AUSGABE 06/2023

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

Google Calendar im Griff

Eines der meistgefragten Themen von Lesern dreht sich um die Synchronisation von Daten aus den Office-Anwendungen wie Outlook, Excel oder Access mit dem Kalender von Google. Bisher sind wir ein wenig um dieses Thema herumgeschlichen. Die Programmierung von Rest-APIs ist uns grundsätzlich nicht fremd und technisch kein Problem. Allerdings gibt es Rest-APIs mit einfachen Authentifizierungsarten und es gibt solche, die OAuth 2.0 verwenden. Letzteres ist recht aufwendig zu programmieren. Wir sind das Thema nun doch angegangen und haben dabei verschiedene Techniken genutzt – .NET, um das notwendige Token zu holen und VB/VBA für den Rest.

Die meisten Webservices wehren sich nicht gegen das Auslesen und Ändern von Daten per VBA. Bei denen, die OAuth 2.0 verwenden, sieht es anders aus: Wir haben noch keine zuverlässige Möglichkeit gefunden, dies per VB/VBA zu erledigen. Also haben wir nun, da wir in diesem Magazin ja auch über VB.NET berichten wollen, eine Kombination gewählt, die gut funktioniert:

Wir holen wahlweise mit einer .NET-DLL oder mit einer .NET-Windows-Anwendung das Token und verwenden dieses dann, um per VBA auf die Daten von Google Calendar zuzugreifen.

Herausforderung OAuth 2.0

Aber was ist nun überhaupt die Hürde bei der Authentifizierung mit OAuth 2.0? Dazu muss man sich erst einmal ansehen, wie die Authentifizierung damit funktioniert.

Als Erstes benötigen wir dazu ein Projekt bei Google, dass wir unter unserem Google-Konto anlegen. Über dieses Projekt erhalten wir, nachdem wir eine kleine Odyssee hinter uns gebracht haben, die wir im Artikel **Google Calendar programmieren – Vorbereitungen** ab Seite 19 beschreiben, zwei wichtige Daten: eine Client-ID und ein Client-Secret. Diese dienen dabei beispielsweise für den Zugriff auf die Daten von Google Calendar. Als Zweites benötigen wir eine Funktion, mit welcher der Benutzer seine Zugangsdaten mit Client-ID und Client-Secret verknüpfen kann, um so einen Access-Token zu bekommen. Mit diesem kann er dann beispielsweise per Rest-API über VBA auf seinen Kalender bei Google zugreifen.

Um dieses Access-Token zu holen, stellen wir zwei Varianten vor – erstens über eine DLL, die wir von VBA aus Aufrufen können (siehe Artikel **Google-Token per DLL holen** ab Seite 30) und alternativ über eine .NET-Windows-Anwendung, die wir im Artikel **OAuth2-Token für Google per .NET-App holen** ab Seite 36 beschreiben.

Danach kann der Benutzer den Code aus unserem Artikel **Google Calendar per Rest-API programmieren** (ab Seite 46) nutzen, um mit dem zuvor gewonnenen Access-Token per VBA auf seine Kalender und die enthaltenen Termine zugreifen kann – beispielsweise um Termine auszulesen, anzulegen oder zu löschen.

Um die notwendigen Informationen zu kodieren und an Google zu senden, verwenden wir Daten im JSON-Format. Wie wir diese einfach zusammenstellen, beschreiben wir ab Seite 10 im Artikel **JSON-Dokumente per Objektmodell zusammenstellen**.

Schließlich schauen wir uns an, wie wir die verwendete Client-ID, Client-Secret und die verschiedenen Token per VBA-Code in der Registry sichern und wieder abrufen – siehe **Anwendungsdaten in der Registry** ab Seite 4.

Das Heft war rasend schnell gefüllt – wir hoffen, Du findest Gefallen an den Themen!

Dein André Minhorst

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.

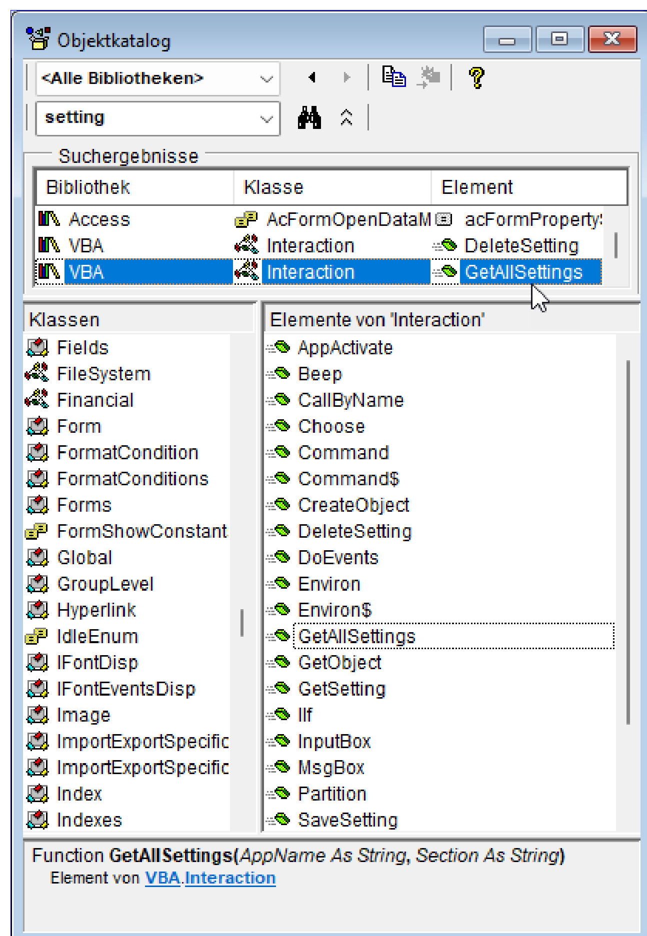

Bild 1: Elemente der **Interaction**-Klasse

Registry-Funktionen der Interaction-Klasse

Wenn wir im Objektkatalog nach Befehlen suchen, die einen Text wie Registry enthalten, suchen wir vergeblich.

Mit ein wenig Forschergeist finden wir jedoch schnell die **Interaction**-Klasse und darin die Elemente aus Bild 1.

Hier sehen wir die folgenden Funktionen:

- **DeleteSetting**: Löscht die mit den Parametern angegebene Einstellung.
- **GetAllSettings**: Liefert alle Einstellungen des mit den Parametern angegebenen Bereichs.
- **GetSetting**: Liefert die mit den Parametern angegebene Einstellung.
- **SaveSetting**: Speichert eine Einstellung in dem mit den Parametern angegebenen Bereich.

Die vier Funktionen verwenden alle ähnliche Parameter:

- **AppName**: Name der Anwendung, unter welcher die Einstellung gespeichert werden soll
- **Section**: Name des Bereichs unterhalb der Anwendung
- **Key**: Name des Schlüssels, also der Option
- **Setting**: Wert der Option

Wo werden die Werte in der Registry gespeichert?

Das könnten wir vermutlich leicht über das Internet erfahren, aber manchmal macht ein wenig Suche von Hand Spaß. Dazu legen wir überhaupt erst einmal einen Eintrag an und nutzen dazu den Befehl **SaveSetting**. Diesen rufen wir über das Direktfenster des VBA-Editors wie folgt auf und verwenden Werte für die Parameter, die den Parameternamen selbst entsprechen:

```
SaveSetting "AppName", "Section", "Key", "Setting"
```

Hier kann eigentlich kein Fehler passieren – selbst wenn die Einstellung bereits vorhanden ist, wird diese mit dem neuen Wert im vierten Parameter überschrieben.

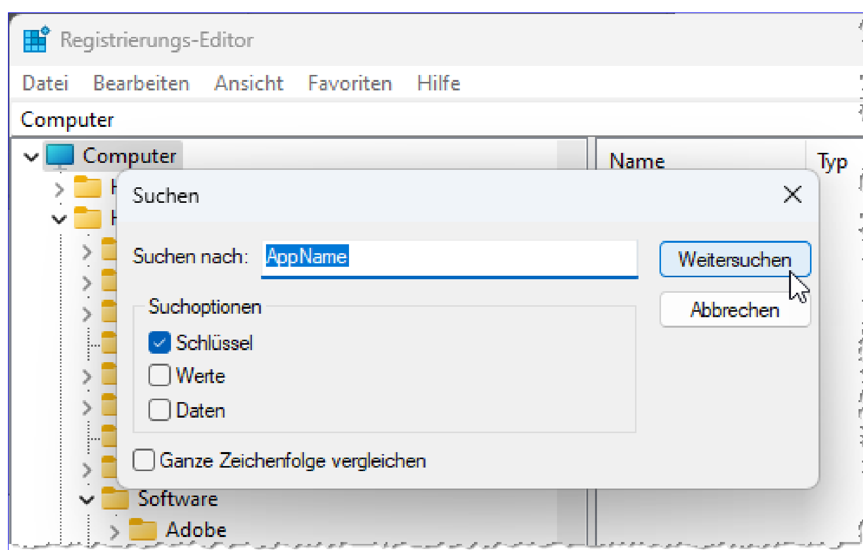

Bild 2: Suche nach dem neuen Registry-Eintrag

Danach öffnen wir die Registry mit dem Befehl **Registrierungs-Editor**, den wir in das **Suchen**-Feld von Windows eingeben.

Hier wählen wir den Menübefehl **Bearbeiten|Suchen** aus und geben dort den Wert **AppName** ein.

Außerdem deaktivieren wir die beiden Optionen **Werte** und **Daten**, damit nur in den Schlüsseln gesucht wird (siehe Bild 2).

Je nach der Menge der Einträge, die sich im Laufe der Zeit in der Registry angesammelt haben, dauert die Suche einige Sekunden.

Kurz danach zeigt die Registry jedoch das Suchergebnis wie in Bild 3 an. Hier sehen wir die Elemente aus den Parametern **AppName** und **Section** und darin auf der rechten Seite den Schlüssel **Key** mit dem Wert **Setting**.

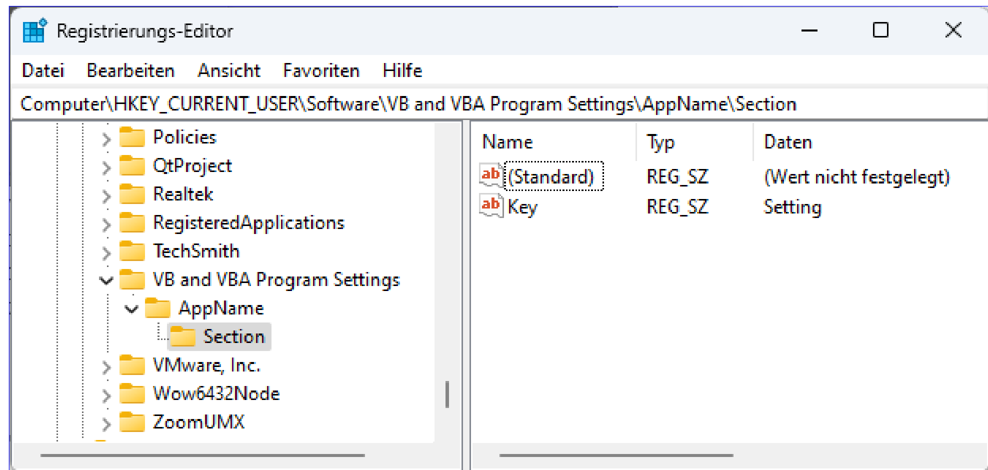

Bild 3: Der neue Eintrag in der Registry

Spannend ist nun, wo diese Elemente untergebracht wurden. Es handelt sich um einen Bereich unterhalb des Root-Elements **HKEY_CURRENT_USER**, was der Bereich für den aktuellen Benutzer ist.

