Structuring and Scripting SAS code with Proc Lua instead of SAS Macro

Martin Sjöberg

Handelsbanken

Structuring and Scripting SAS code with Proc Lua instead of SAS Macro

Martin Sjöberg

- We also use SAS Macro
  - Data analysis and ad-hoc tasks
- Proc Lua for larger, more complex projects
  - Model implementations or automation/scripting
- ”That can be done with macro”

Agenda

- What is Lua?
- The choice between Proc Lua and SAS Macro
- Functionality overview
- How we use it – a risk classification model example
- Lessons learned – 2+ years with Proc Lua

Martin Sjöberg

- Branch office at Handelsbanken
- Quantiative analyst – Financial Crime Prevention
  - AML/CTF Business intelligence, risk and data analysis
- In the SAS world since 2017
  - Previously: Java, Python and MATLAB

What is Lua?

- Lightweight, open source
- A ”regular” programming language
- Used in e.g. games and UI’s – often embedded
  - Angry Birds, World of Warcraft, The Sims
  - UI in Adobe Photoshop Lightroom
  - http://www.lua.org/uses.html

- Proc Lua: In SAS since 9.4M3

The choice between Proc Lua and SAS Macro

- Overview

Proc Lua

- Parameterization
- Rich data structures
- Encapsulation & Abstractions
- Modules (hierarchical structure)
- Helpful log & error messages

SAS Macro

- Parameterization
- Text based variables
- Abstractions
- Libraries, e.g. the autocall facility

- ”SCL like”

For small projects: ”Which side of the bed did I wake up on today?”

The choice between Proc Lua and SAS Macro

- Small example

Lua: Array/list

```
local names={"George", "Paul", "Ringo", "John"}
for i,v in ipairs(names) do
     print(v)
end
```

Macro: Text based variable

```
/* Macro */
%let names=George Paul Ringo John;
%let n = 1;
%let nm = %scan(&names, &n, " ");
%do %while( &nm ne %str());
     %put &name;
     %let n = %eval(&n + 1);
     %let nm = %scan(&names, &n, " ");
%end;
```

Functionality overview

- SAS built-in functions – everything* from the data step
- FCMP functions
- OOP (encapsulation)
- All sorts of data structures
  - Matrices, multidimensional arrays, queues etc.

We mainly use:

- Arrays and hash tables (a.k.a. dictionaries)
  - Combinations of the above
- Functions and modules

How we use it – general principles

- Keep it SAS-close
  - No Proc Lua for stuff SAS does great – data processing
    - The sas.submit() function
  - Minimal use of complex Lua code
  - Low entry into our code bases
- Functions and modules for everything
  - Typically a main module – high level control

How we use it – a risk classification model

- The main module

How we use it – a risk classification model

- Example – aggregation of a dynamic number of tables

```
function module.calculate_risk_factor_scores(setup, parameters, risk_factor_results)
    local assert_string = setup.util.luax.assert_string
    local return_var = {}
    return_var.table_names = {}
    local func_rfs = require('60_risk_factor_scores.calculate_rfs1101                    )
    return_var.table_names.rfs1101_          = assert_string(func_rfs("rfs_1101                    ,
                                                             setup.info,
                                                             parameters.table_names.rf1101_scores,
                                                             risk_factor_results.table_names.rfr1101))
```

```
local function calculate_rfs1101                    (table_name, setup_info, risk_factor_scores_table_name, rfr1101                    _table_name)
    local work_libname = setup_info.work_libname
    local sub_model_key = setup_info.sub_model_key_individuals
    if setup_info.skip_existing_tables and sas.exists(work_libname .. '.' .. table_name) then return table_name end
    sas.submit
    [[
        proc sql noprint;
            create table @work_libname@.@table_name@ as
            select rfr.ip_rk,
                   risks.rf_score as rfs1101
            from @work_libname@.@rfr1101                    _table_name@ rfr
            left join @work_libname@.@risk_factor_scores_table_name@ risks
            on rfr.rfr1101 = risks.rf_result
            and risks.crr_parm_appl = @sub_model_key@;
        quit;
    ]]
    return table_name
end
```

