JudoShiai Lua

JudoShiai Lua is intended to be used with the SVG functionality. Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description. Learn Lua here: https://www.lua.org/

Introduction

There are many ways to express the same thing, like competitor’s points. For example, if a competitor has lost you may not want to display “0” but text “LOSS”. Or if there is hikiwake you may want to show text “DRAW”. JudoShiai is quite flexible, but there are limits to what can be done. JudoShiai Lua introduces some extra help.

You may want to have reports not provided by the JudoShiai at the moment. You can create Lua scripts to create SVG pages with the required data.

Example: You want to show the text “LOSS” if the competitor has lost. Now you can do that by writing to SVG file text

```
%m23p1-1==0'LOSS'
```

Here

%m	Match related text
23	Match number
p	Points
1	1 = white, 2 = blue competitor
-1==0’LOSS’	Print LOSS if points are 0

What if that is not enough? How to print DRAW in case of hikiwake and still print LOSS? Perhaps you want to have text IPPON flashing red? That is possible by using Lua. The SVG text above is replaced by %, a Lua expression, and match identification:

```
%[require 'points']m23
```

Lua code in the example is

```
require 'points'
```

m23 sets global Lua variables to values related to match #23.

You have a file points.lua that can write anything on the SVG sheet.

JudoShiai Lua

There are many ways to use Lua scripts.

Use code in SVG files. Example:

```
Hello %[JS.writesvg('World!')]
```

SVG file will have text Hello World! It is unpractical to write long pieces of code to the SVG sheet. It is better to have Lua code in a separate file. Example:

```
Hello %[require 'myhello']
```

File myhello.lua has a line

```
JS.writesvg('World!')
```

Again you will display text Hello World! File must be in one of the directories

HOMEDIR/lualib
    /home/myname/lualib   (Linux)
    C:\Users\myname\lualib (Windows)

HOMEDIR/lualib/lib
  (for example /home/myname/lualib/lib or C:\Users\myname\lualib\lib)

JudoShiai/svg-lua
  (for example /usr/lib/judoshiai/svg-lua or C:\Program Files\JudoShiai\svg-lua)

JudoShiai/svg-lua/lib
  (for example /usr/lib/judoshiai/svg-lua/lib or
   C:\Program Files\JudoShiai\svg-lua/lib)

You can have Lua file anywhere, but then you must give the whole path:

```
Hello %[@/PATH/TO/myhello.lua]
```

Note: @-character indicates that rest of the text is a path to a file. Also on Windows you must use forward slashes.

Global variables defined by JudoShiai

There are some predefined read-only variables that you can use after their values are set. To set match related variables call your Lua function from an SVG page like this:

```
%[require 'handle_match']m23
```

m23 sets the following global predefined read-only variables for the match #23 before running Lua file handle_match:

JS_category	Index of the category. This has not much use except in SQL command.
JS_number	Number of the match.
JS_comp_1	Internal index of the white competitor.
JS_comp_2	Blue competitor. Indexes have not much use except in SQL commands.
JS_score_1	White competitor’s scores (ippons, waza-aris, shidos)
JS_score_2	Scores for the blue competitor.
JS_points_1	Value of the win for the white (ippon = 10 points).
JS_points_2	Blue competitor’s winning points.
JS_match_time	Match time in seconds.
JS_comment	Match is forced to be next or preparing or delayed.
JS_tatami	Default contest area for the match.
JS_group	Group number category belongs to.
JS_flags	Misc. flags.
JS_forcedtatami	Contest area where fight has been moved manually.
JS_forcednumber	Number of the fight, if moved manually.
JS_date	Date information of the match as seconds since the Epoch.
JS_legend	Legend for the match.
JS_roundnum	Round number of the match (first round, repechage, etc.).

