LogicMachine Italia

LUA per OpenRB

Manuale istruzioni di base

Marco Melis

Ver 1. - 2022

LUA per OpenRB

INTRODUZIONE

La presente mini guida racchiude alcune delle istruzioni LUA disponibili nel framework OpenRB/LogicMachine, ma non tutte le istruzioni disponibili.

Link utili (in lingua inglese) per la comprensione piu' allargata delle istruzioni sono

Elenco completo di istruzioni LUA disponibili >> https://openrb.com/docs/lua.htm

Supporto in lingua inglese >> https://forum.logicmachine.net/

Gruppo Facebook riservato agli installatori e clienti LogicMachine Italia >> https://www.facebook.com/groups/logicmachine/

FUNZIONI DI BASE

La libreria *basic* di LUA include alcune funzioni native. Se questa libreria non viene inclusa nei Vostri script, occorre verificare se il codice richiede una delle funzioni qui' sotto elencate.

NOTA: la libreria è nativamente inclusa nell'ambiente di sviluppo e non è necessaria alcuna azione per richiamarla

---

Funzione *ipairs(t)*

All'interno di una istruzione *for … do … end* restituisce il valore della tabella *t* fino ad arrivare al primo indice NON numerico.

Esempio >>

```
      local t = { "alpha","beta", ["charlie"] = delta }
      for _ , v in ipairs(t) do
            log(_ , v)
      end
restituisce >>    [1] , alpha
                  [2] , beta
```

NOTA: la funzione for si interrompe al primo indice non numerico e non alla fine della tabella stessa

---

Funzione *pairs(t)*

All'interno di una istruzione *for … do … end* restituisce il valore della tabella *t* per ogni tipo di indice (numerico e non)

Esempio >>

```
      local t = { “alpha”,”beta”, [”charlie”] = delta}
      for _ , v in ipairs(t) do
            log(_ , v)
      end
restituisce >>    [1] , alpha
                  [2] , beta
                  [charlie] , delta
```

Funzione ***next(t [, indice])***

Restituisce l’indice e il valore successivo della tabella *t*

Quando non viene specificato (valore *nil*) il secondo parametro (*indice*) viene restituito il valore successivo, quando viene indicato un *indice* oltre la fine della tabella o in un valore *t* non tabella, viene restituito *nil,* se invece viene indicato una valore *indice* non compreso negli indici esistenti della tabella *t*, viene restituito come errore *undefined*

L’ordine di restituzione dei valori non è noto e sicuramente non è in ordine crescente per indice (numerico o non)

La funzione *next* puo’ essere utilizzata come alternativa per verificare se l’argomento *t* fornito è una tabella (se restituisce *nil*… non lo è)

Esempio >>

```
      local t = { 1, 40, ”ciao” }
      log(next(t))            -- Restituisce 1
      t = 300
      log(next(t))            -- Restistuisce nil
```

Funzione ***tonumber(e [, base])***

Converte il valore *e* in un numero, qualora la conversione non sia possibile o fallisca, restituisce *nil*

Il parametro facoltativo *base* permette di specificare la base di conversione da 2 a 36 (inclusi)

Se non specificato, la base di default è 10.

Esempio >>

```
      local t = “11”
      log(tonumber(t))
      log(tonumber(t,2))
      log(tonumber(t,8))
      log(tonumber(t,16))

restituisce >>    11
                  1011
                  13
                  B
```

Funzione ***tostring(e)***

Converte l’argomento fornito *e* (qualunque sia la sua natura, purché sia definito) in stringa.

Per una comprensione piu' esaustiva di come vengono convertiti i numeri è possibile vedere l'istruzione ***string.format*** più avanti

Se il parametro *e* è una tabella e contiene la chiave *__tostring*, verrà utilizzato il valore della chiave per convertire la tabella.

Funzione ***type(v)***

Restituisce, in formato stringa, il tipo di valore *v* passato.

Il risultato è necessariamente uno dei seguenti valori:

"number"	Numero (intero o virgola mobile)
"string"	Stringa / Testo
"boolean"	Logico (vero/falso)
"table"	Tabella
"function"	Funzione (sia user che core)
"thread"	esecuzione di processo
"userdata"	Blocco di codice C usato nello script corrente

La funzione restituisce *nil* in formato stringa quando viene passato un parametro *v* non definito

MANIPOLAZIONE DELLE STRINGHE / TESTI

Questa libreria fornisce funzioni generiche per la manipolazione delle stringhe, come la ricerca e l'estrazione di sottostringhe e le corrispondenze (pattern).

Lua restituisce gli indici di ricerca partendo a 1 e non da 0

Gli indici positivi si intendono da sinistra verso destra, gli indici negativi da destra verso sinistra.

La libreria è disponibile anche come *istruzione in tabella*, pertanto è possibile ottimizzare il codice:

string.byte(s,i) >> s:byte(i)

La libreria di stringhe presuppone codifiche di caratteri a un byte.

Funzione ***string.byte(s [, i [, j]])***

Restituisce il codice carattere in formato numerico della stringa *s* partendo dall'indice *i*, fino all'indice *j*.

