

Einstieg in OOO-BASIC

Table of contents

1 Neues Makro anlegen.....	2
1.1 Bereits vorhandener Code.....	3
1.2 Die MsgBox-Anweisung.....	3
1.3 Schaltfläche im Dokument.....	5
2 Eine Abfolge von Dialogen.....	6
2.1 Die Variablendeklaration.....	6
2.2 Der Titeltext des Dialogfensters.....	8
2.3 Der Text im Dialogfenster.....	8
3 Die interaktive Dialogbox.....	10
3.1 Die Abbrechen-Schaltfläche.....	10
4 Die ersten Verzweigungen.....	11
5 Die zweite Sub-Routine.....	12
5.1 Die InputBox-Funktion.....	13
5.2 Die Select-Case-Anweisung.....	13
6 Der Bug.....	15
6.1 Debugging.....	15
6.2 Beobachter einsetzen.....	16
6.3 Debugging per Print-Anweisung.....	17
6.4 Der Patch des Bugs.....	18
6.5 Umgang mit fehlerhaftem Code.....	19

Im Zusammenhang mit MS Office Makros ist Ihnen eventuell bereits Visual Basic bzw. VBA ein Begriff. OpenOffice.org-BASIC (identisch mit StarOffice BASIC) ist für denselben Zweck gedacht und ebenso einsetzbar: als schnell erlernbare und am besten in OpenOffice.org implementierte Skriptsprache, um die Office-Suite nach eigenen Bedürfnissen zu optimieren. Nachdem Sie über die Programmierschnittstelle (API) auf die gesamte Office-Suite zugreifen können, sind hier die Möglichkeiten grenzenlos.

Für einen ersten Einstieg lassen sich Makros in der in OpenOffice.org integrierten Entwicklungsumgebung (BASIC-IDE) allerdings auch völlig unabhängig von den Dokumenten des Programms erstellen. Auf diese Weise können Sie sich noch ohne Objekte und die Programmierschnittstelle (API) mit BASIC schon etwas vertraut machen.

An einem ersten kleinen Beispiel-Makro sehen Sie, wie einfach Standard-Dialoge erzeugt werden können. Anschließend wird anhand eines etwas größeren Makros die wichtige Programmieretechnik der Fall-Unterscheidungen in einem Frage-und-Antwort-Spiel demonstriert. Das Makro reagiert dabei auf Benutzereingaben.

In diesem Zusammenhang lernen Sie bereits wichtige Grundfunktionen kennen, das Erstellen einer längeren Dialogfolge und auch das evtl. nötige Debugging. Das erworbene Know-How zu Datentypen, Anweisungen, Funktionen, Verzweigungen etc. können Sie in weiteren Workshops zu konkreten Beispiel-Makros einsetzen und erweitern.

1. Neues Makro anlegen

Grundsätzlich gelangen Sie zur BASIC-Entwicklungsumgebung stets aus einem geöffneten OpenOffice.org-Dokument heraus, egal welchen Typs.

1. Über Extras/Makros/Makros verwalten/OpenOffice.org-Basic legen Sie den Speicherort für Ihr Makro fest, in diesem Fall in einem Dokument.
2. Dazu markieren Sie in der Verzeichnisübersicht das Dokument und legen über die Schaltfläche Neu ein neues Modul an. Standardmäßig wird es als Module1 betitelt, was Sie für einen besseren Überblick im Makro in der Regel ändern sollten. Auch vermeiden Sie so Namensgleichheit von Modulen, falls Sie Ihre Module in andere Makros übernehmen möchten.

Die Verzeichnisübersicht und das neue Modul vor der Umbenennung

Im Beispiel wird als neuer Name des Moduls einfache_MsgBox eingetragen, und nach Bestätigung des Dialogs erscheint automatisch die Entwicklungsumgebung, d.h. der bereits vom letzten Kapitel bekannte BASIC-Editor für Ihren Code.

3. Falls Sie das noch nicht getan haben, speichern Sie jetzt Ihr Dokument wie gewöhnlich über Speichern ab, wahlweise im Editor oder auch im Dokument selbst.

Im Beispiel geschieht dies als `basic_dialogbox.odt`, d.h. im neuen Textdokumentformat (OASIS OpenDocument XML-Dateiformat) von OpenOffice.org 2.0. Wie bereits erwähnt, wird Ihr Makro dabei automatisch mitgespeichert.

1.1. Bereits vorhandener Code

Haben Sie ein neues Modul angelegt, so enthält es automatisch bereits drei Zeilen Code.

```
REM ***** BASIC *****  
Sub Main  
End Sub
```

Bei der ersten Zeile handelt es sich um einen Kommentar, was Sie stets an einem vorangestelltem REM bzw. alternativ einem Hochkomma ' erkennen. Kommentare werden nicht mit ausgeführt, sondern dienen lediglich der Information, in diesem Fall über die verwendete Skriptsprache.

Note:

So wie diesen automatisch generierten Kommentar können Sie Kommentare auch selbst verfassen, um sich selbst im eigenen Code leichter zu orientieren oder anderen den Code zu erklären. Kommentare, die Sie an jeder Stelle einfügen können, gehören zum guten Ton beim Programmieren, sollten aber auch nicht ausufern.

Die beiden anderen Zeilen kennzeichnen Anfang und Ende einer Sub-Routine (auch als Prozedur bezeichnet) mit dem Namen Main. Beim modularen Aufbau von OpenOffice.org-BASIC stellt eine Sub-Routine die kleinste Einheit dar, d.h. eine oder mehrere Sub-Routinen sind in einem Modul enthalten. Eines oder mehrere Module sind wiederum Bestandteil einer Bibliothek.

Note:

In Dialogen und Menü-Einträgen von OpenOffice.org werden Sub-Routinen als Makros bezeichnet, dies ist begrifflich etwas unscharf. Zwar wird stets eine Sub-Routine als Einstieg in ein Makro verwendet, ein Makro kann sich aber über mehrere Sub-Routinen erstrecken.

