Lua Grundlagen

Einführung in die Lua Programmiersprache

05.05.2014
Ingo Berg
berg@atvoigt.de
Automatisierungstechnik Voigt GmbH

Die Lua Programmiersprache

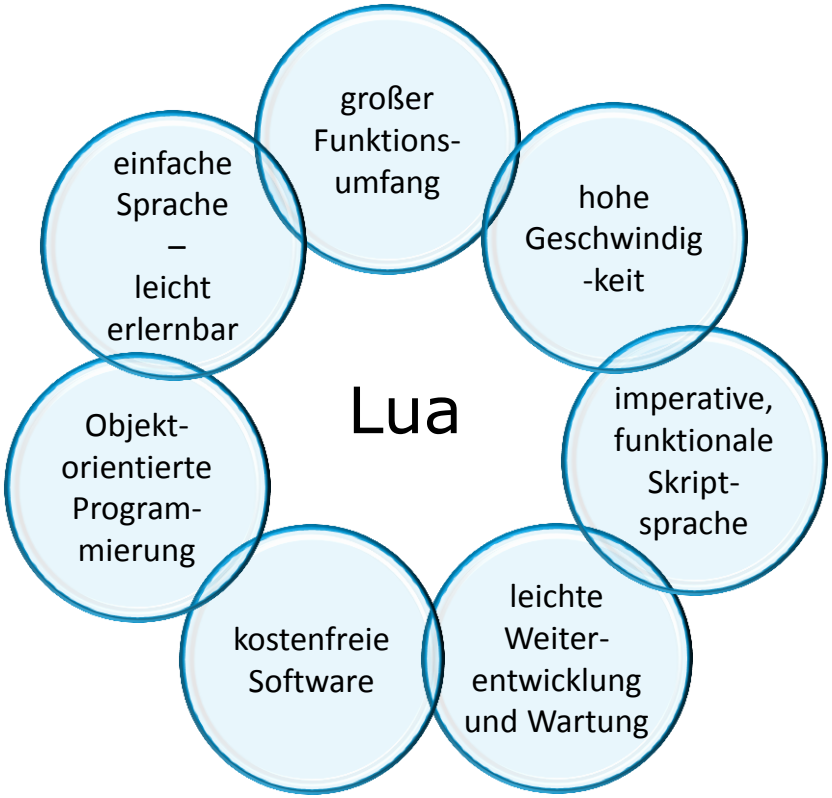
AutomatisierungsTechnik Voigt GmbH

Was ist Lua?

- freie Programmiersprache
- speziell entwickelt für eingebettete Systeme von *Computer Graphics Technology Group* der Päpstlich Katholischen Universität von Rio de Janeiro (Brasilien)

```
for i = 1, 100 do
  xval[i] = i-50

  if (sys.control_var==2) then
    yval[i] = 3 * math.sin(n/5) * math.sin(i/5 + n/3)
  elseif (sys.control_var==3) then
    yval[i] = 3 * math.sin(i/5 + n/3)
  elseif (sys.control_var==4) then
    yval[i] = 3 * math.tan(i/5 + n/3)
  end
end
```



