2** (**www.vbentwickler.de/415**) noch erweitert haben. Die Grundlagen für die Authentifizierung für den Austausch von Daten mit dem Google Calendar haben wir in den Artikeln **OAuth2-Token für Google per .NET-App holen** (**www.vbentwickler.de/413**), **Google Calendar per Rest-API programmieren** (**www.vbentwickler.de/410**) und **Google-Token per DLL holen** (**www.vbentwickler.de/409**).

Grundlegende Techniken für den Zugriff auf Outlook-Termine findest Du im Artikel **Outlook: Kalender und Termine programmieren** (**www.vbentwickler.de/415**).

Ablauf beim Übertragen von Terminen

Grundsätzlich ist das Übertragen eines Termins von Outlook zum Google Calendar kein Hexenwerk, wenn wir die Techniken aus den oben vorgestellten Artikeln kennen. Wir brauchen nur den zu übertragenden Termin zu identifizieren, die relevanten Informationen wie Betreff, Inhalt, Startzeit und Endzeit in einer JSON-Datei zusammenzufassen und diese an Google zu senden.

Allerdings kann es sein, dass sich ein solcher Termin einmal ändert. Wie können wir sicherstellen, dass diese Änderung auch in den Google Calendar übertragen wird? Dazu müssen wir als Erstes ein eindeutiges Merkmal des Termins im Google Calendar ermitteln und dieses mit dem übertragenen Termin speichern. Wenn sich dieser dann ändert, können wir die ID nutzen, um den Google-Termin entsprechend anzupassen. An diese ID kommen wir problemlos heran. Speichern können wir diese in der Eigenschaft **BillingInformation** – das ist eine Eigenschaft, die von Outlook selbst nicht genutzt wird und die auch über die Benutzeroberfläche nicht verfügbar ist.

Außerdem benötigen wir noch Automatismen, die dafür sorgen, dass Aktionen wie das Ändern oder Löschen eines Termins in Outlook direkt in den Google Calendar übertragen werden.

Modul im VBA-Projekt von Outlook anlegen

Um die benötigten VBA-Prozeduren unterzubringen, legen wir ein neues Modul im VBA-Projekt von Outlook an:

- Wechseln zum VBA-Editor mit der Tastenkombination **Alt + F11**
- Erstellen eines neuen Moduls mit **Einfügen|Modul**
- Aktivieren des Eigenschaftenfensters mit der Taste **F4**
- Markieren des Moduls
- Ändern des Namens in **mdlGoogleCalendar**

Dem neuen Modul fügen wir nun erst einmal die beiden Funktionen **GetDefaultCalendar** und **GetSelectedAppointmentItem** hinzu, die wir im Artikel **Outlook: Kalender und Termine programmieren** (**www.vbentwickler.de/415**) beschreiben. Diese nutzen wir, um den aktuell selektierten Termin in Outlook zu referenzieren. Genau diesen wollen wir nun in den Google Calendar übertragen.

Weitere Vorbereitungen

Wir haben alle Module, die wir für die Export-Funktion von Terminen zum Google Calendar benötigen, im Download bereitgestellt. Außerdem findest Du dort

ein funktionsfähiges VBA-Projekt für Outlook. Wenn Du dieses ausprobieren möchtest, kannst Du dieses zeitweise statt Deines eigenen VBA-Projekts unter Outlook nutzen. Dazu öffnest Du im Windows Explorer das Verzeichnis **C:\Users\[Benutzername]\AppData\Roaming\Microsoft\Outlook**. Hier findest Du eine Datei namens **VbaProject.OTM** vor. Dieses benennst Du um und fügst die gleichnamige Datei aus dem Download zu diesem Artikel hinzu.

