Visual Basic

Einführung

Inhalt

 Mag. Rupert Obermüller

Module

Visual Basic unterscheidet drei Modularten, wobei jedes Modul in einer Datei untergebracht ist:

- **Formularmodul (`*.frm`)**
  Benutzerschnittstelle auf Basis eines Formulars, eines Behälters für Kontrollelemente.
  besteht in erster Linie aus Ereignisprozeduren, welche auf vom Benutzer oder dem System ausgelösten Ereignissen reagieren.

- **Standardmodul (`*.bas`)**
  Dient zur Aufnahme von Prozeduren, welche von allen Modulen in Anspruch genommer werden können.

- **Klassenmodul (`*.cls`)**
  Dient zur Erstellung von Klassen, welche dann als Objekte instanziiert werden können. In VB ist jedes Objekt automatisch als COM-Objekt einsetzbar.

Code-Grundlagen

Der Basic-Interpreter ist nicht case-sensitiv.
Anweisungen auf mehrere Zeilen verteilen `_`
Mehrere Anweisungen pro Zeile angeben `:`
Zeile als Kommentar ausweisen `'`

Variable

Der Standard-Datentyp ist der Datentyp `Variant`. Er kann alle Basisdatentypen aufnehmen und bei Bedarf auch konvertieren. Zur Verwendung dieses Datentyps ist keine Deklaration erforderlich.

Ein besseres Laufzeitverhalten ergibt sich allerdings bei der Deklaration eines bestimmten Datentyps

```
dim var [ as typ ]
```

Das Schlüsselwort `dim` legt immer Variable mit lokaler Lebensdauer und Sichtbarkeit an.
Dies kann durch andere Modifizierer geändert werden:

`static` *var*	Sichtbarkeit innerhalb der Prozedur, globale Lebensdauer (nur innerhalb einer Prozedur gültig)
`public` *var*	Sichtbarkeit in allen Modulen, globale Lebensdauer (nur außerhalb einer Prozedur gültig)
`private` *var*	Sichtbarkeit im betreffenden Modul, globale Lebensdauer (nur außerhalb einer Prozedur gültig)

Durch Angabe von

```
option explizit
```

wird man gezwungen, Variablen zu deklarieren.

Konstante

Konstante können nur außerhalb von Prozeduren definiert werden:

```
[ public | private ] const name [ as typ ] = ausdruck
```

Datentypen

Numerische Typen	`integer, long`
Gleitkommatypen	`single, double`
Festkommatypen	`currency`
Binärtypen	`byte`
Zeichenketten	`string` (variable Länge)
	`string*`*`länge`* (fixe Länge)
Wahrheitswert	`boolean`
Datum	`date`
Verweis auf COM-Automation Objekte	`object`

Zeichenkettenkonstante werden in doppelte Anführungszeichen eingeschlossen.
Zeichenketten werden mit dem & Operator konkateniert.

Konvertierungen

Zur expliziten Konvertierung von Datentypen stehen folgende Prozeduren zur Verfügung

Konvertierungsfunktion	auf Datentyp
`cbool`	`boolean`
`cbyte`	`byte`
`ccur`	`currency`
`cdate`	`date`
`cdbl`	`double`
`cint`	`integer`
`clng`	`long`
`csng`	`single`
`cstr`	`string`
`cvar`	`variant`
`cverr`	`error`

Der Datentyp Variant

Dieser muß nicht explizit konvertiert werden.

Beispiel:

```
dim wert
wert = 15
wert = wert - 5
wert = "ergebnis" & wert
```

Er kann außerdem bestimmte Konstante aufnehmen:

`empty`	es wurde noch kein Wert zugewiesen. Prüfung mit `isempty(`*`var`*`)`
`null`	es ist kein Wert verfügbar. Abfrage mit `isnull(`*`var`*`)`, Zuweisung mit `var = null`
`error`	

Arrays

Arrays können durch die Angabe einer Index-Obergrenze quantifiziert werden. Der verfügbare Index läuft dann von 0 bis Anzahl

```
dim var ( Anzahl ) [ as Typ ]
```

Der Indexbereich kann auch in der Deklaration angegeben werden

```
dim var ( Untergrenze to Obergrenze ) [ as Typ ]
```

