LotusScipt Kurzreferenz

by udo junghans (junghans<at>eras.de)
version 0.9 (15.05.2013)

1. Einfache Syntax

Print "Hello" eine Anweisung, eine Zeile
Print "H" & _ 'Kommentar
"ello" eine Anweisung, zwei Zeilen mit Kommentar
Print "He" : Print "llo" zwei Anweisungen auf einer Zeile
„", {}, || Zeichenkette (String)
' Kommentar
%REM … %END REM Kommentar über mehrere Zeilen
… _ Codezeile umbrechen
twoLiner = |Diese Message geht
über mehrere Zeilen|

Literale:
„1234"; „123.34"; „13E-8";
„&B1010" bin
„&O12" octal
„&H47" hex

2. Variablen and Datentypen

Dim | Public | Private | Static name as type
einfache Variable
Public → öffentlich sichtbar (Klasse, Modul)
Private → nur für Klasse, Modul nutzbar
Static → nur in prozeduren, Inhalt bleibt bis zum nächsten Aufruf erhalten
Dim Name(Borders) As Type Array
Dim Name(0 To 8, 17 To 20, 0 To 4) As Type
Multidimensionales Array
Dim Name List as Type Liste
Dim Object As (New) Class Klasse
Const Name = Expr Konstante
Type TypeName Typdefinition
Liste von Elementen
End Type

Vordefinierte Datentypen:
Boolean (True,False)
Byte (0,255)
Currency (-922.337.203.685.477,5807;+…)
Double (-1,79769313486231*10^308;+…) #
Integer (-32768;+…) %
Long (-2.147.483.648;+…) &
Single (-3,402823*10^38;+…) !
String (Speicherabhängig) $
Variant (im Moment der Belegung festgelegt)

Initialisieren mit
Dim A As String oder Dim A$
Dim A$, B$, C$
ReDim →Dynamic Array [Dim arr() as String _ for i=1 to 10 _ Redim Preserve arr(i)_arr(i)="mm"_next]

```
With NotesDocument
        .FieldOne = "Value One"
        .FieldTwo = "Value Two"
End With
```

3. Operatoren

=	Zuweisung
+, -, *, /	Addition, Subtraktion, Multiplikation, Division
****	Integer Division (VB)
Mod	Rest der Division
^	Exponent
&, +	Strings (Zeichenketten) verbinden

4. Boolsche Operatoren

NOT Nicht
AND Und
OR Oder
XOR entweder oder
Eqv Boolsches oder bitweises logisches Equivalent:
-> Wahr, wenn A und B gleichzeitig wahr oder gleichzeitig falsch sind, sonst falsch
Imp Boolsche oder bitweise Implikation:
->Wenn es regnet, dann ist die Strasse nass -> möglich oder nicht)

5. Vergleich (Comparison)

=, >, <, >=, =>, <=, =<, <>, ><, LIKE
Is, IsA Testet den Objekttyp
strCompare Zeichenketten vergleichen
OPTION COMPARE [NoCase | Binary]

6. Typumwandlung (Type conversion)

CBool [Irgenwas nach Bool], **CByte**, **CCur** [Irgenwas nach Währung, **CDat** or **CVDat** [Irgenwas nach Datum (Variant)] **CDbl** [nach double], **CInt**, **CLng**, **CSgn** [nach single], **CStr** [nach String], **CVar** [nach variant], **Str** [Zahl nach String], **Format**(value, $Format), **Val** [String nach double]

7. Mathematische Funktionen

Abs() Betrag, Winkelfunktionen: **ACos**() Ergebnis in Rad, **ASin**(), **ATn**(), **ATn2**(), **Cos**(),**Sin**(), **Exp**() Potenzierung, **Fix**() ganzahliger Teil einer Zahl), **Fraction**() gibt den gebrochenen Anteil einer Zahl zurück, **Int**() Integer immer abgerundet!, **Log**(), Mod() Modulo, **Rnd**() Zufallszahl 0..1, **Round**(1288;-2) -> 1300, **Sgn**() Vorzeichen, **Sqr**() Quadratwurzel, **Tan**()

8. Date functions

Date() Today
DateNumber(nYear, nMonth, nDay) → dd.mm.YYYY
DateValue(strDate) → dd.mm.YYYY
Day(DateString) → dd
Hour(DateString) → HH
Minute(DateString) → MM
Month(DateString) → mm
Now(), **Time**() → actual Time
Second(DateString) → ss
TimeNumber(), **TimeSerial**(nHour, nMinute, nSec) →timeString
TimeValue(strDateTime) → Timestring
Timer() → seconds since midnight
Today() → actual Date
Weekday(DateTime) → 1…7 ^= Sun…Sat
Year(DateTime) → Year%