1.2. Die MsgBox-Anweisung

Zwischen Anfang und Ende der Sub-Routine schreiben Sie Ihren Code für die erste Message-Box. Zweck einer Message-Box ist die Bildschirmausgabe von Meldungen in einem einfachen Dialogfenster, in diesem Fall einfach "Hallo!". Dazu genügt es, eine einzige Zeile zu schreiben: `MsgBox "Hallo!"`

Der gesamte Code sieht danach so aus:

```
REM ***** BASIC *****  
Sub Main
```

```
MsgBox "Hallo!"
End Sub
```

Wie Sie beim Schreiben schnell feststellen werden, erhalten die einzelne Bestandteile des Codes automatisch unterschiedliche Farben. Dieses so genannte Syntax-Highlighting dient der besseren Lesbarkeit, was Sie bei längeren Listings schätzen werden.

- Sub und End erscheinen blau, da es sich hierbei um Schlüsselwörter handelt. Schlüsselwörter oder auch reservierte Wörter sind Bezeichnungen, die eine spezielle Funktion in OOO-BASIC belegen. D.h. sie sind fester Bestandteil von OOO-BASIC und dürfen nicht als Variablennamen vergeben werden.

- Grün wie hier Main und MsgBox werden Bezeichner dargestellt. Dies sind primär Namen von Variablen, Funktionen, Anweisungen, Sub-Routinen, die Sie selbst vergeben können. In einigen Fällen wie bei MsgBox handelt es sich allerdings um vorgelegte Bezeichner, mit denen Sie bei Ihren eigenen Bezeichnungen nicht in Konflikt kommen sollten.

Note:

BASIC ist nicht case-sensitiv, d.h. es wird nicht zwischen Groß- und Kleinschreibung unterschieden: MsgBox = msgbox. Auf eine korrekte Schreibweise sollten Sie trotzdem achten, um nicht später bei API-Zugriffen Fehler zu produzieren.

- Rot wie hier "Hallo!" sind Literale, d.h. alles, was an konstanten Werten eingegeben wird. Während Variablen als Hüllen dienen, kann ein Literal einer Variablen zugewiesen werden, also die Hülle mit Inhalt füllen. Literale können Zeichen, Zeichenfolgen oder auch Zahlen sein.

- Grau erscheinen die bereits erwähnten Kommentare im Code, wie hier REM ***** BASIC *****.

Note:

Alle in OpenOffice.org fest eingebauten Anweisungen und Funktionen mit den erforderlichen Parametern finden Sie, teilweise mit Beispielen erklärt, in der Hilfe zu OpenOffice.org BASIC unter "Arbeiten mit OpenOffice.org-BASIC" beim Thema Runtime-(Laufzeit-)Funktionen.

1.2.1. Die Ausführung des Codes

Haben Sie Ihre Code-Zeile geschrieben, können Sie über die Schaltfläche BASIC-Programm ausführen in der Symbolleiste das Ergebnis ansehen.

Das Dialogfenster unter Windows und Linux(KDE)

Viel passiert in diesem Fall logischerweise noch nicht, aber Sie erhalten bereits ein beschriftetes Dialog-Fenster mit einer OK-Schaltfläche. Auch dieses Mini-Makro, bei dem Sie nur eine Zeile Code schreiben mussten, ist plattformunabhängig. Es kann unverändert auf jedem Betriebssystem ausgeführt werden, auf dem OpenOffice.org läuft.

Die Erzeugung des Dialogfensters im jeweiligen Betriebssystem/Fenstermanager übernimmt Openoffice.org, inklusive Schaltflächen, Beschriftung und Möglichkeit zum Schließen des Dialogs.

Note:

Es wurde im Beispiel gezielt nicht die ebenfalls mögliche Print-Anweisung vorgestellt, da hier die User-Eingaben komplizierter zu behandeln wären. Auf die Möglichkeiten der Print-Anweisung wird später beim Thema Debugging eingegangen.

1.3. Schaltfläche im Dokument

Auf in ein Dokument eingebundene Makros können Sie schnell zugreifen, indem Sie dafür im Dokument eine Schaltfläche anlegen. Praktischerweise lässt sich bei Bedarf auch Erläuterungstext zum Makro in dasselbe Dokument eintragen.

1. Um eine Schaltfläche ins Dokument einzufügen, besorgen Sie sich zunächst über Ansicht/Symbolleiste/Formularsteuerelemente das passende Werkzeug. Die neue Symbolleiste lässt sich durch Ziehen mit der Mauslinks, rechts, oben oder unten im Dokumentfenster andocken.

Links in der Symbolleiste: der aktivierte Entwurfsmodus, rechts die aktivierte Schaltfläche

2. Nach Aktivierung des Entwurfsmodus wählen Sie Schaltfläche und ziehen diese dann – wie eine Form - im Dokument auf.

3. Per Doppelklick in die Schaltfläche oder Rechtsklick und Kontrollfeld legen Sie im Register Allgemein des Dialogs den Titel, also die Beschriftung Ihrer Schaltfläche fest, hier z.B. Hallo.

Note:

Über die Schaltfläche rechts neben Schrift passen Sie bei Bedarf noch die Formatierung Ihrer Beschriftung wie Größe oder Schriftart an.

4. Im Register Ereignisse wählen Sie Beim Auslösen, wenn das Makro per Klick loslegen soll.

Die Schaltfläche mit der gewünschten Beschriftung; das Ereignis wird gerade ausgewählt.

5. Über die zugehörige Schaltfläche rechts und die Schaltfläche Zuweisen im folgenden Dialog gelangen Sie zu Ihren Makro-Verzeichnissen. Dort wählen Sie Ihr Makro aus, bestätigen die Zuweisung mit Ok und schließen den Kontrollfeld-Dialog.

