Manuale xxter
Lua scripts

Versione 1.8 – Agosto 2026

Introduzione agli script xxter

Con gli script xxter è possibile creare i tuoi piccoli programmi all'interno di xxter. Questi script sono molto flessibili e possono essere utilizzati per aggiungere un'ampia varietà di funzionalità a un'installazione di automazione domestica o di un edificio. È possibile creare funzioni logiche, ritardare azioni, estendere i tuoi scenari con sequenze RGB e molto altro. Gli script possono essere attivati utilizzando uno scenario, una pianificazione, un'azione (trigger) o un altro script.

xxter supporta sia un linguaggio di scripting nativo che il supporto per lo scripting Lua. Questo manuale fornisce una spiegazione sugli script Lua. Per gli script nativi xxter, consultare la pagina della documentazione del nostro sito Web (https://www.xxter.com/documentation/). Abbiamo anche diversi esempi disponibili su questa pagina e anche sul nostro forum (https://forum.xxter.com).

Per saperne di più su Lua in generale, consultare il manuale Lua:
https://www.Lua.org/manual/5.3/

Elementi base

Gli script xxter Lua sono programmi scritti che consistono in una o più righe. È possibile aggiungere commenti a uno script scrivendo "--". Questo può essere fatto su una riga separata o alla fine di un comando.

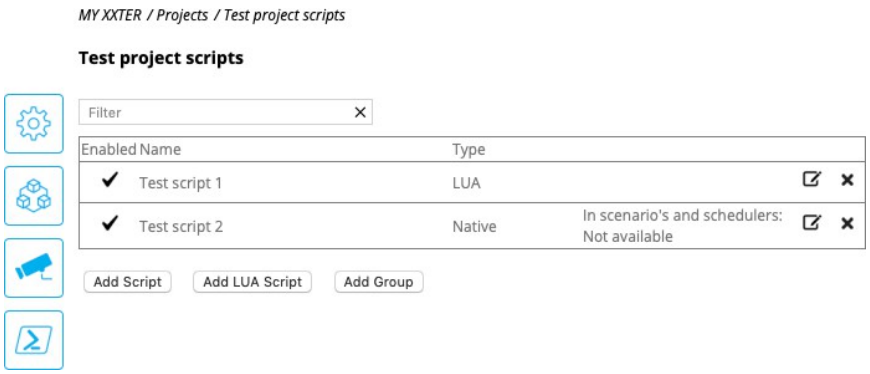
Un esempio di uno script xxter Lua:

```
1 -- this is an example test script.
2 xxter.userlog("starting test script")
3 -- Setting dimmer to 50%
4 xxter.setcomponent(16, 50)
5 -- Waiting for 5 seconds
6 xxter.sleep(5000)
7 -- Setting switch on
8 xxter.setcomponent(56, 1)
9
```

Gli script possono essere avviati e arrestati con molti mezzi, ad esempio con uno scenario o una pianificazione. Gli script possono essere script "unending" che eseguono determinate azioni a intervalli ripetuti o possono essere definiti come una sequenza di comandi che vengono eseguiti una sola volta, ogni volta che lo script viene avviato. Quando si utilizzano script "unending" in loop continuo, ricordarsi di includere sempre un periodo di "sospensione" per evitare che lo script crei un carico molto pesante sul controller xxter.

Gestione script

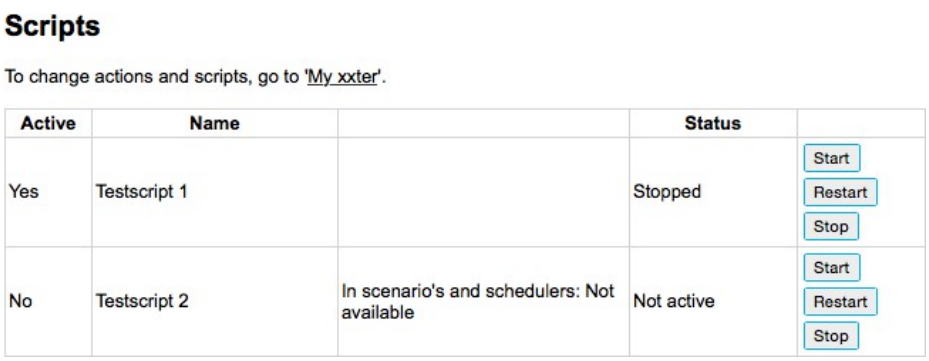
Gli script xxter possono essere creati e aggiornati utilizzando l'editor xxter. Questo editor può essere trovato nella configurazione del progetto di "My xxter". Selezionare il progetto per il quale si desidera gestire gli script e selezionare "Script" nella barra dei menu. Tutti gli script esistenti verranno mostrati qui e possono essere modificati ed eliminati. Quando si crea un nuovo script, è possibile scegliere se si desidera creare uno script xxter nativo o uno script Lua.