9. String Functions

!! **Suffixes**: possible form for instance Len, LenB, LenBP, LenC;
without suffix: calculate pos in characters
…**B**: pos in bytes; …**BP**: pos in Byte depending on OS, …**C** for Asian langs

& concatenates strings

CStr (*expr*) convert any to string

InStr([pos,]Str1, str2,comp) searches/returns Position of str2 in str1 after pos

Join(List,delim$) Joins Members of a list into one string

Left[B,BP,C](str$,pos%) return substring from left to pos%

Len[B,BP,C](str$) String length

LCase wandelt String in Kleinschreibung (→UCase)

InStr[B,BP,C]([pos%],str$,delim$) gives pos of delimiter in str$ starting from pos%

LSet, RSet str1$ = str2$ ersetzt einen String durch einen Teilstring

Mid[B,BP,C](str$,start%,end%) cut/change substring from str$

Replace (*sourceArray***,** *findArray***,** *replacementArray* **[,** *start* **[,** *count* **[,** *compMethod*]]]**)** replaces substrings in strings, sourceArray, findArrayreplacementArray can!! be arrays

Right[B,BP,C](str$,pos%) return substring from right to pos%

String(*stringLen*,*stringExpr*) Returns a string of Length stringLen consisting of StringExpr

StrCompare (*string1*,*string2*[,*compMethod*]) compares 2 strings

StrConv konvertiert eine Zeichenkette hinsichtlich Groß- und Kleinschreibung und in verscheidene Zeichensätze

String(Zeichen,Anzahl) erzeugt einen String aus Anzahl mal dem Zeichen

StrLeft, StrRight(str$, Pattern$[, comp%, occurrence&]) search from left (right) and returns chars left (right) of pattern$ (**StrLeftBack, StrRightBack** search from right(left) and returns pattern to the left (right))

StrToken cut/change substring from str$ searching for pattern$

Space makes space characters

Split(Str$[, delim$, count%, comp%]) exploding of strings to array

Trim, LTrim, RTrim, FullTrim cut off spaces, Fulltrim cuts multiple spaces in string, \t and \n

UCase wandelt String in Großschreibung (->LCase)

Zeichencodes:

Asc(String) gibt den ASCII-Wert des ersten Zeichens des Strings zurück

Uni(String) gibt den UniCode-Wert des ersten Zeichens des Strings zurück

Chr(Integer)**, UChr**(Integer) Umwandlung der Integer in ASCII oder UniCode-Zeichen

Escape sequences:

\a Alarm or bell character
\b Backspace character
\f Form feed character
\n New line character
\t Tab character
\r Carriage return
**** Single backslash
\v Vertical tab character

\0bbb Octal conversion (where bbb is the octal number to convert to a character)

\0xhh Hexadecimal conversion (where hh is the hexadecimal number to convert to a character)

\nnn Integer conversion (where nnn is the decimal number to convert to a character)

10. Array Funktionen

→ Indizierung von Array beginnt normalerweise mit 0

ArrayAppend(Arr1, Arr2) Arrays verketten

ArrayGetIndex(Array, Str) gibt die Position der von Str im Array zurück

ArrayReplace(Array, [StrAlt/ArrAlt], [StrNeu/ArrNeu]) ersetzt StrAlt durchStrNeu im Array (ersetzt nur komplette Elemente)

ArrUnique(Array) entfernt doppelte Elemente

Dim Name(Borders) As Type Array deklarieren

Dim Name(0 To 8, 17 To 20, 0 To 4) As Type Multidimensionales Array deklarieren

Erase(Array) Inhalt des Arrays löschen

ForAll element in Array … end ForAll Iteration über (eindimensionales) Array

For Iteration über Array – bei mehrdimensionalen Arrays zeilen- oder spaltenweises Vorgehen möglich

FullTrim(Array) entfernt Leerzeichen am Anfang und Ende aller Elemente des Arrays

Implode(Array[Trenner]) String aus Array erzeugen

LBound(Arr [,Dimension]) gibt den niedrigsten Indexwert des Arrays aus

ReDim(Array) Neudimensionierung dynamischer Arrays)

Replace(ArrAlt, ArrScan, ArrReplace) Arrayweite Ersetzung von Zeichen(-Ketten) z.B. wenn Element 5 aus ArrScan in einem Element in ArrAlt gefunden wird, wird es durch Element 5 aus ArrReplace ersetzt

UBound(Arr [,Dimension]) gibt den höchsten Indexwert des Arrays aus

11. ListenFunktionen

Listen sind unsortierte Sammlungen nach Schema „Name -> Element“; können nicht mit Index angesprochen werden