6. Nach dem Deaktivieren des Entwurfsmodus wird das Makro ausgeführt, sobald Sie auf die Schaltfläche klicken.

2. Eine Abfolge von Dialogen

Das folgende Makro zeigt Ihnen einige Möglichkeiten, wie Sie mit OpenOffice.org-BASIC flexibel auf das Verhalten des Anwenders reagieren können. Demonstriert wird dies an einer Abfolge von Smalltalk-Dialogen zum beliebigen Thema Wetter. Das aus Standard-Dialogen bestehende Makro wird dabei Stück für Stück aufgebaut. Es werden so lange Ja-Nein-Entscheidungen oder Zahleneingaben des Benutzers abgefragt, bis genug Informationen für eine eindeutige Antwort vorliegen. Diese wird wiederum als Standard-Dialog ausgegeben.

Mit dieser Message-Box geht's los.

Wenn Sie das Makro (smalltalk.odt) über die hier bereits vorhandene Schaltfläche Smalltalk im Dokument smalltalk.odt starten und ablaufen lassen, sehen Sie die unterschiedlichen Auswirkungen, je nachdem, was Sie anklicken bzw. eingeben. Unterschiedliche Antworten bzw. Eingaben wurden beim Schreiben des Makros bereits berücksichtigt.

Wichtig für den reibungslosen Ablauf des Programms ist die logische Baumstruktur im Hintergrund. Auch in diesem Fall ist diese Struktur nicht auf Anhieb ganz perfekt, aber auch nachträgliches Debugging funktioniert problemlos. Wie das konkret aussieht, erfahren Sie beim Thema Debugging. Der Einsatz der Debugging-Werkzeuge hilft Ihnen ferner auch beim Durchblick im eigenen Code.

- Zum Code des mitgelieferten Makros gelangen Sie auf dem üblichen Weg, indem Sie es aus dem Makro-Verzeichnis heraus über Bearbeiten im Editor öffnen. Dies empfiehlt sich, da Sie auf diese Weise den Überblick über den gesamten Code behalten.

- Wenn Sie das Makro einmal schrittweise selbst nachbauen möchten, öffnen Sie dazu ein neues Dokument, in dem Sie Ihr Makro speichern können.

2.1. Die Variablendeklaration

Nicht anders als beim ersten, kleinen Makro mit dem Hallo!-Dialog handelt es sich beim ersten Dialog des Makros um eine MsgBox-Anweisung mit einem schlichten OK-Button. Allerdings werden in diesem Fall im Code zunächst Variablen mit der Dim-Anweisung

deklariert, dies dient der Übersichtlichkeit des Codes. Andernfalls müsste bei der MsgBox-Anweisung diesmal eine sehr lange Zeichenfolge eingetragen werden.

1. Damit jede Variable deklariert werden muss und nicht versehentlich bei ihrer ersten Verwendung automatisch deklariert wird, fügen Sie ganz oben im Modul die Schlüsselwörter Option Explicit für die entsprechende Anweisung ein.

```
REM ***** BASIC *****  
Option Explicit
```

Note:

Eine automatische Variablendeklaration ist nicht unbedingt praktisch, weil Sie auf diese Weise leichter fehlerhaften Code produzieren. Von daher ist der Arbeitsaufwand für die Deklaration die bessere Wahl.

2. Variablen, die nur innerhalb einer Sub-Routine benötigt werden, deklarieren Sie per Dim-Anweisung logischerweise stets am Anfang der Sub-Routine (Prozedur), d.h. direkt nach Sub Main und vor der ersten Verwendung Ihrer Variablen.

```
Dim sText As String  
Dim sTitel As String  
Dim iAntwort As Integer
```

Mit dem Schlüsselwort As wird der Variablen ein fester Datentyp zugewiesen. Dies wäre in BASIC nicht unbedingt erforderlich, da ohne Typ-Festlegung die Variablen automatisch als Variant deklariert werden. Der Variant-Datentyp nimmt wiederum automatisch den Datentyp des Wertes an, der ihm zugewiesen wird. Bei umfangreicheren Makros führt dies allerdings häufig zu Fehlern. Oft wird es schwierig nachzuvollziehen, welchen Datentyp die Variable aktuell besitzt.

- Hier wurde zwei Variablen der Typ String zugewiesen und somit festgelegt, dass nur Zeichenfolgen in diesen Variablen gespeichert werden sollen. Damit die Datentypen später leichter identifizierbar sind, können Sie mit einem vorangestellten Kleinbuchstaben den Datentyp kennzeichnen wie hier z.B. bei den Variablen sText und sTitel.

Note:

Auch der Rest des Variablennamens sollte aussagekräftig sein. Sie erleichtern sich und anderen damit die Weiterarbeit am Makro! Für die Ausführbarkeit des Codes spielt der Variablenname keine Rolle, solange er gültig ist. So darf er nur mit einem Buchstaben oder dem Unterstrich _ beginnen und später zwar Zahlen, aber insgesamt keine Umlaute und Sonderzeichen enthalten.

- Der dritten Variablen iAntwort wurde der Datentyp Integer zugewiesen, d.h. die Variable soll mit Ganzzahlen im Bereich zwischen -32768 und 32767 gefüllt werden. Im Beispiel sollen darin später die Rückgabewerte für die Antworten Ja (6) oder Nein(7) des Anwenders landen.

2.2. Der Titeltext des Dialogfensters

Beim ersten Dialogfenster handelt es sich - wie bereits erwähnt - um eine MsgBox-Anweisung, die Sie bereits kennen. Allerdings wird hier für die Dialogfenster ein eigener Titel vergeben, im Beispiel Smalltalk.

```
sTitel = "Smalltalk"
```

- Der Text des Titels, also die Zeichenfolge (=String) Smalltalk, wird der vorher als String deklarierten Variablen sTitel zugewiesen.
- Die Zuweisung erfolgt durch den Operator = (daher auch Zuweisungsoperator).
- Die Zeichenfolge (Smalltalk) muss in Anführungszeichen gesetzt werden, damit der BASIC-Interpreter sie nicht für einen Variablennamen hält.

