Tutorial

2

EDV INNOVATION & CONSULTING

Stefan Kulpa

Gerhard-Domagk-Str. 6
D-41540 Dormagen
http://www.kulpa-online.com
Email: stefan@kulpa-online.com

VBA

EINFÜHRUNG

LEVEL INTERMEDIATE

Inhaltsverzeichnis

Kapitel 1

EFFEKTIVES DATEI-HANDLING, ODER: VIELE WEGE FÜHREN NACH ROM

Neben der Nutzung von Datenbanken stellt der Zugriff auf (lokalen) Dateien die wohl häufigste Art der Datenspeicherung dar. Streng genommen handelt es sich bei Datenbanken letztendlich auch nur um Dateien, nur deren Verwaltung ist deutlich komplexer und nicht ohne „Hilfstechnologien“ zu bewerkstelligen.

Für den Zugriff auf Dateien stehen uns im Wesentlichen drei Technologien zur Verfügung:

- Zugriff mit VB/A-Hausmitteln
- Zugriff mit dem „Windows Scripting Host“ (nachfolgend WSH genannt)
- Zugriff mit der Win32-API

Alle drei Varianten haben ihr Vor- und Nachteile, die letztendlich von der zu lösenden Aufgabe abhängig sind. Eine pauschale Bewertung ist daher nicht möglich.

Bevor man eine Datei bearbeiten kann, muss man sie ggf. suchen. Wir beginnen also damit, einen vorgegebenen Ordner nach bestimmten Dateien zu durchsuchen und das Ergebnis anzuzeigen.

Aufgabe: Suche alle INI-Dateien im Windows-Ordner.

Zunächst müssen wir ermitteln, welches der Windows-Ordner ist.

Unter Windows NT (2000, XP) könnte man voraussetzen, dass es sich um den Ordner C:\WINNT handelt, unter Windows 95 bzw. Windows Me müsste es sich um den Ordner C:\Windows handeln.

Jetzt könnte man versuchen, dass Betriebssystem zu ermitteln und danach einfach annehmen, dass o.g. Pfade je nach Betriebssystem gültig sind. Das ist jedoch nicht zu empfehlen, da sich beispielsweise Windows XP auch auf einem anderen Laufwerk als C: installieren lässt.

Das Windows-Verzeichnis lässt sich zuverlässig auf zwei verschiedene Art und Weisen ermitteln, mithilfe des WSH und mithilfe des Win32-API.

Um mit dem WSH arbeiten zu können, ist es notwendig, in seinem VBA Projekt einen Verweis auf die entsprechende Bibliothek zu setzen. Dazu muss man im VBA-Editor-Menü **EXTRAS|VERWEISE** auf die **SCRRUN.DLL** im System32-Verzeichnis verweisen.

Hierdurch wird ein Verweis **MICROSOFT SCRIPTING RUNTIME** in die Liste der Verweise aufgenommen bzw. angehakt. Dieser Verweis sollte fortfolgend für diesen Workshop gesetzt sein!

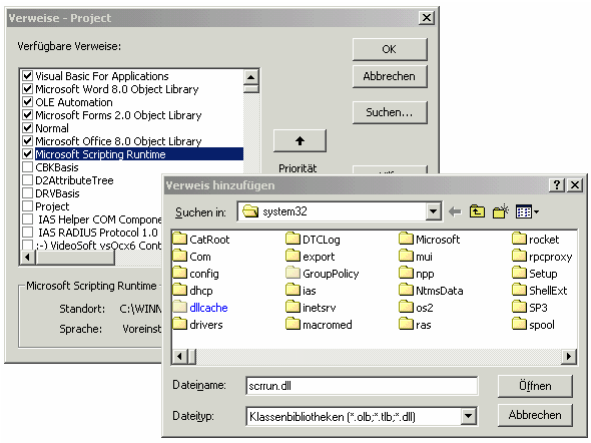

Abbildung 1 - Verweis auf Scrrun.dll (WSH)

Windows-Verzeichnis mithilfe des WSH ermitteln

Das Objektmodell des Windows Scripting Host ist sehr vielschichtig. Für unsere Belange benötigen wir (zunächst) das sog. FileSystemObject, mit dessen Hilfe wir das Windows-Verzeichnis ermitteln wollen.

```
Function WSH_GetWindowsDir() As String

    Dim objFso As New FileSystemObject

    WSH_GetWindowsDir = objFso.GetSpecialFolder(WindowsFolder)

End Function
```

Abbildung 2 - Windows-Verzeichnis mit WSH ermitteln

Neu ist hier ggf. die Objekterstellung über das Schlüsselwort **New**; dies ist bei manchen Objekten notwendig.

Nachdem wir ein FileSystemObject-Objekt erzeugt haben, nutzen wir dessen Methode **GetSpecialFolder** und dem Argument **WindowsFolder**, um die gewünschte Information zu erhalten. Bei dem Argument **WindowsFolder** handelt es sich um eine Konstante mit dem Wert 0.

Hinweis: Es ist bei der Nutzung des WSH grundsätzlich anzuraten, einen Blick in den Objektbrowser **(F2)** zu werfen, um herauszufinden, welche Routinen und Konstanten zur Verfügung stehen.

Windows-Verzeichnis mithilfe des Win32-API ermitteln

Das Win32-API ist nicht VBA! Man könnte es eher mit C/C++ vergleichen und leider gelten deshalb auch „andere“ Regeln. Das fängt bereits damit an, dass C/C++ (nativ) eigentlich keine Strings kennt – traurig aber wahr. Wir müssen bei der Nutzung des Win32-API die „heile und unbeschwerte Welt des VBA“ verlassen und in die „Abgründe der C-Welt“ hinabsteigen.

Das Win32-API besteht aus einer Vielzahl (besser: Unmenge) von Funktionen, die im Betriebssystem fest implementiert sind. Man benutzt de facto die gleichen Funktionen wie das Betriebssystem selbst. Der Großteil der Funktionen ist auf eine Handvoll sog. Bibliotheksdateien (DLLs) verteilt (kernel32.dll, user32.dll, gdi32.dll etc.). Wir müssen „nur noch“ wissen, wo sich die jeweilige Funktion befindet und wie man sie nutzt.

Über das Win32-API (und auch andere API-Varianten) wurden dicke Bücher geschrieben. Dieses Know-how lässt sich im Rahmen dieses Workshops nicht mal annähernd erreichen. Daher mag nachfolgend einiges im Unklaren bleiben und man ggf. in Büchern oder im Internet „nachlesen“ muss – sorry!