Importante: Quando si aggiunge o si modifica uno script, questo deve essere caricato sul controller xxter prima di poter essere utilizzato.

Ogni volta che è stata apportata una modifica, è possibile visualizzare il testo "Project has changed" nell'angolo in alto a destra. Quando si fa clic su questo e successivamente su "Push configuration to xxter device", il controller scaricherà la nuova configurazione. Questa funzione funziona meglio quando xxter è configurato per l'uso all'esterno (vedere il manuale di installazione). In alternativa è possibile ricaricare il profilo tramite l'app (vedi manuale utente). Naturalmente, è anche possibile ricaricare il progetto dal controller xxter stesso, accedendo al controller xxter e facendo clic sul pulsante "Load configuration" nell'angolo in alto a sinistra del menu.

Quando si è connessi al controller xxter, è possibile visualizzare gli script caricati facendo clic sull'opzione "Script" nel menu. Qui uno script può anche essere avviato, riavviato o arrestato manualmente a scopo di test.

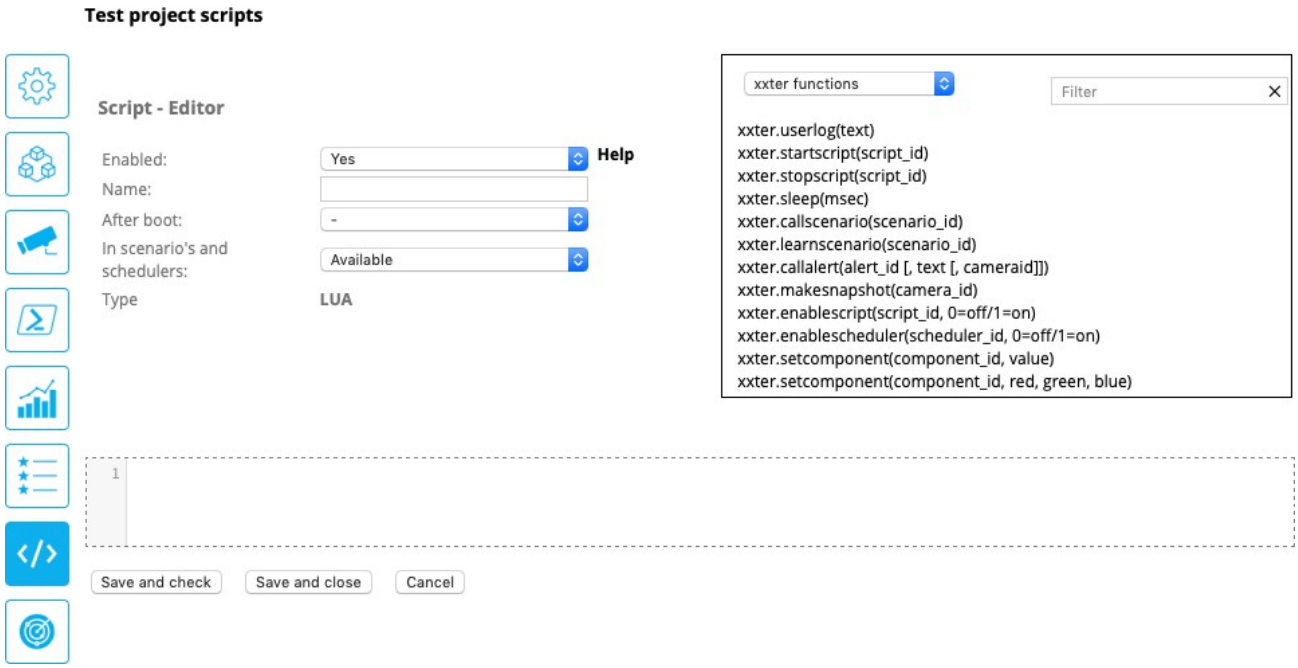


Per la risoluzione dei problemi degli script, i registri del dispositivo sono molto utili. Sul controller xxter, è possibile abilitare "Script" in Accesso utente nella pagina *Settings – basic* . Questo registrerà ogni riga eseguita di ogni script nel registro del dispositivo. È possibile accedere al registro tramite "Open user log window" nella pagina *Status* .