To set competitor related variables call your Lua function like this:

```
%[require 'print_name']m23-1
```

This will first set white competitor’s name and other data to global variables and then run your Lua file. The following competitor’s variables are set:

JS_first	First name.
JS_last	Last name.
JS_club	Club.
JS_country	Country.
JS_comment	Comment text.
JS_compid	Competitor’s ID.
JS_coachid	Competitor’s coach ID.
JS_compix	Internal index. Used with SQL commands.
JS_birthyear	Year of birth.
JS_grade	Grade as an integer. Default: 0 = unknown, 1 = 6. kyu, 2

	= 5. kyu, … 7 = 1. dan, 8 = 2. dan, …
JS_regcategory	Registered category.
JS_category	Real category.
JS_weight	Weight in grams.
JS_seeding	Seeding 1-8.
JS_clubseeding	Clubseeding.
JS_gender	Gender. Male = 1, female = 2.

For the blue competitor the Lua call would be:

```
%[require 'print_name']m23-2
```

Lua code is the one surrounded by brackets. You can have any name and arguments.

Other global read-only variables:

InstallationDir	Installation directory of the JudoShiai software.
LuaScriptDir	Directory where the script is run.

Functions defined by JudoShiai

There are useful functions that start with JS.:

JS.sqlrows(sqlcmd)
Run SQL command.
Parameters
sqlrow String. Sql command to perform.
Returns
Table of tables.

JS.writesvg(str)
Writes SVG code on canvas.
Parameters
str   String. SVG code to write to the document.
Returns
Nothing.

JS.getdata(index)
Get data of competitor or category using index.
Parameters
index   Integer. Index of the competitor or category.
Returns
Table of data.

JS.getmatchbytatami(tatami, number)
Get data about match.
Parameters
tatami   Integer. Number of the tatami (1...)
number   Integer. Number of the match (1...)
Returns
Table of data.

JS.getroundname(round)
Get round name of a match.
Parameters
round   Integer. Round code from database or getmatchbytatami().
Returns
String that describes the round.

JS.setpages(numpages)
Set total number of pages.
Parameters
numpages   Integer. Number of pages in the SVG document.
Returns
Nothing.

JS.getpage()
Get the current page number. First page is 1.
Parameters
None
Returns
Current page number (integer) starting from 1.

JS.flag2base64(country)
Writes flag of a country as base64 string to SVG document. Data is from JudoShiai/etc/flags-ioc.
Parameters
country   String. IOC abbreviation of the country.
Returns
Nothing.

JS.error(message)
Opens a window and shows error text.
Parameters
message   String. Error text.
Returns
Nothing.

JS.print(text)
Prints texts to an open window.
Parameters
text   String. Text to print.
Returns
Nothing.

JS.getcatdefs()
Get defined category names.
Parameters
None.
Returns
Table of categories for men and women.
Example:

```
m, w = JS.getcatdefs()
returns table m for men:
  {
   [1] = "MenU18",
   [2] = "MenU21",
   [3] = "M",
   numentries = 3,
  }
and w for women:
  {
   [1] = "WomenU18",
   [2] = "WomenU21",
   [3] = "W",
   numentries = 3,
  }
```
Tables have entries 'numentries' that tell the lengths of the tables.

JS.getsvgbyid(filename, id)

Returns group whose id is id.

Parameters

filename	String. Path to the SVG file
id	String. Id of the <g id="xxx">...</g>

Returns

String that contains the group data.

Example

```
svg = JS.getsvgbyid(InstallationDir .. '/svg-lua/lib/mypage.svg',
                    'header')
```

JS.getsvggeometry(filename, id)

Returns position and size of a svg element.

Parameters

filename	String. Path to the SVG file
id	String. Id of the <g id="xxx">...</g>

Returns

X and y coordinates of the svg element.
Width and height of the svg element.

Example

```
local x, y, w, h = JS.getsvggeometry(InstallationDir ..
                                     '/svg-lua/lib/mypage.svg', 'header')
```

JS.askoption(title, options)

Asks user to select one of the options. NOTE: Opened window is modal.
While asking all the other functionality is blocked.
Please make your selection without delay.

Parameters

title	String. Question to ask.
options	String. Options to select from. Options are separated by slashes.

Returns

Number of the selected option starting from 1.

Example

```
local color = JS.askoption('Select a color', 'Red/Green/Blue')
```
Color will be 1 for Red, 2 for Green, and 3 for Blue.

Example

In the beginning we had a problem how to print competitor’s points.
Requirements:

- Write nothing if the match has not been fought.
- If match ends up hikiwake (draw) write DRAW.
- Write LOSS if competitor has no points.
- We want to write 5 for waza-ari win. Database marks that with number 7.