Grundsätzlich müssen sämtliche Win32-API-Funktionen auf Modulebene deklariert werden. Dabei wird das „Grundgerüst“ der Funktion genutzt; z.B.:

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias _
                 "GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Abbildung 3 - Grundgerüst der API-Deklaration

Bevor nun jemand versucht, diese Deklaration abzuschreiben: Stopp! Kein Mensch tippt API-Deklarationen ab. Zum einen ist das hochgradig fehleranfällig (API-Funktionen sind case-sensitive!) und zum anderen gibt es genügend Referenzlisten (s. auch Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**).

Die Funktion **GetWindowsDirectory** entstammt der Bibliothek **Kernel32.dll** und erwartet 2 Argumente: **lpBuffer** und **nSize**. Zu beachten ist hierbei, dass fast alle Argumente bei API-Funktionen als Kopie (ByVal) übergeben werden.

Ohne Sekundärliteratur ist man jetzt schon fast verloren, denn was mag in den Argumenten stehen?

- ➔ Das Argument **lpBuffer** entspricht einem **dimensionierten** String für die **Rückgabe** des gesuchten Ordners!
- ➔ Das Argument **nSize** beinhaltet die Länge des Strings **lpBuffer**!

Da es sich (wie fast alle API-Routinen) um eine Funktion handelt, gibt sie auch etwas zurück.

In diesem Fall gibt die Funktion die Länge des Strings zurück, die für die Angabe des Windows-Ordners benötigt wurde (oder auch nicht...). War der String „zu kurz“ dimensioniert, erhalten wir die Länge zurück, die wir hätten benutzen müssen. Kam es jedoch zu einem Fehler, erhalten wir lediglich 0 als Rückgabe; sehr aussagekräftig…

Was ist also zu tun?

- Wir müssen eine String-Variable deklarieren.
- Diese Stringvariable muss entsprechend groß dimensioniert werden.
- Wir übergeben der Funktion **GetWindowsDirectory** den dimensionierten String nebst Längenangabe.
- Aus dem übergebenen String extrahieren wir die Anzahl Zeichen, die die Funktion als Rückgabewert meldet.

Ganz einfach, oder?

Neu ist hier das Dimensionieren von Strings, was innerhalb von VB/A unnötig ist. Wir müssen uns in der Regel nicht um Stringlängen kümmern und diese schon gar nicht explizit setzen.

Um nun eine String-Variable „aufzublasen“, müssen wir sie mit entsprechend vielen Zeichen „befüllen“. Theoretisch ist es egal mit welchem Zeichen dies erfolgt, aber es hat sich zum Quasi-Standard entwickelt, dies mit Leerzeichen oder mit dem ASCII-Zeichen für 0 zu tun.

Um von vornherein die richtige Länge nicht dem Zufall zu überlassen, gibt es auch hier eine Faustregel. Windows erlaubt grundsätzlich nur Pfadlängen von maximal 255 Zeichen. Also „blasen“ wir unseren String auf 260 Zeichen auf, und es kann nichts mehr geschehen.

Dieses Dimensionieren kann man nun C-like über eine Schleife durchführen, in dem wir einen String in 260 Schleifendurchläufen befüllen...

```
Dim sBuffer As String
Dim iLoop   As Integer

For iLoop = 1 To 260
    sBuffer = sBuffer & " "
Next
```

Abbildung 4 - Dimensionierung des String-Variable

...oder wir nutzen die verfügbaren VBA-Hausmittel.

Die VBA-Funktionen **Space()** und **String()** helfen uns dabei, recht einfach das gewünschte Ergebnis zu erhalten.

```
Dim sBuffer As String

sBuffer = Space(260)
'bzw.
sBuffer = String(260, 0)
```

Abbildung 5 - Einsatz der Space- bzw. String-Funktion

Also kann es losgehen:

```
Function API_GetWindowsDir() As String

    Dim sBuffer As String
    Dim lResult As Long

    sBuffer = Space(260)
    lResult = GetWindowsDirectory(sBuffer, Len(sBuffer))
    If lResult <> 0 Then
        API_GetWindowsDir = Left$(sBuffer, lResult)
    End If

End Function
```

Abbildung 6 - Windows-Verzeichnis mit API ermitteln

Ganz einfach, oder?

Nachdem wir jetzt herausgefunden haben, wo sich der Windows-Ordner befindet, können wir versuchen, alle INI-Dateien in diesem Ordner zu suchen. Dies erfolgt mittels VBA-Hausmitteln, dem WSH und dem Win32-API.

Dateien mithilfe von VBA ermitteln

Für diesen Zweck stellt uns VBA den **Dir()**-Befehl zur Verfügung.

Dir-Funktion

Siehe auch Beispiel Zusatzinfo

Gibt eine Zeichenfolge (**String**) zurück, die den Namen einer Datei, eines Verzeichnisses oder eines Ordners darstellt, der mit einem bestimmten Suchmuster, einem Dateiattribut oder mit der angegebenen Datenträger- bzw. Laufwerksbezeichnung übereinstimmt.

Syntax

Dir[(*Pfadname*[, *Attribute*])]

Die Syntax der **Dir**-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
Pfadname	Optional. Zeichenfolgenausdruck, der einen Dateinamen angibt. Der Dateiname kann ein Verzeichnis oder einen Ordner sowie ein Laufwerk enthalten. Eine Null-Zeichenfolge ("") wird zurückgegeben, wenn der Pfad aus *Pfadname* nicht gefunden werden kann.
Attribute	Optional. Eine Konstante oder ein numerischer Ausdruck, deren Summe die Dateiattribute angibt. Wenn das Argument nicht angegeben wird, werden alle Dateien, die dem Argument *Pfadnamen* entsprechen, zurückgegeben.

Einstellungen

Das Argument *Attribute* hat die folgenden Einstellungen:

Konstante	Wert	Beschreibung
vbNormal	0	Normal.
vbHidden	2	Versteckt.
vbSystem	4	Systemdatei.
vbVolume	8	Datenträgerbezeichnung. Falls angegeben, werden alle anderen Attribute ignoriert.
vbDirectory	16	Verzeichnis oder Ordner.

Anmerkung Diese Konstanten werden durch Visual Basic für Applikationen definiert und können beliebig im Code anstelle der tatsächlichen Werte verwendet werden.

