VBComponents

Autor & Copyright: Dipl.-Ing. Harald Nahrstedt

Version: 2016 / 2019 / 2021 / 365

Erstellungsdatum: 09.09.2022

Überarbeitung: 01.12.2023

Beschreibung:

Die VBComponents-Objektliste verwaltet die in einem Projekt enthaltenen Komponenten. Eine Komponente kann ein Formular, ein Modul oder eine Klasse sein.

Anwendungs-Datei: 17-01-02_VBComponents.mpp

1 VBComponents-Auflistung

Die VBComponents-Objektliste verwaltet die in einem Projekt enthaltenen Komponenten. Eine Komponente kann das Projekt-Objekt (*ThisProject*), ein Formular *(UserForm)*, ein Modul (*Modul*) oder eine Klasse (*Klasse*) sein, die in *Klassenmodule* oder *Standardmodule* unterteilt werden.

Mit einer *For-Each-Next*-Schleife können die einzelnen Komponenten angesprochen werden.

Codeliste 1. Die Prozedur liest alle Komponenten eines Projekts

```
Sub ReadAllComponents()
   Dim objComponent As Object

   For Each objComponent In ActiveProject.VBProject.VBComponents
      Debug.Print objComponent.Type, objComponent.Name
   Next
End Sub
```

Das Ergebnis wird im Direktfenster ausgegeben (Bild 1)

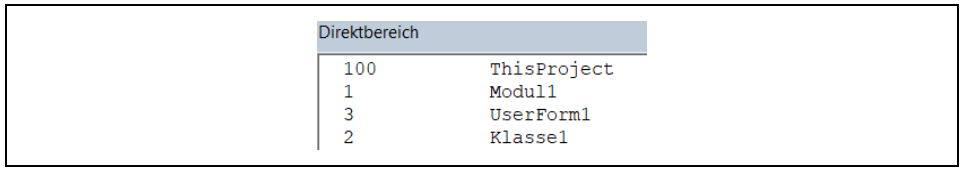

Bild 1. Liste der vorhandenen Komponenten im aktiven Projekt

2 VBComponents-Eigenschaften

Die *Count*-Eigenschaft liefert die Anzahl der Komponenten in der Auflistung. Mit einer *For-Next*-Schleife lassen sich die Komponenten in der Auflistung ebenfalls lesen.

Codeliste 2. Die Prozedur liest alle Komponenten eines Projekts

```
Sub ReadAllComponents2()
   Dim iCount As Integer

   For iCount = 1 To ActiveProject.VBProject.VBComponents.Count
      With ActiveProject.VBProject.VBComponents.Item(iCount)
         Debug.Print .Type, .Name
      End With
   Next
End Sub
```

Die *Type*-Eigenschaft einer Komponente verweist auf das VBComponent-Objekt. Die *Name*-Eigenschaft liefert den Namen der Komponente. Die *Item*-Methode ermöglicht den Zugriff auf die einzelnen Elemente der Auflistung.

Codeliste 3. Die Prozedur exportiert alle vorhandenen Komponenten

```
Sub AllComponentsExport()
   Dim objVBComponent   As Object
   Dim sType            As String

   For Each objVBComponent In ActiveProject.VBProject.VBComponents
      With objVBComponent
         Select Case .Type
         Case 1: sType = ".bas"
         Case 2: sType = ".cls"
         Case 3: sType = ".frm"
         Case 100: sType = ".mpp"
         End Select
      End With
      objVBComponent.Export ("C:/Temp/Komponenten/" & _
         objVBComponent.Name & sType)
   Next
End Sub
```

Nach der Ausführung stehen die Komponenten im Verzeichnis C:/Temp/Komponenten (Bild 2).