- Otherwise, write the database points value.

We have to modify the SVG file. Write text `%[require 'pts'; pts(1)]m<n>` if you want to print the points of the white competitor and `%[require 'pts'; pts(2)]m<n>` for the blue competitor (<n> is the match number).

Make a file `pts.lua`. Start with function `pts`:

```
function pts(who)                               -- line 1
    if JS_points_1 > 0 or JS_points_2 > 0 then      -- line 2
        if JS_points_1 == JS_points_2 then          -- line 3
            JS.writesvg('DRAW')                     -- line 4
        else                                    -- line 5
            if who == 1 then                        -- line 6
                print_points(JS_points_1)           -- line 7
            else                                -- line 8
                print_points(JS_points_2)           -- line 9
            end
        end
    end
end
```

Explanation line by line:

1. Define function pts. It has one argument, who. If who is 1 it means white competitor, 2 means blue competitor.

2. Check if the match has been fought. Either white or blue must have points. JS_points_1 > 0 is true if white has more points than 0. Or is true if blue or white have points.

3. If previous test was true check if the points are the same. This means hikiwake or draw.

4. This line is executed if the points are the same. Write DRAW on the SVG page.

6. Otherwise, check if who is 1.

7. If who is 1 (= white) call function print_points with the white points as its argument.

9. If who was not white call function print_points with the blue points as its argument.

Finally implement function print_points:

```
function print_points(p)               -- line 1
    if p == 0 then                     -- line 2
        JS.writesvg('LOSS')            -- line 3
    elseif p == 7 then                 -- line 4
        JS.writesvg('5')               -- line 5
    elseif p == 11 then                -- line 6
        JS.writesvg('DRAW')            -- line 7
    else                               -- line 8
        JS.writesvg(tostring(p))       -- line 9
    end
end
```

Explanation:

1. Function print_pts has one argument p, the points to print.

2, 3. If p is 0 print LOSS.

4, 5. Otherwise, if p is 7 print 5. In this case a national rule says that waza-ari scores 5 points.

6, 7. Otherwise, if p is 11 print DRAW. In principle draw was detected in the previous function, but let's have it here too for completion.

9. Otherwise, print points as is.

Points in the database can have the following values:

1:	Win by shido
2:	Win in golden score when that always scores 1 point
3:	Win by koka (not used)
5:	Win by yuko (not used)
7:	Win by waza-ari
10:	Win by ippon
11:	Hikiwake

Real points and how they are shown depend on the national rules.

SVG + Lua scripts

There are two ways to run Lua scripts:

- Embed code to SVG sheets. Lua code is embedded to text by using format
  %[lua_code_here]
- Run Lua file. There will be a default SVG file where all the printing goes. Note: The default file has no Inkscape specific definitions. When creating SVG files to include them save them as plain SVG.

Lua embedded in SVG sheets

You can create and print pages generated from SVG files that contain embedded Lua code. Usually Lua code is divided in the two parts:

- Simple code in the SVG file, that calls the more complicated functions.
- Code in a Lua file that contains most of the code.

JudoShiai is delivered with example code in directory svg-lua.

There are two ways to use embedded Lua code:

- Lua in the bracket sheets to use for example non-standard name formats.
- Separate SVG file that can display for example statistics. In JudoShiai select *Tournament → SQL Dialog → Run Script*. Select an SVG file to show.

Whenever JudoShiai finds text like %[*lua code*] the Lua code will be executed. Lua code writes whatever you like, and the output will replace the original %[…].

However, this way you cannot draw graphics since the output from the Lua code will be surrounded by <text...> and </text> tags. Solution is to have one (and only one) command that is called in the end of the SVG file. Text has an exclamation mark between the "%" and "[":

%![*lua code to the end]*

Lua file with the default SVG

You can run a Lua script by selecting *Results → Run Script.* Everything you write goes between main <g> and </g> tags. Next example code draws a 50x60 mm rectangle at position 10, 10:

```
JS.writesvg("<rect x='10' y='10' height='50' width='60' />")
```

Normally there are many more parameters, like colors, line style, etc. However, it is not necessary to know the SVG format. You can draw a file and name the objects and import the objects one by one. In the following example we assume you have your Lua code in JudoShiai/svg-lua/mytest.lua and SVG file in JudoShiai/svg-lua/svg/mytest.svg.