Abbildung 7 - Auszug aus der OL-Hilfe: Dir-Funktion

Diese Routine schaut dann schon ein bissel umfangreicher aus:

```
Sub VBA_GetINIFiles()

    Dim sFolder     As String           'Variable für das Windows-Verzeichnis
    Dim sFile       As String           'Variable für die Dateisuche
    Dim asFiles()   As String           'Datenfeld für die Ergebnisdateien
    Dim lCounter    As Long             'Zählervariable

   'Windows-Pfad ermitteln
    sFolder = WSH_GetWindowsDir()
   'Endenden Backslash überprüfen und ggf. hinzufügen
    If Right$(sFolder, 1) <> "\" Then sFolder = sFolder & "\"
   'Zählervariable initialisieren
    lCounter = 0
   'Erster Suchbefehl für die Dir-Funktion
    sFile = Dir(sFolder & "*.ini")      'Ersten Eintrag abrufen
   'Wenn es einen Treffer gibt, diesen in dem Array speichern und weitersuchen
    If Len(sFile) > 0 Then
       'Zählervariable für das Array vergrößern
        lCounter = lCounter + 1
       'Array um 1 Eintrag erweitern (Re-Dimensionieren)
        ReDim Preserve asFiles(lCounter)
       'In dem neuen Array-Eintrag den ersten Dateinamen speichern
        asFiles(lCounter - 1) = sFile
       'In einer Schleife solange weitersuchen, bis es keine Treffer mehr gibt
        Do While sFile <> ""            'Schleifenstart
            sFile = Dir                 'Nächsten Eintrag abrufen
            If Len(sFile) > 0 Then
               'Zählervariable für das Array vergrößern
                lCounter = lCounter + 1
               'Array um 1 Eintrag erweitern (Re-Dimensionieren)
                ReDim Preserve asFiles(lCounter)
               'In dem neuen Array-Eintrag den ersten Dateinamen speichern
                asFiles(lCounter - 1) = sFile
            End If
        Loop
    End If
   'Ergebnistest - alle Array-Einträge im Direktfenster ausgeben
    For lCounter = LBound(asFiles) To UBound(asFiles)
        sFile = asFiles(lCounter)
        If Len(sFile) > 0 Then Debug.Print "Eintrag"; lCounter; ": "; sFile
    Next

End Sub
```

Abbildung 8 - Dateien mithilfe von VBA ermitteln

Hinweis zur Redimensionierung von Arrays: wenn Arrays dynamisch vergrößert werden sollen, erfolgt dies mithilfe des Befehls **ReDim**. Dabei gehen jedoch die Inhalte der bestehenden Array-Einträge verloren. Um dies zu verhindern, muss das Schlüsselwort **Preserve** benutzt werden, um neben der Vergrößerung des Arrays auch die bisherigen Inhalte zu erhalten.

Dateien mithilfe des WSH ermitteln

Im Rahmen des Windows Scripting Hosts benötigen wir ein paar Objekte aus dem WSH. Das Erfragen des Windows-Ordners erfolgt sinnvollerweisen direkt in der Funktion, da wir eh auf das WSH zugreifen.

```
Sub WSH_GetINIFiles()

    Dim objFso      As Scripting.FileSystemObject
    Dim objFld      As Scripting.Folder
    Dim objFile     As Scripting.File

    Dim asFiles()   As String           'Datenfeld für die Ergebnisdateien
    Dim lCounter    As Long             'Zählervariable


    Set objFso = New Scripting.FileSystemObject
    Set objFld = objFso.GetSpecialFolder(WindowsFolder)

    For Each objFile In objFld.Files
    '// ---------------------------------------------------------------------------
    '// Die Funktion GetExtensionName gibt uns die Dateierweiterung zurück, nur kann
    '// diese mal ini und mal INI sein. Ein direkter Vergleich kann also fehlschlagen!
    '// Daher nutzen wir die VBA-Funktion StrComp, um ein korrektes Ergebnis zu erhalten
    '// ---------------------------------------------------------------------------
        If StrComp(objFso.GetExtensionName(objFile.Path), "ini", vbTextCompare) = 0 Then
        '// -----------------------------------------------------------------------
        '// Bei einem Treffer inkrementieren wir die Zählervariable, re-dimensionieren
        '// das Datenfeld um einen weiteren Eintrag und beschreiben diesen mit dem
        '// aktuell gefundenen Wert (INI-Datei).
        '// -----------------------------------------------------------------------
            lCounter = lCounter + 1
            ReDim Preserve asFiles(lCounter)
            asFiles(lCounter - 1) = objFile.Name
        End If
    Next
   'Ergebnistest - alle Array-Einträge im Direktfenster ausgeben
    For lCounter = LBound(asFiles) To UBound(asFiles)
        If Len(asFiles(lCounter)) > 0 Then
            Debug.Print "Eintrag"; lCounter; ": "; asFiles(lCounter)
        End If
    Next

End Sub
```

Abbildung 9 - Dateien mithilfe von WSH ermitteln

Diese Variante ist schon deutlich schlanker als die **Dir()**-Variante. Ein Nachteil ist, dass wir sämtliche Dateien im Windows-Ordner nach deren Endung untersuchen müssen. Der **Dir()** Befehl nimmt uns diese Arbeit ab – und ist daher in der Regel auch langsamer.

Dateien mithilfe des Win32-API ermitteln

Man kann es sich fast denken – die Win32-API Funktion ist die umfangreichste Variante.

Bevor überhaupt eine Routine erstellt werden kann, sind einige modulglobale Deklarationen notwendig.

Darüber hinaus benötigen diese Funktion 2 bestimmte Strukturen und Konstanten. Der nachfolgende Teil ist modulglobal zu hinterlegen; d.h. außerhalb einer Routine im Kopfbereich eines Moduls:

```
Option Explicit

Const MAX_PATH = 260
Const INVALID_HANDLE_VALUE = -1

Type FILETIME
     dwLowDateTime      As Long
     dwHighDateTime     As Long
End Type

Type WIN32_FIND_DATA
     dwFileAttributes   As Long
     ftCreationTime     As FILETIME
     ftLastAccessTime   As FILETIME
     ftLastWriteTime    As FILETIME
     nFileSizeHigh      As Long
     nFileSizeLow       As Long
     dwReserved0        As Long
     dwReserved1        As Long
     cFileName          As String * MAX_PATH
     cAlternate         As String * 14
End Type

Declare Function GetWindowsDirectory Lib "kernel32" Alias _
                "GetWindowsDirectoryA" _
                (ByVal lpBuffer As String, _
                 ByVal nSize As Long) As Long

Declare Function FindFirstFile Lib "kernel32" Alias _
                "FindFirstFileA" _
                (ByVal lpFileName As String, _
                 lpFindFileData As WIN32_FIND_DATA) As Long

Declare Function FindNextFile Lib "kernel32" Alias _
                "FindNextFileA" _
                (ByVal hFindFile As Long, _
                 lpFindFileData As WIN32_FIND_DATA) As Long

Declare Function FindClose Lib "kernel32" _
                (ByVal hFindFile As Long) As Long
```

