Lispschnittstelle für VBA

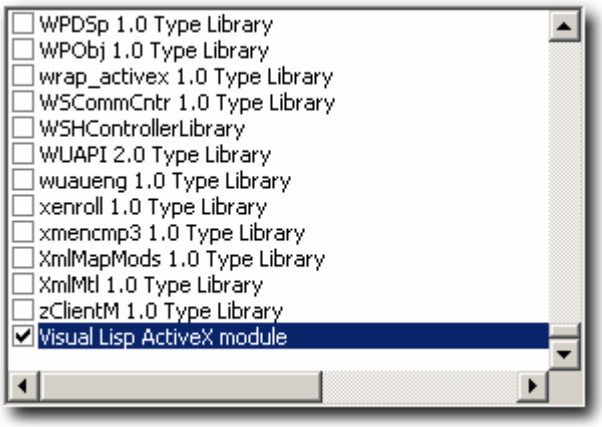
1. Vorweg

Normalerweise wird ein Lisp-Befehl in VBA über *ThisDrawing.SendCommand "(<LISP-BEFEHL MIT PARAMETERN>)" & vbCr* aufgerufen. Eine sinnvolle Rückgabe ist dabei logischerweise nicht zu erwarten.
Die hier vorgestellte Methode benutzt einen anderen Weg. Sie ruft Lisp-Befehle über das Visual-Lisp ActiveX Modul auf. Dadurch kann nicht nur der Rückgabewert sinnvoll weiterverwendet werden, sondern Variablen gesetzt und ausgelesen, ja ganze Lisp-Funktionen ferngesteuert werden.
Die Zusammenstellung ist durch diverse Internetrecherchen entstanden. Hier ein paar der wichtigsten Seiten:

1.1. http://www.augi.com/publications/hotnews.asp?page=578
1.2. http://ww3.cad.de/foren/ubb/Forum259/HTML/000399.shtml
1.3. http://discussion.autodesk.com/thread.jspa?messageID=378137

2. Installation der tlb in VBA

Fügen Sie im VBA-Editor unter Extras – Verweise das Visual Lisp ActiveX Modul hinzu. Sollte es nicht aufgeführt sein, finden Sie es im Programmverzeichnis von AutoCAD unter dem Namen VL16.TLB.
Bevor wir das Interface jedoch nutzen können, muss in Autocad einmal der Befehl (vl-load-com) ausgeführt werden.



3. Initialisieren des Interface-Objekts

Zunächst werden im Programmcode im Deklarationsabschnitt die Variablen für eine spätere Bindung deklariert:

```
Dim VL As Object
Dim VLF As Object
```

Danach kann die eigentliche Initialisierung erfolgen:

```
Sub VL_Initialize()
  ThisDrawing.SendCommand ("(vl-load-com)" & vbCr)
  Set VL =
ThisDrawing.Application.GetInterfaceObject("VL.Application.16")
    Set VLF = VL.ActiveDocument.Functions
End Sub
```

Nun kann über das VLF-Objekt (VisualLispFunktionen) auf die Lisp-Objekte zugegriffen werden, Lisp-Funktionen ausgeführt und Variablen gesetzt werden.

Die zentrale Funktion ist des VLF-Objektes ist die ITEM()-Funktion. Hiermit können alle bekannten Lispfunktionen ausgeführt werden.
Sehr häufig gebraucht werden die Funktion READ und EVAL. READ deshalb, weil alle Initialübergaben an das Interface in Strings erfolgen, und EVAL um eine Evaluation der Funktion durchzuführen.
Daher sollte man diese Funktionen gleich an ein weiteres Objekt binden:

```
Dim VLRead As Object
Dim VLEval As Object
Set VLRead = VLF.Item("read")
Set VLEval = VLF.Item("eval")
```