Meldet sich also anschließend ein anderer Benutzer auf diesem Rechner an, kann er nicht auf diese Einträge zugreifen – außer, er hat sie für sich selbst ebenfalls angelegt.

Unterhalb von **HKEY_CURRENT_USER** finden wir die Elemente **Software** und **VB and VBA Program Settings**, bevor bereits die von uns angelegten Elemente folgen.

Eine Einstellung aus der Registry auslesen

Damit kommen wir zum zweiten Befehl namens **GetSetting**. Dieser erwartet mit Ausnahme des letzten Parameters die gleichen Parameter wie der Befehl, mit dem wir den Eintrag angelegt haben, also **AppName**, **Section** und **Key**. Wir geben den Befehl so im Direktbereich ein, dass das Ergebnis direkt dort ausgeben wird, und erhalten das folgende Ergebnis:

```
? GetSetting("AppName", "Section", "Key")
Setting
```

Einstellung ändern

Wollen wir die Einstellung ändern, zum Beispiel auf den Wert **New Setting**, rufen wir erneut den Befehl **SaveSetting** wie folgt auf:

```
SaveSetting "AppName", "Section", "Key", "New Setting"
```

Dies schlägt sich nach dem Aktualisieren des Registrierungs-Editor mit der Taste **F5** nieder (siehe Bild 4).

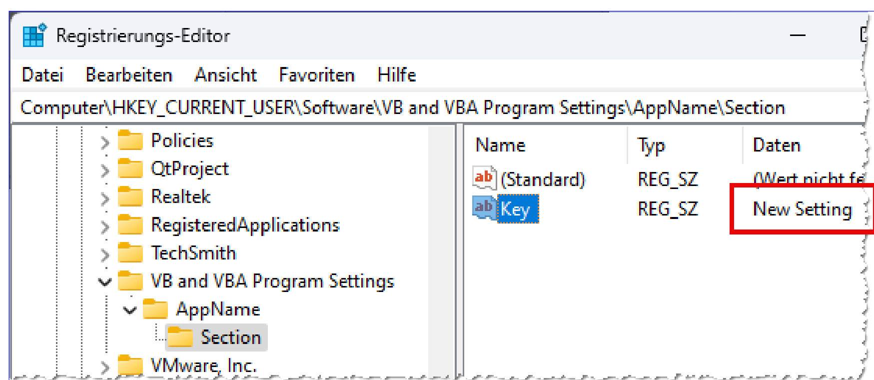

Bild 4: Der geänderte Eintrag in der Registry

JSON-Dokumente per Objektmodell zusammenstellen

Der Umgang mit JSON-Dokumenten wir immer wichtiger, da immer mehr Rest-APIs dieses Format nutzen, um Daten auszutauschen. In einem weiteren Artikel namens Mit JSON arbeiten (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** (**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

Das Problem ist nur: Wenn wir diese Zeilen zusammenstellen wollen, ist das ein erheblicher Aufwand.

Schauen wir uns die folgende Zeile des zu erzeugenden JSON-Dokuments an und gehen wir davon aus, dass wir das Dokument in VBA basierend auf einer Vorlage zusammenstellen wollen:

```
    "dateTime": "2023-11-22T17:00:00",
```

Dann müssen wir zuerst am Anfang der Zeile **strJSON = strJSON & "** voranstellen, hinten ein **" & vbCrLf** anfügen und schließlich noch alle enthaltenen Anführungszeichen verdoppeln, damit diese auch als Anführungszeichen innerhalb des Variableninhalts interpretiert werden und nicht als das Ende der Zeichenkette:

```
strJSON = strJSON & "    ""dateTime"": ""2023-11-22T17:00:00""," & vbCrLf
```

Das ist nicht nur sehr aufwendig, wenn man es von Hand macht, sondern auch noch extrem fehleranfällig.

Stellen wir uns dann vor, wir wollten zwischendrin eine Zeile entfernen – beispielsweise die Zeile mit dem Wert für das Feld **description**:

```
strJSON = strJSON & "  ""summary"":""Summary""," & vbCrLf
strJSON = strJSON & "  ""description"":""description""" & vbCrLf
strJSON = strJSON & "}" & vbCrLf
```

Dann müssen wir hier nicht nur die Zeile entfernen, sondern auch in der vorherigen Zeile das Komma, denn sonst wäre das JSON-Dokument fehlerhaft.

Vereinfachung dank JSON-Bibliothek

Hier kommt die Bibliothek beziehungsweise das Modul von Tim Hall ins Spiel, die wir zum Beispiel hier herunterladen können:

```
https://github.com/VBA-tools/VBA-JSON/blob/master/JsonConverter.bas
```

Diese bietet einige nützliche Funktionen. Eine davon, **ParseJSON**, haben wir bereits in dem oben genannten Artikel verwendet, um aus einem JSON-Dokument ein Objekt zu generieren, das alle Elemente des JSON-Dokuments in strukturierter Form als Dictionaries und Collections enthält.

In diesem Artikel zeigen wir auch unsere eigene Erweiterung, mit der wir noch den Code ausgeben, den wir für den Zugriff auf die Elemente verwenden können. Auf die Elemente des obigen, zugegebenermaßen einfach aufgebauten JSON-Dokuments, greifen wir damit wie folgt zu:

```
objJSON.Item("start").Item("dateTime")
objJSON.Item("start").Item("timeZone")
objJSON.Item("end").Item("dateTime")
objJSON.Item("end").Item("timeZone")
objJSON.Item("summary")
objJSON.Item("description")
```

Wie aber können wir nun den umgekehrten Weg gehen – also den, ein JSON-Dokument über **Dictionary**- und **Collection**-Elemente zusammenzustellen und daraus dann mit der Funktion **ConvertToJSON** ein JSON-Dokument im **String**-Format abzuleiten? Dies schauen wir uns anhand einiger Beispiele an.

Objekte erstellen

Die wichtigste Information vorab:

- Eckige Klammern deuten immer auf die Auflistung von Elementen hin, diese landen in einem **Collection**-Objekt.
- Geschweifte Klammern weisen auf ein oder mehrere Name-Wert-Paare hin, diese landen in einem **Dictionary**-Objekt.

Einfachstes Beispiel: Ein Name-Wert-Paar

Im ersten Beispiel wollen wir dieses Dokument erzeugen:

```
{
  "Vorname":"Andre"
}
```

Dazu brauchen wir noch verhältnismäßig viel Code:

```
Public Sub NameWertPaar()
    Dim dicJSON As New Dictionary
    Dim strJSON As String
    dicJSON.Add "Vorname", "Andre"
    strJSON = ConvertToJson(dicJSON)
    Debug.Print strJSON
End Sub
```

Mehrere Name-Wert-Paare

Nun wollen wir wie folgt ein oder mehrere Name-Wert-Paare hinzufügen:

```
{
  "Vorname":"Andre",
  "Nachname":"Minhorst"
}
```

Dazu brauchen wir je Name-Wert-Paare nur die **Add**-Methode des **Dictionary**-Objekts erneut aufzurufen:

```
Public Sub ZweiNameWertPaare()
    Dim dicJSON As New Dictionary
    Dim strJSON As String
    dicJSON.Add "Vorname", "Andre"
    dicJSON.Add "Nachname", "Minhorst"
    strJSON = ConvertToJson(dicJSON)
    Debug.Print strJSON
End Sub
```

Wir sehen: Die Hauptarbeit ist bereits getan, alle weiteren Elemente auf diese Ebene sind Einzeiler.

Name-Wert-Paare mit untergeordneten Elementen

Nun wollen wir wie folgt die Adresse hinzufügen, welche die Adressdaten im Element **Adresse** aufführt:

```
{
  "Vorname":"Andre",
  "Nachname":"Minhorst",
  "Adresse":{
    "Strasse":"Borkhofer Str. 17",
    "PLZ":"47137",
    "Ort":"Duisburg"
  }
}
```

Dazu legen wir einfach ein neues **Dictionary**-Element namens **dicAdresse** an. Dieses fügen wir zuerst als Unterelement mit dem Namen **Adresse** zum Hauptelement **dicJSON** hinzu.

Erst danach füllen wir **dicAdresse** mit den drei Name-Wert-Paaren:

```
Public Sub NameWertMitUnterelementen()
    Dim dicJSON As New Dictionary
    Dim dicAdresse As New Dictionary
    Dim strJSON As String
    dicJSON.Add "Vorname", "Andre"
    dicJSON.Add "Nachname", "Minhorst"
    dicJSON.Add "Adresse", dicAdresse
    dicAdresse.Add "Strasse", "Borkhofer Str. 17"
    dicAdresse.Add "PLZ", "47137"
    dicAdresse.Add "Ort", "Duisburg"
    strJSON = ConvertToJson(dicJSON)
    Debug.Print strJSON
End Sub
```

Dies können wir endlos so weiterbetreiben. Interessant ist aber nun noch, wie wir mit mehreren gleichartigen Elementen umgehen, die in eckige Klammern eingefasst werden sollen.

Mehrere gleiche Objekte

Nun stellen wir uns vor, wir wollten mehrere Adressen für den Kontakt speichern – beispielsweise zwei:

```
{
  "Vorname":"Andre",
  "Nachname":"Minhorst",
  "Adressen":[
    {
      "Strasse":"Borkhofer Str. 17",
      "PLZ":"47137",
      "Ort":"Duisburg"
    },
    {
      "Strasse":"Gerichtsstr. 2a",
      "PLZ":"12344",
      "Ort":"Testhausen"
    }
  ]
}
```

Dafür benötigen wir nun auch zwei **Dictionary**-Elemente namens **dicAdresse1** und **dicAdresse2**. Diese fügen wir einem **Collection**-Element namens **colAdressen** hinzu, das wir wiederum dem Element **Adressen** untergeordnet haben:

```
Public Sub NameWertMitMehrerenGleichenUnterelementen()
    Dim dicJSON As New Dictionary
    Dim colAdressen As New Collection
    Dim dicAdresse1 As New Dictionary
    Dim dicAdresse2 As New Dictionary
    Dim strJSON As String
    dicJSON.Add "Vorname", "Andre"
    dicJSON.Add "Nachname", "Minhorst"
    dicJSON.Add "Adressen", colAdressen
    colAdressen.Add dicAdresse1
    dicAdresse1.Add "Strasse", "Borkhofer Str. 17"
    dicAdresse1.Add "PLZ", "47137"
    dicAdresse1.Add "Ort", "Duisburg"
    colAdressen.Add dicAdresse2
    dicAdresse2.Add "Strasse", "Gerichtsstr. 2a"
    dicAdresse2.Add "PLZ", "12344"
    dicAdresse2.Add "Ort", "Testhausen"
    strJSON = ConvertToJson(dicJSON)
    Debug.Print strJSON
End Sub
```

Nun könnte man auf den Gedanken kommen, dass man, wenn man Daten beispielsweise aus einer Tabelle per Code zu einem JSON-Dokument zusammenfügen will, für jedes Element eine neue Variable erzeugen muss.

Das ist aber nicht der Fall: Wir können die gleiche Variable für mehrere Elemente nutzen und müssen diese nur neu füllen.

Für das Beispiel von oben sieht dies wie folgt aus:

```
Public Sub NameWertMitMehrerenGleichenUnterelementen_Eine-
Variable()
    ...
    Dim dicAdresse As Dictionary
    ...
    Set dicAdresse = New Dictionary
    colAdressen.Add dicAdresse
    dicAdresse.Add "Strasse", "Borkhofer Str. 17"
    dicAdresse.Add "PLZ", "47137"
    dicAdresse.Add "Ort", "Duisburg"
    Set dicAdresse = New Dictionary
    colAdressen.Add dicAdresse
    dicAdresse.Add "Strasse", "Gerichtsstr. 2a"
    dicAdresse.Add "PLZ", "12344"
    dicAdresse.Add "Ort", "Testhausen"
    ...
End Sub
```

Objekte in der ersten Ebene

Dann gibt es noch den Fall, dass die Objekte in der ersten Ebene in Klammern erscheinen sollen, also beispielsweise mehrere Personen mit ihrem Namen:

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.