Praktische Anwendung

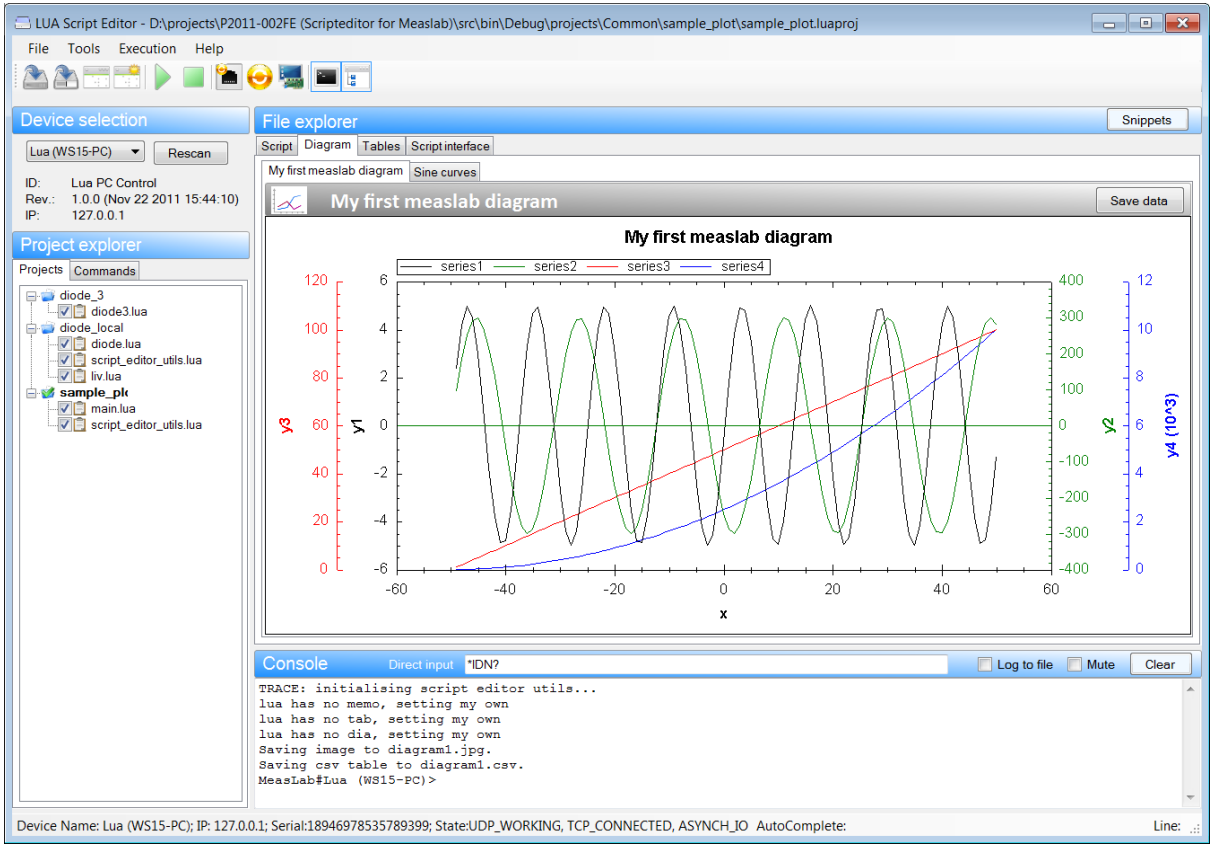
AutomatisierungsTechnik Voigt GmbH

TSP Skripte

- TSP steht für „Test Script Processor“
- Programmiersprache für Messgeräte von KEITHLEY Instruments
- Lua (5.1) ist die Programmiersprache der TSP-Skripte
- TSP Skripte erweitern Lua mit speziellen Befehlen für die Steuerung von KEITHLEY Messgeräten

Lua Script Editor

- dient zur Visualisierung von Messdaten (Diagramme, Tabellen…)
- Integrierte graphische Benutzeroberfläche (GUI)
- Lua-Skripte mit Benutzeroberfläche versehen



Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Variablentypen (Einfache Typen)

Beispiel 1: Gleitkommazahlen

```
a = -1.234
print(a)
```

(Dezimaltrennzeichen ist der Punkt!)

Beispiel 2: Stringvariablen

```
b = "Hallo Welt!"
print(b)
```

Beispiel 3: Logische Variablen

```
c = true
print(c)
```

(Nur zwei Zustände: true / false)

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Variablentypen (Spezielle Typen)

Beispiel 4: Not In List (Bedeutung: Diesen Wert Gibt es nicht!)

```
d = 1.23          (Zuweisen)
d = nil           (Wert wieder löschen)
print(d)
```

Beispiel 5: Tabellen (Feldvariablen)

```
e = {1,2,"hallo"}                (Gemischte Typen)
print(e)                         (wenig hilfreich!)
print(e[1], e[2], e[3])          (viel besser!)
```

Beispiel 6: Funktionen

```
f = print
f("Hallo Welt")
```

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Variablentypen (Tabellen)

Der Tabelleninhalt ist entweder nummeriert:

```
tab1 = {1,2,"hallo"}
print(tab1[1], tab1[2], tab1[3])
```

Oder benannt:

```
tab2 = { x=0, y=0, z=9.81}
print(tab2.x, tab2.y, tab2.z)
```

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Was sind Blöcke?