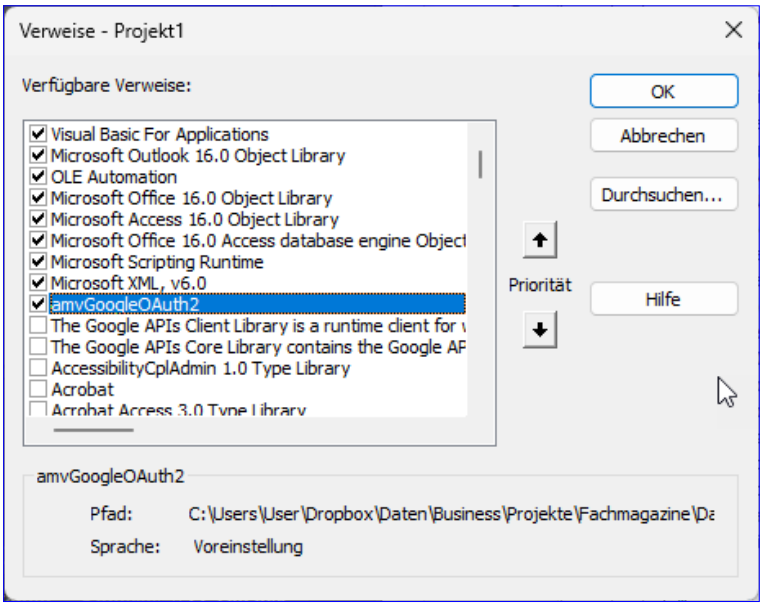

Bild 2: Notwendige Verweise für den Einsatz unserer Beispiellösung

Du kannst auch die Module aus dem Download einzeln in das VBA-Projekt importieren. Dazu ziehst Du diese einfach aus dem Windows Explorer in den Projekt-Explorer des VBA-Editors von Outlook. Du musst dann allerdings noch einige Verweise zum VBA-Projekt hinzufügen – in diesem Fall die unteren drei aus Bild 2:

- **Microsoft Scripting Runtime**
- **Microsoft XML, v6.0**
- **amvGoogleOAuth2**

Für den letzten Verweis musst Du außerdem noch die DLL **amvGoogleOAuth2.dll** installieren. Alle notdar geschrieben werden soll. Für den Rückgabewert der Funktion definieren wir den Datentyp **String**. Die Funktion soll im Erfolgsfall die **EventID** des neu hinzugefügten Kalendereintrags liefern.

Die Funktion liest das Token und die ID des Zielkalenders aus der Registry ein. Mehr zu diesem Thema liest Du im Artikel **Anwendungsdaten in der Registry** (**www.vbentwickler.de/411**). Im Artikel **Anwendungsdaten in der Registry** (**www.vbentwickler.de/411**) beschreiben wir, wie diese Werte zuvor in die

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 Calender holen: **Google-Authentifzierung 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:

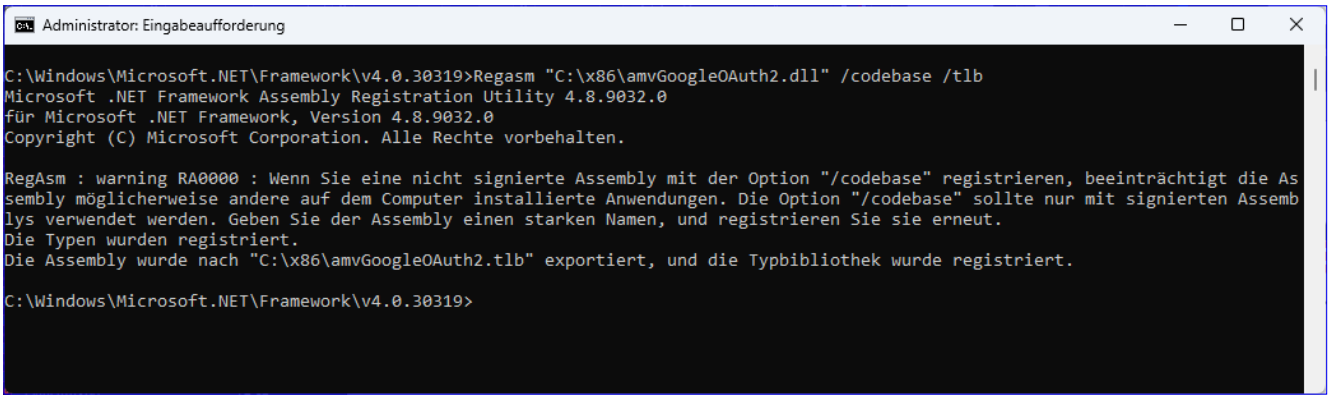

Bild 1: Registrieren der .NET-DLL

- Kopiere den kompletten Pfad zur Datei **amvGoogleOAuth2.dll** in die Zwischenablage.