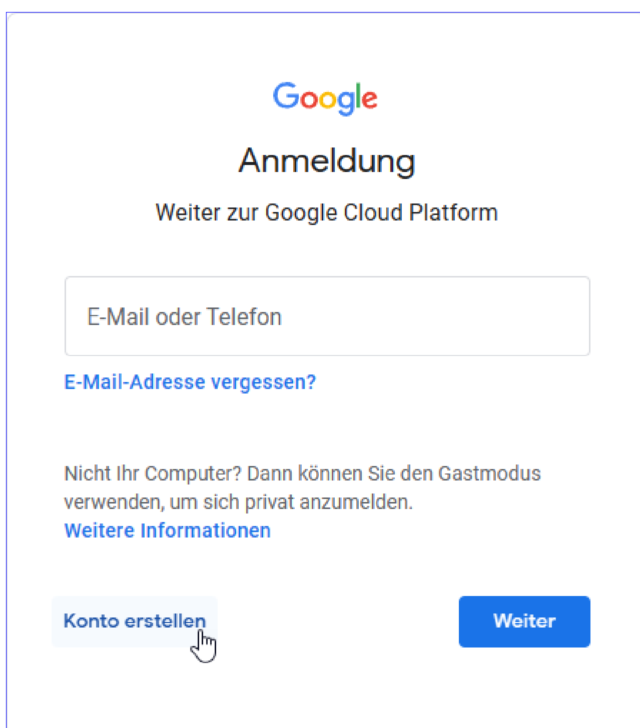

Bild 1: Anmelden oder neues Konto erstellen

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 2: Auswahl des Kontotyps

stehenden Kontos angeben, unter dem Du ein Google-Projekt erstellen möchtest, oder Du klickst auf **Konto erstellen** und startest den Vorgang, um ein neues Konto zu erstellen.

Beim Erstellen eines neuen Konto gibst Du an, für welchen Zweck das Konto angelegt werden soll – für die private Nutzung, für ein Kind oder für Arbeit/Unternehmen (siehe Bild 2). Wenn Du nicht gerade Dein Unternehmen auf Google vorstellen möchtest, reicht hier der erste Eintrag für die private Nutzung aus.

Danach geben wir den Namen ein, wobei der Nachname sogar optional ist. Der nächste Schritt fragt das Geburtsdatum und das Geschlecht ab.

Bei der Angabe der E-Mail-Adresse im nächsten Schritt können wir eine eigene, bereits vorhandene E-Mail-Adresse angebe oder eine Gmail-Adresse erstellen (ebenfalls kostenlos). Wir entscheiden uns für eine vorhandene E-Mail-Adresse (siehe Bild 3).

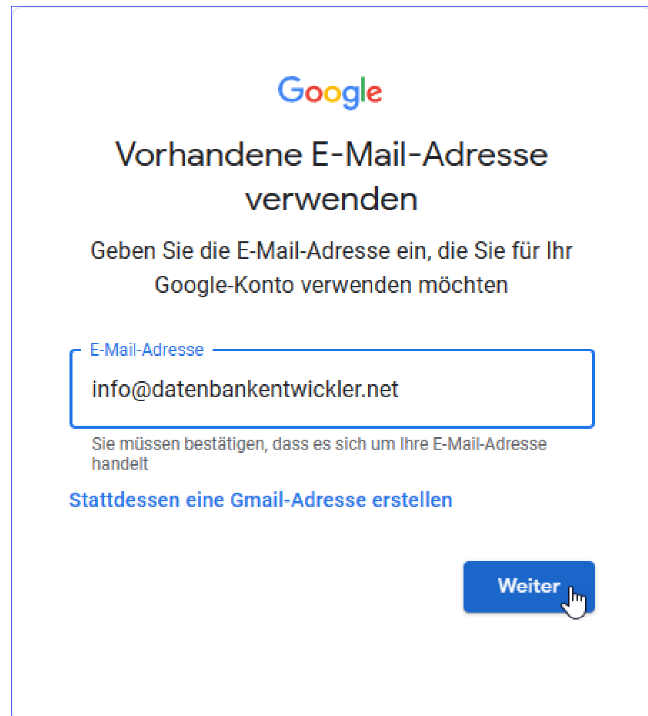

Bild 3: Angabe der E-Mail-Adresse

Diese E-Mail-Adresse muss im nächsten Schritt bestätigt werden. Dazu sendet Google uns eine E-Mail an die angegebene Adresse. Den enthaltenen Code geben wir im folgenden Dialog ein.

Schließlich erstellen wir noch ein Kennwort, fügen eine Telefonnummer hinzu, falls gewünscht (diese wird nur für Sicherheitszwecke verwendet) und fragt noch einige weitere Informationen ab. Haben wir diese eingegeben, können wir mit den nächsten Schritten die eigentliche Aufgabe angehen.

Neues Projekt anlegen

Nachdem wir nun auf jeden Fall ein Google-Konto haben, können wir nochmals zu **https://console.cloud.google.com** gehen und uns dort anmelden. Hier stimmen wir nach der Anmel-

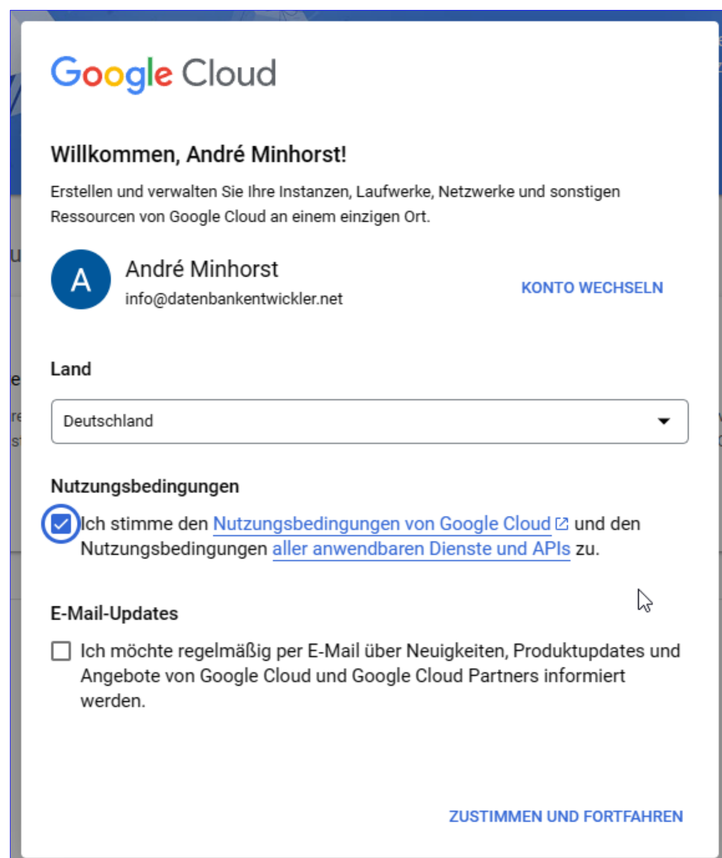

Bild 4: Nutzungsbedingungen akzeptieren

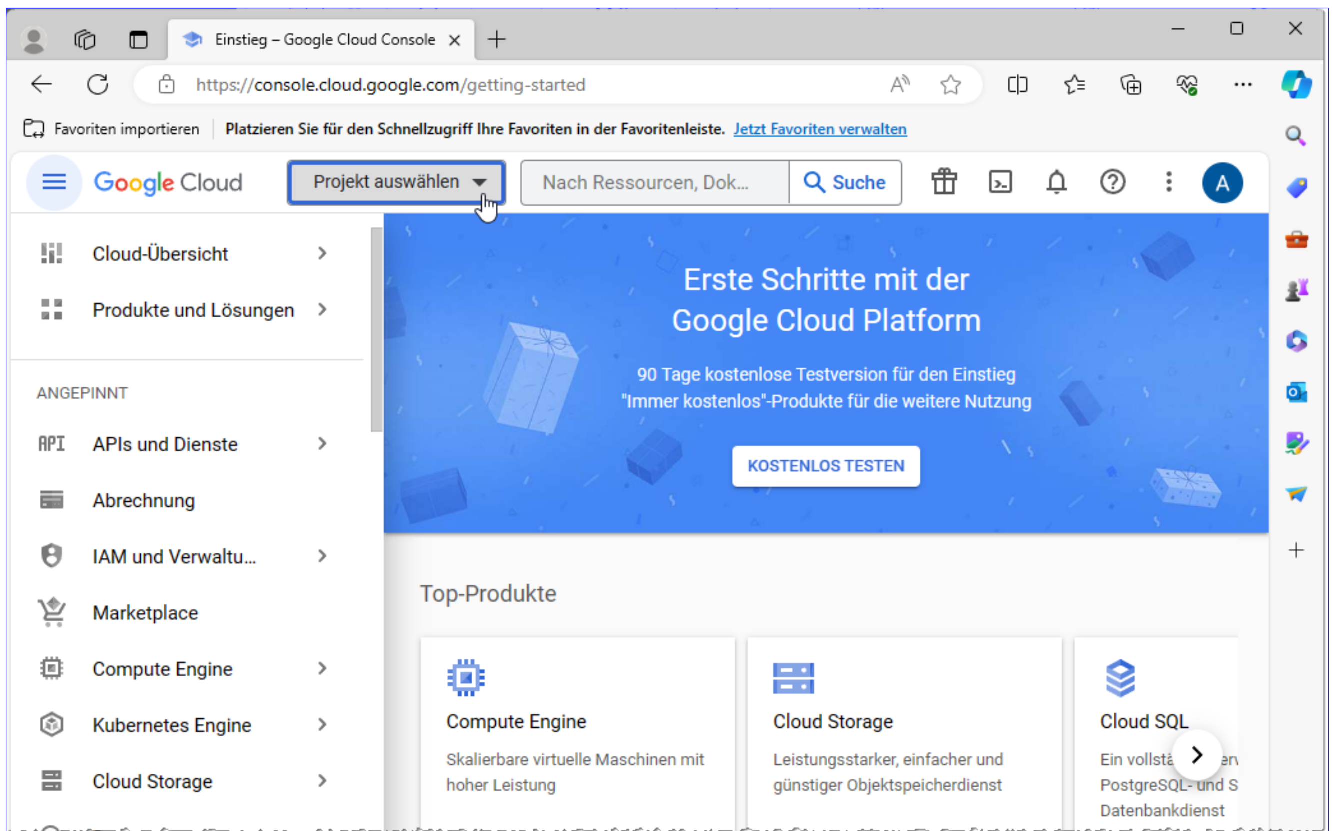

Bild 5: Startbildschirm der Google Cloud

dung auch noch den Nutzungsbedingungen zu (siehe Bild 4).

Danach zeigt Google den Startbildschirm an, der wie in Bild 5 aussieht. Hier klicken wir nun auf **Projekt auswählen**.

Dies blendet den Dialog **Projekt auswählen** aus Bild 6 ein. Da wir zuvor noch kein Projekt angelegt haben, ist dieser Dialog noch leer und wir können mit einem Klick auf **Neues Projekt** direkt zur Tat schreiten.