Editor di script

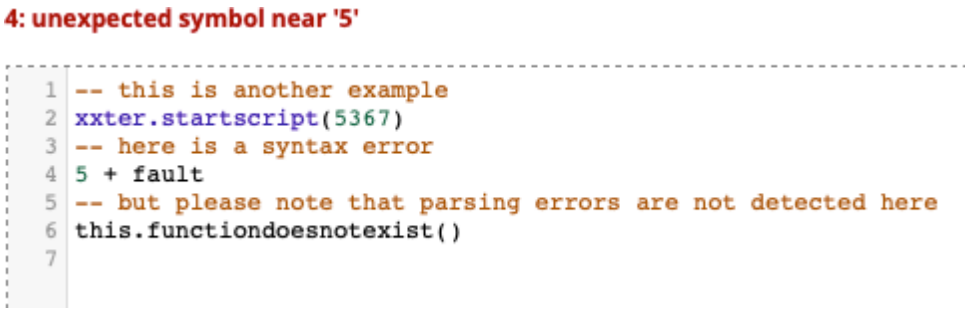
Quando si aggiunge o si modifica uno script, è possibile selezionare se deve essere abilitato e se lo script deve essere disponibile per l'utente finale, da utilizzare negli scenari o nello scheduler. È anche possibile selezionare se lo script deve essere avviato automaticamente, ogni volta che il controller xxter viene (ri) avviato.

IMPORTANTE: Gli script disabilitati non possono essere eseguiti e non verranno triggerati quando inclusi in uno scenario, un'azione o uno scheduler. Questo può essere utile per le proposte di test e risoluzione dei problemi. Tuttavia, tieni presente che uno script può essere abilitato o disabilitato da un altro script!

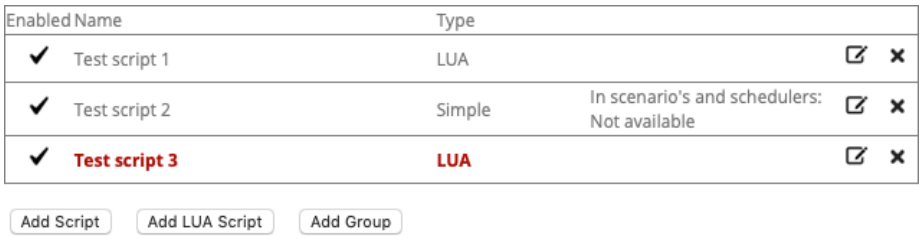


I numeri di riga visualizzati accanto allo script nell'editor sono solo informativi e non vengono utilizzati negli script stessi. I comandi possono essere aggiunti digitandoli direttamente o utilizzando lo strumento di selezione dei comandi, situato nell'angolo in alto a destra. Assicurati di essere nella posizione giusta nell'editor di script quando aggiungi un comando con lo strumento comandi. Quando si aggiunge un comando con lo strumento comandi, le variabili devono essere impostate sui valori appropriati. Le funzioni forniranno informazioni sul tipo di valori da aggiungere. Gli ID dei componenti possono essere selezionati dallo strumento di selezione, selezionando il tipo di valore necessario (ad esempio "script", "integer" o “floating points”). Facendo clic su un componente, questo verrà aggiunto allo script nella posizione del cursore.

Dopo aver aggiunto uno o più comandi nell'editor, è possibile verificare se sono validi facendo clic sul pulsante "Save and check". Lo script corrente verrà controllato e visualizzato nuovamente nell'editor. Se si verifica un errore di sintassi nello script, verrà visualizzato un messaggio sopra la schermata di modifica. Tuttavia, gli errori di analisi (ad esempio, la convalida dell'esistenza di funzioni o della giusta quantità e tipo di variabili) non possono essere rilevati nell'editor.



Dopo aver corretto o aggiunto righe, puoi facilmente verificare nuovamente lo script utilizzando il pulsante "Save and check". Quando tutto è corretto è possibile utilizzare il pulsante "Save and close" per chiudere l'editor.
Gli script che vengono salvati ma contengono un errore sono mostrati in rosso.



Nota: gli script che contengono errori possono essere salvati ma non avviati.

Gli script si interromperanno automaticamente dopo l'esecuzione dell'ultimo comando dello script.

Gli script sono scritti utilizzando funzioni e valori. Inoltre, gli script possono essere estesi utilizzando variabili e strutture di controllo come istruzioni IF e loop WHILE. È anche possibile definire le proprie funzioni, con parametri di input e valori di ritorno.

Per saperne di più su Lua, consultare il manuale Lua: https://www.Lua.org/manual/5.3/

Funzioni xxter

Sono disponibili le seguenti funzioni specifiche xxter Lua.

---

xxter.userlog(text)

Descrizione:
Scrive nel registro utente, utile per il debug degli script Lua. La scrittura sul registro utente in questo modo avverrà sempre, anche quando la registrazione degli script è stata disabilitata nella pagina *Settings – basic* del dispositivo, vedere pagina 3 di questo manuale.

Parametri:
text : una stringa con il testo da scrivere nel log utente.

Ritorni: -

Esempio:
```
xxter.userlog("Running script part X")
```

---

xxter.startscript(script_id)

Descrizione:
Avvia uno script xxter, se non è già in esecuzione.