4. Arbeiten mit dem Interface-Objekt

4.1. auslesen von Variablenwerten

Um den Wert einer Variablen abzurufen muss sie evaluiert werden. Dazu haben wir bereits ein Objekt erstellt – VLEval.
Um an eine Funktion einen (oder mehrere) Parameter zu übergeben, muss sie mit der Methode funcall(<Parameter1>, <Parameter2>, ...) aufgerufen werden. Bei unserem VLEval also mit *VLEval.funcall(symbolname).* Ebenso verhält es sich natürlich mit der Funktion VLRead.

```
Function GetLispSymbol(symbolname As String)
  Dim sym As Object
  Set sym = VLRead.funcall(symbolname)
  GetLispSymbol = VLEval.funcall(sym)
End Function
```

Nun können wir unsere erste Variable abrufen. Angenommen, in Autocad hätten wir mit
`(setq var1 12.23)`
die dem Symbol var1 den Wert 12.23 zugewiesen, dann können wir diesen Wert nun abrufen:

```
Sub test()
  VL_Initialize
  result = GetLispSymbol("var1") 'zahl
  VL_Terminate
End Sub
```

VL_Terminate ist dabei das Gegenstück zu VL_Initialize. Die Bindungen werden wieder freigegeben:

```
Sub VL_Terminate()
  Set VLEval = Nothing
  Set VLRead = Nothing
  Set VLF = Nothing
  Set VL = Nothing
End Sub
```

4.2. setzen von Variablenwerten

Und wie setzt man einen Variablenwert?
Der Lisp-Befehl dazu lautet SET. Er wird wieder mit der ITEM-Funktion übergeben:

```
Function SetLispSymbol(symbolname As String, byval value)
  Dim sym As Object, ret
  Set sym = VLRead.funcall(symbolname)
```

```
  ret = VLF.Item("set").funcall(sym, value)
End Function
```

Dabei ergibt sich ein kleines Problem. Die Datentypen werden dabei nicht übertragen, sondern werden in Lisp alle zu variant. Um sie korrekt in Lisp verwenden zu können, müssen Sie noch in den jeweiligen Typ umgewandelt werden.

4.3. ausführen von Lispausdrücken

Am einfachsten wäre es, wenn wir in Lisp bereits eine entsprechende Funktion definiert hätten:

```
(defun translate-variant (data)
  (cond
    ((= (type data) 'list)
     (mapcar 'translate-variant data))
    ((= (type data) 'variant)
      (translate-variant (vlax-variant-value data)))
    ((= (type data) 'safearray)
      (mapcar 'translate-variant (vlax-safearray->list data)))
    (t data)
  )
)
```

Wir haben ja bereits einen Befehl zum Evaluieren von Lispvariablen kennengelernt: VLEval. Mit der gleichen Funktion können wir aber auch ganze Lisp-Ausdrücke evaluieren:

```
Function EvalLispExpression(lispStatement As String)
    Dim sym As Object, retval
    Set sym = VLRead.funcall(lispStatement)
    On Error Resume Next
    retval = VLEval.funcall(sym)
    If Err Then
        EvalLispExpression = ""
    Else
        EvalLispExpression = retval
    End If
End Function
```

Mit dieser neuen Funktion können wir jetzt die Varablenwerte korrekt zuweisen:

```
Function SetLispSymbol(symbolname As String, byval value)
    ...
    EvalLispExpression "(setq " & symbolname & "(translate-variant " &
symbolname & "))"
End Function
```