Note:

Legen Sie selbst keinen Titel fest, so erscheint standardmäßig soffice in der Titelzeile Ihres Dialogfensters.

2.3. Der Text im Dialogfenster

Generell sollte Text in Dialogfenstern kurz gehalten werden, auch wenn sich Dialogfenster in der Größe automatisch anpassen. Im Idealfall ist Ihr kleines Programm trotzdem selbsterklärend und somit benutzerfreundlich. Sind Erläuterungen zu Ihrem Makro nötig, nutzen Sie dafür das zugehörige Dokument.

Gerade wegen der Übersichtlichkeit kann sich der Text in einem Dialogfenster allerdings einmal über zwei oder mehr Zeilen erstrecken. Auch Zeilenumbrüche legen Sie im Code des Makros fest.

1. Zunächst wird die erste Zeile des Textes wieder der String-Variablen sText zugewiesen.

```
sText = "Es folgt ein typischer Smalltalk."
```

2. Der Zeilenumbruch muss als gesondertes Zeichen übergeben werden, da ein Zeilenumbruch im Code nicht das gewünschte Ergebnis erzeugt. Hierzu wird die Nummerierung der ASCII-Zeichentabelle verwendet, hier 13. Um daraus wieder ein Zeichen zu machen, muss der ASCII-Code mit der Chr(=Convert to Char)-Funktion konvertiert werden.

```
sText = sText + Chr(13)
```

Note:

Dieses Verfahren gilt für alle ASCII-Zeichencodes. So ließe sich über Chr(34) auch ein Anführungszeichen erzeugen. Benötigen Sie einmal weitere ASCII-Zeichen, finden Sie entsprechende Übersichten leicht im Internet wie z.B. unter: <http://de.wikipedia.org/wiki/ASCII-Tabelle>.

- Die Verkettung von Zeichenfolgen findet in BASIC über den gleichen Operator statt wie die Addition von Zahlen, d.h. über +.

- Nach dem Ausführen dieser Zeile enthält die Variable sText nunmehr die ursprüngliche Zeichenfolge "Es folgt ein typischer Smalltalk" und den Zeilenumbruch.

Beachten Sie, dass der BASIC-Interpreter stets die rechte Seite einer Zuweisung zuerst evaluiert, d.h. deren Wert ermittelt. Das kann eine Berechnung sein wie z.B. 3 + 5 oder wie im obigen Fall eine Zeichenverkettung. Mit dem neuen Wert wird dann bei der Zuweisung der alte Wert in der Variablen links überschrieben.

Note:

Der in manchen Programmiersprachen vorhandene Operator += existiert nicht in BASIC, und Sie müssen leider zwei Mal die Variable tippen.

3. In der dritten Codezeile wird wiederum eine Zeichenfolge an die bereits in der Variablen vorhandene angehängt, in diesem Fall der Text für die zweite Zeile des Dialogtexts.

```
sText = sText + "Über was wohl? - Das Wetter!"
```

4. Im letzten Beispiel wurde bei der MsgBox-Anweisung die Standard-Einstellung verwendet. Sowohl Titel als auch Button wurden durch den Standard belegt (OK-Button und soffice als Fenstertitel). Diesmal soll zwar ebenfalls der OK-Button, aber ein anderer Fenstertitel verwendet werden.

```
MsgBox sText, 0, sTitel
```

Der Fenstertitel sTitel wird als dritter Parameter an die MsgBox-Anweisung übergeben. Somit muss auch der zweite Parameter für Buttons und Icons explizit angegeben werden, hier nur 0 für den OK-Button.

Bei einem Parameter handelt es sich um eine Variable, die von einer Anweisung oder Funktion verarbeitet wird. In diesem Fall liefern die Parameter die "Bauteile" für das Dialogfenster. Getrennt werden die Parameter durch Kommata. Während Sie die Variablennamen für die Parameter frei vergeben können, sind der Datentyp und die Platzierung festgelegt.

Note:

Nachdem Sie bei einer MsgBox-Anweisung keinen Rückgabewert erhalten, macht hier nur der OK-Button wirklich Sinn.

Andere Schaltflächen sind zwar darstellbar, aber es ist nicht feststellbar, welche Schaltfläche vom Anwender angeklickt wurde. Dazu benötigen Sie die anschließend erläuterte MsgBox-Funktion. Durchaus Sinn macht dagegen die Verwendung von Symbolen (s.u.).

3. Die interaktive Dialogbox

Nach dem Bestätigen des ersten Dialogs über den OK-Button, erscheint der nächste Dialog des Makros.

Hier hat der Anwender die Wahl.

Ohne weitere Anweisungen wird der Code einer Sub-Routine zeilenweise bis zu ihrem Ende abgearbeitet. Eine Dialogbox wie z.B. die MsgBox unterbricht die Code-Ausführung, solange die Dialogbox sichtbar ist, d.h. bis eine Aktion des Anwenders erfolgt. Danach wird die Ausführung des Codes in der nächsten Zeile fortgesetzt.

1. Der erste Teil des folgenden Codes dürfte Ihnen schon bekannt vorkommen, auch hier wird wieder eine Zeichenfolge in einer Variablen verpackt.

```
sText = "Regnet es gerade?"
```

2. In der nächsten Zeile wird der oben deklarierten Integer-Variablen iAntwort der Rückgabewert der MsgBox-Funktion zugewiesen.

```
iAntwort = MsgBox(sText, 4 + 32, sTitel)
```

Im Gegensatz zur MsgBox-Anweisung oben kann bei der MsgBox-Funktion die Anwender-Aktion zurückgegeben werden. D.h. je nachdem, auf welchen Button der Anwender klickt, gibt die Funktion eine andere, vordefinierte Ganzzahl (Integer) zurück, in diesem Fall 6 für Ja und 7 für Nein.

