PROC LUA and why you should know it

Introduction

"Programming in SAS® has just been made easier" ... "Lua offers you a fresh way to write SAS programs" ...
... as Paul Tomas, Proc Lua developer at SAS Institute Inc., writes in his paper Driving SAS® with Lua (https://support.sas.com/resources/papers/proceedings15/SAS1561-2015.pdf) (SAS Global Forum, 2015)

"The Lua language seems likely to play an increasing role in the SAS world"
... as Amadeus Software, SAS Consultancy, UK, write here (https://amadeus.co.uk/sas-tips/using-lua-instead-of-sas-macro-language/).

What is Lua?

- Lua is a "powerful, efficient, lightweight, embeddable scripting language" (as Lua describes itself)
- Lua was developed by Roberto Ierusalimschy (et al) at the Pontifical Catholic University of Rio de Janeiro in Brazil
- Lua is Portuguese for "moon"

Why Lua?

Paraphrased from A Comparison of the LUA Procedure and the SAS Macro Facility (Vijayaraghavan, 2017) (https://support.sas.com/resources/papers/proceedings17/SAS0212-2017.pdf):

- Need for an alternative to Macro language was felt by various solutions groups at SAS due to the inherent limitations of the latter
- Not feasible to enhance Macro language to the level of a modern scripting language like Lua
- Purpose of Lua is to script C-based software and SAS is written in C, so the two are a good fit for each other
- PROC LUA provides an implementation of Lua 5.2 within Base SAS® (from 9.4)

Benefits:

- Low "entry requirements" for you as a SAS programmer:
  - User-friendly syntax, gentle learning curve
  - Direct access to the vast majority of SAS functions
  - PROC LUA is a brand new way to generate SAS code and it provides a realistic and powerful alternative to the SAS Macro language and `call execute()`
- Reputation for performance and memory efficiency
- Superior debugging information
  - Points to specific line
- Support for OOP
- No more macro quoting!
- Many open source available Lua libraries available
- Scoping for functions
- Functions can return multiple values
  - How do you return a *single* value from a macro which contains PROC or data step code?
- Persistence across multiple calls to PROC LUA
  - Compare with hash object in a data step
- Access to vast majority of SAS functions
- Built-in functions for handling tables and SAS datasets
- Support for highly flexible data structures (via tables)
  - SAS is not good at handling dynamic data structures like JSON
  - PROC JSON can write a JSON file but it cannot read one!

Lua basics

First things first

```
-- Line comments in Lua start with two dashes
--[[
   And this is a
   block comment
]]
```

```
PROC LUA <restart|terminate>;
    SUBMIT;
       -- Lua code goes inside a SUBMIT block...
    ENDSUBMIT;
RUN;
```

- Or an external file (this is the only way inside a SAS macro):

  ```
  FILENAME LuaPath '/my/path/luafiles';
  PROC LUA INFILE='myluafile';
  RUN;
  ```

- Lua is case sensitive
- Semicolons are optional and usually omitted

Data types

- Lua is dynamically typed
- Types are: number, string, boolean, table, nil, function, userdata and thread
- `nil` represents absence of a value, and is different from a SAS missing value
- Boolean values are `true` and `false`
  - **Only** `nil` and `false` evaluate to `false`
  - **Everything** else evaluates to `true` (including zero and SAS missing values!)

Naming rules

Same as SAS, except: can be as long as you like (subject to GPP)

Declaring variables

Variables have global scope within the current Lua state unless explicitly declared as local

```
local pi
pi = 3.1415926

local pi = 3.1415926
local v1, v2 = 'Hello', 'World'   -- Note list-style declaration and value assignment
```

Writing to the log

```
print(v1..', '..v2..'!')          -- Writes "Hello, World!"
```

Compare macro and Lua:

In [ ]:

```
%macro get_max(list=);
    %sysfunc(max(%unquote(&list)));
%mend get_max;

%put >>> %get_max(list=%str(1,4,2,5,7,7.2,2));
```

In [ ]:

```
PROC LUA;
    SUBMIT;
        function get_max(list)
            table.sort(list)
            return list[#list]
        end

        print('>>> '..get_max({1,4,2,5,7,7.2,2}))
    ENDSUBMIT;
RUN;
```

Simple for-loop:

In [ ]:

```
PROC LUA;
    SUBMIT;
        for i = 1, 10 do
            print(i..' squared is '..i^2)
        end
        print(i)
    ENDSUBMIT;
RUN;
```

Simple array and ipairs() function:

In [ ]:

```
PROC LUA;
    SUBMIT;
        local colours = {'red', 'blue', 'green', 'yellow', 123}
        for i, colour in ipairs(colours) do
            print(i, colour)
        end

        for i = 1, #colours do
            print(i, colours[i])
        end
    ENDSUBMIT;
RUN;
```

Hash table / dictionary and pairs() function:

In [ ]:

```
PROC LUA;
    SUBMIT;
        local sp_domains = {CO='Comments',
                            DM='Demographics',
                            SE='Subject Elements',
                            SV='Subject Visits'}
        for code, decode in pairs(sp_domains) do
            print(code, decode)
        end
    ENDSUBMIT;
RUN;
```

