SM.EXEC - Lua wrapper

[anton.bondarenko@gmail.com]

Load module

`sm = require("sm")`

SMEvent

Implemented methods

- Create: `e = sm.new_event(is, data_size)`
- Set data: `e:set(str)`
- Get data: `str = e:get()`
- Set Id: `e:setid(id)`
- Get Id: `id = e:getid()`
- Chain: `e1..e2`
- Get next in chain: `e3 = e1:next()`
- Unlink: `-e1`
- __tostring: `#e`
- __gc: `collectgarbage()`

Inherited methods

- Assign: `e1 = e2`
- Check equality: `e1 == e2`

SMQueue

Implemented methods

- Create: `q = sm.new_queue(qsize, plsize, sync)`
- Get top event handler: `q:top()`
- Enqueue: `q.enqueue()`
- Dequeue: `q.dequeue()`
- __len: `#q`
- __tostring: `print(q)`
- __gc: `collectgarbage()`

Inherited methods

- Assign: `q1 = q2`
- Check equality: `q1 == q2`

SMQueue2

Implemented methods

- Create: `q = sm.new_queue2()`

- Get next low priority event handler: `e = q:get()`
- Get next high priority event handler: `e = q:gethigh()`
- Enqueue with low proority: `q:enqueue(e)`
- Enqueue with low proority and with lock: `q:lockenqueue(e)`
- Enqueue with high proority: `q:enqueuehigh(e)`
- Enqueue with high proority and with lock: `q:lockenqueuehigh(e)`
- Dequeue: `q.dequeue()`
- Dequeue with lock: `q.lockdequeue()`
- __len: `#q` - Always return 0
- __tostring: `print(q)`
- __gc: `collectgarbage()`

Inherited methods

- Assign: `q1 = q2`
- Check equality: `q1 == q2`

SMApp

Implemented methods

- Lookup: `app = sm.lookup(handle, name)` - Library method
- Invoke: `app(e)`
- __tostring: `print(app)`

Inherited methods

- Assign: `app1 = app2`
- Check equality: `app1 == app2`
- __gc: `collectgarbage()`

SMAppTable

Implemented methods

- Load library: `handler = sm:loadlib(file)` - Library method
- Create: `at = sm.new_apptab()`
- Set application: `at:set(app, name)`
- Get application: `at:get(name)`
- Remove application: `at:remove(name)`
- __tostring: `print(at)`
- __gc: `collectgarbage()`
- __len: `#at`

Inherited methods

- Assign: `at1 = at2`
- Check equality: `at1 == at2`

SMFSM

Implemented methods

- Create: `fsm = sm.new_fsm(fsm_json, at, type)` - Type = { "mealy" | "moore" }
- __tostring: `print(fsm)`
- __gc: `collectgarbage()`

Inherited methods

- Assign: `fsm1 = fsm2`
- Check equality: `fsm1 == fsm2`

SMState

Implemented methods

- Create: `s = sm.new_state(fsm, plsize)`
- Add event to state trace: `s:traceadd(e)`
- Get state trace stack: `e = s:traceget()`
- Set hash key: `s:setkey(str)`
- Get hash key: `str = s:getkey()`
- Set state data: `s:set(str)`
- Get state data: `str = s:get()`
- Set state id: `s:setid(id)`
- Get state id: `id = s:getid()`
- Purge state content: `s:purge()`
- Apply event to state: `s:apply(e)`
- __tostring: `print(s)`
- __gc: `collectgarbage()`

Inherited methods

- Assign: `s1 = s2`
- Check equality: `s1 == s2`

SMArray

Implemented methods

- Create: `a = sm.new_array(stsize, plsize)` - Stack size & payload size
- Find state by key (return handler): `s = a:find(str)`
- Get state by key (activate if not active): `s = a:get(str)`
- Release state (deactivate): `a:release(s)`
- __tostring: `print(a)`
- __gc: `collectgarbage()`
- __len: `#a`

Inherited methods

- Assign: `a1 = a2`
- Check equality: `a1 == a2`