Abbildung 10 - Modulglobale Deklarationen für die API-Variante

Nach diesen Vorbereitungen sind wir in der Lage, die API-Lösung zu erstellen:

```
Sub API_GetIniFiles()

    Dim uWFD        As WIN32_FIND_DATA  'Strukturvariable für API-Suche
    Dim sBuffer     As String           'Hilfsvariable
    Dim sWinDir     As String           'Windows-Ordner
    Dim lCounter    As Long             'Zählervariable
    Dim lResult     As Long             'API-Rückgabewert
    Dim lFile       As Long             'Datei-Handle
    Dim asFiles()   As String           'Datenfeld für die Ergebnisdateien

   'Zunächst das Windows-Verzeichnis ermitteln
    sBuffer = Space(260)
    lResult = GetWindowsDirectory(sBuffer, Len(sBuffer))
    If lResult <> 0 Then
          sWinDir = Left$(sBuffer, lResult)
         'Falls nicht vorhanden, einen Backslash anfügen
          If Right$(sWinDir, 1) <> "\" Then sWinDir = sWinDir & "\"
    Else: Exit Sub
    End If
   'Erste Suche starten
    lFile = FindFirstFile(sWinDir & "*.ini", uWFD)
   'Wenn es ein gültiges Dateihandle gibt
    If lFile <> INVALID_HANDLE_VALUE Then
        lCounter = 0
        Do
           'Zählervariable für das Array vergrößern
            lCounter = lCounter + 1
           'Array um 1 Eintrag erweitern (Re-Dimensionieren)
            ReDim Preserve asFiles(lCounter)
           'In dem neuen Array-Eintrag den ersten Dateinamen speichern
            asFiles(lCounter - 1) = uWFD.cFileName
           'Feld in der Struktur initialisieren
            uWFD.cFileName = vbNullString
       'Solange weitersuchen, bis kein Treffer mehr vorliegt
        Loop While FindNextFile(lFile, uWFD)
       'Dateihandle explizit schließen!
        lFile = FindClose(lFile)
    End If
   'Ergebnistest - alle Array-Einträge im Direktfenster ausgeben
    For lCounter = LBound(asFiles) To UBound(asFiles)
        If Len(asFiles(lCounter)) > 0 Then
            Debug.Print "Eintrag"; lCounter; ": "; asFiles(lCounter)
        End If
    Next

End Sub
```

Abbildung 11 - Dateien mithilfe von API ermitteln

Mit dieser 3. Variante schließen wir die Dateisuche ab – welche Variante zu bevorzugen ist, ist letztendlich Geschmacksache. Je nach Art und Menge der zu suchenden Dateien gibt es durchaus zeitliche Unterschiede, wobei wohl die API-Variante die Schnellste (aber auch Aufwändigste) ist.

Nachdem wir jetzt wissen, wie wir Dateien suchen und finden können, werden wir versuchen, mit diesen Dateien zu arbeiten. Speziell INI-Dateien eignen sich hervorragend für das strukturierte Speichern von Daten. Dabei folgen INI-Dateien grundsätzlich folgendem Dateiaufbau:

```
[Abschnitt01]
Schlüssel01=Wert
Schlüssel02=Wert
Schlüssel03=Wert
[Abschnitt02]
Schlüssel01=Wert
Schlüssel02=Wert
Schlüssel03=Wert
usw.
```

Die Bearbeitung von INI-Dateien sollte ausschließlich mithilfe des Win32-API durchführen, da uns hier eine Reihe von Funktionen zur Verfügung gestellt bekommen, die speziell für diese Arbeit geschaffen wurden.

Mit VBA-Hausmitteln oder mit dem WSH INI-Dateien zu bearbeiten ist Strafarbeit, da uns hier keinerlei Funktionalität zur Verfügung steht, um strukturiert auf diese Dateien zugreifen zu können. Daher versuchen wir es erst gar nicht…

Kapitel 2

BEARBEITUNG VON INI-DATEIEN MIT DEM WIN32-API

Obwohl Microsoft seit Jahren versucht, Entwickler von der Nutzung von INI-Dateien abzubringen und stattdessen die **Registry** zu benutzen, erfreut sich die Nutzung dieser sog. Konfigurationsdateien großer Beliebtheit.

Aufgabe: Lese und schreibe in INI-Dateien

Für (fast) jeden Bearbeitungsschritt stellt uns das Win32-API eine Funktion zur Verfügung:

Aufgabe	API-Funktion
Sämtliche Sektions- bzw. Abschnittsnamen ermitteln	GetPrivateProfileSectionNames
Einen kompletten Abschnitt „lesen“	GetPrivateProfileSection
Einen kompletten Abschnitt „schreiben“	WritePrivateProfileSection
Einen einzelnen Schlüsselwert „lesen“	GetPrivateProfileString
Einen einzelnen Schlüsselwert „schreiben“	WritePrivateProfileString

und speziell nur für den Zugriff auf die WINI.INI:

Aufgabe	API-Funktion
Einen einzelnen Schlüsselwert „lesen“	GetProfileString
Einen einzelnen Schlüsselwert „schreiben“	WritePrivateProfileString

Das war die gute Nachricht! Der Umgang mit diesen Funktionen stellt sich zum Teil nicht ganz so simpel dar.

Zunächst benötigen wir natürlich wieder die modulglobalen Deklarationen dieser Funktionen. Hierzu sollte man ein eigenes Modul erstellen, um diese Zugriffe auf INI-Dateien zu kapseln.

Fügen wir unserem VBA-Projekt also ein neues Modul hinzu und nennen dieses Modul **modINI**.