Il valore di default di *i* è 1, il valore di default di *j* è *i*

Gli indici *i* e *j* sono intesi come indici fisici della stringa

In caso di restituzione di valori multipli, questi vengono restituiti come singoli numeri e non come tabella, di conseguenza vanno assegnate variabili in congruo numero per evitare di perdere dati

Esempio >>

```
      local s = "ciao"
      local a, b = string.byte(s,1,2)
      log(a,b)
restituisce >>    99
                  105
```

Funzione ***string.char(s [, a [, ...]])***

Fornendo una serie di numeri, restituisce una stringa di caratteri di uguale lunghezza

I valori numerici possono essere indicati anche in formato esadecimale con il prefisso *0x*

Esempio >>

```
log (string.char(0x63, 0x69,0x61, 0x6f))        -- Restituisce “ciao”
log (string.char(99, 105, 97, 111))             -- Restituisce “ciao”
```

NOTA: Non puo’ essere fornita una tabella come lista di codici da trasporre, questi andranno passati uno per volta

Esempio >>

```
local t = {99,105,97,111}
local s = ''
for _, v in pairs(t) do
      s = s..string.char(v)
end
log(s)                                          -- Restituisce “ciao”
```

Funzione ***string.find(s, pattern [, init [, plain]])***

Ricerca il *pattern* all’interno della stringa *s* e restituisce l’indice numerico della posizione di inizio e di fine della corrispondenza, restituisce *nil* se invece non viene trovata
Il parametro *init* indica da quale carattere (>0 da sinistra verso destra, <0 da destra verso sinistra) di *s* cominciare la ricerca.

Se p*lain* viene dicato a *true* allora verrà ricercato il *pattern* come stringa senza considerare nessun carattere speciale. Inoltre, se viene specificato un valore per *plain* è obbligatorio specificare un valore per *init* (default = 1)

Funzione ***string.format(formatstring, param1 [,...])***

Restituisce una stringa formattata secondo le direttive e per tutti i parametri specificati.

All’interno di *formatstring* devono essere incluse le direttive di formattazione che verranno sostituiti da *paramX* nell’ordine in cui vengono specificati.

CONSIDERAZIONE: La formattazione delle stringhe è un argomento estremamente vario, difficilmente puo’ esistere un solo modo di rappresentare un testo.

Le direttive più comuni (precedute dal carattere %) sono:

%d | %i	Intero Decimale con segno
%u	Intero Decimale senza segno
%o	Numero a base 8
%x | X	Numero base 16 (minuscolo|maiuscolo)
%f | F	Numero decimale a virgola mobile (minuscolo|maiuscolo)
%e | E	Notazione scientifica mantissa/esponente (minuscolo|maiuscolo)
%s	Stringa
%%	Scrive il carattere %

Ogni direttiva può specificare ulteriori parametri come ad esempio la lunghezza massima del campo (specificando il numero), la giustificazione (senza indicazione è intesa da destra verso sinistra, con – si indica da sinistra verso destra) e il numero massimo di decimali (indicando con .n)

Poiché l’istruzione prende origine dall’istruzione *printf* di C++ a questo link è possibile visualizzare altre possibili opzioni (ad eccezione delle opzioni *, l, L, n, p)

Esempio >>

```
local s = 'La temperatura esterna è di %.2f°C, Umidità al %.1f%%'
log (string.format(s,36.999546,74.88))

Restituisce >> “La temperatura esterna è di 37.00°C, Umidità al 74.9%”

local s = 'La temperatura esterna è di %-10.2f°C, Umidità al %7.1f%%'
log (string.format(s,36.929546,74.88))

Restituisce >> “La temperatura esterna è di 36.93     °C, Umidità al    74.9%”
```

Funzione ***string.gmatch(s, pattern)***

Restituisce la corrispondenza (*pattern*) indicata espressa come direttiva all’interno della stringa *s*

Inserita all’interno di un ciclo *for…do…end* restituisce tutte le corrispondenze trovate.

Esempio >>

```
local s = ‘LUA è semplice’
for v in string.gmatch(s, “(%a+”) do
   log(v)
end
restituisce >>    “LUA”
                  “è”
                  “semplice”

local s = ‘ipAddress=192.168.1.254’
for v in string.gmatch(s, “(%d+)”) do
   log(v)
end
log(string.gmatch(s, "%d+.%d+.%d+.%d+")()) -- restituisce “192.168.1.254”

restituisce >>    “192”
                  “168”
                  “1”
                  “254”
```

Funzione ***string.gsub(s, pattern, repl [, n])***

Restituisce una copia della stringa *s* al cui interno la corrispondenza *pattern* è stata sostituita con il valore *repl* per un numero di interazioni *n* (se non specificato si intendono tutte le corrispondenze presenti)

La funzione restituisce come secondo parametro il numero di sostituzioni

Se *repl* è una tabella, la sostituzione avverrà usando il valore della corrispondenza trovata come chiave della tabella

Esempio >>

```
      local t = {['nome'] = 'marco', ['cognome'] = 'rossi'}
      local s = 'Nome = $nome$, Cognome = $cognome$'
      log(string.gsub(s, '%$(%w+)%$', t))
restituisce >> “Nome = marco, Cognome = rossi”
```

Funzione ***string.len(s)***

Restituisce il numero di caratteri della stringa *s* passata.

