



One giant PROC for mankind: Using PROC LUA for code generating code and automating TFLs

What is PROC LUA?

PROC LUA is a procedure added to SAS in mid-2015 designed to allow users to run statements from the Lua programming language in SAS as an alternative to SAS macro code.



The basics of PROC LUA

```
proc lua;  
submit;  
    print("Hello") --Comments are written using double hyphens  
    print("World"); --Lines can end with a semicolon but it is not  
required  
    --[  
    Block Comments are written using 2 hyphens and a square bracket  
    ]--  
endsubmit;  
run;  
  
filename LUAPATH "<directory>";  
proc lua infile="<filename>";  
/*This searches for <filename> in <directory> and runs it*/  
run;
```

Tables in Lua

The table type implements associative arrays. An associative array is an array that can be indexed not only with numbers, but also with strings or any other value of the language, except nil.

- From Programming in Lua, Chapter 2.5

```
y = {1,1}  
y[4] = 2  
y['Hello'] = 'World'
```

$y \rightarrow \{[1] = 1, [2] = 1, [4] = 2, ['Hello'] = 'World'\}$

In the case of a SAS dataset **t** with n observations and m variables. It is represented as an table in the form

$t = \{[1] = t[1], [2] = t[2], \dots, [n] = t[n]\}$

where

$t[x] = \{['<Var 1>'] = <\text{any valid data type}>, \dots, ['<Var m>'] = <\text{any valid data type}>\}$

SAS datasets in PROC LUA

	Dataset_name	Order	Variable_name	Variable_Label	Variable_type	Variable_Length	Display_format	Codelist	Controlled_terms	Source	Derivation
1	ADSL		1 STUDYID	Study Identifier	char	13				DM.STUDYID	
2	ADSL		2 USUBJID	Unique Subject Identifier	char	35				DM.USUBJID	
3	ADSL		3 SUBJID	Subject Identifier for the Study	char	9				DM.SUBJID	
4	ADSL		4 SITEID	Study Site Identifier	char	12				DM.SITEID	
5	ADSL		5 DSUBJID	Display Subject ID	char	6				Concatenate SITEID and SUBJID	
6	ADSL		6 AGE	Age	num	8				DM.AGE	
7	ADSL		7 AGEU	Age Units	char	6				Set to "YEARS"	
8	ADSL		8 SEX	Sex	char	2				DM.SEX	
9	ADSL		9 RACE	Race	char	2				DM.RACE	
10	ADSL		10 COUNTRY	Country	char	3				DM.COUNTRY	
11	ADSL		11 TRT01P	Planned Treatment for Period 01	char	10		TRT01		Assigned from ARM using the TRT01 codelist	
12	ADSL		12 TRT01PN	Planned Treatment for Period 01 (N)	num	8		TRT01N		Assigned from ARM using the TRT01N codelist	
13	ADSL		13 TRT01A	Actual Treatment for Period 01	char	10		TRT01		Assigned from ACTARM using the TRT01 codelist	
14	ADSL		14 TRT01AN	Actual Treatment for Period 01 (N)	num	8		TRT01N		Assigned from ACTARM using the TRT01N codelist	
15	ADSL		15 TRTSDT	Date of First Exposure to Treatment	num	8	e8601da.			Create a numeric version of DM.RFSTDTC and apply the e8601da. format	
16	ADSL		16 TRTEDT	Date of Last Exposure to Treatment	num	8	e8601da.			Create a numeric version of DM.RFENDTC and apply the e8601da. format	
17	ADSL		17 SAFFL	Safety Population Flag	char	1				Populated as "Y" where ARM is either "PLACEBO" or "DUMMY DRUG" (as dictated in SAP section 3.1)	
18	ADSL		18 MSAFFL	Modified Safety Population Flag	char	1				Populated as "Y" as directed by SAP Section 3.1.	

When we use `sas.read_ds("work.specs")` to read our SAS dataset as a Lua table it'll create entries for each observation containing the entries for each variable. It'll also create entries for each variable containing metadata for each variable.

SAS datasets in PROC LUA

An example of a SAS dataset stored in Lua

```
{  
  [1] = { ['Dataset_name'] = 'ADSL', ['Order'] = 1, ... }  
  [2] = ...  
  [...] = ...  
  ['Dataset_name'] = { ['type']='c', ... }  
  [...] = ...  
}
```

Submitting SAS code in Lua

In PROC LUA you can submit sas code in two ways.

```
proc lua;  
submit;  
x=1  
sas.submit([[  
  data @name@;  
  x = @x@;  
  run;  
]], {name="ex"})  
endsubmit;  
run;
```

data ex;
x = 1;
run;

```
proc lua;  
submit;  
x=1  
sas.submit_  
  ([[data @name@;]], {name="ex"})  
sas.submit_([[x = @x@]])  
sas.submit_([[;]])  
sas.submit([[run;]])  
endsubmit;  
run;
```

'ex' dataset	Var x
Obs 1	1

Submitting SAS code in Lua

```
dsin='adsl'  
vars={  
  usubjid='order order=internal'  
  age='display'  
  country='display'  
}
```

```
sas.submit_([[  
proc report data=$(dsin);  
column]])
```

```
for k,v in pairs(vars) do  
sas.submit_([[ k]])  
End
```

```
sas.submit_([[;]])
```

```
for k,v in pairs(vars) do  
sas.submit_([[ define $(k) / $(v);]])  
end  
sas.submit([[run;]])
```