Parametri:
script_id: L'identificativo dello script (numero)

Ritorni: -

Esempio:
```
xxter.startscript(35)
```

---

xxter.stopscript(script_id)

Descrizione:
Interrompe uno script xxter.

Parametri:
script_id: L'identificativo dello script (numero)

Ritorni: -

Esempio:
```
xxter.stopscript(35)
```

---

xxter.enablescript(script_id, status)

Descrizione:
Abilita o disabilita uno script xxter, quindi può (non) essere avviato.

Parametri:
script_id: L'identificativo dello script (numero)
status: Stato dello script desiderato (Enum: 0 = off, 1 = on)

Ritorni: -

Esempio:
```
xxter.enablescript(35,1)
```

---

xxter.include(script_id)

Descrizione:
Include tutto il codice dello script designato nello script corrente. Questo, ad esempio, consente di definire un insieme di funzioni in uno script e utilizzare lo stesso codice in più script.

Parametri:
script_id: L'identificativo dello script (numero)

Ritorni: -

Esempio:
```
xxter.include(17)
```

---

xxter.sleep(time)

Descrizione:
Attende il tempo indicato (in millisecondi), prima di continuare.

Parametri:
time: Il periodo di tempo in millisecondi di attesa (numero)

Ritorni: -

Esempio:
```
xxter.sleep(5000)
```

---

xxter.callscenario(scenario_id)

Descrizione:
Richiama lo scenario xxter specificato per eseguire le azioni configurate.

Parametri:
scenario_id: L'identificativo dello scenario (numero)

Ritorni: -

Esempio:
```
xxter.callscenario(17)
```

xxter.learnscenario(scenario_id)

Descrizione:
Aggiorna le azioni configurate dello scenario xxter specificato ai loro valori attuali

Parametri:
scenario_id: L'identificativo dello scenario (numero)

Ritorni: -

Esempio:
```
xxter.learnscenario(17)
```

---

xxter.callalert(alert_id[, text [,camera_id]])

Descrizione:
Richiamare l'avviso xxter specificato, facoltativamente con testo specifico e istantanea della telecamera. Il testo (opzionale) fornito viene posizionato nella posizione "[x]" del servizio di avviso, se disponibile.

Parametri:
alert_id: L'identificativo della segnalazione (numero)
text: *Optional,* il testo da inviare con l'avviso (stringa)
camera_id: *Optional,* l'identificativo della telecamera per scattare un'istantanea (numero)

Ritorni: -

Esempi:
```
xxter.callalert(65)
xxter.callalert(65, "Scripted alert")
xxter.callalert(65, "Scripted alert", 103)
```

---

xxter.makesnapshot(camera_id)

Descrizione:
Realizza un'istantanea della telecamera fornita.

Parametri:
camera_id: L'identificativo della telecamera per scattare un'istantanea (numero)

Ritorni: -

Esempi:
```
xxter.makesnapshot(103)
```

---

xxter.enablescheduler(schedule_id, status)

Descrizione:
Abilita o disabilita uno scheduler xxter, in modo che possa (non) essere avviato.

Parametri:
schedule_id: L'identificativo del programma (numero)
status: Stato dello scheduler desiderato (Enum: 0 = off, 1 = on)

Ritorni: -

Esempio:
```
xxter.enablesceduler(18,1)
```

---

xxter.setcomponent(component_id, value [, value_2, value_3])

Descrizione:
Imposta un componente sul valore fornito.

Parametri:
component_id: L'identificativo del componente (numero)
value: Valore desiderato del componente (il tipo dipende dal componente).
Nel caso in cui il componente sia RGB, questo valore si riferisce a R (numero)
value_2: *Only for RGB,* si riferisce a G (numero)
value_3: *Only for RGB,* si riferisce a B (numero)

Ritorni: -

Esempi: `xxter.setcomponent(38,75)`
```
xxter.setcomponent(45,75,230,176)
```

---

xxter.getcomponent(component_id)

Descrizione:
Ottiene il valore corrente di un componente.

Parametri:
component_id: L'identificativo del componente (numero)

Ritorni:
value: Valore corrente del componente (il tipo dipende dal componente).
Componenti RGB, restituiscono tre valori

Esempio:
```
var = xxter.getcomponent(38)
varR, varG, varB = xxter.getcomponent(45)
```

---

xxter.readcomponent(component_id)

Descrizione:
Esegue un comando di lettura sul bus KNX del componente. Se il componente KNX risponde alla richiesta, xxter aggiornerà il suo stato attuale (ciò potrebbe richiedere del tempo). Successivamente, il valore corrente può essere utilizzato nello script con *xxter.getcomponent*).

Parametri:
component_id: L'identificativo del componente (numero)

Ritorni: -

Esempio:
```
xxter.readcomponent(38)
```

---

xxter.httpcommand(http_id [, result])

Descrizione:
Esegue il comando http fornito.