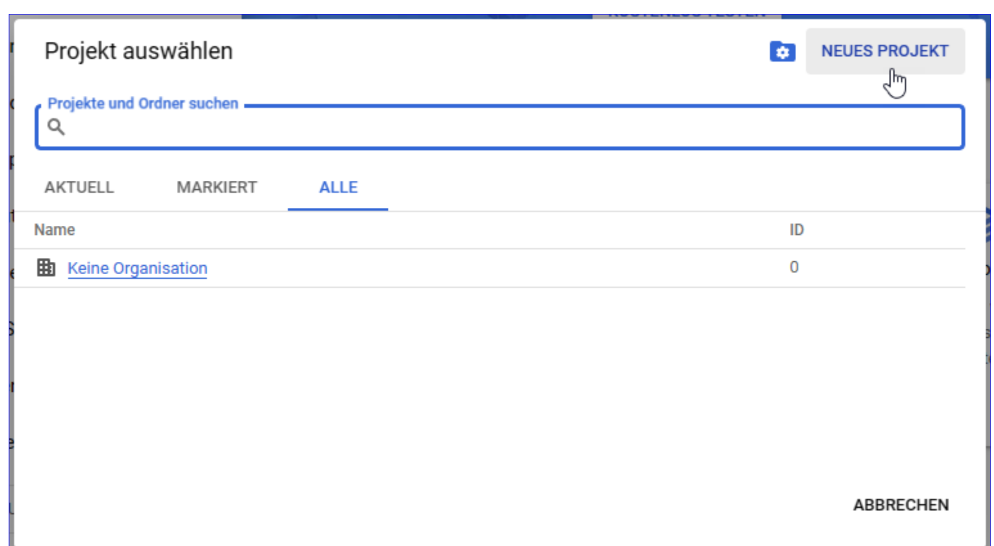

Bild 6: Anlegen eines neuen Projekts

Google öffnet nun eine neue Seite mit dem Titel **Neues Projekt** (siehe Bild 7). Hier geben wir den Projektna-

men ein, zum Beispiel **amvGoogleRestAPI**, und belassen den Wert für **Speicherort** auf **Keine Organisation** und klicken auf **Erstellen**.

Nun erstellt Google das Projekt, was einige Sekunden dauert. Danach zeigt es eine Benachrichtigung wie in Bild 8 an.

Wenn diese verschwindet, ohne dass wir das neue Projekt ausgewählt haben – kein Problem, wir finden dieses nun auch vor, wenn wir erneut auf den Link **Projekt auswählen** klicken.

Hier sehen wir jetzt den Eintrag **amvGoogleRestAPI** (siehe Bild 9).

Danach sehen wir die Übersichtsseite für unser Projekt. Hier finden wir unter Schnellzugriff den Befehl **APIs und Dienste**, den wir nun mit der Maus überfahren und im folgenden Untermenü den Eintrag **Bibliothek** auswählen (siehe Bild 10).

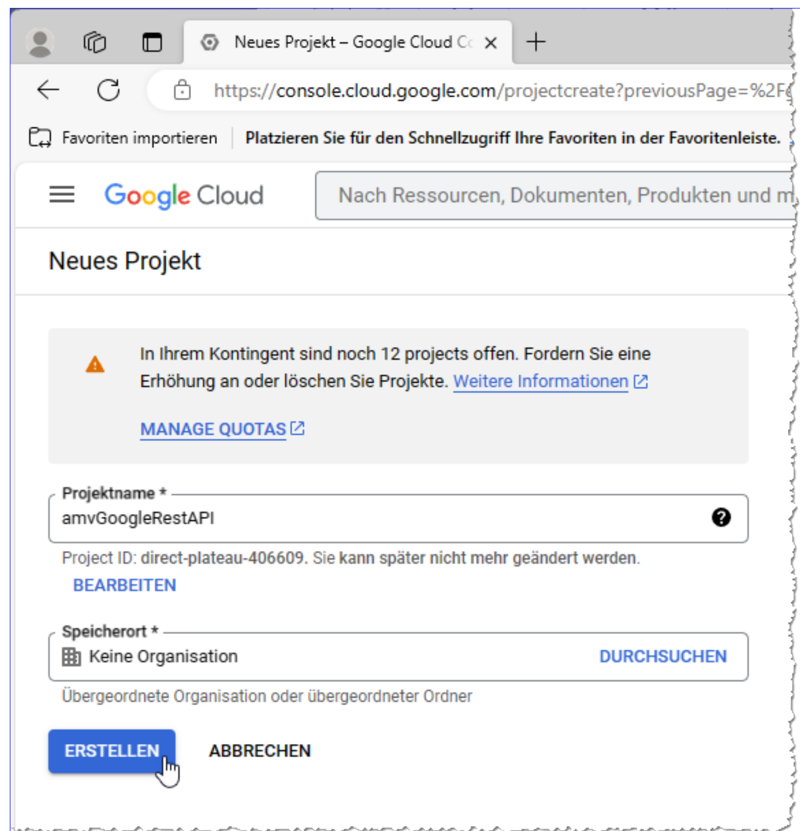

Bild 7: Eingeben des Projektnamens

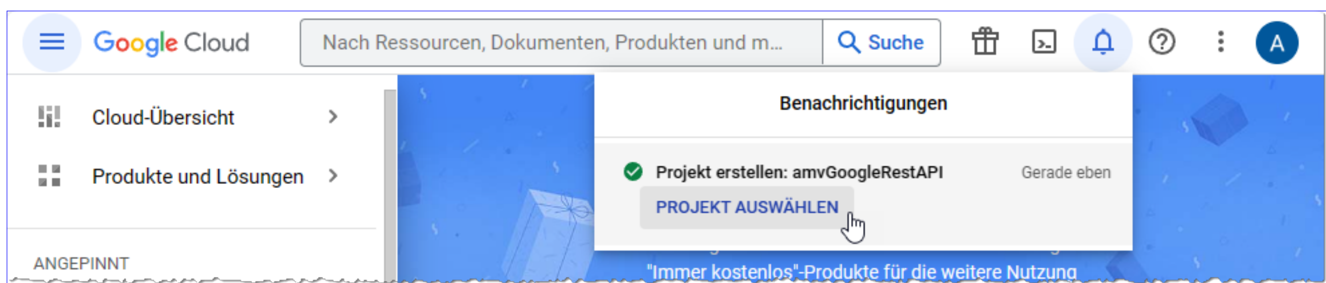

Bild 8: Möglichkeit, das Projekt auszuwählen

Bild 9: Unser Projekt im Dialog **Projekt auswählen**

Google Calendar API hinzufügen

Auf der nun erscheinenden Seite geben wir oben

im Suchfeld **Google Calendar** ein und finden als Ergebnis unter anderem den Eintrag **Google Calendar API** vor (siehe Bild 11).

Mit einem Klick auf diesen Eintrag zeigen wir dessen Detailansicht an. Hier finden wir die Schaltfläche **Aktivieren**, um die API zu aktivieren – was wir auch direkt erledigen (siehe Bild 12). Dies dauert wiederum einige Sekunden.

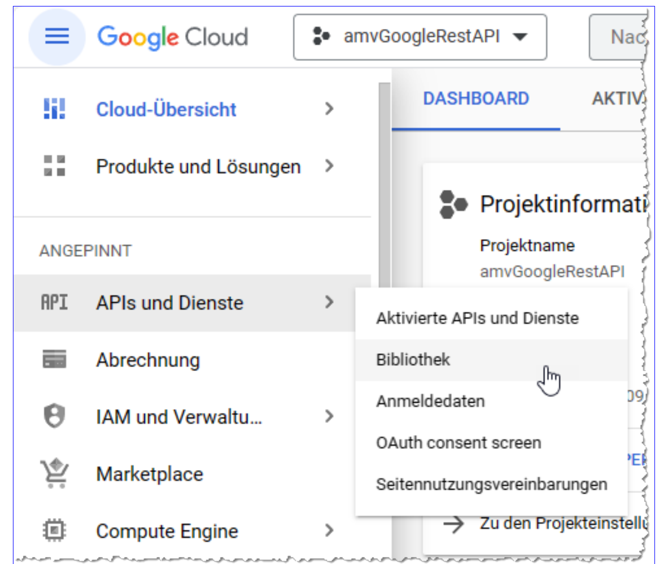

Bild 10: Anzeige der Bibliotheken

Anmeldedaten erstellen

Damit kommen wir zu einem sehr wichtigen Punkt, den wir bereits in der Einleitung angesprochen haben:

Das Ermitteln der für den ersten Schritt der Anmeldung nötigen Daten, konkret der Client-ID und des Client-Secret.

Dass diese notwendig sind, zeigt uns auch Google nun an und stellt direkt die Schaltfläche **Anmeldedaten erstellen** bereit (siehe Bild 13).

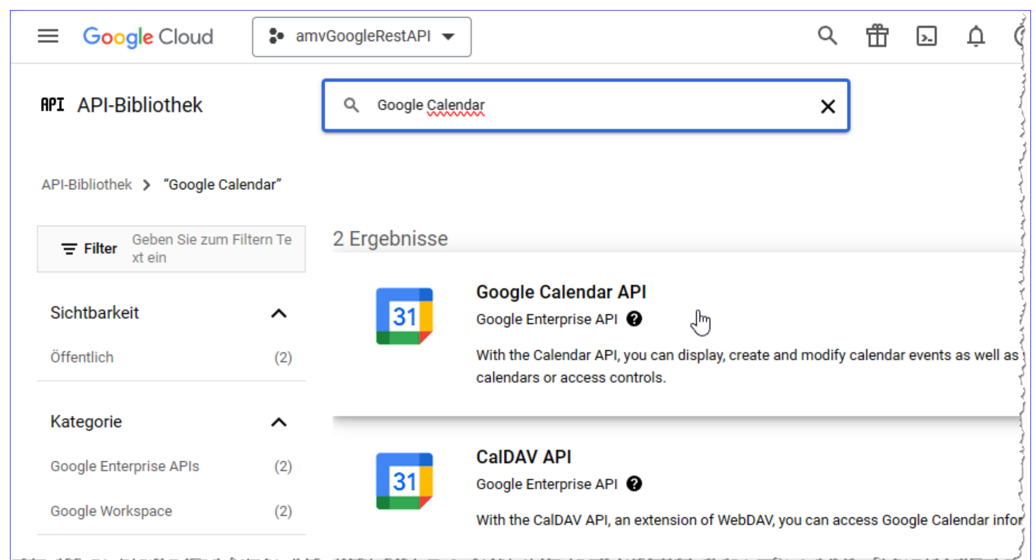

Bild 11: Auswahl der Bibliothek **Google Calendar API**

API auswählen

Nun folgen einige Schritte, in denen Informationen über die Anmeldedaten abgefragt werden.

Im ersten Schritt behalten wir unter **API auswählen** den Eintrag **Google Calendar API** bei.

Unten selektieren wir die Option **Nutzerdaten**, denn wir wollen einen OAuth-Client erstellen (siehe Bild 14).

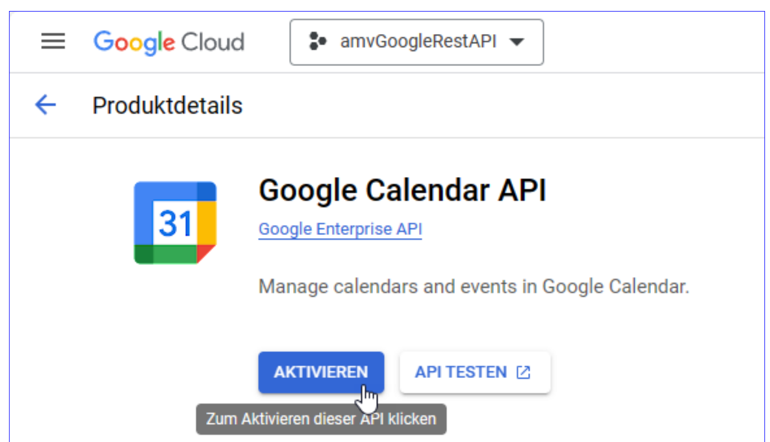

Bild 12: Aktivieren der Bibliothek **Google Calendar API**

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**

(**.NET Framework**). Die einzige Klasse, die wir nun im Projektmappen-Explorer vorfinden, benennen wir von **Class1** in **GoogleOAuth2** um. Stimmen wir der folgenden Meldung zu, wird der Klassenname auch gleich in den Code übernommen.