Use Inkscape to draw a rectangle. Set its id = myrect. Save it using *Save As → Plain SVG* (important!) to mytest.svg. Your file looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->

<svg
   width="210mm"
   height="297mm"
   viewBox="0 0 210 297"
   version="1.1"
   id="svg5"
   xmlns="http://www.w3.org/2000/svg"
   xmlns:svg="http://www.w3.org/2000/svg">
  <defs
     id="defs2" />
  <g
     id="layer1">
    <rect
       style="fill:#d5e5ff;stroke:#000000;stroke-width:0.3;...MORE..."
       id="myrect"
       width="23.7"
       height="18.4"
       x="11.9"
       y="11.5" />
  </g>
</svg>
```

Style is a long line. Note the id="myrect". Write the following lines to your Lua file:

```
local rect = JS.getsvgbyid(LuaScriptDir .. '/svg/mytest.svg', 'myrect')
```

Function JS.getsvgbyid reads object myrect to local variable rect. You can draw it on the SVG canvas simply like this:

```
JS.writesvg(rect)
```

It will appear at position 11.9, 11.5. Maybe you want to have another one next to it. First you have to know its dimensions:

```
local x, y, width, height = JS.getsvggeometry(LuaScriptDir .. '/svg/mytest.svg',
                            'myrect')
```

Now you can draw a copy shifted by width:

```
JS.writesvg(string.format('<g transform="translate(%d, 0)">', width))
JS.writesvg(rect)
JS.writesvg('</g>')
```

This way you can make lists just by creating one line that is copied.

string.format replaces %d with the corresponding integer. Why integer since width is a float? Using floats can be dangerous. More in the next paragraph.

Float values in Lua

A decimal separator is a symbol used to separate the integer part from the fractional part of a number written in decimal form (e.g., "." in 3.14). Different countries officially designate different symbols for use as the separator. Separator can be a dot or a comma. Example:

```
print(string.format('%f', 3.14))
```

In many countries this will print 3.14, but in the others 3,14. SVG accepts only the dot as a decimal separator. To be on the safe side it is better to convert the float to a string and then replace possible comma with a dot. Function FloatToStr does that:

```
function FloatToStr(a)
  local r = tostring(a)
  return r:gsub(',', '.')
end
```

Now, instead of %f you will use %s for a string. The previous translate line

```
JS.writesvg(string.format('<g transform="translate(%d, 0)">', width))
```

changes to

```
JS.writesvg(string.format('<g transform="translate(%s, 0)">',
                     FloatToStr(width))
```

Page size

The default page size is A4 (210mm x 297mm). You can define your own size by having a comment on the first line of the Lua file:

```
-- paper width=300 height=200
```

It is possible to have many sizes for the same Lua code. Create a base file. Then include that in the other files:

```
-- paper width=300 height=200
paper_w = 300
paper_h = 200
require 'mybase'
```

Use `paper_w` and `paper_h` for layout design in `mybase.lua`.

Multiple pages

SVG has no standard for multiple pages. However, you can draw as many pages as you wish and send them to a printer that can be a PDF file, too.

Set the total number of pages first:

```
JS.setpages(3)
```

Next draw you page based on the current page number:

```
current_page = JS.getpage()
if current_page == 1 then
  print_first_page()
else
  print_something_else()
end
```

Page changes when you click the visible SVG page with the left or right mouse button. In printer dialog you can select the pages as usual.

SQL commands

Example: Print members of category `M-90`:

```
local numrows, tbl = JS.sqlrows(
    'select last,first from competitors where category="M-90"')

```

`Numrows` tells how many rows `tbl` has. `Tbl` looks like this:

```
{
  1=     {
      last= "SCHMIDT",
      first= "Leon",
    },
  2=     {
      last= "NEUMANN",
      first= "Maximilian",
    },
  3=     {
      last= "CHRISTENSEN",
      first= "Frederik",
    },
}
```

There are three rows. You can get last name of the second competitor like this:

```
last_name = tbl['2']['last']
```