Submitting SAS code in Lua

```
proc report data=ads1;  
column  
  usubjid  
  age  
  country  
;  
  
  define usubjid / order order=internal;  
  define age / display;  
  define country / display;  
run;
```

Generating SAS code in Lua

- Lua is a language designed to be simple and provide the mechanisms to allow the user to create whatever is required
 - Lua can create its own macro processor to generate code
 - Lua can write to a file
- Whilst SAS can achieve both of these features, Lua can do it with more complexity

Lua's 'Macro' processor

Using a *Simple Lua Preprocessor*^[1] we can generate SAS code using Lua code. It can then be written to a file to be ran like any other SAS programs (and crucially, edited easily)

```
function prep(file)
  local chunk = {n=0}
  for line in file:lines() do
    if string.find(line, "^#") then
      table.insert(chunk, string.sub(line, 2) .. "\n")
    else
      local last = 1
      for test, expr, index in string.gmatch(line, "(.-)$(%b())()" do
        last = index
        if test ~= "" then
          table.insert(chunk, string.format('io.write %q', test))
        end
        table.insert(chunk, string.format('io.write %s', expr))
      end
      table.insert(chunk, string.format('io.write %q\n',
                                         string.sub(line, last) .. "\n"))
    end
  end
  return load(table.concat(chunk)) ()
end
```

[1] <http://lua-users.org/wiki/SimpleLuaPreprocessor>

Example Program

Lines starting with # are executed as Lua.
Other lines are sent through as is,
except that \$(...) appearing anywhere in them is executed.

```
example.txt >
proc report data=$(dsin);
column
#for k in pairs(vars) do
    $(k)
#end
;

#for k,v in pairs(vars) do
    define $(k) / $(v);
#end
run;
```

Example Program

example.lua >

<our prep function from earlier goes here>

```
dsin='adsl'
```

```
vars={
```

```
    usubjid='order order=internal'
```

```
    age='display'
```

```
    country='display'
```

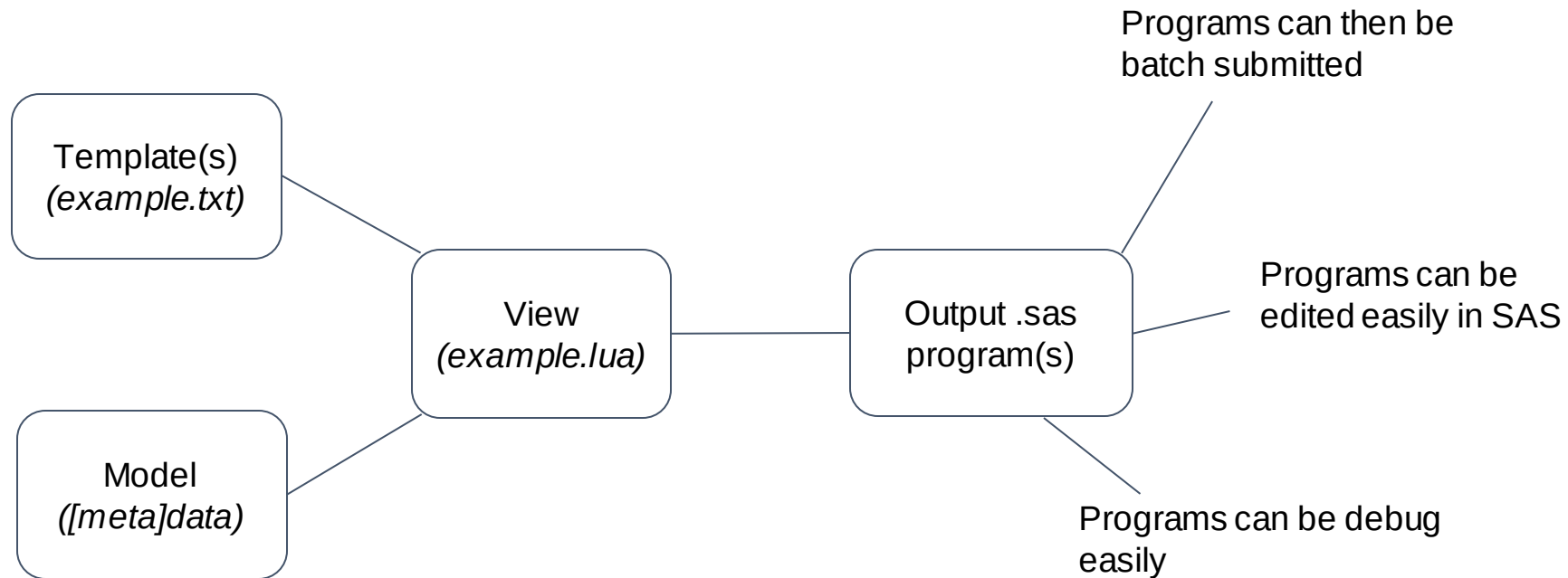
```
}
```

```
prep(assert(io.open("example.txt")))
```

Example Program

```
proc report data=adsl;  
column  
  usubjid  
  age  
  country  
;  
  
  define usubjid / order order=internal;  
  define age / display;  
  define country / display;  
run;
```

An Example Framework





Get In Touch

(T) +44 20 3696 7240

(E) info@veramed.co.uk

(A) Regal House, 70 London Rd,
Twickenham
TW1 3QS

Thank You

tom.ratford@veramed.co.uk