- Ein Block ist definierter Abschnitt im Quellcode:
  - Eine Funktion
  - Eine Schleife
  - Der Inhalt einer if-then-Anweisung

- Blöcke werden in der Regel mit dem Schlüsselwort „end“ beendet

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Was sind Blöcke?

- Ein Block enthält eine oder mehrere Lua Anweisung.
- Das Lua Skript selbst kann auch als Block gesehen werden
- Schlüsselworte wie end, else, elseif weisen auf ein Blockende hin
- Blöcke können verschachtelt sein.

Beispiel 1: Eine Funktion

```
function test()
   …
end
```

Beispiel 2: Eine Schleife

```
for i=1,5 do
   …
end
```

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Lokale und Globale Variablen

- Lokale Variablen werden mit dem Schlüsselwort „local“ gekennzeichnet.
- Sie sind nur in dem Block gültig, in dem sie definiert wurden

- Beispiel:

```
a = nil; b=nil   (evtl. ist a oder b noch von vorherigen Beispielen definiert)

function testVar()
  a = "Hallo Welt"
  local b = "Lokale Variable"
end

testVar()          (Wir müssen die Funktion aufrufen, sonst passiert nichts)
print(a,b)
```

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Kommentare

- Kommentare werden bei der interpretation des Skriptes ignoriert.
- Kommentare sollen helfen sich im Skript besser zurecht zu finden.
- Einzeiliger Kommentar:

```
-- Das ist ein einzeiliger Kommentar
```

- Mehrzeiliger kommentar:

```
--[[ Das ist ein mehrzeiliger Kommentar
 Diese Zeile gehört noch zum Kommentar
]]
```

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Schlüsselwörter

Schlüsselwörter sind reservierte Begriffe, die nicht in Variablen oder Funktionsnamen verwendet werden dürfen:

```
and       break     do        else
elseif    end       false     for
function  if        in        local
nil       not       or        repeat
return    then      true      until
while
```

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Mathematikoperatoren

Lua kennt folgende binäre Operatoren:

+	Addition	c = a + b
-	Subtraktion	c = a - b
*	Multiplikation	c = a * b
/	Division	c = a / b
^	Potenzieren	c = a ^ b

Lua kennt folgende unäre Operatoren:

-	Negation	c = -a

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Vergleichsoperatoren

Vergleichsoperatoren überprüfen, ob eine Bedingung wahr ist oder nicht. Lua kennt folgende binäre Vergleichsoperatoren:

Operator	Bedeutung	Beispiel	Ergebnis
==	Ist gleich	"Hallo" == "Hallo"	true
~=	Ungleich	"Hallo" ~= "Welt"	true
<	Kleiner als …	5 < 3	false
>	Größer als…	5 > 3	true
<=	Kleiner oder gleich	4 <= 5	true
>=	Größer oder gleich	4 >= 5	false

Das Ergebnis von Vergleichen ist entweder true oder false.

Lua Sprachkonstrukte

AutomatisierungsTechnik Voigt GmbH

Logische Operatoren

Logische Operatoren können nur auf Booleanwerte angewendet werden. Lua kennt folgende binären Logikoperatoren:

Operator	Bedeutung	Beispiel
`and`	Logisches und	`(a>b) and (a<c)`
`or`	Logisches oder	`(a>b) or (a>c)`

Für die Umkehrung einer Bedingung gibt es den Negationsoperator:

Operator	Bedeutung	Beispiel
`not`	Logische Negation	`not (a<b)` ist identisch mit `a>=b`