Erkennbar ist eine Funktion immer an den Klammern um die Parameter. Erster und dritte Parameter sind hier identisch mit denen der oben verwendeten MsgBox-Anweisung, auch wenn die Variable sText hier einen anderen Inhalt hat. Der zweite Parameter 4 + 32 gibt die Verwendung der Buttons Ja und Nein (4) und das Fragezeichen-Symbol (32) vor.

Note:

Nachdem der Parameter 4 + 32 vor seiner Zuweisung sofort berechnet wird, könnten Sie genauso 36 schreiben. Allerdings ist bei 4 + 32 auf den ersten Blick zu sehen, welche Schaltflächen und Symbole hier verwendet werden. Die möglichen Integer-Werte für diesen Parameter finden Sie – ebenso wie die Rückgabewerte – in der OpenOffice.org-Hilfe zu BASIC bei der MsgBox-Funktion.

3.1. Die Abbrechen-Schaltfläche

In diesem Beispiel-Makro wurde auf die zusätzliche Abbrechen-Schaltfläche verzichtet, um den Code einfach und übersichtlich zu halten.

Wollen Sie dem Anwender eine Möglichkeit zum Abbruch zur Verfügung stellen, verwenden Sie die MsgBox-Funktion mit zusätzlichem Abbrechen-Button, d.h. MsgBox(Text, 1, Titel) für Ok, Abbrechen oder MsgBox(Text, 3, Titel) für Ja, Nein, Abbrechen.

Sie erhalten dann den Rückgabewert 2, wenn der Anwender Abbrechen gewählt hat und müssen eine entsprechende Verzweigung (Erläuterung s.u.) einbauen, in der Sie mit der End-Anweisung den Abbruch des Makros herbeiführen können. Dafür genügt es, in einer eigenen Codezeile ein schlichtes End einzufügen.

```
Dim iAntwort As Integer
iAntwort = MsgBox("Meldungstext", 3, "Meldungstitel")
If iAntwort = 2 Then
    End
End If
```

4. Die ersten Verzweigungen

Die Antwort des Anwenders auf die Frage "Regnet es?" wurde an die Variable iAntwort übergeben. Es muss jetzt überprüft werden, ob er mit Ja (Rückgabewert 6) oder Nein (7) geantwortet hat, um ihm im Anschluss den passenden Dialog präsentieren zu können.

Diese zwei unterschiedlichen Dialoge erscheinen im Fall von Antwort Ja bzw. Nein .

Dies geschieht mit Hilfe einer If-Then-Else-Verzweigung, wobei der zweite, hier abgebildete Dialog die Notwendigkeit einer weiteren If-Verzweigung bereits deutlich macht. Diese wird in die erste If-Verzweigung eingebaut, d.h. die Verzweigungen werden ineinander verschachtelt. Eine If-Verzweigung umfasst dabei stets die Codezeilen von If bis End If.

Die Einrückung der Codezeilen spielt dabei keine Rolle und wurde nur zur besseren Lesbarkeit verwendet.

```
If iAntwort = 6 Then
    sText = "Dann bleiben Sie besser am Computer!"
    MsgBox sText, 0 + 64, sTitel
Else
    sText = "Scheint die Sonne?"
    iAntwort = MsgBox(sText, 4 + 32, sTitel)
    If iAntwort = 6 Then
        AntwortSonne
    Else
        sText = "Dann ist es wohl bewölkt."
```

```

        MsgBox sText, 0 + 48, sTitel
    End If
End If
End Sub

```

1. Ist der Ausdruck nach If eine wahre Aussage (true), wird der Code nach Then ausgeführt. Dies wäre bei Ja (6) der Fall, und somit wird die MsgBox-Anweisung mit dem Beschriftungstext "Dann bleiben Sie besser am Computer!" ausgeführt. Damit wäre die Dialogfolge bereits beendet.

2. Antwortet der Anwender mit Nein (7), wird der Code ab Else ausgeführt. Der Ausdruck nach If ist falsch (false), und somit wird der Code zwischen If und Else übersprungen. Entsprechend wird jetzt die nächste MsgBox-Funktion ausgeführt, die wieder eine Frage für den Anwender bereithält und erneut eine Antwort entgegennimmt.

Eine erneute Frage an den Anwender bedeutet eine weitere Verzweigung im Code.

3. So folgt logischerweise wieder die nächste Überprüfung, wiederum mit einer If-Else-Verzweigung.

- Im positiven Fall wird mit der Anweisung AntwortSonne eine weitere Sub-Routine aufgerufen. AntwortSonne ist der selbst vergebene Name für die Sub-Routine, in der die Dialogfolge im Fall von Sonne fortgeführt wird.

- Im negativen Fall endet das Makro mit der MsgBox-Anweisung mit der Beschriftung "Dann ist es wohl bewölkt".

Dieser Dialog wird bei einem Nein auf die zweite Frage ausgegeben.

4. Abgeschlossen wird eine If-Verzweigung stets mit End If, wie hier zwei ineinander verschachtelten Verzweigungen, die auch jede für sich wieder geschlossen werden müssen.

Wie bereits erwähnt, springt das Makro von AntwortSonne in die anschließend beschriebene zweite Sub-Routine Sub AntwortSonne.

5. Die zweite Sub-Routine

Die Variablen, die in der ersten Sub-Routine deklariert wurden, gelten außerhalb dieser Sub-Routine nicht. Wird dies dennoch gewünscht, müssten sie außerhalb der Sub-Routine deklariert werden, im Beispiel also zwischen Option explicit und Sub Main. Es handelt sich dann um globale Variablen, die auch in anderen Sub-Routinen dieses Moduls gelten.

Solange es nicht zwingend erforderlich ist, sollten keine globalen Variablen verwendet werden. In umfangreichen Makros können sie durch ihre Allgegenwärtigkeit zu

irrtümlichen Fehlbelegungen führen.