Parametri:
http_id: L'identificativo del comando http (numero)
result: Parametro opzionale per richiedere anche il risultato del comando (booleano)

Ritorni:
result: Se richiesto e disponibile, il risultato restituito dal comando HTTP (stringa)

Esempio:
```
xxter.httpcommand(12)
var = xxter.httpcommand(13,true)
```

---

xxter.tcpcommand(tcp_id)

Descrizione:
Esegue il comando TCP fornito.

Parametri:
tcp_id: L'identificativo del comando tcp (numero)

Ritorni: -

Esempio:
```
xxter.tcpcommand(17)
```

---

xxter.ircommand(ir_id)

Descrizione:
Esegue il comando infrarosso fornito.

Parametri:
ir_id: L'identificativo del comando IR (numero)

Ritorni: -

Esempio:
```
xxter.ircommand(7)
```

---

xxter.openKNXtunnel(status)

Descrizione:
Apre o chiude il tunnel KNX.

Parametri:
status: Comando per aprire o chiudere il tunnel (enum; 0=close, 1=open)

Ritorni: -

Esempio:
```
xxter.openKNXtunnel(1)
```

---

xxter.setsimulation(status)

Descrizione:
Imposta la simulazione di presenza allo stato desiderato.

Parametri:
status: Stato desiderato della simulazione di presenza (enum; 0=stop, 1=record, 2=play)

Ritorni: -

Esempio:
```
xxter.setsimulation(2)
```

---

xxter.getsimulation()

Descrizione:
Ottiene lo stato corrente della simulazione di

presenza. Parametri: -

Ritorni:
Stato attuale della simulazione di presenza (enum; 0=stop, 1=record, 2=play)

Esempio:
```
var = xxter.getsimulation()
```

---

xxter.connectKNX(status)

Descrizione:
Collegare o scollegare la connessione KNX del dispositivo.

Parametri:
status: Comando per la connessione KNX (enum; 0=disconnect, 1=connect)

Ritorni: -

Esempio:
```
xxter.connectKNX(1)
```

---

xxter.setpersistent(varname, value)

Descrizione:
Crea o aggiorna una variabile persistente su tutti gli script, per passare informazioni tra gli script.

Parametri:
varname: Nome della variabile persistente (stringa, alfanumerico: 0-9a-zA-Z)
value: Valore che la variabile persistente dovrebbe avere (tipo a seconda della variabile)

Ritorni: -

Esempi:
```
xxter.setpersistent("maxValue",21.23)
xxter.setpersistent("message","Valve error")
```

---

xxter.getpersistent(varname)

Descrizione:
Recuperare una variabile persistente su tutti gli script, per passare informazioni tra gli script.

Parametri:
varname: Nome della variabile persistente (stringa, alfanumerico: 0-9a-zA-Z)

Ritorni:
Valore della variabile persistente (tipo a seconda della variabile)

Esempi:
```
var = xxter.getpersistent("maxValue")
```

---

xxter.getSunAzimuth()

Descrizione:
Ottenere l'angolo del sole corrente, in relazione al nord (direzione), in base alla posizione configurata sul controller xxter.

Parametri: - Ritorni:
Valore della direzione del sole.

Esempi:
```
var = xxter.getSunAzimuth()
```

---

xxter.getSunAltitude()

Descrizione:
Ottenere l'angolo di sole corrente, in relazione all'orizzonte (altitudine), in base alla posizione configurata sul controller xxter.

Parametri: - Ritorni:
Valore dell'altitudine del sole.

Esempi:
```
var = xxter.getSunAltitude()
```

---

xxter.setaudio(device_id, command [, option_setting])

Descrizione:
Esegue un comando Sonos/BluOS

Parametri:
device_id : Può essere impostato su "global" per i comandi globali o per fornire un identificatore di dispositivo audio specifico (formato: RINCON_949F3EC192E401400).
command: quando device_id è impostato su "global" è possibile utilizzare i seguenti comandi: "creategroup": richiede option_setting per groupid (numero) "muteall"
“unmuteall”
“pauseall”
“playall”

quando il device_id di un dispositivo audio è impostato, è possibile utilizzare i seguenti comandi: "groupvolume": richiede option_setting per il volume (numero)
"groupmute"
"volume": richiede option_setting per il volume (numero)
"mute"
“unmute”
“play”
“pause”
"seek": richiede option_setting per la posizione (numero)
“next”
“previous”
"selectplaylist": richiede option_setting per il nome della playlist (string)
"selectfavorite" : richiede option_setting per il nome del preferito (string)
"selectnextfavorite"
“selectlinein”
“selectHDMI”

"playaudioclip": richiede due option_setting, uno per il tipo di clip audio (enum) e uno per il numero di clip (numero). I tipi di clip audio sono:
0: Standard Sonos/BluOS
1: xxter standard
2: custom

optional_setting : a seconda del comando utilizzato (vedi parametri del comando sopra)

Ritorni: -

Esempi:
```
xxter.setaudio("global", "creategroup", 3)
xxter.setaudio("global", "muteall")
xxter.setaudio("RINCON_949F3EC192E401400", "selectfavorite", "Radio 4")
xxter.setaudio("RINCON_949F3EC192E401400", "playaudioclip", 2, 4)
```

Gli esempi 1 e 2 sono comandi globali, per creare un gruppo di altoparlanti o per disattivare tutti gli altoparlanti. Gli esempi 3 e 4 sono per un altoparlante, per riprodurre "Radio 4" o riprodurre una clip audio personalizzata.