NOTA: i caratteri di controllo all’interno della stringa non contribuiscono al conteggio.

Funzione ***string.lower(s)***

Restituisce una copia della stringa *s* convertendo tutti i caratteri in minuscolo

Funzione ***string.match(s, pattern [, init])***

Restituisce la posizione del carattere della stringa *s*, dove è stato trovata la coorispondenza *pattern*

L’Opzione *init* (default = 1) specifica la posizione di partenza della ricerca

---

Funzione ***string.rep(s, n)***

Ripete la stringa *s* per il numero di volte *n*.

Funzione ***string.reverse(s)***

Restituisce la stringa *s* invertita.

Funzione ***string.sub(s,i [, j])***

Restituisce una porzione della stringa *s* a partire dalla posizone *i*, fino alla posizione *j*

In caso di numeri negativi l’estrazione verrà effettuata da destra verso sinistra.

Funzione ***string.upper(s)***

Restituisce una copia della stringa *s* convertendo tutti i caratteri in maiuscolo

PATTERN DI RICERCA

Classi

Le classi di ricerca sono i caratteri di controllo impiegati per identificare le varie tipologie di stringhe e numeri. Le loro combinazioni, in aggiunta alle opzioni, rappresentano la regola di corrispondenza.

Le direttive ricercano UN SINGOLO carattere

x	(ad esclusione dei caratteri speciali ^$()%.[ ]*+-?) rappresenta il carattere raffigurato
.	Rappresenta tutti i caratteri
%a	Rappresenta tutte le lettere
%c	Rappresenta i caratteri di controllo (\n \r \t …)
%d	Rappresenta tutti i numeri
%l	Rappresenta lettere minuscole
%p	Rappresenta caratteri di punteggiatura
%s	Rappresenta il carattere spazio
%u	Rappresenta lettere MAIUSCOLE
%w	Rappresenta caratteri alfanumerifici
%x	Rappresenta numeri in base esadecimale
%z	Rappresenta numeri con 0 iniziali
%*x*	(dove x non è nessun carattere alfanumerico) rappresenta il carattere indicato
[set]	Rappresenta l'unione di 2 o più classi insieme, l'impiego del carattere – indica range di inclusione

Per ogni classe indicata, la corrispondente lettera maiuscola, indica la classe indicata negata:
%s indica gli spazi, %S indica i caratteri NON spazi

Indicatori

Poiché le classi ricercano un singolo carattere, è possibile affiancare degli indicatori per migliorare la ricerca:

- La classe ricerca un solo carattere/numero
- Classe e * restituisce 0 o più corrispondenze. La ripetizione restituisce il massimo numero di corrispondenze
- Classe e + restituisce 1 o più corrispondenze. La ripetizione restituisce il massimo numero di corrispondenze
- Classe e - restituisce 0 o più corrispondenze. La ripetizione restituisce il minimo numero di corrispondenze
- Classe e ? restituisce 0 o 1 corrispondenze.
- Classe e %*n* (con *n* compreso fra 1 e 9) estende il pattern al numero di caratteri indicato
- %b*xy* crea un pattern dove *x* e *y* rappresentano i caratteri di inizio e fine ricerca
    - Esempio: %b() ricerca tutti i caratteri inclusi fra parentesi

Pattern

Un pattern è una sequenza organizzata di classi ed indicatori.

Utilizzando il carattere ^ all'inizio del pattern si bloccherà il conteggio dei caratteri all'inizio della corrispondenza.

Utilizzando il carattere $ all'inizio del pattern si bloccherà il conteggio dei caratteri alla fine della corrispondenza.

L'impiego di ^ e $ all'interno dello stesso pattern non ha alcun valore e viene ignorato

Cattura

Ogni pattern puo' contenere al suo interno ulteriori sub-pattern che vengono specificati con le parentesi ( )

Quando una corrispondenza viene trovata (catturata), questa viene passata al pattern piu' interno

Esempio >>

```
local s = 'ipAddress=192.168.1.254'

ip1,ip2,ip3,ip4,ip5 = string.gmatch(s,"((%d+).(%d+).(%d+).(%d+))")()

log(ip1,ip2,ip3,ip4,ip5)
```

Restituisce >> "192.168.1.254"
"192"
"168"
"1"
"254"

MANIPOLAZIONE DELLE TABELLE

Funzione ***table.concat(table [, sep [, i [, j]]])***
Restituisce una stringa composta da ogni valore della tabella *table* separati dal valore *sep* partendo dall'indice numerico *i* fino ad arrivare all'indice numerico *j*

NOTA: la funzione è valida solo in caso di array (quindi con indice tabella numerico), tabelle con indici testuali NON vengono considerati dalla funzione

