SM.EXEC binding to Lua

```
sm = require("sm")
```

SM.DCB

DCB (Data Control Block) - is a Lua data element linked to data element of any of { `sm.event` , `sm.state` , `sm.tx` , `sm.exec` }. DCB is used to access data blocks within SM.EXEC objects indexed by application key string. DCB keeps track of the current read/write position within its data block.

Get DCB

```
e1 = sm.new_event(size)
eData = e1:get_DCB(key1)
s1 = sm.new_state()
sData = s1:get_DCB(key2)
```

Write integer

```
eData:write_int(int)
```

Read integer

```
eData:read_int(int)
```

Allocate array of integers

```
eData:allocate_int_array(size)
```

Read value at pos in array

```
i = eData:int1d_get(pos)
```

Write value at pos in array

```
eData:int1d_set(pos)
```

Allocate 2d array of integers

```
eData:allocate_2d_int_array(size)
```

Read value at (row, col) in 2d array

```
i = eData:int2d_get(row, col)
```

Write value at (row, col) in 2d array

anton.bondarenko@gmail.com

```
eData:int2d_set(row, col)
```

Write string

```
eData:write_str(str)
```

Read string

```
str = eData:write_str(str)
```

Skip n bytes

```
eData:skip(n)
```

Aligb

```
eData:align()
```

Rewind

```
eData:rewind()
```

Size

```
#eData
```

SM.EXEC

Main Lua wrapper process singleton. Created on init, exposed to Lua as global name "sm_exec".

Get exec data block as state_tostring

```
str = exec[app_key]
```
Deprecated, use DCB access instead)

Get DCB

```
execData = _G.sm_exec:getDCB(key)
```

SM.DIRECTORY

Not accessible in Lua other than through stored elements special getters and setters. Currently supported only for FSM-s and for Applications. Other architecture block to follow. Initialized on load of SM module.
Get operations are not implemented for containers as they are not needed actually.

anton.bondarenko@gmail.com

Register app in the dir

```
a1:set_app(key)
```

Get app from dir

```
a2 = sm.get_app(key)
```

Register fsm in the dir

```
f1:dir_set(key)
```

Get fsm from dir

```
f2 = sm.dir_get(key)
```

Register queue/queue2/pqueue/array

```
q:dir_set(key)
```

SM.EVENT

The main trolley of SM.EXEC.

Create event:

```
e = sm.new_event(size)
```

Set event data:

```
e:set(string)
```
(Deprecated, use DCB access instead)

Get event data:

```
str = e:get()
```
(Deprecated, use DCB access instead)

Set event id:

```
e:set_id(int)
```

Get event id:

```
e:id() -> int
```

Set event priority:

anton.bondarenko@gmail.com

`e:set_priority(int, int)`

Get event priority:

`[p0, p1] = e:get_priority()`

Get DCB

`eventData = e1:get_DCB(key)`

Chain events

`e1..e2`

Next event in chain

`e3 = e1:next()`

Unlink events

`-e1`

Event data block size

`#e`
(Deprecated)

Get event data block as string

`str = e[app_key]`
Deprecated, use DCB access instead)

SM.QUEUE

Basic event queue

Create queue:

`q = sm.new_queue(queue_size, event_size, sync)`

Get top event (w/o extracting)

`e = q:top()`

Dequeue event

`e = q:dequeue()`

anton.bondarenko@gmail.com

Enqueue event

```
q:enqueue(e)
```

Register queue in the Directory

```
q:dir_set(key)
```

Get a number of events

```
#q
```

SM.PQUEUE

Priority queue

Create priority queue:

```
pq = sm.new_pqueue(capacity, sync)
```

Get top event (w/o extracting)

```
e = pq:top()
```

Dequeue event

```
e = pq:dequeue()
```

Enqueue event

```
pq:enqueue(e)
```

Register queue in the Directory

```
pq:dir_set(key)
```

Get a number of events

```
#pq
```

SM.QUEUE2

Biproprity queue `q2 = sm.new_queue2()`

anton.bondarenko@gmail.com

Get top event from normal line

```
e = q2:get()
```

Get top event from 'fast' line

```
e = q2:get_high()
```

Enqueue event

```
q2:enqueue(e)
```

Enqueue event in 'fast' line

```
q2:enqueue_high(e)
```

Enqueue event with lock

```
q2:lock_enqueue(e)
```

Enqueue event in 'fast' line with lock

```
q2:lock_enqueue_high(e)
```

Dequeue event

```
e = q:dequeue()
```

Dequeue event with lock

```
e = q:lock_dequeue()
```

Register queue in the Directory

```
q2:dir_set(key)
```

Get a number of events

```
#q2
```

SM.APP

Applicationfeature(elementarnative application) Lua handler

Load dynamic lib (.so) with apps

```
handle = sm.load_applib(filename)
```


anton.bondarenko@gmail.com

Find app in the lib (handle)

```
app = sm.lookup(handle, name)
```

Call application

```
app(event, state)
```

Register app in the dir

```
a1:dir_set(key)
```

Get app from dir

```
a2 = sm.dir_get(key)
```

SM.FSM

FSM LUA handler

Create new state machine

```
sm = sm.new_fsm(json, type)
```
`json` - state machine description in JSON
```
type = {"mealy" | "moore"}
```

Register fsm in the dir

```
f1:dir_set(key)
```

Get fsm from dir

```
f2 = sm.dir_get(key)
```

Make fsm collectable

```
f:free()
```

SM.STATE

State of the service containing application context with reference to FSM and specific current state in it.

Create new state machine

anton.bondarenko@gmail.com

```
s1 = sm.new_state(fsm, data_block_size)
```

Add event to the trace

```
s1:trace_add(e1)
```

Get top the top event in trace (w/o removing)

```
e2 = s1:trace_get()
```

Set state data:

```
s1:set(string)
```
(Deprecated, use DCB access instead)

Get state data:

```
str = s1:get()
```
(Deprecated, use DCB access instead)

Set state key:

```
s1:set_key(string)
```

Set state key:

```
str = s1:get_key()
```

Set state id:

```
s1:set_id(int)
```

Get state id:

```
i = e:id()
```

Get state DCB

```
stateData = s1:get_DCB(key)
```

Get state data block as string

```
str = s1[app_key]
```
Deprecated, use DCB access instead)

Apply event to state

```
s1:apply(e1)
```


anton.bondarenko@gmail.com

Purge all state information & detach

```
s1:purge()
```

SM.ARRAY

Hash array of states (applicaion contexts)

Create new state array

```
a = sm.new_array(stack_size, state_data_block_size)
```

Get number of states in stack

```
#a
```

Get state by key

```
s1 = a:get(key)
```

Find state by ey

```
s1 = a[key]
```

Release state

```
a:release(s1)
```

Register array in the dir

```
f1:dir_set(key)
```

SM.TX

Thread-worker descritor (Lua handler for).

Create new tx

```
tx = sm.new_tx(fsm, queue2, sync, tx_data_block_size)
```
`fsm` and `queue2` are given by Directory names

Get tx data block as string by key

```
str = tx[app_key]
```


anton.bondarenko@gmail.com

Deprecated, use DCB access instead)

Get tx DCB

```
txData = tx:get_DCB(key)
```

Run thread-worker

```
tx:run()
```


anton.bondarenko@gmail.com