Mehrdimensionale Arrays können erzeugt werden, indem die Indexgrenzen pro Dimension durch Beistriche getrennt angegeben werden:

```
dim var ( Anzahl1, Anzahl2, ... ) [ as Typ ]
```

Prozeduren

Man unterscheidet drei Typen

- Sub-Prozeduren besitzen keinen Rückgabewert
- Funktionen besitzen einen Rückgabewert
- Property-Prozeduren zur Abfrage und zum Setzen von Eigenschaften in einer Klasse

Sub-Prozeduren

Deklaration

```
[ private | public | static ] Sub name ( argumente )
   Anweisungen
end sub
```

Man unterscheidet

- allgemeine Sub-Prozeduren beliebige Namesgebung
- Ereignis-Sub-Prozeduren vorgegebene Namensgebung

  als Reaktion auf Steuerelement-Ereignisse
  elementname_ereignisname(), z. Bsp. `cmdButton_Click()`

  als Reaktion auf Formular-Ereignisse
  `form_`*ereignisname*`()`, z. Bsp. `Form_Load()`

Funktionen

Deklaration

```
[ private | public | static ] Function name ( argumente ) [ as Typ]
   Anweisungen
end function
```

Der Aufruf kann als Ausdruck verwendet werden. Wird kein Rückgabetyp angegeben, wird der Typ Variant angenommen. Die Wertrückgabe erfolgt durch Zuweisung eines Ausdrucks an den Funktionsnamen.

Beispiel:

```
function getx() as integer
   getx = 1
end function
```

Property-Prozeduren

Um die Daten eines Klassenmoduls zu schützen werden sie meist als private deklariert. Um dennoch einen kontrollierten Zugriff darauf zu gestatten können die Property-Prozeduren verwendet werden.

Dabei deklariert man eine Prozedur für den Zugriff:

```
public property get name() [ as typ ]
   Anweisungen
end property
```

und eine Prozedur für das setzen der Eigenschaft:

```
public property let name(byval value [ as typ ] )
   Anweisungen
end property
```

Prozeduraufrufe

Sub-Prozeduren und Funktionen können auf zwei Arten aufgerufen werden:

```
call prozname ( arg1, arg2, ... )
prozname arg1, arg2, ...
```

Prozeduren in Formularen müssen mit mit dem Formularnamen qualifiziert werden

```
formularname.prozname arg1, arg2, ...
```

Prozeduren in Klassenmodulen können erst dann angesprochen werden, wenn eine Instanz der entsprechenden Klasse existiert

```
dim objektname as new klassenname
objektname.prozname arg1, arg2, ...
```

Prozeduren in Standardmodulen können im allgenmeinen mit ihrem Namen allein aufgerufen weden. Bei Namensgleichheit von Prozeduren in anderen Modulen muß die Prozedur mit dem Modulnamen qualifiziert werden:

```
modulname.prozname arg1, arg2, ...
```

Argumente von Prozeduren

Wertübergabe

```
byval argname [ as typ ]
```

Referenzübergabe (Standard)

```
[ byref ] argname [ as typ ]
```

Optionale Argumente

```
optional argname [ as typ ] [ = wert ]
```

Kontrollstrukturen

Verzweigungen

```
if bedingung then anweisung

if begingung then
   Anweisungen
endif

if bedingung then
   Anweisungen
[ elseif bedingung then
   Anweisungen ]
[else
   Anweisungen ]
endif

select case ausdruck
case Ausdruckliste
   Anweisungen
case Ausdruckliste
   Anweisungen
 ...
case else
   Anweisungen
end select
```

Schleifen

```
do while Bedingung
   Anweisungen
loop

do
   Anweisungen
loop while bedingung

for var = startvar to endvar [ step schrittweite ]
   Anweisungen
next [ var ]

for each Element in Gruppe
   Anweisungen
next element
```

`Gruppe` kann eine Auflistung oder ein Array sein.
`Element` kann ein beliebiger Datentyp sein.