- Öffne die Eingabeaufforderung im Administrator-Modus. Dazu gibst Du den Suchbegriff **cmd** in die Windows-Suche ein und klickst mit der rechten Maustaste auf den Eintrag **Eingabeaufforderung**. Hier wählst Du den Befehl **Als Administrator ausführen...** aus.

- Navigiere zur Datei **Regasm.exe**. Hier ist wieder die Unterscheidung zwischen Office 32-Bit und Office 64-Bit nötig. Für 32-Bit navigierst Du beispielsweise zum Verzeichnis **C:\Windows\Microsoft.NET\Framework\v4.0.30319**, für 64-Bit zum Verzeichnis **C:\Windows\Microsoft.NET\Framework64\v4.0.30319**. Hier gibst Du für die 32-Bit-Version den folgenden Befehl ein: **Regasm "C:\...\x86\amvGoogleOAuth2.dll" /codebase /tlb**. In der Eingabeaufforderung erhalten wir das Ergebnis aus Bild 1.

Ob die .NET-DLL korrekt installiert ist, können wir in einem beliebigen VBA-Projekt testen. Hier öffnen wir den **Verweise**-Dialog (**Extras|Verweise**) und suchen nach dem Eintrag **amvGoogleOAuth2**. Ist der Eintrag wie in Bild 2 vorhanden, können wir fortfahren.

COM-Add-In für Outlook installieren

Als Nächstes installieren wir das COM-Add-In, dessen Erstellung wir später in diesem Artikel erläutern. Das COM-Add-In kommt ebenfalls in zwei Versionen für die 32-Bit- und die 64-Bit-Fassung von Outlook.

Die entsprechenden Dateien heißen **amvAppointmentsToGoogle_win32.dll** und **amvAppointmentsToGoogle_win64.dll**. Die Installation ist etwas einfacher:

- Als Erstes öffnen wir Outlook.

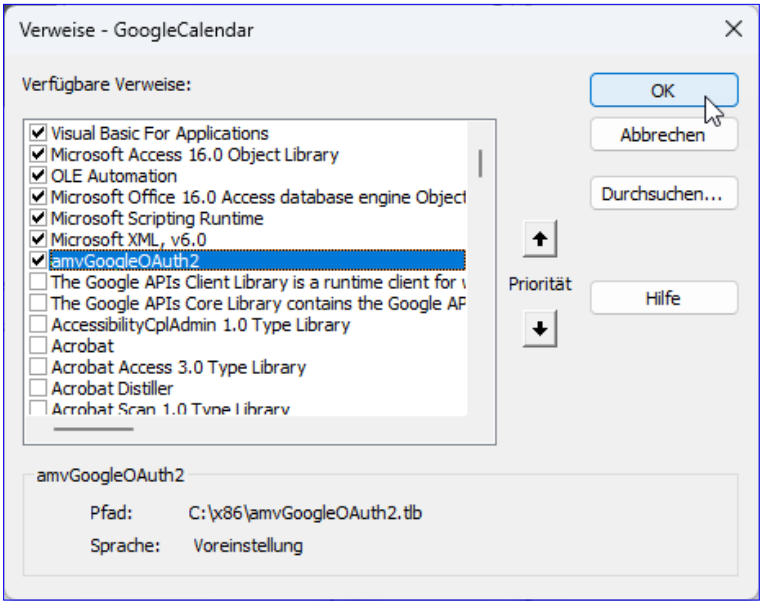

Bild 2: Prüfen, ob die .NET-DLL erfolgreich installiert wurde

- Dann zeigen wir mit dem Ribbonbefehl **Datei|Optionen** den Optionen-Dialog von Outlook an.

- Hier wechseln wir zum Bereich **Add-Ins**.

- Dort finden wir neben dem Auswahlfeld **Verwalten** mit dem Wert **COM-Add-Ins** die Schaltfläche **Los...**, die wir nun betätigen.