Entsprechend werden im Beispiel die dort verwendeten Variablen zu Beginn der zweiten Sub-Routine, d.h. nach Sub AntwortSonne, wieder deklariert. Dies gilt aus o.g. Gründen auch, wenn sie wie z.B. sText in der ersten Sub-Routine deklariert und benutzt wurden.

```
Sub AntwortSonne
  Dim sText As String
  Dim sTitel As String
  Dim sStandard As String
  Dim sAntwort As String
  Dim iAntwort As Integer
```

5.1. Die InputBox-Funktion

Anders als die MsgBox-Anweisung/-Funktion existiert die InputBox nur als Funktion, da sie immer einen Rückgabewert haben soll. In diesem Fall wird vom Anwender eine Zahl für die Temperatur eingegeben.

Im Eingabefeld soll die Zahl für die Temperaturangabe landen.

Zu den Parametern der InputBox gehören wie bei der MsgBox Beschriftungstext und Titel. Andere Schaltflächen und Symbole sind nicht vorgesehen. Neu hinzugekommen ist die Möglichkeit, an dritter Stelle einen Standard-Wert vorzugeben. Dieser erscheint dann bereits markiert in der Eingabe-Zeile des Dialogs.

Note:

Es wäre hier ebenso möglich, einen Standard-Temperaturwert einzugeben, wie z.B. 20. In der Praxis werden Eingabezeilen aber häufig "missbraucht", um eine weitere Erläuterung vor der Eingabe unterzubringen, so auch in diesem Beispiel.

```
sText = "Wieviel Grad hat es?"
sTitel = "Smalltalk"
sStandard = "Bitte eine Zahl eingeben"
sAntwort = InputBox(sText, sTitel, sStandard)
iAntwort = CInt(sAntwort)
```

Der Rückgabewert einer InputBox ist in jedem Fall eine Zeichenfolge (String). Daher muss im Beispiel die Zeichenfolge sAntwort mit der CInt-Funktion (Convert to Integer) in eine Ganzzahl (Integer) konvertiert werden, bevor sie der Integer-Variablen iAntwort zugewiesen wird.

5.2. Die Select-Case-Anweisung

Da die Antwort des Anwenders in diesem Fall mehr Möglichkeiten zulässt als Ja oder

Nein, muss auch der Rückgabewert etwas komplexer untersucht werden. Es sollen für verschiedene Temperatur-Bereiche verschiedene Antworten ausgegeben werden.

Diese Dialoge werden bei Temperaturen zwischen -20 und 24 Grad (links) bzw. zwischen 25 und 34 Grad (rechts) ausgegeben.

Bei Temperaturen zwischen 35 und 44 Grad (links) bzw. bei allen anderen, noch nicht berücksichtigten Zahleneingaben wie z.B. 60 oder -30 (rechts) erscheinen diese Dialoge.

Mit Hilfe der Select Case-Anweisung lässt sich eine Vielzahl an Fällen übersichtlich abfragen. If-Then-Konstrukte würden hier dagegen schnell unlesbar werden.

```

Select Case iAntwort
  Case -20 To 24
    sText = "Machen Sie doch einen Spaziergang!"
  Case 25 To 34
    sText = "Was schwitzen Sie hier noch? Gehen Sie baden!"
  Case 35 To 45
    sText = "Hoffentlich haben Sie gute Lüfter im PC!"
  Case Else
    sText = "Solche Temperaturen sind sehr ungewöhnlich!"
End Select
MsgBox sText, 0+48, sTitel
End Sub

```

1. Zunächst wird die zu untersuchende Variable festgelegt, hier iAntwort.
2. In den folgenden Zeilen werden verschiedene Fälle einzeln abgefragt, wie z.B. ob der Inhalt der Variablen im Bereich zwischen -20 und 24 liegt.
3. Ist dies der Fall, so wird die nächste Zeile ausgeführt, falls nicht, zur nächsten Fallunterscheidung Case weiter gesprungen. Auf diese Weise wird ein Fall nach dem anderen untersucht, bis ein Fall zutrifft.
4. Die Case Else-Anweisung kann optional verwendet werden. Der Code würde dann ausgeführt werden, wenn keiner der anderen Fälle zutrifft. Fehlt die Case Else-Anweisung, würde die Select-Anweisung ohne weitere Code-Ausführung bei End Select enden.
5. Wurde auf Basis der Fallunterscheidung hier der passende Text ausgewählt, gibt die MsgBox-Anweisung ihn auf dem Bildschirm aus.

Das Makro scheint zunächst einmal beendet, da nach diesem Dialog sichtbar nichts mehr ausgeführt wird. Tatsächlich kehrt der Interpreter nach dem Ende der zweiten Sub-Routine zu der Zeile in der ersten Sub-Routine zurück, die auf jene Zeile folgt, aus

der die zweite Sub-Routine aufgerufen wurde.

Nachdem die If-Unterscheidung bereits getroffen wurde, springt er jedoch zu End If . Da sich die äußere If-Unterscheidung ebenfalls bereits erledigt hat, geht's gleich weiter über End If zu End Sub.

Jetzt ist die Ausführung des Codes tatsächlich beendet. Würde zwischen End If und End Sub noch Code stehen, würde dieser selbstverständlich noch ausgeführt werden.

6. Der Bug

Wenn Sie etwas mit dem Makro herumspielen und verschiedene Antworten durchprobieren, stoßen Sie eventuell auf einen Bug. Er tritt nämlich dann auf, wenn Sie die Frage Scheint die Sonne? mit Ja beantwortet haben und im anschließenden Eingabe-Dialog sofort den OK-Button klicken oder einen nicht numerischen Wert eingeben wie z.B. heiß. Als Antwort erhalten Sie die Meldung Machen Sie doch einen Spaziergang!, die eigentlich für den Temperatur-Bereich von -20 bis 24 Grad gedacht war.