Eigenschaften des Projekts ändern

Nun öffnen wir die Projekteigenschaften und aktivieren den Bereich **Anwendung**. Hier klicken wir auf die Schaltfläche **Assemblyinformationen...** und öffnen so den Dialog **Assemblyinformationen**. Hier können wir alle Einstellungen beibehalten außer der Option **Assembly COM-sichtbar machen**, die wir aktivieren (siehe Bild 1).

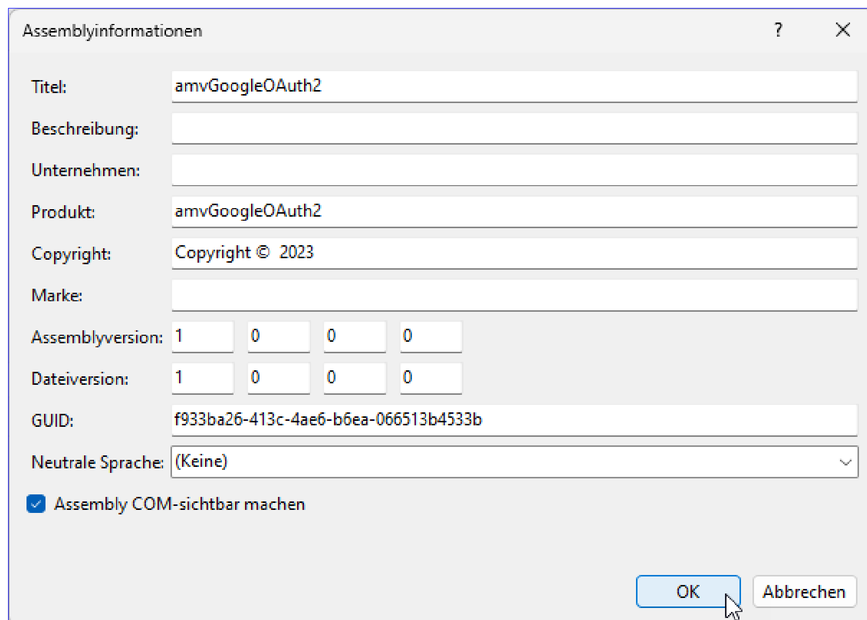

Bild 1: Aktivieren der Option **Assembly COM-sichtbar machen**

Danach wechseln wir zum Bereich **Kompilieren** und aktivieren dort die Option **Für COM-Interop registrieren** (siehe Bild 2). Außerdem müssen wir die Einstellung für die Eigenschaft **Ziel-CPU** prüfen. Wenn diese auf **AnyCPU** eingestellt ist, kann es sein, dass direkt

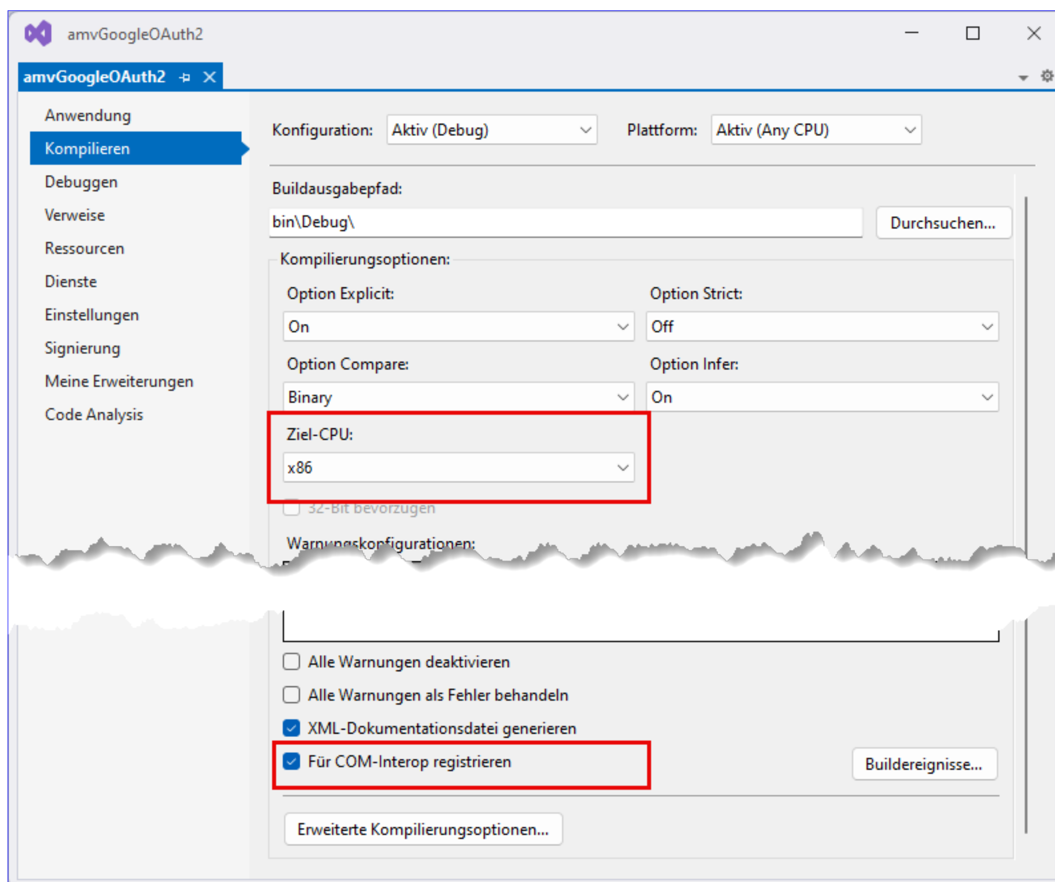

Bild 2: Einstellen der richtigen Ziel-CPU

für x64 kompiliert wird – was ein Problem ist, wenn wir von einer 32-Bit-Office-Anwendung darauf zugreifen wollen. Solltest Du also Office in der 32-Bit-Version nutzen, stellst Du sicherheitshalber hier den Wert **x86** ein.

Damit können wir die Eigenschaften speichern und schließen.

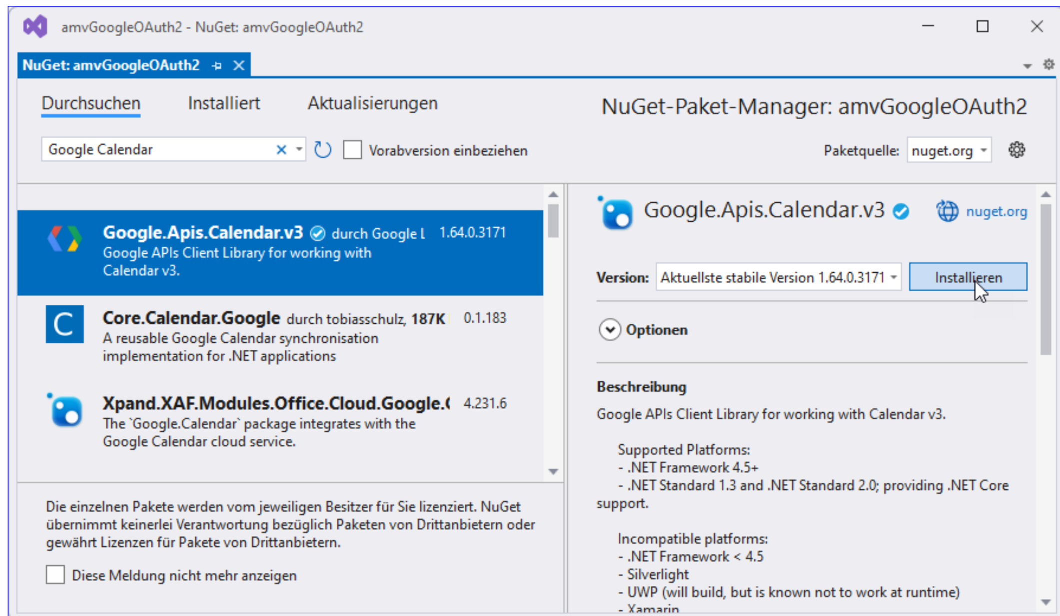

Bild 3: Installieren des Pakets **Google.Apis.Calendar.V3**

NuGet-Paket hinzufügen

Nun öffnen wir mit dem Menübefehl **Projekt|NuGet-Pakete verwalten...** den Bereich zum Verwalten der **NuGet**-Pakete. Hier wechseln wir zur Registerseite **Durchsuchen** und geben den Suchbegriff **Google Calendar** ein.

Den nun erscheinenden Eintrag **Google.Apis.Calendar.v3** wählen wir aus und fügen diesen mit einem Klick auf die Schaltfläche **Installieren** zum Projekt hinzu (siehe Bild 3).

Code zum Ermitteln des Tokens

Damit können wir den Code eingeben. Dieser sieht in der Übersicht wie in Listing 1 aus. Wir benötigen als Erstes einige Namespaces, die wir mit der **Imports**-Anweisung festlegen:

```
Imports System.Runtime.InteropServices
Imports System.Threading
Imports Google.Apis.Auth.OAuth2
Imports Google.Apis.Calendar.v3
```

Dann folgt die Schnittstelle **IGoogleOAuth2**, die festlegt, welche Elemente durch die DLL nach außen hin sichtbar gemacht werden. Diese enthält die beiden Eigenschaften **ClientID** und **ClientSecret** sowie eine Funktion namens **GetToken**:

```
<InterfaceType(ComInterfaceType.InterfaceIsDual)>
Public Interface IGoogleOAuth2
    Property ClientID As String
    Property ClientSecret As String
    Function GetToken() As String
End Interface
```

Danach implementieren wir diese Schnittstelle. Das erledigen wir mit der Klasse **GoogleOAuth2**. Diese definiert die beiden Eigenschaften **ClientID** und **ClientSecret**:

```
<ClassInterface(ClassInterfaceType.None)>
Public Class GoogleOAuth2
    Implements IGoogleOAuth2
    Private Property ClientID As String _
        Implements IGoogleOAuth2.ClientID
    Private Property ClientSecret As String _
```

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-Token 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 Projektnamen 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

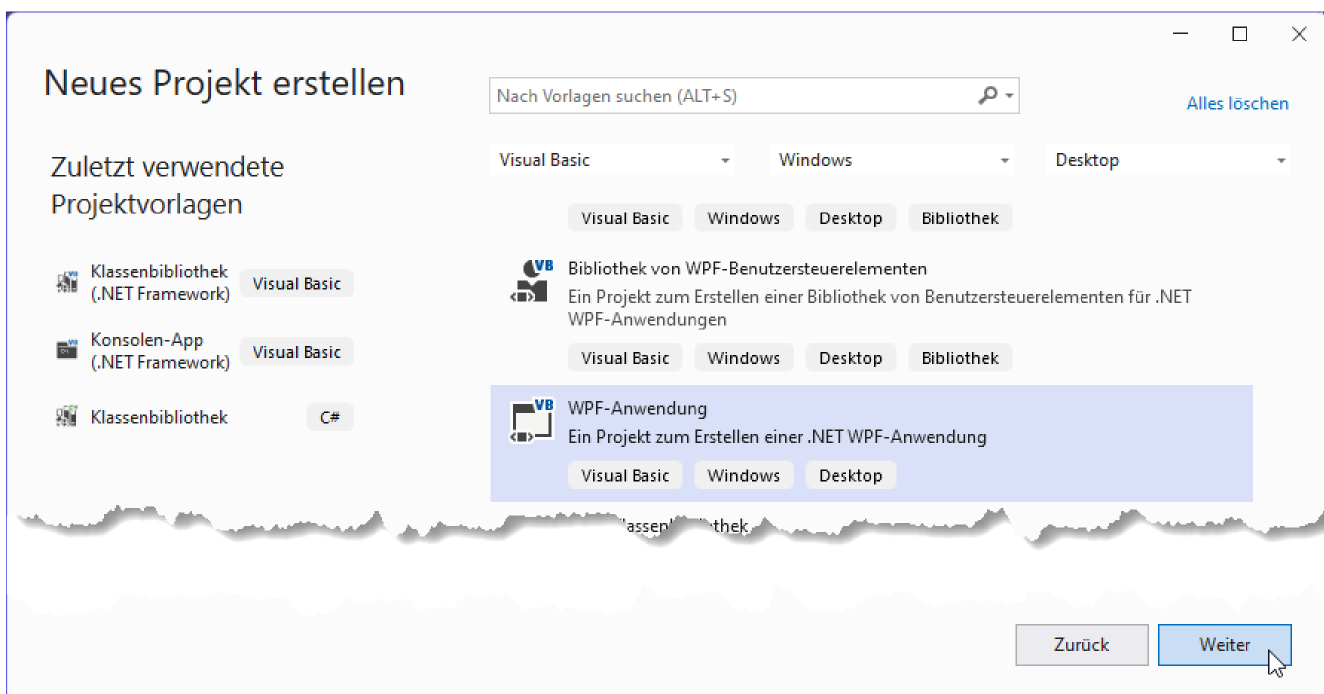

Bild 1: Anlegen einer WPF-Anwendung

für die Anwendung benötigen. Wir benötigen:

- **txtClientID**: Feld zum Eingeben der Client-ID
- **btnLoadClientData**: Öffnet einen Dateiauswahl-Dialog, um eine JSON-Datei mit Client-ID und Client-Secret auszuwählen, deren Daten in die beiden Felder **txtClientID** und **txtClientSecret** eingelesen werden.
- **txtClientSecret**: Feld zum Eingeben des Client-Secrets

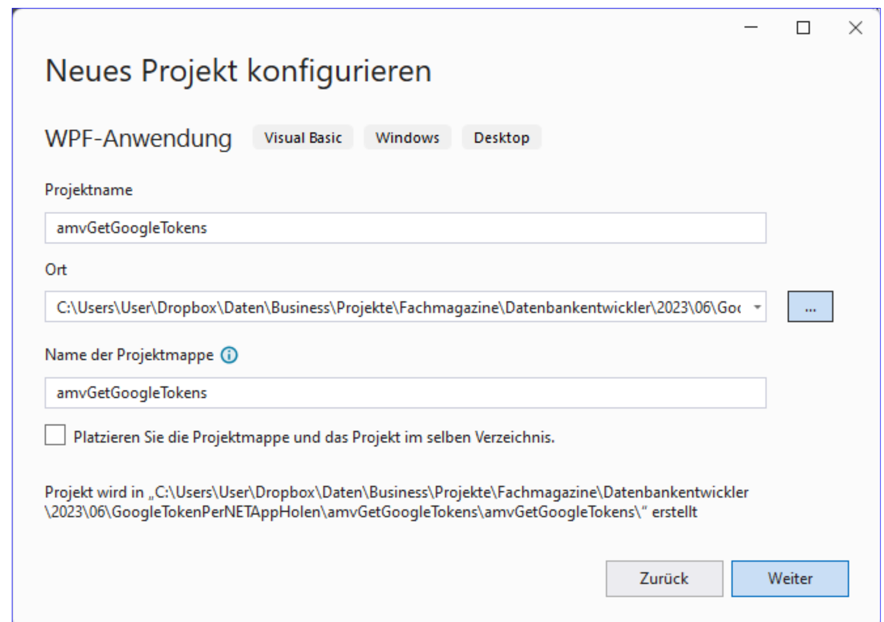

Bild 2: Konfigurieren des Projekts

- **txtRefreshToken**: Feld, in welches das ermittelte Refresh-Token geschrieben werden soll.
- **btnCopyRefreshToken**: Schaltfläche zum Kopieren des Refresh-Tokens in die Zwischenablage
- **txtAccessToken**: Feld, in welches das Access-Token geschrieben werden soll
- **btnCopyAccessToken**: Schaltfläche zum Kopieren des Access-Tokens in die Zwischenablage
- **txtExpiresAt**: Feld, in das wir das Verfallsdatum des Access-Tokens eintragen
- **btnGetTokens**: Schaltfläche zum Aktualisieren beider Token
- **btnRefreshToken**: Schaltfläche zum Aktualisieren des Access-Tokens mit dem Refresh-Token
- **btnOK**: Schaltfläche zum Beenden des Dialogs

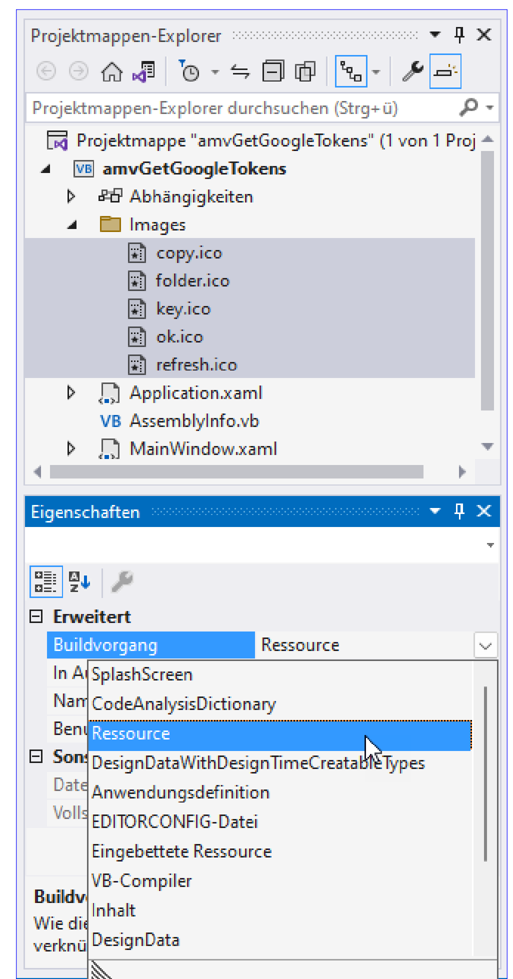

Bild 3: Einstellen der Bilddatei-Eigenschaften

Icons für die Schaltflächen

Damit wir die Schaltflächen klein halten können, verwenden wir, soweit möglich, Icons zur Beschreibung der Funktion. So können wir zum Beispiel für die Schaltflächen **btnCopyRefreshToken** und **btnCopyAccessToken** das übliche Icon für das Kopieren in die Zwischenablage verwenden.

Um Icons zu verwalten, fügen wir dem Projekt einen Ordner namens **Images** hinzu. In dieses ziehen wir die zu verwendenden Icons aus dem jeweiligen Ordner aus dem Windows Explorer hinein. Dazu können wir eine spezielle Instanz des Windows Explorers öffnen, aus dem wir sicher Dateien in das Projekt ziehen können. Dazu betätigen wir, während Visual Studio aktiviert ist, die Tastenkombination **Strg + O**. Dann stellen wir für die Elemente des Ordners **Images** noch die Eigenschaft **Buildvorgang** auf **Ressource** ein. Nur so werden diese in den Buildvorgang integriert und erscheinen auch in der Anwendung (siehe Bild 3).

Code des WPF-Fensters für die Anwendung

Neben den Schaltflächen soll auch das Hauptfenster selbst ein Icon erhalten – und eine entsprechende Beschriftung. Dazu stellen wir in dem Gerüst der Seite **MainWindow.xaml**, das beim Erstellen des Projekts automatisch angelegt wurde, für das Hauptelement **Window** die Attribute **Title** und **Icon** entsprechend ein (siehe Listing 1).

Danach folgt der Abschnitt **Windows.Resources**. Hier legen wir Standardwerte für einige Attribute bestimmter Steuerelemente fest. Danach folgt das **Grid**-Element, in dem wir im Abschnitt **Grid.RowDefinitions** die Zeilen definieren und im Abschnitt **Grid.ColumnDefinitions** die Spalten. Die Höhe der **RowDefinition**-Elemente legen wir mit dem Attribut **Height** jeweils auf **Auto** fest, damit diese sich an die Höhe der enthaltenen Steuerelemente anpasst. Einzige Ausnahme ist die letzte Zeile, welche wir auf * einstellen und die so den verbleibenden Platz einnimmt.

Die Breite der Spalten definieren wir mit dem Attribut **Width** der **ColumnDefinition**-Elemente. Die erste und die letzten Spalte sollen sich an der Breite der enthaltenen Steuerelemente orientieren, die mittlere soll den verbleibenden Platz einnehmen. Dadurch erstrecken sich die in der mittleren Spalte enthaltenen Textfelder, sofern nicht anders festgelegt, über die ge-