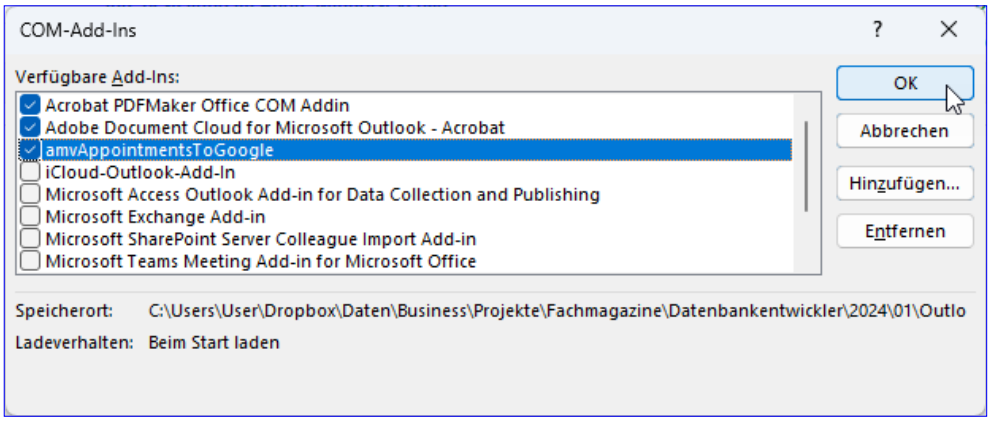

Bild 3: Installieren des COM-Add-Ins in Outlook

- Damit zeigen wir den Dialog **COM-Add-Ins** an, in dem wir auf die Schaltfläche **Hinzufügen...** klicken.

- Dies öffnet einen Dateiauswahl-Dialog, mit dem wir je nach Office-Version die Datei **amvAppointmentsToGoogle_win32.dll** oder **amvAppointmentsToGoogle_win64.dll** selektieren.

- Nach dem Hinzufügen zeigt der Dialog den neuen Eintrag wie in Bild 3 an.

Wenn nicht nur dieser Eintrag vorhanden ist, sondern Du unter Datei ganz unten auch noch den Eintrag **amvAppointmentToGoogle**, hat alles funktioniert.

Google-App für den Zugriff auf den Kalender

Um fortzufahren, benötigen wir die Client-ID und das Client-Secret einer Google-App. Wie Du diese erhältst, haben wir in **Google Calendar programmieren – Vorbereitungen** (**www.vbentwickler.de/408**) ausführlich gezeigt.

Outlook starten und Optionen eintragen

Damit können wir, falls nicht bereits geöffnet, Outlook starten und die soeben ermittelten Werte in die entsprechenden Optionen eintragen. Dazu wählen wir im Ribbon-Tab **Datei** den Eintrag **amvAppointmentToGoogle** aus. Dies zeigt den Bereich aus Bild 4 an.

Hier sehen wir oben die beiden Eigenschaften **Client-ID** und **Client-Secret**, die noch leer sein dürften. Hier trägst Du die beim Erstellen der Google-App erhaltenen Informationen ein.

Authentifizieren

Danach klicken wir auf die Schaltfläche **Token aktualisieren**, um erstmalig das Access-Token zu ermitteln, das wir für den Zugriff auf den Google Calendar im Kontext eines Benutzers benötigen.

Für diesen Schritt benötigen wir die .NET-DLL. Die darin enthaltenen Funktionen konnten wir nicht mit vertretbarem Aufwand in einer DLL auf Basis von VB6/twinBASIC erstellen geschweige denn mit VBA realisieren, daher haben wir hier .NET gewählt.

Danach öffnet sich ein Webbrowser und zeigt den Dialog an, mit dem wir angeben wollen, welches Konto wir als Kontext für den Zugriff auf den Google Calendar nutzen wollen (siehe Bild 5).

Durch die Anmeldung mit diesem Konto erfolgt die Authentifizierung und die .NET-DLL liefert uns das Access-Token, ein Refresh-Token und die Zeit, wie