Dim ListName **List As** Type Listen deklarieren

Erase(Liste(Element)) Entfernt Eintrag aus Liste

Erase(Liste) Entfernt Liste

IsElement prüft, ob das Element in der Liste vorkommt

IsList ist die Variable eine Liste

ForAll element in List … end ForAll Iteration über Liste

ListTag ermittelt den Namen zu einem Element

12. Proof Datatypes

Int = **DataType**(Expr), **IsArray**, **IsDate**, **IsEmpty**, **IsList**, **IsNull**, **IsNumeric**, **IsObject**, **IsScalar**, **IsUnknown**, **TypeName** gibt Datentyp zurück

13. Program flow

IF Cond THEN … [ELSEIF… THEN…|ELSE…] END [IF]

IF… GOTO Label ELSE … → Bad Style!!

SELECT CASE … Case Is… Case x To y …. CASE ELSE … END SELECT

DO WHILE (Cond is True) …. LOOP
DO … LOOP WHILE (Cond is True)
→ loop as long cond is true
DO UNTIL (Cond is False) …. LOOP
DO … LOOP UNTIL (Cond is False)
→ loop as long cond is false
Goto → jump to Label in this routine → Bad Style!!
GoSub → Jump to Label in same Function/Sub/Property → Bad Style!!
ON nDiscriminant Gosub Label1$, Lab2$, Lab3$...
FOR nCount=nStart TO nEnd [STEP nInc.]
…….NEXT [nCount]
FORALL element In [List | Array]…. END FORALL
WHILE (true) … WEND (cannot be escaped with Exit!)

EXIT [Sub | Function | Property | Do | For | Forall]
Return → go back after GoSub / ON…GoSub
END MessageCode% → Ends Script execution

Errorhandling:
ON ERROR nErrornum GOTO subErrorHandler

subErrorhandler:
…
Err = 0
[RESUME (resumes at error line) | EXIT SUB (Exit Script) | RESUME NEXT (resumes at next line)]

ON ERROR nErrNumber RESUME NEXT
Error nErrorNum, ErrText$ - lost Error aus
Erl – line number of last error
Err – errornumber of last error

14. Function and Procedure

Functions:
[Static][Public | Private] **Function** *functionName* [(%|&|!|#|@|$][(*param1, param2,*)][As *dataType*]
…
functionName = "ReturnValue"
End Function
Static: Optional. Specifies that the values of the function's local variables are saved between calls to the function.

Public | Private: Optional. Public specifies that the function is visible outside the scope (module or class) where the function is defined, as long as that remains loaded. Private specifies that the function is visible only within the current scope.
A function in module scope is Private by default; a function in class scope is Public by default.
Definition of returntype: either "%|&|!" or "As DataType"
Param: [ByVal] *paramName* [() | List] [As *dataType*]
ByVal – Call by Value, default call by reference

Procedures:
[Static] [Public | Private] Sub *subName* [(*param1, param2*)]
…
End Sub
Param: [ByVal] *paramName* [() | List] [As *dataType*]
ByVal – Call by Value, default call by reference

Invoke functions or procedures:
Call *subOrFunction* [([*argList*])] Parentheses can be omitted if there are no args
subOrFunction [*arg1, arg2,…*] without parentheses args are called by reference
subOrFunction [(*arg1, arg2,…*)] with parentheses args are called by value
returnVal = *function* [**(** [*argList*] **)**]

15. Property Procedure

[Public | Private | Friend] [Static] Property Get name [(arglist)] [As type]
[statements]
[Exit Property]
[statements]
End Property

16. Objects

Class Name
Eigenschaften
Methoden
End class

Set xx = yy.nn Zuweisung von Objektreferenzen

17. File Handling

fileNum% = Freefile()
fileName$ = "…"
Open FileName$ for *Accesstype* As fileNum%
Print #fileNum% , "Complex String"
Write
Write
Close fileNum%

18. Code Options on Module level

This Options should be declared in "**(Options)**"
Option DefType DataType set the default datatype if no type is mentioned (normally this option is set to Variant)
Option Base int sets first index of arrays to 1,2,… (Normally set to 0)
Option Compare (Binary | Case | Nocase, Pitch |NoPitch] how should strings be compared
Option Declare or Option Explicit if set, Variable must be declared with Dim before used
Option Public every Sub or class is available public if not declared with "Private"

19. Including external Code

Var = Evaluate(Macro$[, Notesdocument]) Call @Formula Statement from within Script

%include "Pathstring" includes and compiles external .lss-File, Source is hidden from foreign developers

%if Const1 … %ElseIf Const2 … %Else … %END IF
asking for OS-Constants: [WIN32, WINDOWS, UNIX, SOLARIS]