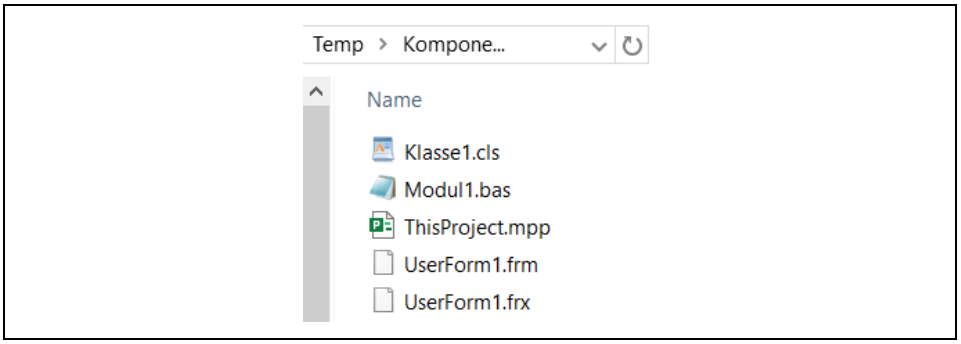

Bild 2. Verzeichnis mit den Komponenten

Die *Parent*-Eigenschaft der Auflistung liefert das Projekt, indem sich die Auflistung befindet.

3 VBComponents-Methoden

Die *Add*-Methode erzeugt eine neue Komponente im aktiven Projekt. Zur Erzeugung wird die Typangabe benutzt.

Codeliste 4. Die Prozedur fügt eine neue Komponente in das aktive Projekt ein

```
Sub InsertCodeModul()
   ActiveProject.VBProject.VBComponents.Add (1)
End Sub
```

Nach der Ausführung besitzt das aktive Projekt ein weiteres Code-Modul mit dem Namen *Modul2*. Mit jeder neuen Einfügung erhöht sich die angehängte Zahl (Bild 3).

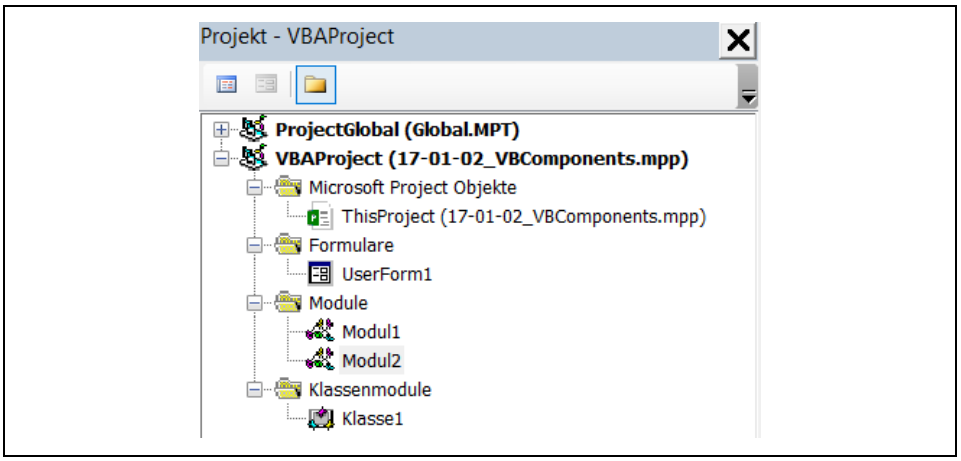

Bild 3. Die Komponenten des Projekts im Projekt-Explorer

Mit der *Import*-Methode können Komponenten dem Projekt hinzugefügt werden.

Codeliste 5. Die Prozedur importiert ein Code-Modul

```
Sub ImportCodeModul()
   ActiveProject.VBProject.VBComponents.Import _
      ("C:/Temp/Komponenten/Modul2.bas")
End Sub
```

Mit der *Remove*-Methode lassen sich Komponenten aus dem Projekt entfernen. Dazu muss das entsprechende Objekt angegeben werden.