Im Kopfbereich unseres neuen Moduls werden die „**Declares**“ hinterlegt. Auf die beiden Funktionen für den Zugriff auf die WIN.INI wurde hier verzichtet, da wir nur auf unsere eigene INI-Datei zugreifen wollen. Diese „Declares“ sind hier nun „Private“ deklariert, was bedeutet, dass außerhalb dieses Moduls nichts und niemand direkt darauf zugreifen kann. Für den indirekten Zugriff werden wir entsprechende „**Wrapper**“-Funktionen erstellen. Dabei handelt es sich um Funktionen, die diese API-Funktionen kapseln und „von außen“ recht einfach genutzt werden können.

```
Option Explicit

Private Declare Function GetPrivateProfileSectionNames Lib "kernel32" Alias _
                         "GetPrivateProfileSectionNamesA" _
                         (ByVal lpszReturnBuffer As String, _
                          ByVal nSize As Long, _
                          ByVal lpFileName As String) As Long

Private Declare Function WritePrivateProfileSection Lib "kernel32" Alias _
                         "WritePrivateProfileSectionA" _
                         (ByVal lpAppName As String, _
                          ByVal lpString As String, _
                          ByVal lpFileName As String) As Long

Private Declare Function WritePrivateProfileString Lib "kernel32" Alias _
                         "WritePrivateProfileStringA" _
                         (ByVal lpApplicationName As String, _
                          ByVal lpKeyName As Any, _
                          ByVal lpString As Any, _
                          ByVal lpFileName As String) As Long

Private Declare Function GetPrivateProfileSection Lib "kernel32" Alias _
                         "GetPrivateProfileSectionA" _
                         (ByVal lpAppName As String, _
                          ByVal lpReturnedString As String, _
                          ByVal nSize As Long, _
                          ByVal lpFileName As String) As Long

Private Declare Function GetPrivateProfileString Lib "kernel32" Alias _
                         "GetPrivateProfileStringA" _
                         (ByVal lpApplicationName As String, _
                          ByVal lpKeyName As Any, _
                          ByVal lpDefault As String, _
                          ByVal lpReturnedString As String, _
                          ByVal nSize As Long, _
                          ByVal lpFileName As String) As Long
```

Abbildung 12 - Die benötigten API-Declares

Grundsätzliches zur Nutzung dieser Win32-Funktionen

- Jede Funktion erwartet als letztes Argument (**lpFileName**) einen gültigen Pfad zu einer INI-Datei.
- Existiert die INI-Datei, wird sie genutzt. Falls nicht, wird sie automatisch angelegt.
- Es gibt Funktionen, um Sektionen zu schreiben – es gibt aber keine zum Löschen!
- Wie (fast) alle anderen Win32-API Funktion erfolgt deren Nutzung sehr restriktiv hinsichtlich ihrer Sicherheit – es wird grundsätzlich nicht nachgefragt, bevor etwas geändert oder gelöscht wird.

Nachfolgend werden wir uns die notwendigen „Wrapper“-Funktionen erstellen.

Einen Wert in eine INI-Datei schreiben

Dieses Unterfangen stellt sich noch recht einfach dar.

Die Funktion **WritePrivateProfileString** besitzt 4 Argumente:

Argument	Bedeutung	Beispielwerte
lpApplicationName	Name der Sektion bzw. des Abschnitts	Section 01
lpKeyName	Name des Schlüssels	Key 01
lpString	zu schreibender Wert	Testwert
lpFileName	Gültiger Pfad zur INI-Datei	C:\WINNT\Dummy.ini

```
Sub WriteMyINISectionValue()

    Dim sINIPath    As String
    Dim sSection    As String
    Dim sValue      As String
    Dim sKey        As String
    Dim lRes        As Long

   'Pfad zum Windows-Ordner ermitteln
    sINIPath = API_GetWindowsDir()
   'Backslash prüfen und ggf. hinzufügen
    If Right$(sINIPath, 1) <> "\" Then sINIPath = sINIPath & "\"
   'Unseren INI-Dateinamen anfügen
    sINIPath = sINIPath & "Dummy.ini"
   'Unsere Sektion festlegen
    sSection = "Section 01"
   'Unseren Schlüssel festlegen
    sKey = "Key 01"
   'Unseren zu schreibenden Wert festlegen
    sValue = "Testwert"
   'Jetzt können wir die Funktion aufrufen
    lRes = WritePrivateProfileString(sSection, sKey, sValue, sINIPath)
   'Erfolgreich?
    If lRes <> 0 Then MsgBox "Alles klar!"

End Sub
```

Abbildung 13 - Einen Wert in INI schreiben

Und das Ergebnis im Editor:

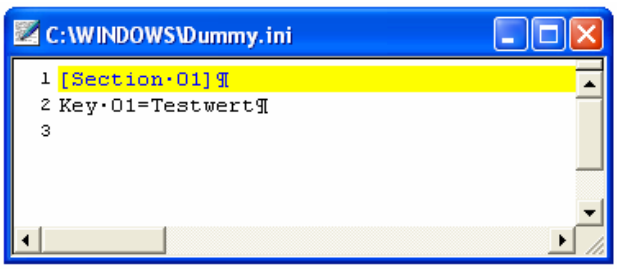

Abbildung 14 - Das Ergebnis

Nach diesem ersten Erfolg erstellen wir für das Schreiben von Schlüsselwerten in INI-Dateien eine Wrapper-Funktion.

```
Public Function SaveMySetting(ByVal sIniFilePath As String, _
                              ByVal sSection As String, _
                              ByVal sKey As String, _
                              ByVal sValue As String, _
                              ByRef sError As String) As Boolean
    Dim lResult As Long
    lResult = WritePrivateProfileString(sSection, sKey, sValue, sIniFilePath)
    SaveMySetting = CBool(lResult <> 0 And Err.LastDllError = 0)
    If Not SaveMySetting Then sError = GetDllErrorDescription(Err.LastDllError)

End Function
```

Abbildung 15 - Die Wrapper-Funktion

Diese Wrapper-Funktion wurde um die Prüfung des Rückgabewertes erweitert. Dabei wird im Fehlerfall der Wert **Err.LastDllError** aus dem Error-Objekt abgefragt.

Ist dieser **<> 0 UND** gab die Funktion selbst **0** zurück liegt ein Fehler vor.

Da unsere Wrapper-Funktion **SaveMySetting** selbst einen Boolean-Wert als Ergebnis zurückgibt, erzeugen wir diesen „typsicher“ über die Konvertierungsfunktion CBool().

Die Funktion ist genau dann erfolgreich (= True), wenn sowohl der API-Funktionsaufruf einen Wert <> 0 zurückgibt UND die VBA-Error-Objekt-Eigenschaft **LastDllError** keinen Wert (=0) besitzt.

Sollte die VBA-Error-Objekt-Eigenschaft **LastDllError** jedoch einen Wert (<> 0) beinhalten, haben wir lediglich einen (API)-Fehlercode, der genauso wenig wie VBA-Fehlercodes aussagekräftig ist. Es besteht jedoch die Möglichkeit, vom Betriebssystem den zugehörigen Fehlertext zu ermitteln.

Um diesen Fehler also „lesbar“ zu machen, benötigen wir eine weitere API-Funktion:

```
FormatMessage
```

Da die Benutzung dieser zusätzlichen API-Funktion nicht gerade simpel ist, habe ich hierfür eine Wrapper-Funktion **GetDllErrorDescription** erstellt, der man lediglich die API-Fehlernummer übergibt und die entsprechende, lesbare Fehlermeldung zurückerhält.