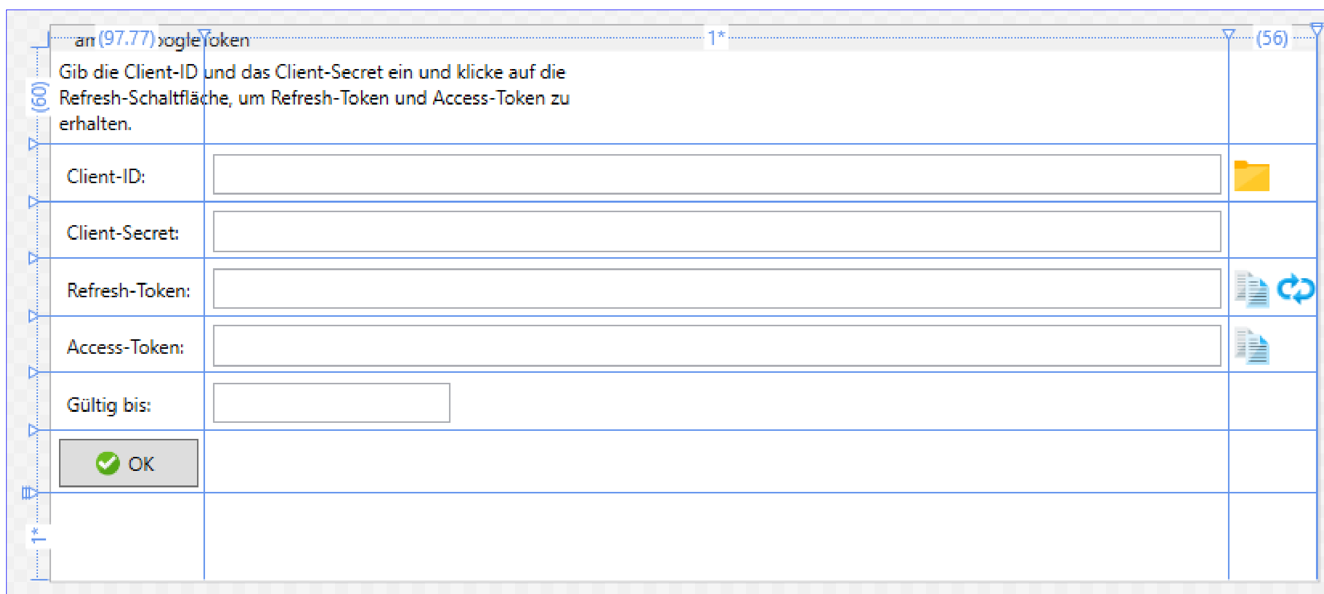

Bild 4: Entwurf des Hauptfensters

```
<Window x:Class="MainWindow"
        ...
        Title="amvGetGoogleToken" Height="350" Width="800" Icon="Images/key.ico">
    <Window.Resources>...</Window.Resources>
    <Grid>
        <Grid.RowDefinitions>...</Grid.RowDefinitions>
        <Grid.ColumnDefinitions>...</Grid.ColumnDefinitions>
        <TextBlock Grid.ColumnSpan="3" Height="50" Width="350" HorizontalAlignment="Left" TextWrapping="Wrap">Gib die
            Client-ID und das Client-Secret ein und klicke auf die Refresh-Schaltfläche, um Refresh-Token und Access-
            Token zu erhalten.</TextBlock>
        <Label Grid.Row="1">Client-ID:</Label>
        <TextBox x:Name="txtClientID" Width="Auto" Margin="5" Grid.Column="1" Grid.Row="1"></TextBox>
        <Button x:Name="btnLoadClientdata" Grid.Row="1" Grid.Column="2" HorizontalAlignment="left"
                Click="btnLoadClientdata_Click" Background="Transparent" BorderBrush="Transparent">
            <Image Source="Images/folder.ico" Width="24" Height="24"></Image>
        </Button>
        <Label Grid.Row="2">Client-Secret:</Label>
        <TextBox x:Name="txtClientSecret" Width="Auto" Margin="5" Grid.Row="2" Grid.Column="1"></TextBox>
        <Label Grid.Row="3">Refresh-Token:</Label>
        <TextBox x:Name="txtRefreshToken" Width="Auto" Margin="5" Grid.Row="3" Grid.Column="1"></TextBox>
        <StackPanel Grid.Row="3" Grid.Column="2" Orientation="Horizontal">
            <Button x:Name="btnCopyRefreshToken" Click="btnCopyRefreshToken_Click" Background="Transparent"
                    BorderBrush="transparent">
                <Image Source="Images/Copy.ico" Width="24" Height="24"></Image>
            </Button>
            <Button x:Name="btnRefresh" Click="btnRefresh_Click" Background="Transparent" BorderBrush="transparent">
                <Image Source="Images/refresh.ico" Width="24" Height="24"></Image>
            </Button>
        </StackPanel>
        <Label Grid.Row="4">Access-Token:</Label>
        <TextBox x:Name="txtAccessToken" Width="Auto" Margin="5" Grid.Row="4" Grid.Column="1"></TextBox>
        <Button x:Name="btnCopyAccessToken" Grid.Row="4" Grid.Column="2" HorizontalAlignment="left"
                Click="btnCopyRefreshToken_Click" Background="Transparent" BorderBrush="Transparent">
            <Image Source="Images/Copy.ico" Width="24" Height="24"></Image>
        </Button>
        <Label Grid.Row="5">Gültig bis:</Label>
        <TextBox x:Name="txtExpiresAt" Width="150" HorizontalAlignment="left" Grid.Row="5" Grid.Column="1"></TextBox>
        <Button x:Name="btnOK" Grid.Row="6" Click="btnOK_Click" Margin="5">
            <StackPanel Orientation="Horizontal">
                <Image Source="Images/ok.ico" Width="16" Height="16"></Image>
                <Label Padding="0">OK</Label>
            </StackPanel>
        </Button>
    </Grid>
</Window>
```

Listing 1: Definition des Hauptfensters der Anwendung

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-**

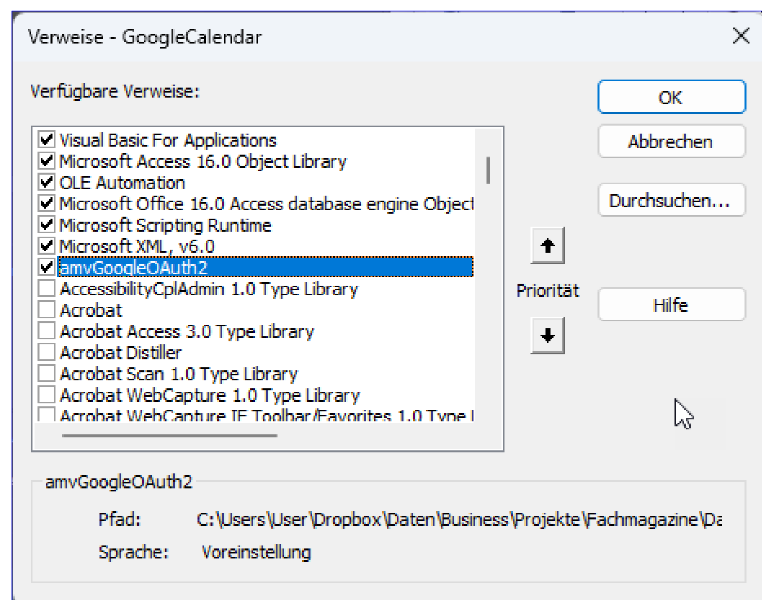

Bild 1: Verweise für die Lösung

crosoft Scripting Runtime und **Microsoft XML, v6.0** (siehe Bild 1). Dies ist das Verweise-Fenster des VBA-Editors, unter twinBASIC sieht dies etwas anders aus.

Außerdem fügen wir dem Projekt die beiden Module **mdlJSON** und **mdlJSONDOM** hinzu. Diese enthalten die Elemente zum Parsen und Lesen der gelieferten JSON-Dokumente.

Im Modul **mdlClipboard** finden wir die Funktion **InZwischenablage**, die immer praktisch ist, um längere JSON-Dokumente in die Zwischenablage zu kopieren, um diese dann beispielsweise formatiert in Visual Studio einzufügen und anzuzeigen.

Das Modul **mdlTools** enthält aktuell nur die Funktion **SQLDatum**, mit dem wir Datumsangabe in das SQL-Format übertragen können.

Das Modul **mdlRegistry** enthält einige Funktionen, mit denen wir Daten wie die ClientID, das ClientSecret, das Access-Token und das Refresh-Token in der Registry speichern und wieder auslesen können.

Diese beschreiben wir im Detail im Artikel **Anwendungsdaten in der Registry** (**www.vbentwickler.de/411**).

Das Modul **mdlAuthentication** enthält die Prozeduren, mit denen wir das Access-Token für die Authentifizierung unserer Anfragen an die Rest-API von Google generieren.

Das Modul **mdlGoogleCalendar** enthält schließlich die eigentlichen Prozeduren, um auf die Daten des Google-Kalenders zuzugreifen.

Client-ID und Client-Secret holen und speichern

Im Artikel **Google Calendar programmieren: Vorbereitungen** (**www.vbentwickler.de/408**) beschreiben wir, wie Du die Client-ID und das Client-Secret über die Webseite von Google holst und dazu auch eine App anlegst. Die beiden Daten sind wichtige Voraussetzungen für das Erhalten eines Access-Tokens und damit letztlich für den Zugriff auf die Rest-API von Google.

Im Modul **mdlAuthentication** findest Du eine Prozedur namens **SetClientData**, mit der Du die gewonnenen Daten für Client-ID und Client-Secret in die Registry schreibst, von wo weitere Prozeduren sich diese Informationen im weiteren Verlauf holen. Wir können diese zwar auch in Form von Konstanten direkt im Modul speichern, aber da die weiteren benötigten Informationen wie das Access-Token und das Refresh-Token sich gelegentlich ändern und wir diese deshalb in der Registry speichern wollen, legen wir dort auch gleich Einträge für Client-ID und Client-Secret an.

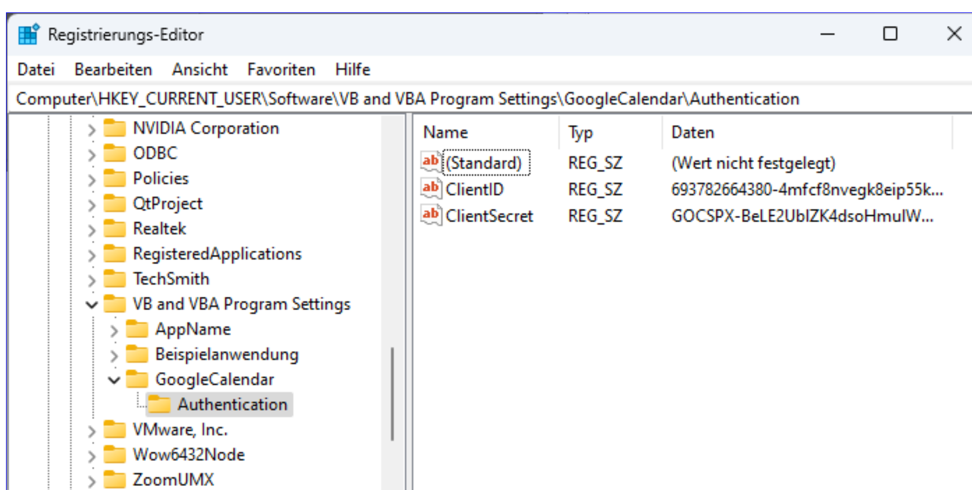

Bild 2: Informationen in der Registry

Wir tragen die entsprechenden Werte also in den folgenden Zeilen für die Platzhalter **xxxxxx** und **yyyyyy** ein und rufen dann die Prozedur auf:

```
Public Sub SetClientData()
    SaveAppSetting "ClientID", "xxxxxx"
    SaveAppSetting "ClientSecret", "yyyyyy"
End Sub
```

Diese landen dann wie in Bild 2 in der Registry.

Token für den Zugriff auf die Rest-API holen

Google liefert zwei Arten von Token:

- Access-Token: Das eigentliche Token, welches wir für den Zugriff auf die Rest-API von Google nutzen. Dieses ist in der Regel nur für 60 Minuten gültig.

- Refresh-Token: Langlebiges Token, mit dem wir das Access-Token aktualisieren können.

Mit der Prozedur **GetTokens** holen wir beide Token und speichern diese in weiteren Einträgen in der Registry. Die Prozedur ermittelt als Erstes die Werte der Registry-Einträge **ClientID** und **ClientSecret** und gibt entsprechende Meldungen aus, wenn diese nicht aus der Registry eingelesen werden können.

Sind beide Werte vorhanden, ruft die Prozedur eine weitere Prozedur namens **GetTokensDLL** auf. Diese erwartet die Client-ID, das Client-Secret und weitere leere Variablen als Parameter, die mit dem Access-Token, dem Refresh-Token und der Haltbarkeit des Access-Tokens in Sekunden gefüllt werden sollen. Liefert diese Funktion den Wert **True** zurück, trägt die aufrufende Prozedur die Werte der nun gefüllten Parameter **strRefreshToken**, **strAccessToken** und **lngExpiresInSeconds** unter den Namen **RefreshToken**, **AccessTo-**

```
Public Sub GetTokens()
    Dim strClientID As String
    Dim strClientSecret As String
    Dim strRefreshToken As String
    Dim strAccessToken As String
    Dim lngExpiresInSeconds As Long
    strClientID = GetAppSetting("ClientID")
    If Len(strClientID) = 0 Then
        MsgBox "Keine ClientID in der Registry gefunden. Verwende die Prozedur SetClientData.", vbOKOnly + _
            vbExclamation, "ClientID fehlt"
            Exit Sub
    End If
    strClientSecret = GetAppSetting("ClientSecret")
    If Len(strClientSecret) = 0 Then
        MsgBox "Kein ClientSecret in der Registry gefunden. Verwende die Prozedur SetClientData.", vbOKOnly + _
            vbExclamation, "ClientSecret fehlt"
           Exit Sub
    End If
    If GetTokensDLL(strClientID, strClientSecret, strRefreshToken, strAccessToken, lngExpiresInSeconds) Then
        SaveAppSetting "RefreshToken", strRefreshToken
        SaveAppSetting "AccessToken", strAccessToken
        SaveAppSetting "TokenExpiresAt", DateAdd("s", lngExpiresInSeconds, Now)
    End If
End Sub
```

Listing 1: Die Prozedur **GetTokens** ruft die Funktion **GetTokensDLL** auf.

```
Public Sub GetTokens()
    Dim strClientID As String
    Dim strClientSecret As String
    Dim strRefreshToken As String
    Dim strAccessToken As String
    Dim lngExpiresInSeconds As Long
    strClientID = GetAppSetting("ClientID")
    If Len(strClientID) = 0 Then
        MsgBox "Keine ClientID in der Registry gefunden. Verwende die Prozedur SetClientData.", vbOKOnly + _
            vbExclamation, "ClientID fehlt"
    End If
    strClientSecret = GetAppSetting("ClientSecret")
    If Len(strClientSecret) = 0 Then
        MsgBox "Kein ClientSecret in der Registry gefunden. Verwende die Prozedur SetClientData.", vbOKOnly + _
            vbExclamation, "ClientSecret fehlt"
    End If
    If GetTokensDLL(strClientID, strClientSecret, strRefreshToken, strAccessToken, lngExpiresInSeconds) Then
        SaveAppSetting "RefreshToken", strRefreshToken
        SaveAppSetting "AccessToken", strAccessToken
        SaveAppSetting "TokenExpiresAt", DateAdd("s", lngExpiresInSeconds, Now)
    End If
End Sub
```

Listing 2: Die Funktion **GetTokensDLL** ermittelt per DLL das Access-Token, das Refresh-Token und die Gültigkeitsdauer.

ken und **TokenExpiresAt** in die Registry ein. Dabei berechnet sie für den Eintrag **TokenExpiresAt** die aktuelle Zeit plus die Anzahl der Sekunden aus **lngExpiresInSeconds** (siehe Listing 1).