---

xxter.getaudio(device_id, parameter)

Descrizione:
Richiedi lo stato corrente di un dispositivo Sonos/BluOS

Parametri:
device_id : Identificatore specifico del dispositivo audio (formato: RINCON_949F3EC192E401400) parameter : Il parametro che deve essere restituito (selezionare una delle seguenti opzioni:
"groupvolume", "volume", "status", "mute", "groupmute", "meta")

Ritorni:
groupvolume : volume corrente del gruppo del dispositivo (numero)
volume : volume corrente del dispositivo audio (numero)
status : stato attuale del dispositivo audio (enum: 0:IDLE, 1:BUFFER, 2:PLAY, 3:PAUSED)
(group)mute : corrente (gruppo) stato muto (1 se disattivato, 0 in caso contrario)
a, b, c, d, e, f, g : metadati del titolo in riproduzione corrente (più stringhe)
A = track_name
B = track_artist
C = album_name
D = album_artist
E = show_name
F = container_name
G = stream_info

Esempi:
```
groupvol = xxter.getaudio("RINCON_949F3EC192E401400", "groupvolume")
a, b, c, d, e, f, g = xxter.getaudio("RINCON_949F3EC192E401400", "meta")
```

L'esempio 1 richiede il volume del gruppo, l'esempio 2 i metadati della traccia corrente.

---

xxter.setupnp(device_id, command[, option_setting])

Descrizione:
Esegue un comando uPnP

Parametri:
device_id : Identificatore specifico del dispositivo uPnP (formato: RINCON_949F3EC192E401400) : è possibile utilizzare i seguenti comandi:: the following commands can be used:

"volume":	richiede option_setting per il volume
(numero) "mute"	
“unmute”	
“play”	
“pause”	
"seek":	richiede option_setting per la posizione
(numero) "next"	
“previous”	

optional_setting : a seconda del comando utilizzato (vedi parametri del comando sopra)

Ritorni: -

Esempi:
```
xxter.setupnp("RINCON_949F3EC192E401400", "volume", 25)
xxter.setupnp("RINCON_949F3EC192E401400", "mute")
```

L'esempio 1 imposta il volume dell'altoparlante, l'esempio 2 mette l'altoparlante in muto.

---

xxter.getupnp(device_id)

Descrizione:
Richiedi lo stato corrente di un dispositivo uPnP

Parametri:
device_id : Identificatore specifico del dispositivo uPnP (formato: RINCON_949F3EC192E401400) parameter : Il parametro che deve essere restituito (selezionare una delle seguenti opzioni: “volume”, “status”, “meta”)

Ritorni:
volume : volume corrente dello stato (numero) del dispositivo uPnP: status corrente del dispositivo uPnP (stringa)
a, b, c, d, e, f, g : metadati del titolo in riproduzione corrente (più stringhe)

A = track_name
B = track_artist
C = album_name
D = album_artist
E = show_name
F = container_name
G = stream_info

Esempi:
```
volume = xxter.getupnp("RINCON_949F3EC192E401400", "volume")
a, b, c, d, e, f, g = xxter.getupnp("RINCON_949F3EC192E401400", "meta")
```

L'esempio 1 richiede il volume dell'altoparlante, l'esempio 2 i metadati della traccia corrente.

xxter.timezonediff()

Descrizione:
Fornisce la differenza in secondi tra il fuso orario locale e UTC (GMT senza ora legale). Parametri: -

Ritorni:
Differenza di fuso orario con UTC in secondi

Esempio:
```
var = xxter.timezonediff()
```

---

xxter.localtime()

Descrizione:
Fornisce l'ora locale in secondi. Funziona in modo simile aos.clock () ma poi per il fuso orario locale anziché UTC (GMT senza ora legale). La funzione può ad esempio essere utilizzata insieme alla funzione Luaos.date (format, time) come variabile temporale .

Parametri: - Ritorni:
Ora locale in secondi

Esempio:
```
var = xxter.localtime()
```

---

xxter.getmeteostatus()

Descrizione:
Indica la quantità di ore trascorse dall'ultimo aggiornamento delle informazioni meteorologiche. Le informazioni meteorologiche vengono aggiornate automaticamente, ma questa funzione può essere utilizzata per sapere quando è stata eseguita l'ultima volta.