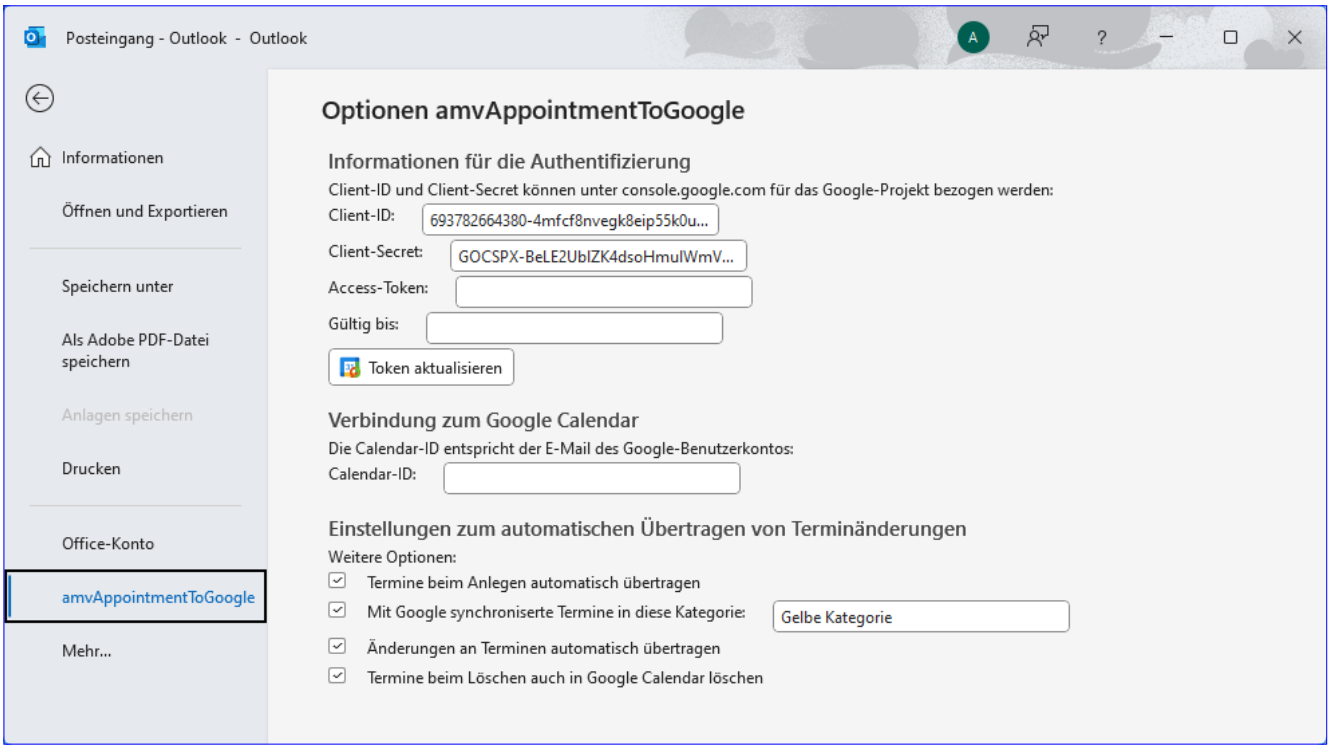

Bild 4: Die Optionen des COM-Add-Ins **amvAppointmentToGoogle**

lange das Token gültig ist und für den Zugriff auf den Google Calendar über die Rest-API erfolgen kann. Das Access-Token wird daraufhin gleich im entsprechenden Feld der Optionen angezeigt.

Wir können uns auch gleich die E-Mail-Adresse merken, die wir für die Authentifizierung verwendet haben. Diese tragen wir nun in das Feld **Calendar-ID** der Optionen des COM-Add-Ins ein. Alle Daten, die Du in den Optionen siehst, werden in der Registry gespeichert – mehr dazu später.

COM-Add-In anwenden

Danach können wir das COM-Add-In in Outlook ausprobieren. Wir haben folgende Funktionen implementiert:

- Manuelles Übertragen eines Termins zum Google Calendar per Kontextmenü oder Ribbon-Eintrag
- Manuelles Löschen eines Termins per Kontextmenü oder Ribboneintrag
- Automatisches Anlegen eines neuen Termins im Google Calendar, wenn ein neuer Termin im Outlook erstellt wird