How we use it – a risk classification model

- Example – aggregation of a dynamic number of tables

```
function module.calculate_risk_factor_scores(setup, parameters, risk_factor_results)
    local assert_string = setup.util.luax.assert_string
    local return_var = {}
    return_var.table_names = {}
    local func_rfs = require('60_risk_factor_scores.calculate_rfs1101                    )
    return_var.table_names.rfs1101_          = assert_string(func_rfs("rfs_1101                    ,
                                                             setup.info,
                                                             parameters.table_names.rf1101_scores,
                                                             risk_factor_results.table_names.rfr1101))
    func_rfs = require('60_risk_factor_scores.calculate_rfs1102                    )
    return_var.table_names.rfs1102_          = assert_string(func_rfs("rfs_1102
                                                             setup.info,
                                                             parameters.table_names.rf1102_scores,
                                                             risk_factor_results.table_names.rfr1102))

    func_rfs = require('60_risk_factor_scores.calculate_rfs0401
    return_var.table_names.rfs1401           = assert_string(func_rfs("rfs_1401
                                                             setup.info.sub_model_key_individuals,
                                                             setup.info,
                                                             parameters.table_names.rf0400_scores,
                                                             risk_factor_results.table_names.rfr1401))

    func_rfs = require('60_risk_factor_scores.calculate_rfs0501
    return_var.table_names.rfs1501                   = assert_string(func_rfs("rfs_1501
                                                                setup.info.sub_model_key_individuals,
                                                                setup.info,
                                                                parameters.table_names.rf0501_scores,
                                                                risk_factor_results.table_names.rfr1501))

    return return_var
end
```

How we use it – a risk classification model

- Example – aggregation of a dynamic number of tables

```
function module.run_model(project_path, work_lib_path, out_lib_path, as_of_date)
    local setup                 = module.setup(project_path, work_lib_path, out_lib_path, as_of_date)
    local assert_table          = setup.util.luax.assert_table
    local parameters            = assert_table(module.extract_parameters(setup))
    local kure_extract          = assert_table(module.extract_kure(setup, parameters))
    local customers             = assert_table(module.extract_customers(setup, parameters, kure_extract))
    local attributes            = assert_table(module.extract_attributes(setup, kure_extract, customers, parameters))
    local risk_factor_results   = assert_table(module.calculate_risk_factor_results(setup, parameters, customers, attributes))
    local risk_factor_scores    = assert_table(module.calculate_risk_factor_scores(setup, parameters, risk_factor_results))
    local category_scores       = assert_table(module.calculate_category_scores(setup, parameters, risk_factor_scores))
```

How we use it – a risk classification model

- Example – aggregation of a dynamic number of tables

```
local function consolidate_rfs(table_name, setup_info, rfs_table_names)
    local work_libname = setup_info.work_libname
    if setup_info.skip_existing_tables and sas.exists(work_libname .. '.' .. table_name
    sas.submit_[[data @work_libname@.@table_name@;
                  length ip_rk 8 rf_id $32 rfs 8;
                  set ]]
    for key,value in pairs(rfs_table_names) do
        local rfs_name = key:sub(1,7)
        sas.submit_[[@work_libname@.@value@(in=@rfs_name@ rename=(@rfs_name@=rfs))
            ]]
    end
    sas.submit_(';')
    for key,value in pairs(rfs_table_names) do
        local rfs_name = key:sub(1,7)
        local rf_id = rfs_name:sub(4)
        sas.submit_[[
          if @rfs_name@ then
            rf_id = '@rf_id@';
        ]]
    end
    sas.submit('run;')
    return table_name
end
```

```
if rfs2104 then
  rf_id = '2104';

if rfs2107 then
  rf_id = '2107';

if rfs2108 then
  rf_id = '2108';

if rfs1101 then
  rf_id = '1101';

if rfs2501 then
  rf_id = '2501';

if rfs1501 then
  rf_id = '1501';

if rfs2602 then
  rf_id = '2602';
```