Parametri: - Ritorni:
Numero di ore dall'ultimo aggiornamento meteo

Esempio:
```
var = xxter. getmeteostatus()
```

`xxter.getmeteoinfo(what [, offset [, unit]])`

Descrizione:
Questa funzione restituisce il valore della proprietà richiesta. Nel caso in cui i dati non siano disponibili, viene restituito *false* .

Parametri:
what: Proprietà che deve essere restituita, vedere la tabella 1 di seguito (stringa)
offset: Offset in ore, per quanto in futuro le informazioni dovrebbero essere (numero)
Ad esempio, "4" fornisce le informazioni meteorologiche previste in 4 ore. Il valore predefinito è 0. unit: Unità del valore richiesto (enum; "C"=Celsius, "K"=Kelvin, "F"=Fahrenheit)
Si applica solo alle temperature, il valore predefinito è "C".

Ritorni:
Valore previsto della proprietà fornita nel periodo di tempo fornito

Esempi:
```
var = xxter.getmeteoinfo("temperature", 8, "F")
var = xxter.getmeteoinfo("rain", 24)
```

L'Esempio 1 fornisce la temperatura prevista in gradi Fahrenheit in 8 ore.
L'Esempio 2 fornisce le precipitazioni previste in 24 ore.

Tabella 1: Possibili proprietà per funzioni meteo	
“temperature”	Temperatura (in C, F o K)
"feels like”	Temperatura per percezione umana (in C, F o K)
“pressure”	Pressione atmosferica a livello del mare (in hPa)
“humidity”	Umidità relativa dell'aria (in %)
"dew point”	Temperatura alla quale le gocce d'acqua iniziano a condensare (in C, F o K)
“clouds”	Copertura nuvolosità (in %)
“visibility”	Visibilità media (in metri) **- n.d. con funzioni min/max**
"wind speed”	Velocità del vento (in m/s)
"wind gust”	Raffiche di vento, ove disponibili (in m/s)
"wind degrees”	Direzione del vento, 0 = N, 90 = E, 180 = S, 270 = W **- n.d. con funzioni min/max**
“rain”	Prevista quantità di pioggia (in mm) **- n.d. con funzione min**
“snow”	Quantità prevista di neve (in mm) **- n.d con funzione min**
“precipitation”	Quantità prevista di precipitazioni (in mm) **- n.d. con funzione min**
"precipitation chance”	Probabilità di precipitazioni (pioggia o neve) **- n.d. con funzione min**

xxter.getmeteoinfomin(what, from_offset, to_offset [, unit])

Descrizione:
Questa funzione restituisce il valore minimo della proprietà richiesta nel periodo di tempo fornito. Nel caso in cui i dati non siano disponibili, viene restituito *false* .

Parametri:
what: Proprietà che deve essere restituita, vedere la tabella 1 nella pagina precedente (stringa) from_offset: Tempo in ore, da quando la previsione dovrebbe essere (numero)
to_offset: Tempo in ore, fino a quando la previsione dovrebbe essere (numero)
unit: L'unità del valore richiesto (enum; "C"=Celsius, "K"=Kelvin, "F"=Fahrenheit) si applica solo alle temperature, il valore predefinito è "C".

Ritorni:
Valore minimo previsto della proprietà fornita nel periodo di tempo fornito

Esempi:
```
var = xxter.getmeteoinfomin("temperature", 0, 8, "F")
var = xxter.getmeteoinfomin("clouds", 24, 48)
```

L'Esempio 1 fornisce la temperatura minima in gradi Fahrenheit tra oggi e le prossime 8 ore.
L'Esempio 2 fornisce la copertura nuvolosa minima prevista tra 24 e 48 ore da ora.

---

xxter.getmeteoinfomax(what, from_offset, to_offset [, unit])

Descrizione:
Questa funzione restituisce il valore massimo della proprietà richiesta nel periodo di tempo fornito. Nel caso in cui i dati non siano disponibili, viene restituito *false* .

Parametri:
what: Proprietà che deve essere restituita, vedere la tabella 1 nella pagina precedente (stringa) from_offset: Tempo in ore, da quando la previsione dovrebbe essere (numero)
to_offset: Tempo in ore, fino a quando la previsione dovrebbe essere (numero)
unit: L'unità del valore richiesto (enum; "C"=Celsius, "K"=Kelvin, "F"=Fahrenheit) si applica solo alle temperature, il valore predefinito è "C".

Ritorni:
Valore massimo previsto della proprietà fornita nel periodo di tempo fornito

Esempi:
```
var = xxter.getmeteoinfomax("temperature", 0, 8, "F")
var = xxter.getmeteoinfomax("rain", 24, 48)
```

L'Esempio 1 fornisce la temperatura massima in gradi Fahrenheit tra oggi e le prossime 8 ore.
L'Esempio 2 fornisce la pioggia massima prevista tra 24 e 48 ore da ora.