Submitting SAS code with substitution:

In [ ]:

```
PROC LUA;
    SUBMIT;
        sas.submit([[
            proc print data=sashelp.class;
                where age > @age@;
                var @vars@;
            run;
        ]], {age=12, vars='name age'})
    ENDSUBMIT;
RUN;
```

Iteration through a SAS dataset:

In [ ]:

```
PROC LUA;
    SUBMIT;
        sas.submit[[
            data class1 class2;
                set sashelp.class;
                if age > 13 then output class1;
                else output class2;
            run;
        ]]

        local dsid = sas.open('sashelp.vtable (where=(libname = "WORK"))')
        for obs in sas.rows(dsid) do
            local ds = obs.memname
            sas.submit[[
                proc print data=@ds@;
                run;
            ]]
        end
        sas.close(dsid)
    ENDSUBMIT;
RUN;
```

In [ ]:

```
%LET INLIB = WORK;
PROC LUA;
    SUBMIT;
        local dsid = sas.open(sas.cat('sashelp.vtable (where=(libname = "', sas.
symget('inlib'),'"))'))
        while sas.next(dsid) do
            local ds = sas.get_value(dsid,'memname')
            sas.submit[[
                proc print data=@ds@;
                run;
            ]]
        end
        sas.close(dsid)
    ENDSUBMIT;
RUN;
```

Table to SAS dataset:

In [ ]:

```
PROC LUA;
    SUBMIT;
        local tbl = {}
        for i = 1, 10 do
            local vars = {}
            vars.n = i
            vars.n2 = i ^ 2
            tbl[i] = vars
        end

        print(table.tostring(tbl))
        sas.write_ds(tbl, 'squares')

        sas.submit[[
            proc print data=squares noobs;
            run;
        ]]
    ENDSUBMIT;
RUN;
```

New SAS dataset from scratch:

In [ ]:

```
PROC LUA;
    SUBMIT;
        sas.new_table('squares', {
                {name='n',  type='n', length=8, label='N'},
                {name='n2', type='n', length=8, label='N squared'},
        })
        local dsid = sas.open('squares', 'u')
        for n = 1, 10 do
            sas.append(dsid)
            sas.put_value(dsid, 'n', n)
            sas.put_value(dsid, 'n2', n ^ 2)
            sas.update(dsid)
        end;
        sas.close(dsid)
        sas.submit [[
            proc print data=squares noobs label;
            run;
        ]]
    ENDSUBMIT;
RUN;
```

Function which returns multiple values:

In [ ]:

```
PROC LUA;
    SUBMIT;
        local function dateparts(sasdate)
            return sas.put(sas.day(sasdate),'z2'), sas.put(sas.month(sasdate),'z
2'), sas.year(sasdate)
        end
        local d, m, y = dateparts(sas.today())
        print("Today's date is: "..y..m..d)
    ENDSUBMIT;
RUN;
```

Function which submits SAS code:

In [ ]:

```
PROC LUA;
    SUBMIT;
        local function mycompare(ds1, ds2)
            sas.submit [[
                proc compare base=class1 comp=class2 noprint;
                run;
            ]]
            return sas.symget('sysinfo')
        end

        print('Comparison result is: '..mycompare('class1', 'class2'))
    ENDSUBMIT;
RUN;
```

A module:

In [ ]:

```
PROC LUA;
    SUBMIT;
        myfuncs = {}

        myfuncs.dateparts = function(sasdate)
            return sas.put(sas.day(sasdate),'z2'), sas.put(sas.month(sasdate),'z
2'), sas.year(sasdate)
        end

        myfuncs.mycompare = function(ds1, ds2)
            sas.submit [[
                proc compare base=@ds1@ comp=@ds2@ noprint;
                run;
            ]]
            return sas.symget('sysinfo')
        end
    ENDSUBMIT;
RUN;

PROC LUA;
    SUBMIT;
        local d, m, y = myfuncs.dateparts(sas.today())
        print(y..m..d)

        sas.submit[[
            data class1 class2;
                set sashelp.class;
                if age > 13 then output class1;
                else output class2;
            run;
        ]]

        print('Comparison result is: '..myfuncs.mycompare('class1', 'class2'))
    ENDSUBMIT;
RUN;
```

Macro interface to Lua:

```
filename LuaPath "/my/path/to/lua/files";

%MACRO _set_graphoption(
    dsname    = _gral_graphopts
  , plot       = .
  , cell       = .
  , object     =
  , type       =
  , attribute  =
  , value      =
  , noset      =
  , module     =
) ;

    proc lua infile='_set_graphoption';
    run;

%MEND _set_graphoption;
```

Acknowledgments

Thanks go to my esteemed colleague Igor Khorlo who takes the blame for getting me interested in Lua in the first place, and who is a constant and ever willing source of good advice and valuable feedback.

Syneos™ Health | Shortening the distance from lab to life.™