Codeliste 6. Die Prozedur entfernt ein Code-Modul

```
Sub RemoveCodeModul()
   Dim objComponent As Object

   For Each objComponent In ActiveProject.VBProject.VBComponents
      If objComponent.Name = "Modul2" Then
         ActiveProject.VBProject.VBComponents.Remove objComponent
      End If
   Next
End Sub
```

4 Application.VBE-Eigenschaft

Eine andere Möglichkeit auf Objekte im aktuellen Projekt-Explorer zuzugreifen, ist die Application.VBE-Eigenschaft. Die folgende Prozedur erzeugt einen Kommentarkopf im Codefenster einer Komponente. Er sollte in keiner Komponente fehlen. Die folgende Prozedur ist auch ein Beispiel dafür, wie Code durch Code erzeugt werden kann. Voraussetzung ist allerdings, dass die Objektbibliothek Microsoft Project .Object Library unter Extras / Verweise im VBA-Tool eingebunden ist. Bei einer Laufzeit-Fehlermeldung 1004 ist die Makrosicherheit zu hoch gesetzt und es sollteZugriff vertrauen gesetzt werden, wenigstens für die Zeit der Anwendung.

Codeliste 7. Die Prozedur erzeugt einen Kommentarkopf in einem neuen Codefenster

```
Public Sub CodeHeader()
   Dim objComp As Object

   Set objComp = ActiveProject.VBProject.VBComponents.Add(1)
   objComp.Activate
   With Application.VBE.ActiveCodePane.CodeModule
      .InsertLines 3, "'VBA-Modultyp:    ."
      .InsertLines 4, "'Modulname:       ."
      .InsertLines 5, "'Autor:           ."
      .InsertLines 6, "'Firma:           ."
      .InsertLines 7, "'Abteilung:       ."
      .InsertLines 8, "'Erstelldatum:    " & Format(Date, "dd.mm.yyyy")
      .InsertLines 9, "'Installation:    ."
      .InsertLines 10, "'Application:     Project"
      .InsertLines 11, "'Beschreibung:    ."
      .InsertLines 12, "'"
      .InsertLines 13, "'Änderungen:      ."
      .InsertLines 14, "'"
   End With
   Set objComp = Nothing
End Sub
```

Die Prozedur *CodeHeader* erstellt zunächst ein neues Modul und erzeugt im leeren Codefenster (nur Option Explicit) den nachfolgenden Eintrag (Codeliste 8). Dieser sollte den eigenen Wünschen entsprechend angepasst werden.

Codeliste 8. MitCodeHeader erzeugter Header

```
Option Explicit

'VBA-Modultyp     .
'Modulname:       .
'Autor:           .
'Firma:           .
'Abteilung:       .
'Erstelldatum:    01.12.2023
'Installation:    .
'Application:     Project
'Beschreibung:    .
'
'Änderungen:      .
'
```

Die folgende Prozedur erzeugt ebenfalls eine neue Komponente und erstellt darin die Prozedur *TestModul*, die sie anschließend auch aufruft (Bild 4).

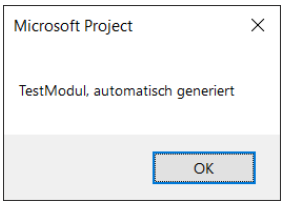

Bild 4. Ausgabe von Prozedur TextModul

Codeliste 9. Die Prozedur erzeugt eine Prozedur in einem neuen Code-Modul

```
Public Sub CreateProcedure()
   Dim objComp As Object

   Set objComp = ActiveProject.VBProject.VBComponents.Add(1)
   objComp.Activate
   With Application.VBE.ActiveCodePane.CodeModule
      .InsertLines 2, ""
      .InsertLines 3, "Public Sub TestModul"
      .InsertLines 4, "'automatisch generiert"
      .InsertLines 5, "   MsgBox ""TestModul, automatisch generiert"""
      .InsertLines 6, "End Sub"
   End With
   Application.Run "TestModul"
   Set objComp = Nothing
End Sub
```

Die nächste Prozedur ändert die Zeile 5 in der zuvor erstellten Prozedur im *Modul3*.

Codeliste 1-10. Prozedur ändert eine Codezeile

```
Sub ÄndereModulCode()
   Dim objComp As Object

   Set objComp = ActiveProject.VBProject.VBComponents ("Modul3")
   With objComp.CodeModule
      .DeleteLines 5
      .InsertLines 5, "   MsgBox ""CodeModul, automatisch generiert"""
   End With
   Application.Run "TestModul"
   Set objComp = Nothing
End Sub
```