Zunächst muss jedoch auch für diese Funktion eine entsprechende „Deklaration“ erfolgen. Wir müssen also die bisher deklarierten API-Funktionen um folgenden Eintrag erweitern:

```
Private Declare Function FormatMessage Lib "kernel32" Alias _
                         "FormatMessageA" _
                         (ByVal dwFlags As Long, _
                          lpSource As Any, _
                          ByVal dwMessageId As Long, _
                          ByVal dwLanguageId As Long, _
                          ByVal lpBuffer As String, _
                          ByVal nSize As Long, _
                          Arguments As Long) As Long
```

Abbildung 16 - FormatMessage-Deklaration

Und hier die „Wrapper“-Variante:

```
Public Function GetDllErrorDescription(ByVal lErrCode As Long) As String
'// ---------------------------------------------------------------
'// Methode:    | Ermittelt die Beschreibung eines API-Fehlers
'// ---------------------------------------------------------------
'// Parameter: | lErrCode = API-Fehlernummer
'// ---------------------------------------------------------------
'// Rückgabe:  | Beschreibung des API-Fehlers
'// ---------------------------------------------------------------
    Const FORMAT_MESSAGE_FROM_SYSTEM = &H1000
    Dim lMsgSize    As Long
    Dim sSysMsg     As String

    If lErrCode <> 0 Then
        lMsgSize = 1000
        sSysMsg = Space(lMsgSize)
        lMsgSize = FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, _
                                 ByVal 0&, _
                                 lErrCode, _
                                 0, _
                                 sSysMsg, _
                                 lMsgSize, _
                                 ByVal 0&)
        If lMsgSize = 0 Then
              sSysMsg = "System-Fehlercode: " & Str$(lErrCode)
        Else: sSysMsg = Str$(lErrCode) & ": " & Left$(sSysMsg, lMsgSize)
        End If
    Else
        sSysMsg = vbNullString
    End If
    sSysMsg = Trim$(sSysMsg)
    If Right$(sSysMsg, Len(vbCrLf)) = vbCrLf Then
        sSysMsg = Left$(sSysMsg, Len(sSysMsg) - Len(vbCrLf))
    End If
    GetDllErrorDescription = sSysMsg

End Function
```

Abbildung 17 - Wrapper: GetDllErrorDescription

Das ursprüngliche Beispiel hat sich wie folgt verändert:

```
Sub INI_Sample()

    Dim sINIPath    As String
    Dim sError      As String
    Dim lRes        As Long

   'Pfad zum Windows-Ordner ermitteln
    sINIPath = API_GetWindowsDir()
   'Backslash prüfen und ggf. hinzufügen
    If Right$(sINIPath, 1) <> "\" Then sINIPath = sINIPath & "\"
   'Unseren INI-Dateinamen anfügen
    sINIPath = sINIPath & "Dummy.ini"
   'Jetzt können wir die Wrapper-Funktion aufrufen
    If SaveMySetting(sINIPath, "Section 01", "Key 01", "Testwert", sError) Then
          MsgBox "Alles klar!"
    Else: MsgBox sError
    End If

End Sub
```

Abbildung 18 - Die veränderte Prozedur

Einen Wert aus einer INI-Datei lesen

Genauso „einfach“ ist es, einen Wert aus einer INI-Datei zu lesen.

Die Funktion **GetPrivateProfileString** besitzt 6 Argumente:

Argument	Bedeutung	Beispielwerte
`lpApplicationName`	Name der Sektion bzw. des Abschnitts	`Section 01`
`lpKeyName`	Name des Schlüssels	`Key 01`
`lpDefault`	zu schreibender Wert	`i.d.R. Leerstring`
`lpReturnedString`	Dimensionierte Rückgabestring	`dimensionierter String`
`nSize`	Länge von `lpReturnedString`	`Länge des dimensionierter Strings`
`lpFileName`	Gültiger Pfad zur INI-Datei	`C:\WINNT\Dummy.ini`

Wie wir bereits in der Funktion **GetWindowsDirectory** kennen gelernt haben, müssen wir auch hier für die Rückgabe des gewünschten Wertes, einen String entsprechend dimensionieren.

Für die notwendige Länge gibt’s hier jedoch keine „Eselsbrücke“ (beim Ermitteln des Windows-Verzeichnisses wurde der String mit 260 Zeichen gefüllt, da ein Dateipfad maximal 255 Zeichen lang sein darf). Man sollte also einen „genügend“ langen Rückgabestring dimensionieren.

Wie (fast) alle API-Funktionen gibt auch **GetPrivateProfileString** einen Wert zurück; leider gestaltet sich die Auswertung dieses Ergebnisses etwas umständlich:

Rückgabewert	Bedeutung
Anzahl benötigter Zeichen	Alles klar, wir können die Rückgabe auswerten
`nSize - 1`	Wenn weder lpAppName noch lpKeyName den Wert NULL besitzen und der Rückgabestring zu klein bemessen war, wird der String abgeschnitten und ein Chr(0) Zeichen angehängt.
`nSize - 2`	Wenn entweder lpAppName oder lpKeyName den Wert NULL besitzen und der Rückgabestring zu klein bemessen war, wird der String abgeschnitten und zwei Chr(0) Zeichen angehängt.

Wir sollten also möglichst vermeiden, einen zu kurzen Rückgabestring zu dimensionieren. Auf dem sicheren Wege sind wir, wenn der Rückgabestring in der Länge der Datei dimensioniert wird, außer wir sind uns über die notwendige Länge im Klaren. Die Dateilänge zu nutzen „kostet“ zwar kurzfristig ein bisschen Speicher, aber der zurückgegebene Wert kann niemals länger als die Datei selbst sein.

Die nachfolgende „Wrapper“-Funktion ist schon sehr sicher, verzichtet aber auf die Längenprüfung **nSize** – 1 bzw. **nSize** – 2; wer das möchte, kann die Funktion gerne entsprechend erweitern. Wir gehen also davon aus, dass ein Rückgabewert der API-Funktion > 0 uns es di-