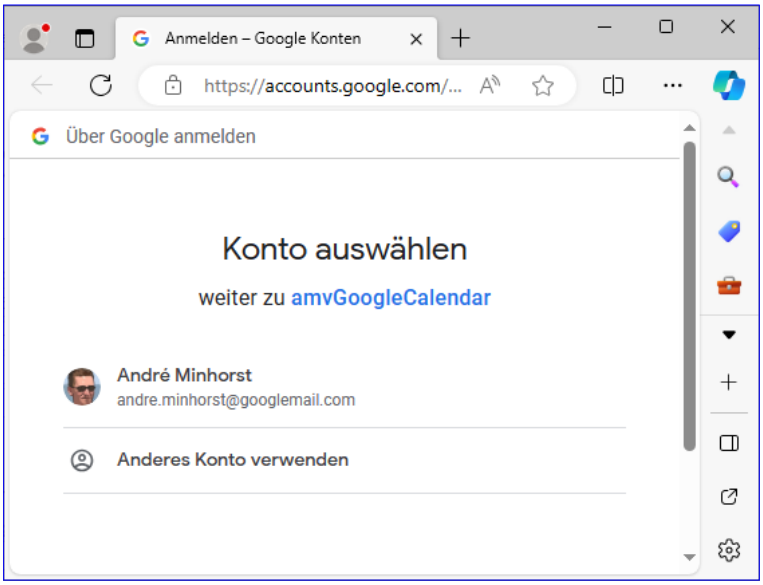

Bild 5: Konto für das Access-Token authentifizieren

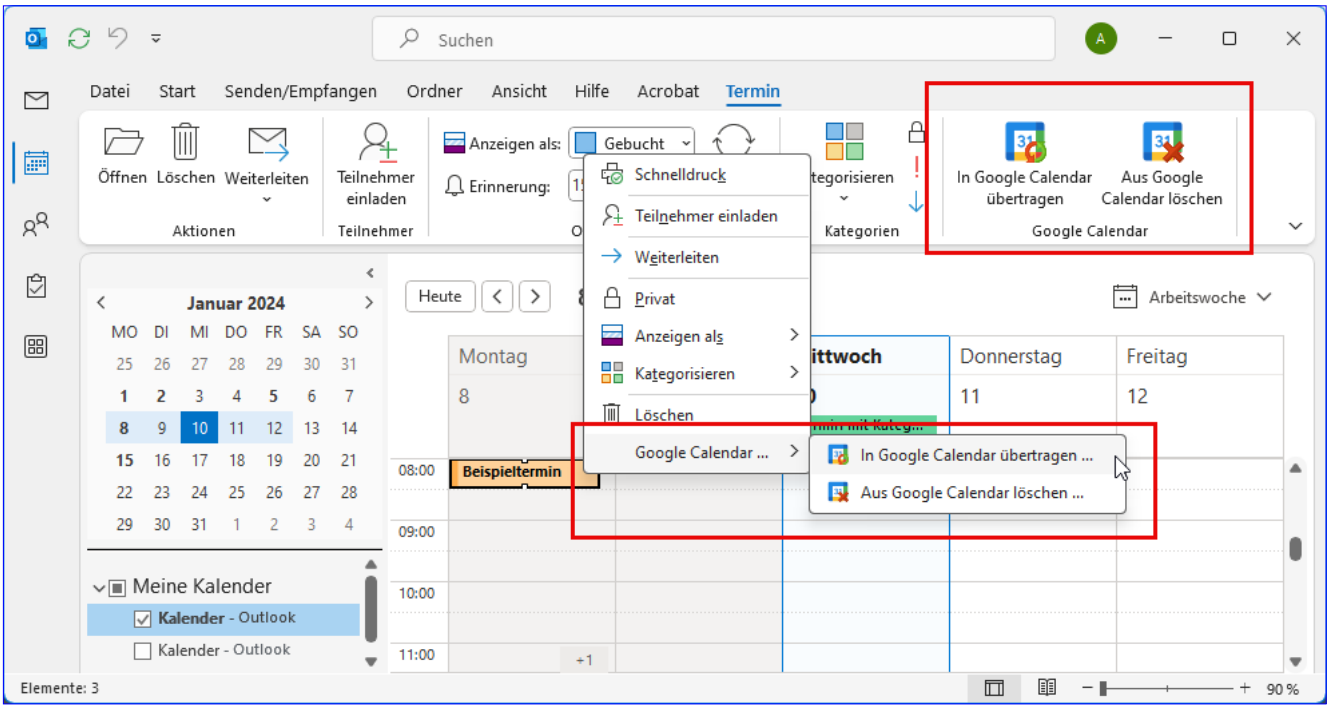

Bild 6: Funktionen zum manuellen Übertragen und Löschen von Terminen

- Automatisches Aktualisieren eines Termins im Google Calendar, wenn dieser in Outlook geändert wurde

- Automatisches Löschen beziehungsweise Setzen von Google-Terminen auf den Status **Cancelled**, wenn der entsprechende Outlook-Termin gelöscht wurde

Wenn wir beispielsweise einen Termin aktivieren, zeigt das Ribbon-Tab **Termin** im Bereich **Google Calendar** zwei Befehle an. Mit **In Google Calendar übertragen** schicken wir den Termin zum Google Calendar.