---

Funzione ***table.insert(table [, pos] , value)***
Inserisce il valore *value* nella tabella *table* nella posizione *pos* specificata
Se la posizione non viene specificata, il valore verrà aggiunto alla fine della tabella

---

Funzione ***table.maxn(table)***
Restituisce l'indice numerico piu' alto della tabella *table*

Funzione ***table.remove(table [, pos])***
Elimina il valore della tabella *table* dell'indice *pos* indicato.
Se *pos* non viene specificato, si assume la fine della tabella e quindi un *table.remove(t)* eliminerà l'ultima riga della tabella

---

Funzione ***table.sort(table [, comp])***
Ordina la tabella *table* in base ai suoi valori secondo quanto specificato in *comp*
comp va espressa come funzione alla quale vengono passati 2 valori di tabella
quando *comp* non viene indicato si assume l'ordinamento crescente
Esempio >>

```
      t = { 10,5,9,3,7 }
      table.sort(t, function(k, s) return k > s end )
      log(t)
restituisce >>    10
                  9
                  7
                  5
                  3
```

FUNZIONI MATEMATICHE
La libreria matematica è assimilabil e a quella del linguaggio C++ e provvede a fornire tutte le operazioni matematiche sotto forma di tabella *math*

Funzione `math.abs(x)`
Restituisce il valore assoluto di *x*

Funzione `math.acos(x)`
Restituisce il valore arco coseno di *x* in radiante

Funzione `math.sin(x)`
Restituisce il valore arco seno di *x* in radiante

Funzione `math.atan(x)`
Restituisce il valore arco tangente di *x* in radiante

Funzione `math.atan2(x, y)`
Restituisce il valore arco tangente di *x*e *y* in radiante, ma in base al segno dei parametri, restituisce anche il quadrante di riferimento

---

Funzione `math.ceil(x)`
Restituisce il valore di *x* arrotondato all'intero per eccesso

---

Funzione `math.cos(x)`
Restituisce il coseno di *x* in radiante

Funzione `math.cosh(x)`
Restituisce il coseno iperbolico di *x*

Funzione `math.deg(x)`
Restituisce l'angolo di *x* (espresso in radiante) in gradi

Funzione `math.exp(x)`
Restituisce il valore *e* elevato a *x*

Funzione `math.floor(x)`
Restituisce il valore di *x* arrotondato per difetto

Funzione `math.fmod(x, y)`
Restituisce il resto della divisione fra *x* e *y*

Funzione `math.frexp(x)`
Restituisce il valore di *m* ed *e* nell'equazione $\mathbf{x = m2^e}$
E è un intero e il valore assoluto di *m* è compreso nel range 0.5-1 (o 0 se x = 0)

Funzione `math.huge`
COSTANTE - Restituisce il valore piu' grande gestibile dal sistema

---

Funzione `math.ldexp(m, e)`
Restituisce il valore dell'equazione **m2^{e}**

Funzione `math.log(x)`
Restituisce il logaritmo naturale di *x*

Funzione `math.log10(x)`
Restituisce il logaritmo base10 di *x*

Funzione `math.max(x, ...)`
Restituisce il valore massimo della sequenza di valori forniti

---

Funzione `math.min(x, ...)`
Restituisce il valore minimo della sequenza di valori forniti

---

Funzione `math.modf(x)`
Restituisce 2 valori contenente le unità i decimali di *x*

Funzione `math.pi`
COSTANTE - Restituisce il valore di piGreco

Funzione `math.pow(x, y)`
Restituisce il valore dell'operazione x^y
Puo' anche essere sostituita dall'operazione matematica *x^y*

Funzione `math.rad(x)`
Restituisce l'angolo di *x* (espresso in gradi) in radiante

Funzione `math.random([m [, n]])`
Genera un numero decimale casuale compreso fra 0 e 1
Se *m* è specificato viene generato un numero intero fra 1 e *m*
Se *n* è specificato viene generato un numero intero fra *m* ed *n*

Funzione `math.randomseed(x)`
Imposta *x* come base per il generatore di numeri psuedo-casuali.
Stesso valore di *x* genera le medesime sequenze di numeri

Funzione `math.sin(x)`
Restituisce il seno di *x* in randiante

Funzione `math.sinh(x)`
Restituisce il seno iperbolico di *x*

Funzione `math.sqrt(x)`
Restituisce la radice quadrata di *x*

Funzione `math.tan(x)`
Restituisce la tangente di *x*

Funzione `math.tanh(x)`
Restituisce il la tangente iperbolica di *x*

GESTIONE INPUT/OUTPUT
La gestione I/O puo' essere effettuata in maniera implicita, attraverso la tabella *io.* Oppure esplicita, attraverso la gestione ad oggetti dei singoli files.

