Flexibler Einsatz von VBA-Code und Access-Elementen durch Kapselung und modularer Programmierung

(Josef Pötzl, www.joposol.com)

Kapselung – Was ist das?

Auszug aus Wikipedia

Kapselung ist auch ein wichtiges Konzept der objektorientierten Programmierung.

Als Kapselung bezeichnet man den kontrollierten Zugriff auf Methoden bzw. Attribute von Klassen. […] Dies verhindert das Umgehen von Invarianten des Programms.
Vom Innenleben einer Klasse soll der Verwender (gemeint sind sowohl die Algorithmen, die mit der Klasse arbeiten, als auch der Programmierer, der diese entwickelt) möglichst wenig wissen müssen (Geheimnisprinzip). Durch die Kapselung werden nur Informationen über das "Was" einer Klasse (was es leistet) nach außen sichtbar, nicht aber das "Wie" (die interne Repräsentation). Dadurch wird eine Schnittstelle nach außen definiert und zugleich dokumentiert.

aus http://de.wikipedia.org/wiki/Datenkapselung_(Programmierung)

Kapselung unter Access

Access unterstützt nicht alle Erfordernisse der *Objektorientierten Programmierung*. Es ist z.B. keine Vererbung möglich.
Daher spricht man unter Access von *Objektbasierter Programmierung*.
Kapselung ist jedoch auch unter Access möglich.

Der folgende Text soll keine Einführung in objektorientiertes Programmieren darstellen, sondern Möglichkeiten zeigen, wie das Prinzip der Kapselung unter Access angewendet werden kann.

VBA-Code kapseln

Kapselung *im objektorientierten Sinn* benötigt Klassen. Das Prinzip der Kapselung bzw. Modularisierung lässt sich aber bereits bei Prozeduren in allgemeinen Modulen als „Hülle“ anwenden, um z. B. oft benötige Funktionsaufrufe zu vereinfachen.

Standard-Prozeduren kapseln

Die MsgBox-Prozedur verwendet vermutlich jeder. Meist wird für diese Prozedur ein identischer Titel-Parameter verwendet.

```
Msgbox "Meldungstext", vbOKOnly, g_conStandardTitel
'g_conStandardTitel = globale Konstante
```

Wird eine eigene MsgBox-Funktion erstellt, muss nicht jedes Mal diese Konstante angegeben werden.

```
Public Function MsgBoxNeu(ByVal Prompt As String, _
                 Optional ByVal Buttons As VbMsgBoxStyle = vbOKOnly, _
                 Optional ByVal Title As String = g_conStandardTitel, _
                 Optional ByVal HelpFile As Variant, _
                 Optional ByVal Context As Variant) As VbMsgBoxResult

   MsgBoxNeu = Msgbox(Prompt, Buttons, Title, HelpFile, Context)

End Function
```

Es besteht außerdem die Möglichkeit die VBA-MsgBox-Funktion mit der neuen Prozedur zu ersetzen. In diesem Fall muss aber die VBA-MsgBox inkl. Bibliothek angesprochen werden.

```
Public Function MsgBox(ByVal Prompt As String, _
              Optional ByVal Buttons As VbMsgBoxStyle = vbOKOnly, _
              Optional ByVal Title As String = g_conStandardTitel, _
              Optional ByVal HelpFile As Variant, _
              Optional ByVal Context As Variant) As VbMsgBoxResult

   MsgBoxNeu = VBA.MsgBox(Prompt, Buttons, Title, HelpFile, Context)

End Function
```

Anm.: Diese Variante hat den positiven Nebeneffekt, dass die MsgBox-Prozedur sogar in einer mdb funktioniert, in der ein VBA-Verweis defekt ist. Der Aufruf der VBA-MsgBox funktioniert bei defektem Verweis nur mittels VBA.MsgBox, das man aber im Code sehr selten schreiben wird.

Sollte einmal die VBA-MsgBox nicht ausreichen, könnte man z.B. als Ersatz ein Formular gestalten und im Dialog-Modus öffnen. Diese Änderung müsste nur in der MsgBox-Prozedur durchgeführt werden. Der restliche Code muss nicht geändert werden, da der Aufruf identisch bleibt.