Schleifen

AutomatisierungsTechnik Voigt GmbH

For Schleifen

- Schleifen führen einen Codeblock mehrfach aus.
- Die Ausführung endet, wenn ein Zähler einen Wert ereicht oder eine Bedingung erfüllt ist bzw. nicht mehr erfüllt ist.
- Lua kennt drei Arten von Schleifen:
  - For-Schleifen
  - While-Schleifen
  - Repeat-Schleifen

Schleifen

AutomatisierungsTechnik Voigt GmbH

For Schleifen

- For Schleifen zählen ausgehend von einem Startwert bis zu einem Endwert in einer vorgegebenen Schrittweite. Der Wert wird in der Schleifenvariable gespeichert (hier: i)
- Der Innerhalb der “for”-Anweisung stehende Codeblock wird ausgeführt bis der Endwert erreicht wird.
- Die Schrittweite ist optional, fehlt sie wird eins angenommen.
- Ist die Schrittweite negativ, zählt die Schleife rückwärts.

Beispiel:

```
-- for Schleifen
startwert=1; endwert=10; schrittweite=1
for i=startwert,endwert,schrittweite do
  print(“Ich kann zählen “..i)
end
```

Schleifen

AutomatisierungsTechnik Voigt GmbH

While - Schleifen

- While Schleifen führen einen Block so lange aus, wie eine Bedingung erfüllt ist.
- Die Überprüfung der Schleifenbedingung erfolgt am Schleifenanfang

Beispiel:

```
i = 0
while (i<10) do
  print("Ich kann zählen: " ..i)
  i = i + 1
end
```

AutomatisierungsTechnik Voigt GmbH

Repeat - Schleifen

- Repeat Schleifen führen einen Block so lange aus, bis eine Bedingung erfüllt ist.
- Unterschied zur While-Schleife: Die Überprüfung der Schleifenbedingung erfolgt am Schleifenende

Beispiel:

```
i = 0
repeat
  print("Ich kann zählen: " ..i)
  i = i + 1
until i>=10
```

Verzweigungen

AutomatisierungsTechnik Voigt GmbH

- Verzweigung führen einen Codeblock in Abhängigkeit einer Bedingung aus:

```
if Bedingung then
  -- Codeblock
end
```

- Verzweigungen können Alternativen enthalten:

```
if Bedingung then
  -- Codeblock
else
  -- Alternativer Codeblock
end
```

Verzweigungen

- Es können auch mehrere Bedingungen aneinander gereiht werden:

```
if (eindruck>1000) then
  print("Sehr beeindruckend!")
elseif (eindruck>100) then
  print("Beeindruckend!")
elseif (eindruck>10) then
  print("Mittelprächtig")
else
  print("wenig beeindruckend")
end
```

Funktionen

- Funktionen sind Codeblöcke, die einen Namen und eine Parameterliste haben. Sie beginnen mit dem Schlüsselwort `function` und hören mit dem Schlüsselwort `end` auf:

```
function MachWas(parameter1, parameter2)
  …
end
```

- Funktionsrückgabewerte werden mit dem `return` Schlüsselwort gekennzeichnet. Sollen mehrere Werte zurück gegeben werden, so werden diese durch Komma getrennt:

```
function MachEinWenigMehr(a, b)
  return a, b
end
```

- Funktionen werden nur ausgeführt, wenn sie aufgerufen werden:

```
a, b = MachEinWenigMehr(10,20)
```

Funktionen

AutomatisierungsTechnik Voigt GmbH

Funktionen können sich auch selbst rekursiv aufrufen:

```
function factorial(n)
  if n == 0 then
    return 1
  else
    return n * factorial(n - 1)
  end
end
```

Online Resourcen

AutomatisierungsTechnik Voigt GmbH

Mehr Details auf www.lua.org

http://www.lua.org/manual/

Kontakt

AutomatisierungsTechnik Voigt GmbH

Haben Sie Fragen?

Kontaktieren Sie uns:

Automatisierungstechnik Voigt GmbH
Löbtauer Straße 67
01159 Dresden

Tel.: + 49 351 213 86 40
Fax: + 49 351 213 86 50
E-Mail: atv@atvoigt.de