Funzione `io.close([file])`
Chiude *file*, se non specificato chiude quello di default
Metodo esplicito: *file:close()*

Funzione `io.exist(path)`
Restituisce un valore logico riguardo l'esistenza del file *path*

Funzione `io.flush()`
Salva il buffer corrente sul file, ma non lo chiude
Metodo esplicito: *file:flush*
NOTA: nella gestione di files in scrittura, questi vengono fisicamente scritti sul filesystem solo alla loro chiusura. Il metodo flush forza la scrittura prima della loro chiusura

---

Funzione `io.input([file]) / io.read()`
Se chiamato con un nome file, apre il file in modalità test e si setta come default.
Quando chiamato con il riferimento file, si setta come default
Quando *file* non viene indicato, restituisce il nome file di default corrente
In caso di errore, genera una eccezione (ed interrompe il codice)

Funzione `io.lines([filename])`
All'interno di un ciclo *for … do … end* apre *filename* in modalità di lettura e restituisce una riga alla volta. Come ultima riga viene restituito *nil*
Se non viene specificato *filename* viene utilizzato quello di default.
La funzione chiude il file al termine del ciclo.

Funzione `io.open(filename [, mode])`
Apre il file *filname* nella modalità *mode*, restituisce l'identificativo (handler)
In caso di errore di apertura, restiruisce come secondo parametro il codice di errore.
mode puo' essere uno di questi:

- "r" Lettura (default)
- "w" Scrittura
- "a" Accodamento
- "r+" Aggiornamento (i dati vengono sovrascritti ma preservati)
- "w+ Sostituzione (i dati vengono sovrascritti)
- "a+" Aggiornamento e accodamento: i dati vengono scritti alla fine del file

Aggiungendo il carattere *b* il file viene aperto in modalità binaria. Questa funzione non è supportata da tutte le piattaforme.

Funzione `io.output([file]) / io.write()`
Come *io.inut,* ma opera con il file output di default.

Funzione `io.popen(prog [,mode])`
Restituisce il riferimento del programma *prog* eseguito in un processo separato (asincrono) e ne permette la lettura del risultato. *Mode* segue la codifica di *io.open*
Questa funzione non è supportata da tutte le piattaforme

Funzione `io.readfile(file)`
Restituisce *file* in un'unica stringa, *nil* in caso di errore

---

Funzione `io.tmpfile()`
Restituisce l'identificativo di un file temporaneo, il file viene aperto in modo Update e viene cancellato non appena lo script termina

---

Funzione `io.type(obj)`
Verifica sel *obj* è un identificatore valido di un file aperto.

Restituisce *file* se *obj* è aperto, *closed file* se il file è chiuso, *nil* se invalido

---

Funzione `io.writefile(file, data)`
Scrive *data* su *file*, se *data* è una tabella ogni chiave verrà scritta su una riga separata.
Restituisce un valore logico a conferma dell'esecuzione dell'istruzione

---

Funzione `file:close()`
Chiude l'identificativo *file*
NOTA: al termine dello script, indipendentemente se specificato o meno, i files vengono chiusi dal sistema; ma non è possibile stabilire con certezza quando il sistema lo effettuerà, di conseguenza conviene sempre chiudere i file tramite istruzione.

Funzione `file:flush()`
Salva ogni informazione scritta su *file*

Funzione `file:lines()`
All'interno di un ciclo for … do … end apre *file* in modalità di lettura e restituisce una riga alla volta.
Come ultima riga viene restituito nil
La funzione, a differenza di *io.lines(filename)* NON chiude il file al termine del ciclo

---

Funzione `file:read(...)`
Legge il file secondo le direttive indicate
Quando non specificate, legge l'intera riga

"*n"	Legge i numeri >> restituisce solo numero e non stringa
"*a"	Legge l'intero file partendo dalla posizione corrente, alla fine restituisce *nil*
"*l"	Legge una singola riga, esclude il carattere di fine riga (CRLF)
numero	Legge il numero di caratteri specificato partendo dalla posizione corrente

---

Funzione *`file:seek([whence] [, offset])`*

Restituisce la posizione (espressa in byte) partendo dalla posizione *whence* e aggiungendo l'*offset* in byte

whence può avere i seguenti valori

- "set" parte dalla posizione 0 (inizio del file)
- "cur" posizione corrente
- "end" si posiziona alla fine del file

La chiamata *file:seek()* restituisce la posizione corrente del puntatore mentre *file:seek("end")* ne restituisce la dimensione in bytes

NOTA: nella gestione dei files 1 byte = 1 carattere, di conseguenza il conteggio puo' essere assimilato al trattamento dei puntatori che abbiamo già visto nel trattamento delle stringhe

---

Funzione *`file:setvbuf(mode [, size])`*

Imposta la dimensione del buffer per i files di putput.