Mit **Aus Google Calendar löschen** entfernen wir den Termin wieder beziehungsweise setzen seinen Status auf **Cancelled** (siehe Bild 6).

Die gleichen Befehle tauchen nochmals auf, wenn wir das Kontextmenü eines Termins öffnen und dort das Untermenü **Google Calendar ...** öffnen.

Automatische Synchronisierung aktivieren

Damit Termindaten automatisch zum Google Calendar übertragen werden, sobald ein Termin angelegt, geändert oder gelöscht wird, müssen wir einige Optionen aktivieren. Diese finden wir im unteren Bereich der Optionen von **amvAppointmentToGoogle**.

Hier können wir festlegen, ob neu angelegte Termine, geänderte Termine oder gelöschte Termine mit dem Google Calendar synchronisiert werden sollen (siehe Bild 7). Mit der Option **Mit Google synchronisierte Termine in diese Kategorie:** können wir festlegen, dass Termine in Outlook, die nach Google übertragen wurden, automatisch eine bestimmte Kategorie erhalten. Den Namen dieser Kategorie geben wir im Textfeld rechts daneben an. Wichtig: Die Kategorie muss vorhanden sein, sonst wird diese nicht zugeordnet.

Programmieren des COM-Add-Ins

Das COM-Add-In haben wir mit twinBASIC programmiert, der neuen Entwicklungsumgebung, die sich als

Nachfolger von VB6 anschickt und über die wir bereits in einigen Artikeln geschrieben haben – mehr liefert die Suchfunktion mit dem Begriff twinBASIC auf **https://www.vbentwickler.de**.

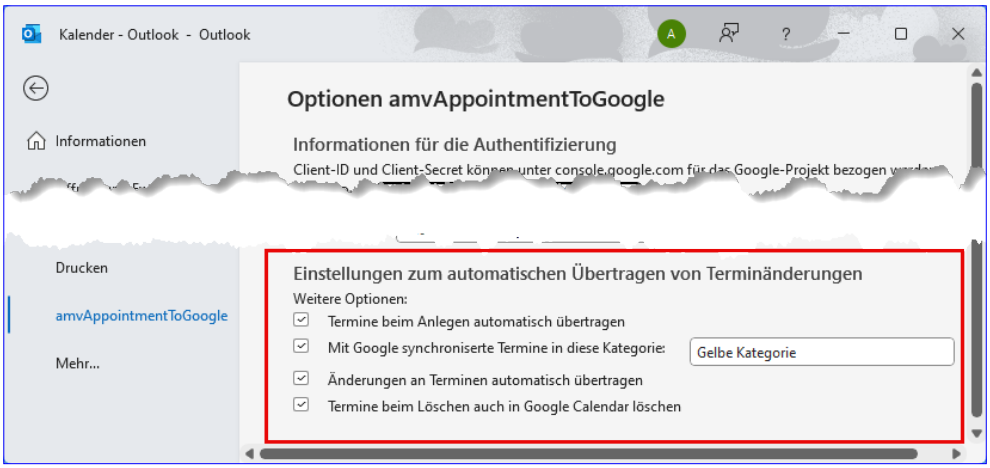

Bild 7: Aktivieren der automatischen Synchronisierung

Anlegen des Projekts und Optionen einstellen

Wir legen ein neues twinBASIC-Projekt an und stellen unter Settings den Projektnamen, die Beschreibung und den Anwendungstitel alle auf den Text **amvAppointmentsToGoogle** ein.

Außerdem fügen wir unter **Library References** noch einige zusätzliche Einträge hinzu (siehe Bild 8). Vor allem ist hier der Verweis auf die .NET-DLL **amvGoogleOAuth2** wichtig.