How we use it – a risk classification model

- Example – aggregation of a dynamic number of tables

```
function module.calculate_risk_factor_scores(setup, parameters, risk_factor_results)
    local assert_string = setup.util.luax.assert_string
    local return_var = {}
    return_var.table_names = {}
    local func_rfs = require('60_risk_factor_scores.calculate_rfs1101
    return_var.table_names.rfs1101 = assert_string(func_rfs("rfs_1101
                                                            setup.info,
                                                            parameters.table_names.rf
                                                            risk_factor_results.table

    func_rfs = require('60_risk_factor_scores.calculate_rfs1102
    return_var.table_names.rfs1102 = assert_string(func_rfs("rfs_1102
                                                            setup.info,
                                                            parameters.table_names.rf
                                                            risk_factor_results.table

    func_rfs = require('60_risk_factor_scores.calculate_rfs0401
    return_var.table_names.rfs1401 = assert_string(func_rfs("rfs_1401
                                                            setup.info.sub_model_key_
                                                            setup.info,
                                                            parameters.table_names.rf
                                                            risk_factor_results.table

    func_rfs = require('60_risk_factor_scores.calculate_rfs0501
    return_var.table_names.rfs1501 = assert_string(func_rfs("rfs_1501
                                                            setup.info.sub_model_k
                                                            setup.info,
                                                            parameters.table_names
                                                            risk_factor_results.ta

    return return_var
end
```

```
if rfs2104 then
  rf_id = '2104';

if rfs2107 then
  rf_id = '2107';

if rfs2108 then
  rf_id = '2108';

if rfs1101 then
  rf_id = '1101';

if rfs2501 then
  rf_id = '2501';

if rfs1501 then
  rf_id = '1501';

if rfs2602 then
  rf_id = '2602';
```

How we use it – a risk classification model

➢ Example – aggregation of a dynamic number of tables

- What does the SAS log look like?
  - **Exactly** the executed code – with resolved variables
  - Compare to e.g. MPRINT

```
local function consolidate_rfs(table_name, setup_info, rfs_table_names)
    local work_libname = setup_info.work_libname
    if setup_info.skip_existing_tables and sas.exists(work_libname .. '.' .. table_name) then return table_name end
    sas.submit_[[data @work_libname@.@table_name@;
                 length ip_rk 8 rf_id $32 rfs 8;
                 set ]]
    for key,value in pairs(rfs_table_names) do
        local rfs_name = key:sub(1,7)
        sas.submit_[[@work_libname@.@value@(in=@rfs_name@ rename=(@rfs_name@=rfs))
            ]]
    end
    sas.submit_(';')
    for key,value in pairs(rfs_table_names) do
        local rfs_name = key:sub(1,7)
        local rf_id = rfs_name:sub(4)
        sas.submit_[[
          if @rfs_name@ then
            rf_id = '@rf_id@';
        ]]
    end
    sas.submit('run;')
    return table_name
end
```

```
data work.rfs_consolidated;
                    length ip_rk 8 rf_id $32 rfs 8;
                    set
work.rfs_2104 (in=rfs2104 rename=(rfs2104=rfs))
work.rfs_2107 (in=rfs2107 rename=(rfs2107=rfs))
work.rfs_2108 (in=rfs2108 rename=(rfs2108=rfs))
work.rfs_1101 (in=rfs1101 rename=(rfs1101=rfs))
work.rfs_2501 (in=rfs2501 rename=(rfs2501=rfs))
work.rfs_1501 (in=rfs1501 rename=(rfs1501=rfs))
work.rfs_2602 (in=rfs2602 rename=(rfs2602=rfs))
work.rfs_1201 (in=rfs1201 rename=(rfs1201=rfs))
```