Objekte

Objekte sind Instanzen von Klassenmodulen, welche eine COM-Klasse beschreiben. Auf diese Instanzen wird mittels einer sogenannten Objektvariablen verwiesen.

```
dim objvar as klassenname
```

Bevor die Eigenschaften und Methoden des Objektes verwendet werden können, muß das Objekt instanziiert werden:

```
set objvar = new klassenname
dim objvar as new klassenname
```

Die Eigenschaften bzw. Methoden müssen dann mit dem objektnamen qualifiziert werden:

```
objvar.property = wert
var = objvar.property

objvar.prozname arg1, arg2, ...
```

Beispiel: Verwendung des TipOfDay-Objektes

Aus dem Menu Project - References wird die betreffende Typbibliothek (TipOfDay Bibliothek) ausgewählt.

Dann können die Objekte mit `New` angelegt werden, was einem `CoCreateInstance()`-Aufruf entspricht.
Visual Basic geht jedoch davon aus, daß die erste definierte Schnittstelle die Standardschnittstelle ist, welche "versteckt" wird und mit dem Objektnamen angesprochen werden kann.

Jede weitere Schnittstelle kann deklariert werden und mit dem `Set`-Schlüsselwort wird ihr die Objektreferenz zugewiesen, was einem `QueryInterface()`-Aufruf entspricht.

```
Private tip As New CoTipOfDay
Private nr  As Integer
Private str As String

Private Sub Form_Load()
    Dim tip2 As ITipOfDay2
    Set tip2 = tip
    tip2.LoadTips "tiptext.tip"
    tip2.GetNextTip nr, str
    TipText.Text = str
End Sub

Private Sub nextButton_Click()
    nr = nr + 1
    tip.GetNextTip nr, str
    TipText.Text = str
End Sub
```

Auflistungsobjekte

Auflistungsobjekte sind eine Sammlung von Variant-Datentypen. Jedes Objekt in der Auflistung kann über einen Zeichenketten-Schlüssel oder einem bei 1 beginnenden Index angesprochen werden

```
auflistungsobj ( "schlüssel" )
auflistungsobj ! schlüssel
auflistungsobj ( index )
```

Beispiele:

`Forms`	alle geladenen Formulare einer Anwendung
`Controls`	alle Kontrollelemente eines Formulars

Methoden

- **`Add`** Element hinzufügen

```
Sub Add ( Element As Variant [, Schlüssel As Variant]
          [, before As Variant] [, after As Variant] )
```

- **`Item`** Zugriff auf ein Element

```
[Set] Variable = Objekt.Item(Index)
```

- **`Remove`** Element löschen

```
Objekt.Remove Index
```

Eigenschaften

- **`Count`** Anzahl der Elemente bestimmen

Erstellung von COM-Server

Visual Basic versteckt bei der Erstellung eines Servers viel Details, sodaß diese sehr schnell vorgenommen werden kann. Folgende Vorgangsweise ist nötig:

- **Erstellen einer ActiveX-DLL**
  Der Projektname dient außerdem als Name der Typbibliothek
  Der Klassenmodulname wird als Name der COM-Klasse verwendet
  Die ProgID wird dann aus diesen beiden Namen zusammengesetzt.

Dann muß man eine Entscheidung treffen:

- **Erstellen des COM-Objektes sozusagen von Anfang an**
  Hier muß man die Einschränkung in Kauf nehmen, daß das Objekt nur eine Schnittstelle besitzen kann. Diese erhält den Namen `_<COM-Klassenname>`.
  In dem Modul hinzugefügte Methoden werden automatisch Methoden der Schnittstelle.

- **Erstellen des COM-Objektes auf Grundlage einer vorhandenen Typbibliothek.**
  Soll das COM-Objekt mehrere Schnittstellen unterstützen, so ist diese Vorgangsweise unerlässlich. Die Typbibliothek der gewünschten COM-Klasse kann mit einem beliebigen Werkzeug (z. Bsp. Visual C++) erzeugt werden.
  In Visual Basic richtet man eine Refernz auf diese Typbibliothek ein und gibt an, welche Schnittstellen man implementieren will, z. Bsp.

```
Implements DataStore.IStringArray
```

  Dann kann man mit dem Editor die Methodenrümpfe erstellen lassen und diese füllen.