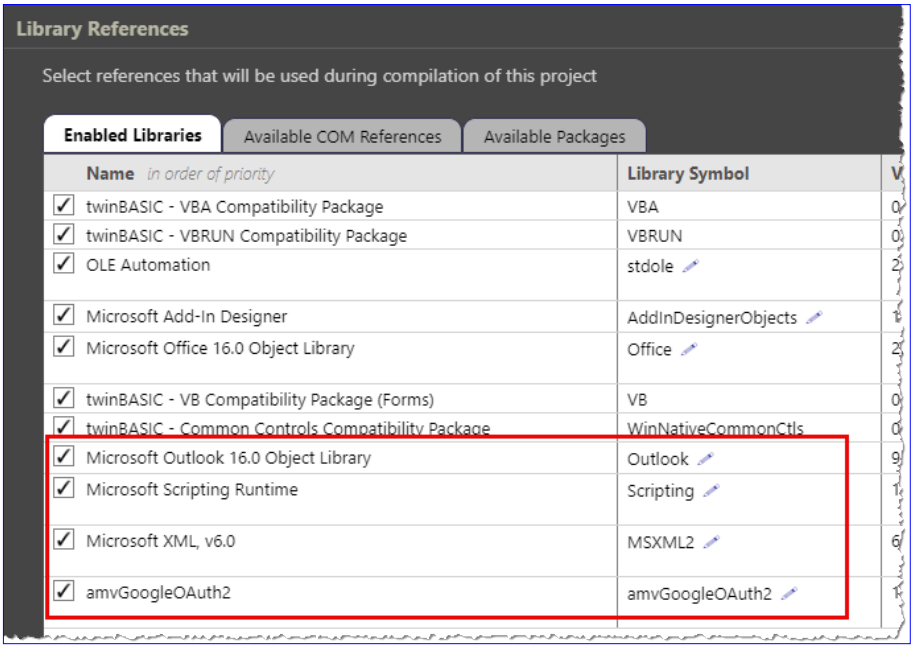

Bild 8: Verweise des twinBASIC-Projekts

Registrierung auf Outlook anpassen

Damit das COM-Add-In beim Start von Outlook ebenfalls gestartet wird, nehmen wir am Modul **DLLRegistration** der COM-Add-In-Vorlage von twinBASIC einige Änderungen vor.

Für die Konstante **AddinClassName** tragen wir **amvAppointmentsToGoogle** ein. Den Zielpfad in der Registry passen wir auf Outlook an:

```
Const RootRegistryFolder As String = "HKCU\SOFTWARE\Microsoft\Office\Outlook\Addins\" & AddinQualifiedClassName & "\"
```

Die restlichen Anpassungen kannst Du im Modul **DLLRegistration** anschauen.

Die Programmierung des Ribbons und der Kontextmenüs von Outlook haben wir bereits in den folgenden Artikel ausführlich erläutert:

- **Outlook: Ribbon per COM-Add-In anpassen (www.vbentwickler.de/376)**

- **Outlook: Kontextmenüs anpassen** (**www.vbentwickler.de/369**)

- **Outlook: Anhang speichern per Kontextmenü** (**www.vbentwickler.de/375**)

Daher gehen wir hier etwas knapper auf die verwendeten Techniken ein und beschreiben diese einmal etwas anders – nämlich nach dem Zeitpunkt des Aufrufs.

Start des COM-Add-Ins

Sobald das COM-Add-In einmal erstellt und registriert ist, stehen in der Registry einige Einträge, die Outlook beim Start ausliest.

Dadurch wird das COM-Add-In direkt beim Start von Outlook ebenfalls geladen.

Die dazu wichtigsten Funktionen sind in der Hauptklasse des COM-Add-Ins namens **amvAppointmentsToGoogle** enthalten.

Beim Starten geschehen zwei Dinge:

```
Public objOutlook As Outlook.Application
```

Drei weitere Variablen benötigen wir nur in der Klasse **amvAppointmentsToGoogle**:

```
Private objRibbon As IRibbonUI
Private WithEvents objCalendar As Outlook.Folder
Private WithEvents objCalendarItems As Outlook.Items
```

- Die Variable **objRibbon** nimmt einen Verweis auf die benutzerdefinierte Ribbondefinition des COM-Add-Ins auf.

- **objCalendar** referenziert den Ordner mit den Kalendereinträgen. Diese Variable wird mit dem Schlüsselwort **WithEvents** deklariert, damit wir die Ereignisse des **Folder**-Elements implementieren können.

- **objCalendarItems** referenziert die **Items**-Auflistung des Kalender-Ordners. Auch diese Objektvariable deklarieren wir mit **WithEvents**.

Die Initialisierung der Variablen erfolgt in der beim