6.1. Debugging

Das Wort Bug kommt nicht umsonst von "Käfer", d.h. er taucht irgendwo auf, ist schnell wieder verschwunden und schwer zu finden. Können Sie durch den Programmablauf bereits eingrenzen, wo sich der Bug ungefähr befinden muss, ist das Debugging relativ einfach.

In diesem Fall kann z.B. festgestellt werden, dass bis zum Erscheinen der InputBox noch alles regulär gelaufen ist. Der nächste Dialog ist aber offensichtlich unpassend. So deutet schon einiges darauf hin, dass der Bug mit einem unerwarteten Rückgabewert der InputBox zu tun hat.

1. Haben Sie den Code im Editor geöffnet, suchen Sie sich die Zeile nach der Konvertierung des Rückgabewerts aus, d.h. hier Select Case iAntwort und setzen dort über das Symbol Haltepunkt ein/aus in der Symbolleiste einen Haltepunkt. Er wird stets am Anfang der Codezeile eingefügt, in der sich gerade der Cursor befindet.

Links neben dem aktivierten Haltepunkt-Symbol sehen Sie die Symbole für schrittweises Ausführen des Codes; die Brille rechts davon symbolisiert den Beobachter

Note:

Alternativ setzen Sie einen Haltepunkt per Doppelklick links in der Spalte neben der betreffenden Code-Zeile. Es können auch mehrere Haltepunkte gesetzt werden.

1. Starten Sie über das Symbol BASIC-Programm ausführen im Editor jetzt Ihr Makro,

wird der Code bis zum Haltepunkt durchlaufen. D.h. die Zeile, in der sich der Haltepunkt befindet, ist zu diesem Zeitpunkt noch nicht ausgeführt worden.

Selbstverständlich müssen Sie dabei die Antworten so geben, dass Sie zum gewünschten Dialog vorstoßen: Also regnet es nicht, die Sonne scheint, und Sie geben keine Zahl als Temperatur ein.

2. Statt des roten Haltepunkts sehen Sie jetzt an der Stelle, an der angehalten wurde, einen gelben Pfeil. Gehen Sie jetzt mit der Maus über die Variable `iAntwort`, wird Ihnen per Tooltip der aktuelle Wert der Variablen angezeigt, in diesem Fall `iAntwort = 0`.

Der Cursor befindet sich über der Variablen, deren aktueller Wert gerade abgefragt wird.

Wie Sie auch ganz unten im Bild schon erkennen können, liegt der Wert 0 im Bereich der ersten Fallunterscheidung, d.h. zwischen -20 und 24. Somit wird dann auch die erste Antwort ausgegeben.

Die 0 kommt dadurch zustande, dass ein nicht numerischer Wert in einen Ganzzahl-Wert konvertiert werden sollte. Hierbei gibt der Interpreter keine Fehlermeldung aus, sondern konvertiert jeden nicht numerischen Wert, sogar eine leere Zeichenfolge in den Ganzzahlwert 0.

Note:

Falls Sie einmal nicht wissen sollten, welche Variable für den Bug verantwortlich ist, können Sie auf die o.g. Weise mit der Maus die momentanen Werte aller Variablen abfragen. Den jeweiligen Datentyp sehen Sie bei dieser Methode allerdings nicht. Dazu können Sie Beobachter definieren, wie anschließend beschrieben.

6.2. Beobachter einsetzen

1. Um den eigenen Code weiter unter die Lupe zu nehmen, markieren Sie eine Variable, deren Belegung Sie verfolgen wollen, mit der Maus und aktivieren per Klick auf das Brillensymbol Beobachter ein/aus in der Symbolleiste die genauere Analysemöglichkeit.

2. Auf diese Weise lassen sich sämtliche Variablen Ihres Codes in die Beobachtung mit einbeziehen, wenn Sie anschließend mit Hilfe des Symbols Step into Zeile für Zeile Ihren Code ausführen lassen.

Note:

Während Sie mit Step into in aufgerufene Sub-Routinen und Funktionen hineingehen, wird bei Step over nur der Aufruf als ein Zeilenschritt verstanden. Code in Sub-Routine oder Funktion läuft ohne Stopp durch. Bei Step out gehen Sie entsprechend aus einer Sub-Routine oder Funktion wieder heraus und lassen den Rest des Codes darin einfach ausführen.

3. Die Variablen werden unten im Editor unter Beobachter angezeigt und zwar mit ihrem

jeweiligen Wert und Typ. Per Tooltip können Sie sich den ganzen Wert anzeigen lassen, falls er nicht mehr in die Spalte passt – oder alternativ die Spalten mit der Maus breiter ziehen.

Ausgewählte Variablen werden hier mit jeweiligem Wert und Typ unter Beobachter angezeigt.

Befinden Sie sich vor einer Codezeile mit einer Variablen-Deklaration, hat noch keine Deklaration stattgefunden. Die Variable existiert noch nicht und entsprechend wird der Beobachter hier den Hinweis Out of Scope (außerhalb des Gültigkeitsbereichs) geben.

4. Im Beobachter können Sie Zeile für Zeile die aktuelle Belegung der Variablen bis zum jeweiligen Ende ihrer Gültigkeit verfolgen. Beachten Sie dabei wieder, dass der gelbe Pfeil stets die als nächstes auszuführende Zeile markiert. Code dieser Zeile ist zu diesem Zeitpunkt noch nicht ausgeführt. Somit können sich Veränderungen, die dieser Code bewirken würde, noch nicht auf die beobachteten Variablen ausgewirkt haben.

Das schrittweise Nachvollziehen des Programmablaufs ist oft sehr hilfreich bei der Fehlersuche. Z.B. würden Sie leichter mitbekommen, wenn der Programmablauf in einer anderen Zeile als erwartet fortgeführt wird, weil die Bedingung einer Verzweigung falsch gestellt wurde.