Die Funktion **GetTokensDLL** deklariert eine Objektvariable auf Basis der Klasse **amvGoogleOAuth2.GoogleOAuth2** aus der per Verweis verfügbar gemachten DLL **amvGoogleOAuth2** und erstellt eine neue Instanz dieser Klasse (siehe Listing 2).

Sie weist den Eigenschaften **ClientId** und **ClientSecret** die Werte aus **strClientID** und **strClientSecret** zu und ruft dann die Funktion **GetTokens** auf, um das Refresh-Token, das Access-Token und die Gültigkeitsdauer des Access-Tokens zu ermitteln.

War dieser Aufruf erfolgreich, liefert die Funktion **GetTokensDLL** den Wert **True** zurück, sodass die aufrufende Prozedur **GetTokens** die Werte der Parameter

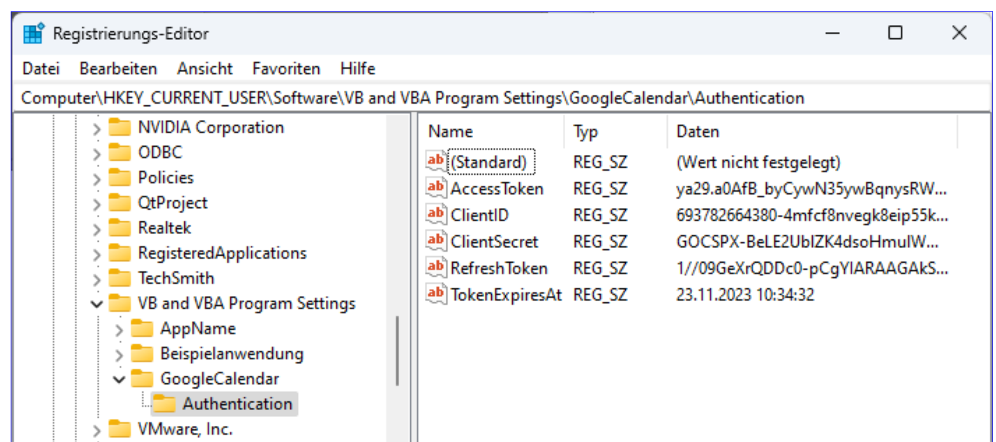

Bild 3: Vervollständigung der Informationen in der Registry

```
Public Sub RefreshAccessToken()
    Dim strClientID As String
    Dim strClientSecret As String
    Dim lngExpiresIn As Long
    Dim strRefreshToken As String
    Dim strAccessToken As String
    Dim lngStatus As Long
    Dim strErrorTitle As String
    Dim strErrorDescription As String
    strClientID = GetAppSetting("ClientID")
    strClientSecret = GetAppSetting("ClientSecret")
    strRefreshToken = GetAppSetting("RefreshToken")
    If RefreshAccessTokenRestAPI(strClientID, strClientSecret, strRefreshToken, strAccessToken, lngExpiresIn, _
            lngStatus, strErrorTitle, strErrorDescription) = True Then
        SaveAppSetting "AccessToken", strAccessToken
        SaveAppSetting "TokenExpiresAt", DateAdd("s", lngExpiresIn, Now)
    Else
        MsgBox "Status: " & lngStatus & vbCrLf & vbCrLf & "Fehler: " & strErrorTitle & vbCrLf & vbCrLf _
            & "Beschreibung: " & strErrorDescription, vbCritical + vbOKOnly, "Fehler bei Refresh Token"
    End If
End Sub
```

Listing 3: Die Prozedur **RefreshAccessToken** startet die Aktualisierung des Access-Tokens.

strRefreshToken, **strAccessToken** und **lngExpiresIn-Seconds** wie oben beschrieben in der Registry speichern kann (siehe Bild 3).

Access-Token aktualieren

Das Access-Token ist genau eine Stunde gültig, und die Stunde ist manchmal schneller herum als man denkt. Deshalb benötigen wir noch eine weitere Funktion, mit der wir das Access-Token aktualisieren können. Diese benötigt nicht mehr die COM-DLL, sondern wir kommen hier mit einem direkten Aufruf der Rest-API von Google aus.

Wir verwenden wieder eine Prozedur und eine Funktion, wobei die Funktion den eigentlichen Zugriff auf die Rest-API kapselt. Die erste Prozedur heißt **RefreshAccessToken** und kümmert sich um das Zusammenstellen der erforderlichen Daten und um die Auswertung des Ergebnisses der Funktion **RefreshAccessTokenRestAPI**. Sie ermittelt als Erstes die Werte für die Client-ID, das Client-Secret und das aktuelle Refresh-Token aus der Registry (siehe Listing 3).

Dann ruft die die Funktion **RefreshAccessTokenRest-API** auf und übergibt einige Parameter, die teilweise erst von dieser Funktion gefüllt werden. Liefert dieser Aufruf den Wert **True** zurück, werden die neuen Werte für die Registry-Einträge **AccessToken** und **Token-ExpiresAt** gespeichert. Falls nicht, wertet die Funktion die zurückgegebenen Informationen aus und liefert eine Meldung wie beispielsweise in Bild 4.

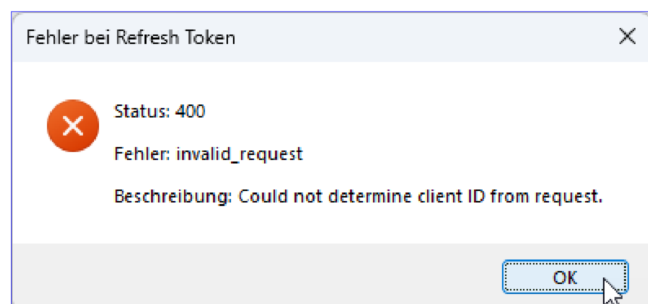

Bild 4: Fehlermeldung

Funktion zum Aktualisieren des Access-Tokens per Rest-API

Die Funktion **RefreshAccessTokenRestAPI** aus Listing 4 erstellt eine **ServerXMLHTTP60**-Objekt und legt diverse Einstellungen für den Aufruf der Rest-API fest. Dazu gehören zum Beispiel der Content-Type, die Methode (**POST**) und die zu verwendende URL, die wie folgt lautet:

```
https://accounts.google.com/o/oauth2/token
```

```
Public Function RefreshAccessTokenRestAPI(strClientID As String, strClientSecret As String, _
        strRefreshToken As String, strAccessToken As String, lngExpiresIn As Long, Optional lngStatus As Long, _
        Optional strErrorTitle As String, Optional strErrorDescription As String) As Boolean
    Dim objHTTP As ServerXMLHTTP60
    Set objHTTP = New MSXML2.ServerXMLHTTP60
    Dim strData As String
    Dim strURL As String
    Dim strMethod As String
    Dim strContentType As String
    Dim strResponse As String
    Dim objJSON As Object
    strContentType = "application/x-www-form-urlencoded"
    strMethod = "POST"
    strURL = "https://accounts.google.com/o/oauth2/token"
    strData = "client_id=" & strClientID & "&client_secret=" & strClientSecret & "&refresh_token=" & strRefreshToken _
        & "&grant_type=refresh_token&scope=https://www.googleapis.com/auth/calendar"
    With objHTTP
        .Open strMethod, strURL, False
        .setRequestHeader "Accept", strContentType
        .setRequestHeader "Content-Type", strContentType
        .send strData
        strResponse = .responseText
        Select Case .status
            Case 200
                RefreshAccessTokenRestAPI = True
                Set objJSON = ParseJson(strResponse)
                strAccessToken = objJSON.Item("access_token")
                lngExpiresIn = objJSON.Item("expires_in")
            Case Else
                RefreshAccessTokenRestAPI = False
                Set objJSON = ParseJson(strResponse)
                lngStatus = .status
                strErrorTitle = objJSON.Item("error")
                strErrorDescription = objJSON.Item("error_description")
        End Select
    End With
End Function
```

Listing 4: Die Prozedur **RefreshAccessTokenRestAPI** aktualisiert das Access-Token.

Außerdem stellt sie in **strData** die Parameter zusammen, die an die Rest-API übergeben werden sollen – die Client-ID, das Client-Secret und das Refresh-Token sowie den Typ **refresh_token**, der angibt, dass wir das Access-Token aktualisieren wollen.

Und als **scope** beziehungsweise Gültigkeitsbereich geben wir **https://www.googleapis.com/auth/calendar** an, was den lesenden und schreibenden Zugriff auf den kompletten Kalender erlaubt. Hier können wir auch andere Werte wie hier angegeben eintragen:

```
https://developers.google.com/calendar/api/auth?hl=de
```

Danach verwendet die Funktion die **Open**-Methode des **ServerXMLHTTP60**-Objekts und übergibt die URL. Dann stellt sie verschiedene **Header**-Eigenschaften ein und ruft schließlich die **send**-Methode auf und übergibt den Inhalt von **strData**. Das Ergebnis liefert das **ServerXMLHTTP60**-Objekt mit der Eigenschaft **Response**.

Daneben erhalten wir den Status des Aufrufs. Lautet dieser **200**, hat der Aufruf funktioniert und wir können den das Ergebnis aus **strResponse** mit der **ParseJSON**-Funktion auslesen und in das Objekt **objJSON** schreiben. Auf dieses greifen wir dann über die entsprechende Syntax zu – mehr zum Umfang mit JSON-Dateien unter **Mit JSON arbeiten** (**www.vbentwickler.de/361**). Dabei füllen wir die Rückgabeparameter **strAccessToken** und **lngExpiresIn**. Die Funktion liefert in diesem Fall den Wert **True** zurück.

Falls der Aufruf nicht den Status **200** zurückliefert, soll die Funktion **RefreshAccessTokenRestAPI** den Wert **False** zurückliefern. Außerdem lesen wir die Elemente **error** und **error_description** aus **strResponse** aus und schreiben diese in die Rückgabeparameter **strErrorTitle** und **strErrorDescription**.

Damit haben wir die beiden grundlegenden Aufgaben zum Ermitteln des Refresh-Tokens und des Access-Tokens abgehakt.

```
Public Function HTTPRequest(strURL As String, strAuthorization As String, Optional strMethod As String = "POST", _
        Optional strContentType As String = "application/json", Optional strData As String, _
        Optional strResponse As String) As Integer
    Dim objHTTP As ServerXMLHTTP60
    Set objHTTP = New MSXML2.ServerXMLHTTP60
    With objHTTP
        .Open strMethod, strURL, False
        .setRequestHeader "Accept", strContentType
        .setRequestHeader "Content-Type", strContentType
        .setRequestHeader "Authorization", strAuthorization
        If Not Len(strData) = 0 Then
            .setRequestHeader "Body", strData
        End If
        .send strData
        strResponse = .responseText
        HTTPRequest = .status
    End With
End Function
```

Listing 5: Grundlagende Funktion zum Aufruf der Rest-API für den Zugriff auf die Kalender und ihre Elemente