Mode puo' essere un valore fra

- "no" nessun Buffer, i dati vengono scritti immediatamente
- "full" Buffering completo, i dati vengono scritti alla chiusura o tramite un flush
- "line" Buffer a livello di riga, la scrittura avviene ad ogni nuova riga

In caso di *full* e *line* è possibile specificare una dimensione (*size*) del buffer in bytes (caratteri)

CONSIDERAZIONE: E' buona norma ottimizzare le scritture su disco, specialmente quando queste avvengono su dispositivi a limitata velocità (dischi meccanici) o limitato numero di cicli di lettura/scrittura (SD non industriali)

Funzione *`file:write(...)`*

Scrive la stringa fornita nel *file*

La stringa puo' essere formattata secondo le regole delle funzione *tostring* o *string.format*

SUPPORTO SISTEMA OPERATIVO

LogicMachine è basato su kernel Linux e alcune funzioni sono disponibili anche in ambiente di scripting

Funzione ***os.date([format [, time]])***
Restituisce una stringa o una tabella compilata secondo le direttive di *format*
Se non viene indicato *time* (vedi os.time per la formattazione) si restituirà la data attuale

Format supporta i seguenti parametri:

!	prefisso – Restituisce l'orario UTC (Greenwitch)
*t	Restituisce una tabella con i seguenti campi (non applicabile ad altre impostazioni)
	year *(4 numeri)*
	month *(1-12)*
	day *(1-31)*
	hour *(0-23)*
	min *(0-59)*
	sec *(0-59)*
	wday *(1-7 – Domenica = 1)*
	yday *(giorno giuliano)*
	isdst *logico – flag Ora legale (true = attivo)*

%a	Nome della settimana abbreviato nelle impostazioni locali
%A	Nome completo del giorno della settimana nelle impostazioni locali
%b	Nome mese abbreviato nelle impostazioni locali
%B	Nome mese completo nelle impostazioni locali
%c	Rappresentazione di data e ora appropriata per le impostazioni locali
%C	L'anno diviso da 100 e troncato a un intero, come numero decimale (00-99)
%d	Giorno del mese come numero decimale (01 - 31)
%D	Equivalente a %m/%d/%y
%e	Giorno del mese come numero decimale (1 - 31), dove le singole cifre sono precedute da uno spazio
%F	Equivalente a %Y-%m-%d
%g	Le ultime 2 cifre dell'anno (00 - 99)
%G	Anno basato su settimana ISO 8601 come numero decimale
%h	Nome mese abbreviato (equivalente a %b)
%H	Ora in formato 24 ore (00 - 23)
%I	Ora in formato 12 ore (01 - 12)
%j	Giorno dell'anno come numero decimale (001 - 366)
%m	Mese come numero decimale (01 - 12)
%M	Minuto come numero decimale (00 - 59)
%n	Carattere di nuova riga (\n)
%p	Indicatore delle impostazioni locali A.M./P.M. per 12 ore
%r	Ora dell'ora di 12 ore delle impostazioni locali
%R	Equivalente a %H:%M
%S	Secondo come numero decimale (00 - 59)
%t	Carattere di scheda orizzontale (\t)
%T	Equivalente a %H:%M:%S, il formato di ora ISO 8601
%u	ISO 8601 giorno lavorativo come numero decimale (1 - 7; Lunedì è 1)

%U	Numero di settimana dell'anno come numero decimale (00 - 53), dove la prima domenica è il primo giorno della settimana 1
%V	Numero di settimana ISO 8601 come numero decimale (00 - 53)
%w	Giorno lavorativo come numero decimale (0 - 6; Domenica è 0)
%W	Numero di settimana dell'anno come numero decimale (00 - 53), dove il primo lunedì è il primo giorno della settimana 1
%x	Rappresentazione data per le impostazioni locali
%X	Rappresentazione temporale per le impostazioni locali
%y	Anno senza secolo, come numero decimale (00 - 99)
%Y	Anno con il secolo, come numero decimale
%z	Offset da UTC in formato ISO 8601; nessun carattere se il fuso orario è sconosciuto
%Z	Nome del fuso orario delle impostazioni locali o abbreviazione del fuso orario, a seconda delle impostazioni del Registro di sistema; nessun carattere se il fuso orario è sconosciuto
%%	Segno di percentuale

Funzione `os.difftime( t2, t1 )`
Restituisce il numero di secondi di differenzafra *t1* e *t2* espresso in notazione POSIZ (YYYY-MM-DD HH:mm:ss)

Funzione `os.execute( [command] )`
Esegue il comando *command* e ne restituisce lo status code di sistema quando è terminato.

Se *command* non viene indicato, la funzione restituisce la disponibilità della shell (area di esecuzione comandi del sistema operativo)

Funzione `os.exit([code])`
Forza la chiusura del processo in corso con la possibilità di forzare il codice di uscita *code*

Quando non indicato, viene usato il codice di operazione conclusa con successo (0)

Funzione `os.getenv(varname)`
Restituisce il valore della variabile di ambiente *varname* o *nil* se non è disponibile

---

Funzione `os.remove(filename)`
Cancella fisicamente il file o la directory *filename* (le directory devono essere vuote per essere cancellate). Restituisce *nil* più un messaggio di errore in caso di fallimento del comando

---

Funzione `os.rename(oldname, newname)`
Rinomina file o directory specificato in *oldname* in *newname*. Restituisce *nil* più un messaggio di errore in caso di fallimento del comando

---

Funzione `os.time([table])`
Senza parametri restituisce il numero di secondi trascorsi dal 01-01-1970 ad ora
Se fornito di tabella (i cui campi sono quelli indicati nell'opzione **t* di *os.date*) restituirà la differenza fra 01-01-1970 e la data specificata in tabella