---

xxter.gettariff(delta_min)

Descrizione:
Ottiene la tariffa effettiva per il momento fornito nel futuro (in minuti), come impostato per il Gestore Smart Energy. Se è configurata una tariffa variabile questo sarà un valore dinamico, altrimenti restituirà la tariffa unica o frazionata applicabile per quel tempo.

Parametri:
delta_min: La quantità di minuti in futuro (valore numerico)

Ritorni:
La tariffa effettiva, come impostata per il Gestore Smart Energy.

Esempio:
```
var = xxter.gettariff(60)
```

---

xxter.getusername()

Descrizione:
Ottiene il nome utente dell'utente che ha avviato la funzione. Quando un utente (locale) triggera lo script, ad es. attraverso un'azione o l'app, il suo nome utente verrà restituito con questa funzione. Se lo script viene triggerato con qualsiasi altro mezzo, ad es. da un valore nell'automazione, verrà restituito l'utente principale.

Parametri: - Ritorni:
Il nome dell'utente che ha avviato la funzione.

Esempio:
```
var = xxter.getusername()
```

---

xxter.fadeto(duration_sec, component_id, value [, value 2, value 3] [, step_msec])

Descrizione:
Effettua il fade del componente dalla sua impostazione corrente al/ai valore/i fornito/i per la durata prevista nel tempo. La funzione tornerà immediatamente e il fade avverrà in background.

Parametri:
duration_sec: La quantità di secondi per effettuare il fade al nuovo valore (numero)
component_id: L'identificativo del componente (numero)
value: Valore desiderato del componente (il tipo dipende dal componente).
Nel caso in cui il componente sia RGB, questo valore si riferisce a R (numero)
value_2: *Solo per RGB,* si riferisce a G (numero)
value_3: *Solo per RGB,* si riferisce a B (numero)
step_msec: *La quantità di millisecondi per ogni step (modifica del valore) nel fade (numero) opzionale. Il valore predefinito è 500, l'input può essere compreso tra 250 e 60000.*
Ritorni: -

Esempio:
`xxter.fadeto(30,312,100)` - *avrà fade comp. 312 al valore 100 in 30 secondi, ogni 500 msec.*
`xxter.fadeto(60,75,135,230,176,250)` - *avrà fade comp. 75 al valore RGB 135,230,176 in 60 secondi ogni 250 msec.*

Funzioni Lua generali supportate

Oltre a queste specifiche funzioni xxter, sono supportate anche le seguenti funzioni Lua standard:

Tabella 2: funzioni Lua standard supportate

```
tonumber (e [, base])
tostring (v)
type (v)
coroutine.create (f)
coroutine.isyieldable ()
coroutine.resume (co [, val1, ···])
coroutine.running ()
coroutine.status (co)
coroutine.wrap (f)
coroutine.yield (···)
string.byte (s [, i [, j]])
string.char (···)
string.dump (function [, strip])
string.find (s, pattern [, init
                        [, plain]])
string.format (formatstring, ···)
string.gmatch (s, pattern)
string.gsub (s, pattern, repl [, n])
string.len (s)
string.lower (s)
string.match (s, pattern [, init])
string.pack (fmt, v1, v2, ···)
string.packsize (fmt)
string.rep (s, n [, sep])
string.reverse (s)
string.sub (s, i [, j])
string.unpack (fmt, s [, pos])
string.upper (s)
utf8.char (···)
utf8.charpattern
utf8.codes (s)
utf8.codepoint (s [, i [, j]])
utf8.len (s [, i [, j]])
utf8.offset (s, n [, i])
os.clock ()
os.date ([format [, time]])
```

```
os.difftime (t2, t1)
os.time ([table])
table.concat (list [, sep [, i [, j]]])
table.insert (list, [pos,] value)
table.move (a1, f, e, t [,a2])
table.pack (···)
table.remove (list [, pos])
table.sort (list [, comp])
table.unpack (list [, i [, j]])
math.abs (x)
math.acos (x)
math.asin (x)
math.atan (y [, x])
math.ceil (x)
math.cos (x)
math.deg (x)
math.exp (x)
math.floor (x)
math.fmod (x, y)
math.huge
math.log (x [, base])
math.max (x, ···)
math.maxnumber
math.min (x, ···)
math.minnumber
math.modf (x)
math.pi
math.rad (x)
math.random ([m [, n]])
math.randomseed (x)
math.sin (x)
math.sqrt (x)
math.tan (x)
math.tonumber (x)
math.type (x)
math.ult (m, n)
```

La documentazione per queste funzioni è disponibile nel manuale Lua:
https://www.Lua.org/manual/5.3/