How we use it – a risk classification model

➢ Example – aggregation of a dynamic number of tables

- What does the SAS log look like?
  - **Exactly** the executed code – with resolved variables
  - Compare to e.g. MPRINT

```
local function consolidate_rfs(table_name, setup_info, rfs_table_names)
    local work_libname = setup_info.work_libname
    if setup_info.skip_existing_tables and sas.exists(work_libname .. '.' .. table_name) then return table_name end
    sas.submit_[[data @work_libname@.@table_name@;
                 length ip_rk 8 rf_id $32 rfs 8;
                 set ]]
    for key,value in pairs(rfs_table_names) do
        local rfs_name = key:sub(1,7)
        sas.submit_[[@work_libname@.@value@(in=@rfs_name@ rename=(@rfs_name@=rfs))
            ]]
    end
    sas.submit_(';')
    for key,value in pairs(rfs_table_names) do
        local rfs_name = key:sub(1,7)
        local rf_id = rfs_name:sub(4)
        sas.submit_[[
          if @rfs_name@ then
            rf_id = '@rf_id@';
        ]]
    end
    sas.submit('run;')
    return table_name
end
```

```
if rfs2104 then
  rf_id = '2104';

if rfs2107 then
  rf_id = '2107';

if rfs2108 then
  rf_id = '2108';

if rfs1101 then
  rf_id = '1101';

if rfs2501 then
  rf_id = '2501';

if rfs1501 then
  rf_id = '1501';

if rfs2602 then
  rf_id = '2602';
```

Bonus: Function parameters

- An explosion of Macro parameters

```
%macro CompareWorkDSContents(       RefRunID, RefCustIDVar, RefDSName, RefRunInfoPath, RefRunProcessID, RefIDLookupVar, RefIDL
                             ,      TestRunID, TestCustIDVar, TestDSName, TestRunInfoPath, TestRunProcessID, TestIDLookupVar,
                             ,      OutputTitle, IgnoreCompVars=, PrintDetailedResults=1, OutDS=, NrDiffsAbsTolerance=0, NrDif
                             );
```

```
function module.run_model(project_path, work_lib_path, out_lib_path, as_of_date)
    local setup                 = module.setup(project_path, work_lib_path, out_lib_path, as_of_date)
    local assert_table          = setup.util.luax.assert_table
    local parameters            = assert_table(module.extract_parameters(setup))
    local kure_extract          = assert_table(module.extract_kure(setup, parameters))
    local customers             = assert_table(module.extract_customers(setup, parameters, kure_extract))
    local attributes            = assert_table(module.extract_attributes(setup, kure_extract, customers, parameters))
    local risk_factor_results   = assert_table(module.calculate_risk_factor_results(setup, parameters, customers, attributes))
    local risk_factor_scores    = assert_table(module.calculate_risk_factor_scores(setup, parameters, risk_factor_results))
    local category_scores       = assert_table(module.calculate_category_scores(setup, parameters, risk_factor_scores))
```

Lessons learned – 2+ years of Proc Lua

- Resources:
  - Driving SAS® with Lua* – 90 % of what we use
  - Programming in Lua** – PIL
  - SAS docs*** – Way improved since I started

* https://support.sas.com/resources/papers/proceedings15/SAS1561-2015.pdf

** https://www.lua.org/pil/contents.html

*** https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=proc&docsetTarget=n1csk38ocks0rgn1rr8d302ofqgs.htm&locale=en

Lessons learned – 2+ years of Proc Lua

- Lua centered or SAS centered code?
  - E.g. our risk classification model, 3 versions:
    - Complex Lua – dynamically generated (from Excel files)
    - Only SAS Macro
    - ”SAS-close” Lua – The one we’ve looked at
  - Current preference: Keep it SAS-close
- Project sizes:
  - Small projects: Proc Lua statements in open code (.SAS files)
  - Large projects: Modules in .Lua files
    - Language specificity (syntax highlight, autocompletion etc.)

Thank you

Martin Sjöberg
masj27@handelsbanken.se
https://www.linkedin.com/in/martin-sjoeberg/