Kapseln mit Klassen

COM-Klassen benutzerfreundlich gestalten

Zum Steuern von Word oder Excel von Access aus, werden deren COM-Schnittstellen verwendet. Die Prozeduren für diese COM-Klassen-Steuerung sind meist sehr ähnlich gestaltet. Daher lohnt es sich, diese Abläufe in einer eigenen Access-VBA-Klasse einzubetten. Damit wird der Code für die unterschiedlichen Anwendungsfälle verkürzt. Außerdem sind Codeoptimierungen einfacher durchführbar, da der Code zur Steuerung nur noch in einer einzigen Klasse überarbeitet werden muss.

Recordset-Export-Funktion mit Excel-COM-Steuerung

```
Public Function ExcelExportOhneHilfsklasse( _
                         ByRef rst As DAO.Recordset, _
                Optional ByVal MitUeberschrift As Boolean = False))

   Dim xlsApp As Excel.Application
   Dim xlsWorkBook As Excel.Workbook
   Dim xlsWorkSheet As Excel.Worksheet

   'Excel-Instanz erzeugen
   Set xlsApp = CreateObject("Excel.Application")
   xlsApp.Visible = True

   'neues Workbook/Datei erstellen
   Set xlsWorkBook = xlsApp.Workbooks.Add

   'Worksheet-Referenz
   Set xlsWorkSheet = xlsWorkBook.Sheets(1)

   'Überschrift erstellen
   If MitUeberschrift Then
      '... wegen Übersichtlichkeit gekürzt
   End If

   'Recordset übertragen
   xlsWorkSheet.Range("A1").CopyFromRecordset rst

   'Objekt-Referenzen entfernen
   Set xlsWorkSheet = Nothing
   ...
End Function
```

Recordset-Export-Funktion über Hilfsklasse zur Excel-COM-Steuerung

```
Public Function ExcelExportMitHilfsklasse(_
                          ByRef rst As DAO.Recordset, _
                 Optional ByVal MitUeberschrift As Boolean = False))

   Dim xlsHandler As ExcelHandler

   Set xlsHandler = New ExcelHandler
   With xlsHandler
      .NewFile
      .CopyFromRecordset rst, 1, 1, MitUeberschrift
   End With

   Set xlsHandler = Nothing

End Function
```

Anm.: Die Hilfsklasse `ExcelHandler` *ist in der Beispiel-mdb enthalten.*

Die oben angeführten Prozeduren erleichtern den Export eines Formular-Recordsets:

```
Public Function ExcelExportFormRecordset(ByRef frm As Access.Form)
   ExcelExportMitHilfsklasse frm.Recordset.Clone, True
End Function
```

Anwendung im Formular:

```
Private Sub cmdExcelExport_Click()
   ExcelExportFormRecordset Me.sfrListe.Form, True
End Sub
```

Details siehe Beispiel-Formular `frmAuswahl`

Anm.: unbedingt die Formular-Referenz und nicht den Formular-Namen als Parameter verwenden, sonst lässt sich diese Prozedur nicht für Unterformulare verwenden.

Beispiel:

```
Public Function ExcelExportFormRecordset(Byval frmName As String)
   ExcelExportMitHilfsklasse Forms(frmName).Recordset.Clone, True
End Function
```

Diese Funktion würde für Hauptformulare ohne Probleme funktionieren. Der Datenexport aus einem Unterformular ist damit jedoch nicht mehr möglich. Auch der Export aus mehren Instanzen eines Formulars ist mit dieser Funktion nicht sichergestellt. Daher immer versuchen die Objekt-Referenzen zu übergeben.

Access-Objekte kapseln

Nicht nur VBA-Code sollte flexibel bleiben, sondern auch Access-Formulare sollten für mehrfache Anwendungsbereiche gestaltet werden. Damit ein flexibler Einsatz der Formulare gewährleistet wird, ist es empfehlenswert, jedes Formular als eigenständiges Element zu behandeln.

Auch wenn ein Formular erstmal nur als Hauptformular verwendet wird, besteht immer noch die Möglichkeit, dass es einmal zu einem Unterformular wird.

Falls ein Unterformular verwendet wird, könnte man dieses in mehr als einem Hauptformular verwenden. Das bedeutet aber, dass Zugriffe vom Unterformular auf Methoden des Hauptformulars vermieden werden müssen, da das Formular sonst nicht in anderen Hauptformularen verwendet werden kann, die diese Methode nicht besitzen.

Der Zugriff auf öffentliche Methoden des Unterformulars vom Hauptformular aus ist jedoch erlaubt. Ich empfehle dafür nur selbst definierte Methoden und Eigenschaften statt den eingebauten Formular-Elementen zu verwenden, um eine spätere Umgestaltung des Formulars zu ermöglichen. Nur die Schnittstellen nach außen müssen gleich bleiben, der Rest ist „Formularsache“.