rekt erlaubt, die entsprechende Anzahl Zeichen aus dem Rückgabestring zu extrahieren. Die „Wrapper"-Funktion stellt sich folgt dar:

```
Public Function GetMySetting(ByVal sIniFilePath As String, _
                             ByVal sSection As String, _
                             ByVal sKey As String, _
                             ByRef sResult As String, _
                             ByRef sError As String) As Boolean

    Dim lResult As Long     'API-Rückgabewert
    Dim lLength As Long     'Dateilänge der INI-Datei
    Dim sBuffer As String   'String für den Rückgabewert

   'Rückgabestring initialisieren
    sResult = vbNullString
   'Fehlerhandling einschalten, falls es die Datei nicht gibt
    On Error Resume Next
   'Dateilänge ermitteln
    lLength = FileLen(sIniFilePath)
   'Länge (und Fehler) auswerten
    If lLength = 0 Or Err.Number <> 0 Then Exit Function
   'Fehlerhandling wieder ausschalten
    On Error GoTo 0
   'Rückgabestring dimensionieren
    sBuffer = Space(lLength)
   'API-Funktion ausführen
    lResult = GetPrivateProfileString(sSection, sKey, vbNullString, _
                                      sBuffer, Len(sBuffer), sIniFilePath)
   'Resultat auswerten
    If lResult > 0 Then
       'Rückgabewert "extrahieren"
        sResult = Left$(sBuffer, lResult)
       'Funktionsrückgabewert setzen
        GetMySetting = True
    ElseIf Err.LastDllError <> 0 Then
       'API-Fehlerbeschreibung ermitteln und zurückgeben
        sError = GetDllErrorDescription(Err.LastDllError)
       'Funktionsrückgabewert setzen
        GetMySetting = False
    End If

End Function
```

Abbildung 19 - Wrapper: GetMySetting

Und hier ein „Zugriffsbeispiel:"

```
Sub INI_Sample()

    Dim sINIPath    As String
    Dim sResult     As String
    Dim sError      As String
   'Pfad zum Windows-Ordner ermitteln
    sINIPath = API_GetWindowsDir()
   'Backslash prüfen und ggf. hinzufügen
    If Right$(sINIPath, 1) <> "\" Then sINIPath = sINIPath & "\"
   'Unseren INI-Dateinamen anfügen
    sINIPath = sINIPath & "Dummy.ini"
   'Jetzt können wir die Wrapper-Funktion aufrufen
    If GetMySetting(sINIPath, "Section 01", "Key 01", sResult, sError) Then
          MsgBox sResult
    Else: MsgBox sError
    End If

End Sub
```

Abbildung 20 - Zugriffsbeispiel

Das Lesen und Schreiben einzelner Werte in einer INI-Datei ist im Vergleich zum Lesen und Schreiben ganzer Abschnitte relativ simpel. Nachfolgend werden wir folgenden beispielhaften Abschnitt aus unserer INI-Datei auslesen:

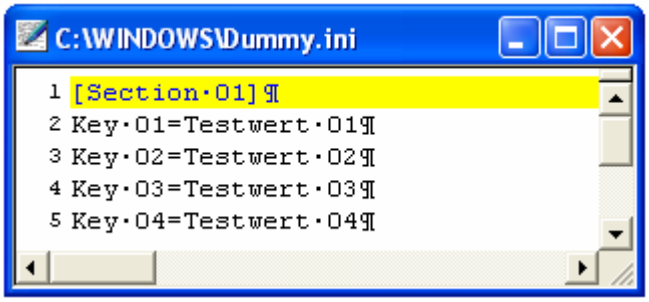

Abbildung 21 - INI Werte auslesen

Einen Abschnitt aus einer INI-Datei lesen

Die hierzu notwendige Funktion GetPrivateProfileSection besitzt folgende 4 Argumente:

Argument	Bedeutung	Beispielwerte
`lpAppName`	Name der Sektion bzw. des Abschnitts	`Section 01`
`lpReturnedString`	Dimensionierte Rückgabestring	`dimensionierter String`
`nSize`	Länge von `lpReturnedString`	`Länge des dimensionierter Strings`
`lpFileName`	Gültiger Pfad zur INI-Datei	`C:\WINNT\Dummy.ini`

Auch hier dimensionieren wir den Rückgabestring am besten wieder in der Länge der Datei, um unliebsame Überraschungen zu erwarten.

Der Rückgabewert der Funktion GetPrivateProfileSection entspricht der Länge des Rückgabestrings. War dieser String nicht genügend lang dimensioniert, entspricht der Rückgabewert der Längenangabe **nSize** – 2.

Etwas problematisch stellen sich die Darstellung der unterschiedlichen Schlüssel und deren Werte in einem einzelnen String dar. Diese Werte werden jeweils durch ein Chr(0)-Zeichen voneinander getrennt.

Da wir uns auch hier eine Wrapper-Funktion erstellen, sorgen wir auch gleichzeitig für eine komfortable Rückgabe der einzelnen Schlüssel in Form eines Arrays. Da jeder Eintrag aus „Schlüssel=Wert“ besteht, gehen wir einen Schritt weiter und teilen auch diese Information jeweils auf. Dazu benötigen wir eine Struktur bzw. für die Übergabe an die Wrapper-Funktion ein Struktur-Array:

```
Private Type INI_SECTION_DATA
             sKeyName   As String
             sKeyValue  As String
End Type
```

Abbildung 22 - Die Type-Struktur

Diese Struktur nimmt für jeden Eintrag in der betroffenen Sektion jeweils den Schlüsselnamen und den Schlüsselwert auf. Um alle Schlüssel der Sektion zu berücksichtigen, nutzen wir ein Struktur-Array vom Typ **INI_SECTION_DATA**. Mithilfe von VBA 6 (ab Office 2000 verfügbar), würde uns VBA hierbei helfen, da ab dieser Version die Funktion Split zur Verfügung steht, welche einen String anhand eines speziellen Kriteriums aufteilt und in einem Array speichert. Da uns für diesen Workshop nur VBA 5 zur Verfügung steht, müssen wir dies selbst erledigen…eine gute Übung um mit Strings zu arbeiten.

```
Public Function GetMySection(ByVal sIniFilePath As String, ByVal sSection As String, _
                             ByRef ausData() As INI_SECTION_DATA, _
                             ByRef sError As String) As Boolean

    Dim lDims       As Long     'Dimensionenzähler für Rückgabe-Array
    Dim lResult     As Long     'API-Rückgabewert
    Dim lLength     As Long     'Dateilänge der INI-Datei
    Dim lOffeset    As Long     'Hilfvariable
    Dim sBuffer     As String   'String für den Rückgabewert
    Dim sRecord     As String   'Daten eines Eintrags (Schlüssel=Wert)
   'Fehlerhandling einschalten, falls es die Datei nicht gibt
    On Error Resume Next
   'Dateilänge ermitteln
    lLength = FileLen(sIniFilePath)
   'Länge (und Fehler) auswerten
    If lLength = 0 Or Err.Number <> 0 Then Exit Function
   'Fehlerhandling wieder ausschalten
    On Error GoTo 0
   'Rückgabe-Array initialisieren
    Erase ausData
   'Rückgabestring dimensionieren
    sBuffer = Space(lLength)
   'API-Funktion ausführen
    lResult = GetPrivateProfileSection(sSection, sBuffer, lLength, sIniFilePath)
   'Rückgabewerte überprüfen
    If lResult > 0 Then
       'Unnötigen 'Anhang' abschneiden
        sBuffer = Left$(sBuffer, lResult)
        While Len(sBuffer) > 0
            lOffeset = InStr(sBuffer, vbNullChar) 'nächstes Trennzeichen ermitteln
            If lOffeset > 0 Then
               'Aktuellen Schlüssel nebst Wert auslesen
                sRecord = Left$(sBuffer, lOffeset - 1)
               'Gesamtstring verkürzen
                sBuffer = Mid$(sBuffer, lOffeset + 1)
               'Dimensionenzähler inkrementieren und Array redimensionieren
                lDims = lDims + 1
                ReDim Preserve ausData(lDims)
               'Schlüssel und Schlüsselwert separat in Struktur schreiben;
               'als Trennzeichen wird "=" vorausgesetzt (Standard bei INI-Dateien)
                ausData(lDims - 1).sKeyName = Left$(sRecord, InStr(sRecord, "=") - 1)
                ausData(lDims - 1).sKeyValue = Mid$(sRecord, InStr(sRecord, "=") + 1)
            End If
        Wend
        GetMySection = True
    ElseIf Err.LastDllError <> 0 Then
       'API-Fehlerbeschreibung ermitteln und zurückgeben
        sError = GetDllErrorDescription(Err.LastDllError)
       'Funktionsrückgabewert setzen
        GetMySection = False
    End If

End Function
```