---

Funzione `os.tmpname()`
Restituisce una stringa con il nome di un file che puo' essere impiegato come file temporaneo.
A differenza di *io.tmpfile* in questo caso il file non viene né aperto, né verrà chiuso

---

LOGICMACHINE – FUNZIONI DI BASE PROPRIETARIE

Queste funzioni sono state disegnate specificatamente per l'ambiente di scripting e non fanno parte delle normali librerie e moduli di LUA

Funzione ***toboolean(value)***
Converte *value* in logico secondo la regola

False	*nil*, *false*, 0, vuoto, '0'
True	tutto il resto

Funzione ***sleep(delay)***
Ritarda l'esecuzione dell'istruzione successiva di *delay* secondi

---

Funzione ***alert(fmt, ...)***
Sfruttando le regole di *string.format* aggiunge una indicazione all'alert system di Logicmachine
Questa istruzione agisce direttamente con l'App ALERT MANAGER e la sezione Allarmi di LM

Funzione ***log(...)***
Aggiunge una riga di log con un numero di variabili separato da ,

---

OPERATORI LOGICI E BITSHIFT

Funzione ***bit.nbot(value)***
Gate logico NOT

Funzione ***bit.band(x1 [, x2 ...])***
Gate Logico AND fra tutti i parametri passati

Funzione ***bit.bor(x1 [, x2 ...])***
Gate Logico OR fra tutti i parametri passati

Funzione ***bit.xor(x1 [, x2 ...])***
Gate Logico XOR fra tutti i parametri passati

Funzione ***bit.lshift(value, shift)***
Bitshift a sinistra

Funzione ***bit.rshift(value, shift)***
Bitshift a destra

CONVERSIONI

Funzione *`cnv.strtohex(str)`*
Converte la stringa binaria fornita in una stringa esadecimale

---

Funzione *`cnv.hextostr(hex [, keepnulls])`*
Converte un numero esadecimale in una stringa binaria binaria, se *keepnulls* è *true* manterrà i caratteri nulli

---

Funzione *`cnv.tonumber(value)`*
Converte in numero secondo la direttiva:

true	1
false	0
numeri	trattati nella stessa maniera (identificatore decimale è . e non ,)
tutto il resto	*nil*

DATATYPE
La tabella *dt* include i datatype disponibili su LM, nonostante non includano tutti i dtp disponibili su bus

dt.bool	logico	1 bit logico (0/1)
dt.bit2	numero	2 bit (1 bit di controllo)
dt.bit4	numero	4 bit (3 bit di controllo)
dt.char	testo	1 Byte carattere singolo
dt.uint8	numero	1 Byte senza segno
dt.scale	numero	1 Byte segna segno
dt.int8	numero	1 Byte con segno
dt.uint16	numero	2 Bytes senza segno
dt.int16	numero	2 Bytes con segno
dt.float16	numero	2 Bytes a virgola mobile
dt.time	tabella:	3 Bytes day (0-7) hour (0-23) minute (0-59) second (0-59)
dt.date	tabella:	3 Bytes day (1-31) month (1-12) yesr (1990-2089)
dt.uint32	numero	4 Bytes senza segno
dt.int32	numero	4 Byte con segno
dt.float32	numero	4 Byte a virgola mobile
dt.access	numero	4 Byte a virgola mobile – non ancora supportato
dt.string	testo,	14 Bytes – caratternu NULL scartati durante la decodifica

INTERAZIONE CON OGGETTI BUS

Funzione ***grp.getvalue(alias)***
Restituisce il valore di *alias* come indirizzo dell'oggetto e nome

---

Funzione ***grp.find(alias)***
Restituisce un singolo oggetto (tabella) per il gruppo *alias* indicato come nome o indirizzo di gruppo
Restituisce *nil* in caso non venga trovato alcun gruppo, diversamente la tabella conteiene:

address	Indirizzo di gruppo
updatetime	ultimo aggiornamento (usare os.date() per formattare in maniera leggibile)

Se è stato stato specificato il DTP dell'oggetto verrà inoltre fornito

Name	Nome degl gruppo
Datatype	tipo di dtp
Decoded	logico – *true* quando è disponibile un valore convertito
Value	il valore attuale del gruppo

NOTA: la funzione restituisce una copia dell'oggetto, di conseguenza il valore riportato è quello al momento dell'esecuzione dell'istruzione. Se l'oggetto cambia valore durante l'esecuzione dello script, questa funzione non ne tiene traccia

---

Funzione ***grp.tag(tag [, mode])***
Restituisce una tabella contente tutti gli oggetti che appartengono ai *tag* indicati (tabella)
in caso di tag multipli, *mode* puo' indicare *all* se si vogliono selezionare tutti i tag, oppure *any* per selezionare almeno un tag

---

Funzione ***grp.alias(alias)***
Specificando *alias* come indirizzo di gruppo viene restituito il nome del gruppo e viveversa