5. kompletter Code mit weiteren Beispielen

```
Dim VL As Object
Dim VLF As Object
Dim VLRead As Object
Dim VLEval As Object

Sub VL_Initialize()
    ThisDrawing.SendCommand ("(vl-load-com)" & vbCr)
    Set VL = ThisDrawing.Application.GetInterfaceObject("VL.Application.16")
    Set VLF = VL.ActiveDocument.Functions
    Set VLRead = VLF.Item("read")
    Set VLEval = VLF.Item("eval")
End Sub

Sub VL_Terminate()
    Set VLEval = Nothing
    Set VLRead = Nothing
    Set VLF = Nothing
    Set VL = Nothing
End Sub

Function EvalLispExpression(lispStatement As String)
    Dim sym As Object, retval
    Set sym = VLRead.funcall(lispStatement)
    On Error Resume Next
    retval = VLEval.funcall(sym)
    If Err Then
        EvalLispExpression = ""
    Else
        EvalLispExpression = retval
    End If
End Function

Function SetLispSymbol(symbolname As String, ByVal value)
    Dim sym As Object, ret
    Set sym = VLRead.funcall(symbolname)
    ret = VLF.Item("set").funcall(sym, value)
    EvalLispExpression "(defun translate-variant (data) (cond ((= (type " & _
      "data) 'list) (mapcar 'translate-variant data)) ((= (type data)
'variant)" & _
      "(translate-variant (vlax-variant-value data))) ((= (type data)
'safearray)" & _
      "(mapcar 'translate-variant (vlax-safearray->list data))) (t data)))"
    EvalLispExpression "(setq " & symbolname & "(translate-variant " &
symbolname & "))"
    EvalLispExpression "(setq translate-variant nil)"
End Function

Function GetLispSymbol(symbolname As String)
    Dim sym As Object, list As Object
    Dim elements() As Variant, i As Long, Count As Integer, art As String
    art = Lispvartype(symbolname)
    Set sym = VLRead.funcall(symbolname)
    If art = "LIST" Then
      Set list = VLEval.funcall(sym)
      Count = VLF.Item("length").funcall(list)
      ReDim elements(0 To Count - 1) As Variant
      For i = 0 To Count - 1
        elements(i) = VLF.Item("nth").funcall(i, list)
      Next
      GetLispSymbol = elements
```

```
    Else
      GetLispSymbol = VLEval.funcall(sym)
    End If
End Function

Function GetLispList(symbolname As String) As Variant
  Dim sym As Object, list As Object
  Dim Count, elements(), i As Long
  Set sym = VLRead.funcall(symbolname)
  Set list = VLEval.funcall(sym)
  Count = VLF.Item("length").funcall(list)
  ReDim elements(0 To Count - 1) As Variant
  For i = 0 To Count - 1
        elements(i) = VLF.Item("nth").funcall(i, list)
  Next
  GetLispList = elements
End Function

Function Entlast() As AcadEntity
  Dim retval As String
  EvalLispExpression ("(defun *vox-entlast* ( / ele) (if (setq ele (entlast))
(cdr (assoc 5 (entget ele)))))")
  retval = VLF.Item("*vox-entlast*")
  EvalLispExpression ("(setq *vox-entlast* nil)")
  If retval <> "" Then Set Entlast = ThisDrawing.HandleToObject(retval)
End Function

Function Lispvartype(symbolname As String) As String
  EvalLispExpression ("(defun *vox-type* (symbolname) (vl-prin1-to-string
(type (eval (read symbolname)))))")
  Lispvartype = VLF.Item("*vox-type*").funcall(symbolname)
  EvalLispExpression ("(setq *vox-type* nil)")
End Function

Sub NullifySymbol(ParamArray symbolname())
    Dim i As Integer
    For i = LBound(symbolname) To UBound(symbolname)
        EvalLispExpression "(setq " & CStr(symbolname(i)) & " nil)"
    Next
End Sub

Sub test()
Dim result0 As AcadEntity, result1, result2, result3, result4
Dim liste(100) As Integer, zahl As Double, text As String
VL_Initialize
For i = 0 To 100
  liste(i) = i
Next
zahl = 12.34
text = "testtext"
result1 = SetLispSymbol("test1", liste) 'Liste
result2 = SetLispSymbol("test2", zahl) 'zahl
result3 = SetLispSymbol("test3", leer) 'nil
result4 = SetLispSymbol("test4", text) 'text
Set result0 = Entlast
result1 = GetLispSymbol("test1") 'Liste
result2 = GetLispSymbol("test2") 'zahl
result3 = GetLispSymbol("test3") 'nil
result4 = GetLispSymbol("test4") 'text
VL_Terminate
End Sub
```