Abbildung 23 - GetMySection Prozedur

Als Test für die zuvor beschriebene Funktion dient folgende Routine:

```
Sub INI_Sample()

    Dim sINIPath    As String
    Dim sError      As String
    Dim sOutput     As String
    Dim lLoop       As Long
    Dim ausData()   As INI_SECTION_DATA

   'Pfad zum Windows-Ordner ermitteln
    sINIPath = API_GetWindowsDir()
   'Backslash prüfen und ggf. hinzufügen
    If Right$(sINIPath, 1) <> "\" Then sINIPath = sINIPath & "\"
   'Unseren INI-Dateinamen anfügen
    sINIPath = sINIPath & "Dummy.ini"
   'Jetzt können wir die Wrapper-Funktion aufrufen
    If GetMySection(sINIPath, "Section 01", ausData, sError) Then
        For lLoop = LBound(ausData) To UBound(ausData)
            If Len(ausData(lLoop).sKeyName) > 0 Then
                sOutput = "Eintrag " & Str(lLoop) & ": " & _
                          ausData(lLoop).sKeyName & "=" & _
                          ausData(lLoop).sKeyValue
                Debug.Print sOutput
            End If
        Next
    Else
        MsgBox sError
    End If

End Sub
```

Abbildung 24 - Test der Prozedur GetMySection

Einen Abschnitt in eine INI-Datei schreiben

Das Schreiben eines kompletten Abschnitts erfolgt anlog dem Lesen bzw. unter Berücksichtigung der dort vorgestellten Besonderheiten. Der gesamte Inhalt der zu speichernden Sektion muss in einen einzelnen String gepackt werden, wobei „als Zeilentrenner“ das Chr(0)-Zeichen dient. Der String selbst muss mit einem endenden Chr(0)-Zeichen abgeschlossen werden.

Achtung: das Schreiben eines kompletten Abschnitts löscht unweigerlich einen vorhandenen Abschnitt; es erfolgt weder eine Warnmeldung noch erfolgt ein „Abgleich“.

Man sollte also entsprechend vorsichtig mit dieser Funktion umgehen. Das Zusammenfassen eines Abschnitts in einen einzelnen String zum Speichern in eine INI-Datei stellt sich deutlich einfacher dar, als das Auslesen und Auswerten eines Abschnitts aus einer INI-Datei; Bsp.:

```
"Key 01=Testwert 01" & Chr(0) & … "Key 04=Testwert 04" & Chr(0) &
Chr(0)
```

Um einen Unterschied zu vorher festzustellen, ändern wir den Abschnittsinhalt für die Speicherung wie folgt ab:

```
[Section 01]
Schlüssel 01=Das
```

```
Schlüssel 02=ist
Schlüssel 03=ein
Schlüssel 04=Test
```

Eine entsprechende Wrapper-Routine erhält bereits den vorbereiteten String, so dass sich diese Routine recht einfach darstellt.

Die hierzu notwendige Funktion **WritePrivateProfileSection** besitzt folgende 3 Argumente:

Argument	Bedeutung	Beispielwerte
lpAppName	Name der Sektion bzw. des Abschnitts	Section 01
lpString	Die Abschnittsdaten	s.u.
lpFileName	Gültiger Pfad zur INI-Datei	C:\WINNT\Dummy.ini

Der Rückgabewert der Funktion **WritePrivateProfileSection** ist bei Erfolg ungleich 0, sonst 0.

```
Public Function WriteMySection(ByVal sIniFilePath As String, _
                               ByVal sSection As String, _
                               ByVal sRecord As String, _
                               ByRef sError As String) As Boolean

    Dim lResult As Long      'API-Rückgabewert

    lResult = WritePrivateProfileSection(sSection, sRecord, sIniFilePath)
    WriteMySection = CBool(lResult <> 0)
    If Not WriteMySection And Err.LastDllError <> 0 Then
        sError = GetDllErrorDescription(Err.LastDllError)
    End If

End Function
```

Abbildung 25 - Wrapper: WriteMySection

Als Test für die zuvor beschriebene Funktion dient folgende Routine:

```
Sub INI_Sample()

    Dim sINIPath    As String
    Dim sRecord     As String
    Dim sError      As String

   'Pfad zum Windows-Ordner ermitteln
    sINIPath = API_GetWindowsDir()
   'Backslash prüfen und ggf. hinzufügen
    If Right$(sINIPath, 1) <> "\" Then sINIPath = sINIPath & "\"
   'Unseren INI-Dateinamen anfügen
    sINIPath = sINIPath & "Dummy.ini"
   'Jetzt basteln wir den Abschnittsstring zusammen
    sRecord = "Schlüssel 1 = Das" & vbNullChar & _
              "Schlüssel 2 = ist" & vbNullChar & _
              "Schlüssel 3 = ein" & vbNullChar & _
              "Schlüssel 4 = Test" & vbNullChar & vbNullChar

    If WriteMySection(sINIPath, "Section 01", sRecord, sError) = True Then
          MsgBox "OK"
    Else: MsgBox sError
    End If

End Sub
```

Abbildung 26 - Test der Prozedur WriteMySection

Kapitel 3

ABBILDUNGSVERZEICHNIS