---

Funzione ***grp.create(config)***
Crea un nuovo oggettosecondo le specifiche della tabella *config* passata.
La tabella deve avere i seguenti campi

obbligatori

datatype	DTP secondo la tabella *dt*

facoltativi

name	Nome univoco del gruppo, se già esistente verrà aggiunto un suffisso numerico automaticamente
comment	Stringa descrittiva del gruppo
units	Suffisso per unità di misura
addess	indirizzo univoco, se non specificato verrà usato il primo libero a partire dal range indicato nell'impostazione generale "indirizzi di gruppo assegnati automaticamente"
tags	Sottoforma di tabella, applica i tag indicati
enums	sottoforma di tabella [valore] = [descrizione valore] è possibile personalizzare la visualizzazione del valore

---

Funzione *grp.write(alias, value [,datatype])*
Scrive *value* nel gruppo *alias*, secondo il *datatype* specificato (diversamente viene utilizzato quello configurato nell'oggetto
Restituisce un valore Logico a risultato dell'operazione
NOTA: la scrittura genera un evento bus ad eccezione dei gruppi virtuali per i quali non viene generato alcun evento bus

---

Funzione *grp.response(alias, value [, datatype])*
Risponde ad un evento di lettura *value* nel gruppo *alias*, secondo il *datatype* specificato (diversamente viene utilizzato quello configurato nell'oggetto)
NOTA: LM non risponde MAI a eventi bus di lettura, occorre forzare tramite lo scripting-evento una eventuale risposta

---

Funzione *grp.read(alias)*
Invia una richiesta di lettura per il gruppo *alias*.
NOTA: questa istruzione si limita alla generazione dell'evento e non attende alcuna risposta

---

Funzione *grp.update(alias, value [, datatype])*
Scrive *value* nel gruppo *alias*, secondo il *datatype* specificato (diversamente viene utilizzato quello configurato nell'oggetto
Restituisce un valore Logico a risultato dell'operazione
NOTA: a differenza di grp.write() questa funzione NON genera alcun evento bus

---

Per gli oggetti/tabella restituiti da grp.tag() e grp.find() sono disponibili le seguenti funzioni

Funzione *oggetto:write(value [, datatype])*
Scrive *value* su tutti gli oggetti inclusi usando *datatype*, se non specificato verrà assunto il valore configurato per l'oggetto

---

Funzione *oggetto:response(value [, datatype])*
Risponde *value* su tutti gli oggetti inclusi usando *datatype*, se non specificato verrà assunto il valore configurato per l'oggetto

---

Funzione *oggetto:read()*
Invia una richiesta di lettura per tutti gli oggetti

Funzione *oggetto:update(value [, datatype])*
Scrive *value* su tutti gli oggetti inclusi usando *datatype*, se non specificato verrà assunto il valore configurato per l'oggetto
Non genera nessun evento bus

FUNZIONI ORARIE

Funzione *os.microtime ()*
Restituisce 2 valori in secondi e nanosecondi dell'orario corrente (partendo a contare dal 00:00)

Funzione *os.udifftime(sec, usec)*
Restituisce come numero decimale la differenza di orario da adesso e i secondi (*sec*)e frazione di secondo *usec*

Funzione os.*sleep(delay)*
Ritarda l'esecuzione dell'istruzione successiva di *delay* secondi

---

SCRIPT STORAGE

Gli storage rappresentano un metodo per salvare valori ed eventualmente passarli fra uno script e l'altro.

A differenza dei files (liberira Input/output) questi vengono salvati in una porzione del sistema a parte e hanno il solo scopo di salvare valori interpretabili da LUA:

Logici
Numeri
Stringhe/testi
tabelle

Funzione *storage.set(key, value)*
Scrive *value* per la chiave *key*
Restituisce un valore logico a risultato dell'operazione

Funzione *storage.get(key [, default])*
Legge la chiave *key*
E' possibile specifcare un valore di *default* qualore *key* non fosse disponibile/non esiste

---

FUNZIONI STRINGHE

Funzione *string.trim(str)*
Restituisce la stringa *str* eliminando gli spazi all'inizio e alla fine del testo

---

Funzione *string.split(str, sep)*
Restituisce una tabella composta da *str* separata secondo il separatore *sep*

SUPPORTO JSON

La libreria JSON non è immediatamente disponibile in ogni script e va invocata mediante un

```
require('json')
```

NOTA: qualora l'impiego della libreria sia persistente in tutti gli script indipendentemente dalla tipologia, è possibile includere la libreria json all'interno delle funzioni comuni per renderla disponibile senza dichiararla

Funzione *json.encode(value)*

Converte la tabella *value* in JSON. Interrompe l'esecuzione dello script in caso di errore

---

Funzione ***`json.pencode(value)`***
Converte la tabella *value* in JSON. Restituisce *nil* in caso di errore
NON interrompe l'esecuzione dello script

Funzione ***`json.decode(value)`***
Converte la stringa JSON *value* in una tabella LUA. Interrompe l'esecuzione dello script in caso di errore

---

Funzione ***`json.pdecode(value)`***
Converte la stringa JSON *value* in una tabella LUA. Restituisce *nil* in caso di errore
NON interrompe l'esecuzione dello script