Zugriffe auf Steuerelement des Unterformular vom Hauptformular aus, verstoßen gegen das Prinzip der Kapselung. Damit wird direkt auf Elemente des Formulars zugegriffen. Das hat zur Folge, dass z.B. eine Listbox nicht mit einem Unterformular ersetzt werden, ohne die Schnittstelle nach außen zu beeinflussen. Stellt man die Information die aus dem Listenfeld benötigt wird, über eine selbst definierte Formular-Eigenschaft zur Verfügung, so muss nur im Unterformular der Code des Zugriffs auf das Listenfeld in einen Code für den Zugriff auf ein Unterformular geändert werden. Das beeinflusst die Schnittstelle nach außen nicht und die Kapselung hat ihren Zweck erfüllt: der Code konnte ohne aufwändige Umgestaltung der zugreifenden Objekte geändert werden.

Methoden und Eigenschaften

Die Änderung der Daten im Unterformular soll vom Hauptformular aus gesperrt bzw. freigegeben werden. Das könnte vom HF mittels Direkt-Zugriff auf AllowEdits erfolgen.

```
Me!sfrUnterformularSteuerelement.Form.AllowEdits = False/True
```

Fügt man später jedoch im Unterformular ein weiteres Unterformular ein, dann wird diese Einstellung nicht weitergereicht und man müsste in jedem Formular, in dem das Unterformular verwendet wird, den Code noch ergänzen.

```
Private Sub setAllowEdits(allowEdit as boolean)
   With Me!sfrUnterformularSteuerelement.Form
      .AllowEdits = allowEdit
      !sfrNocheinUF.Form.AllowEdits = allowEdit
   End With
End Sub
```

Was ist, wenn im Unterformular kein Datensatz verfügbar ist? In diesem Fall löst der Zugriff auf das zweite Unterformular einen Fehler aus.

Mit einer zusätzlichen Prozedur im Formular kann dieses Problem vermieden werden.

```
Public Property Let AllowDataEdits(allowEdit As Boolean)

   Me.AllowEdits = allowEdit
   Me.AllowAdditions = allowEdit

   If allowEdit OR (Me.RecordsetClone.RecordCount <> 0) Then
      Me.sfrNochEinUF.Form.AllowDataEdits = allowEdit
   End If

End Property
```

Aufruf vom Hauptformular:

```
Me!sfrUnterformularSteuerelement.Form.AllowDataEdits = True | False
```

Ereignisse

Um vom Unterformular aus dem Hauptformular Informationen zu übermitteln, können Ereignisse verwendet werden. Damit wird im Unterformular kein direkter Bezug zum Hauptformular benötigt und das Unterformular kann somit beliebig in unterschiedlichen Hauptformularen eingesetzt werden. Nur im Hauptformular wird entschieden, ob auf das Ereignis reagiert wird.

Prinzip

Im Hauptformular wird eine Objektvariable mit WithEvents deklariert, um auf Ereignisse reagieren zu können. (Das Unterformular benötigt diese Information nicht.)

Sobald im Unterformular eine Aktion abläuft, über die das Hauptformular informiert werden soll, wird ein Ereignis ausgelöst, auf das das Haupt-Formular reagiert. Dabei muss das reagierende Formular nicht einmal das Hauptformular sein, in das das Unterformular eingebettet ist. Auch ein anderes Unterformular oder ein weiteres Hauptformular kann auf das Ereignis reagieren. Es können sogar mehrere Formular (im Prinzip jede Instanz einer Klasse) auf das Ereignis reagieren.

Beispiel

In der auslösenden Klasse

- Ereignis deklarieren

```
Public Event RecordsetSelected(ByVal DS_ID As Variant)
```

- Ereignis auslösen

```
RaiseEvent RecordsetSelected(Me!id)
```

In der empfangenden Klasse

- Objektreferenz mit Ereignisbehandlung deklarieren

```
Dim WithEvents m_EndlosForm As Form_frmEndlos
```

- Auf Ereignis reagieren

```
Private Sub m_EndlosForm_RecordsetSelected(ByVal DS_ID As Variant)
    ...
End Sub
```

Beispiel-Code siehe `frmEndlos` und `frmHF`

Download der Beispiel-Anwendung: http://access.joposol.com/download/Kapselung.zip

Josef Pötzl, August 2008