6.3. Debugging per Print-Anweisung

Eine andere Möglichkeit, aus dem laufenden Makro Informationen über den Variableninhalt zu erhalten, ist die Bildschirmausgabe per Print-Anweisung.

```
sAntwort = InputBox(sText, sTitel, sStandard)
Print "Inhalt von sAntwort: " + sAntwort
iAntwort = CInt(sAntwort)
Print "Inhalt von iAntwort: " + iAntwort
```

Die Print-Anweisung wird nach der Belegung der Variablen einfach in den Code eingefügt, hier für die Variablen sAntwort und iAntwort. Der beschreibende Text im String kann natürlich weggelassen werden, dient aber dem besseren Verständnis der Ausgabe.

Note:

Vergessen Sie am Ende des Strings nicht ein Leerzeichen einzufügen, sonst kleben Beschreibungstext und Variableninhalt zusammen.

Die Ausgabe einer Print-Anweisung gleicht einer MsgBox, eignet sich aber nicht so gut für Dialoge. Der von Print erzeugte Dialog verfügt über einen Abbrechen-Button, der zu

einem abrupten Abbruch des Makros mit entsprechender Fehlermeldung führt.

Die Ausgabe der zwei Print-Anweisungen bestätigt hier, dass die Konvertierung vom String zum Integer funktioniert hat.

6.4. Der Patch des Bugs

Da der Fehler im Code des Beispiels offensichtlich darin bestand, Anwendereingaben ungeprüft weiterzuverarbeiten, wird dieses Versäumnis jetzt korrigiert. Im Code des im Dokument `smalltalk_patch.odt` mitgelieferten Makros sehen Sie, wo der Patch genau eingefügt wurde.

Die hinzugefügten Codezeilen sind noch auskommentiert, was Sie am jeweils vorangestellten Hochkomma und der grauen Schriftfarbe erkennen. Entfernen Sie die Hochkommata, können auch die neuen Code-Zeilen ausgeführt werden.

1. Für die Korrektur wird die von der Laufzeitumgebung bereitgestellte Funktion `isNumeric()` als Bedingung für eine weitere Verzweigung verwendet. Die Funktion überprüft, ob die Zeichen einer Zeichenfolge in eine Zahl konvertiert werden können.

```
If IsNumeric(sAntwort) Then
```

- Ist so sichergestellt, dass `sAntwort` einen numerischen Wert enthält, kann der Code ausgeführt werden wie bisher.

- Enthält die Variable keinen numerischen Wert, liegt eine unerwartete Anwender-Eingabe vor, auf die jetzt reagiert werden muss.

```
Else
  MsgBox "Bitte geben Sie eine Zahl ein!"
  AntwortSonne
  Exit Sub
End If
```

2. Für diesen Fall wird jetzt eine weitere `MsgBox` ergänzt, um den Anwender noch einmal deutlich auf die nötige Eingabe einer Zahl hinzuweisen.

3. Daraufhin wird die Sub-Routine `AntwortSonne`, in der sich die Code-Ausführung gerade befindet, rekursiv aufgerufen. D.h. Funktionen oder Sub-Routinen/Prozeduren können auch aus sich selbst heraus aufgerufen werden.

4. Entsprechend bekommt der Anwender jetzt erneut den Dialog für die Eingabe der Zahl zu sehen und kann sie korrekt eingeben. Tut er das, wird der Code bis zur Ausgabe der nun richtigen Meldung per `MsgBox` wieder ausgeführt.

5. Nach Beenden der Sub-Prozedur springt der Interpreter wie bei einem Aufruf aus einer

anderen Sub-Routine wieder zur aufrufenden Anweisung im Patch zurück. Daher wird dort im Anschluss an den Aufruf AntwortSonne die Anweisung Exit Sub eingefügt. Jetzt wird die Sub-Routine wirklich verlassen und nicht ihr unterer Teil inklusive MsgBox erneut ausgeführt.

Note:

Falls Sie Exit Sub einmal probeweise wieder auskommentieren, sehen Sie, dass Sie diesmal zwei Antworten bekommen. Erst dann stoppt die Code-Ausführung.

Natürlich ist dieser Patch nicht perfekt: Was, wenn ein unverbesserlicher Anwender immer wieder falsche Eingaben macht? Allerdings ist die hier vorgestellte Lösung durchaus praxisnah und noch recht diplomatisch.

6.5. Umgang mit fehlerhaftem Code

Fehler im logischen Ablauf des Programms wie beim o.g. Bug führen zwar zu Irritationen beim Anwender, beeinträchtigen aber nicht die Ausführbarkeit des Codes. Anders sieht es aus, wenn relevante Teile des Codes fehlen oder fehlerhaft sind.

1. Bemerkbar machen sich solche Fehler bei Programmausführung sofort. Die Code-Ausführung wird angehalten, eine Fehlermeldung ausgegeben und die fehlerhafte Zeile im Editor markiert.

Hier führte ein Tippfehler zur Fehlermeldung wegen einer nicht definierten Variablen.

Note:

Wenn Sie ein fehlerhaftes Makro aus einem Dokument heraus starten, geschieht dasselbe. Hier wird dazu automatisch der Editor geöffnet.

Bei manchen Fehlern, wie hier beim Tippfehler im Variablennamen, erhalten Sie bereits einen nützlichen Hinweis. Ferner zeigt sich ein Vorteil der per Option explicit erzwungenen Variablendeklaration.

Wurde diese nicht vorgenommen, führt die falsch geschriebene Variable nicht zu einer Fehlermeldung. Allerdings wird eine Dialogbox ausgegeben, in der die erste Zeile der Beschriftung fehlt, weil der entsprechende String in die Zeichenkette nicht mehr aufgenommen wird.

2. Nach dem Bestätigen der Fehlermeldung können Sie die Korrektur im Code vornehmen und anschließend das Makro zur Überprüfung wieder starten.

Ist Ihnen nicht sofort klar, wie der Fehler zustande kommt, empfiehlt sich wieder

schrittweises Debugging mit Beobachtern der Variablen.