

DOMOTICA LABS

IKON SERVER

SCRIPTING LUA

VERSIONE SOFTWARE 2.9.2

INDICE

1 INTRODUZIONE.....5
1.1 COSA E' IKON SERVER.....5
1.2 SCOPO DI QUESTO MANUALE.....5
1.3 REQUISITI.....5
2 PANORAMICA.....6
2.1 SCRIPTS.....6
2.2 RUNNERS.....6
2.3 LIBRERIE.....7
2.4 INGRESSI E USCITE.....8
2.5 INCLUSIONE DI SCRIPTS.....9
2.6 ISPEZIONE VARIABILI.....9
2.7 TABLES.....10
2.7.1 NUMERO DI ELEMENTI.....10
2.7.2 CICLO TRA GLI ELEMENTI.....10
3 SCRIPTS.....12
3.1 INRODUZIONE.....12
3.2 LISTA.....12
3.3 EDITOR.....13
3.4 ESEMPIO.....13
4 RUNNERS.....15
4.1 INTRODUZIONE.....15
4.2 NUOVO RUNNER.....15
4.2.1 INGRESSI.....16
4.2.2 USCITE.....17
4.3 LOOP.....17
4.3.1 ESECUZIONE CICLICA.....17
4.3.2 ESECUZIONE AD INTERRUPT.....18
4.4 DEBUGGING.....19
5 LIBRERIA "OBJECTS".....20
5.1 INTRODUZIONE.....20
5.2 INCLUSIONE.....20
5.3 CARICAMENTO DI OGGETTI.....20
5.4 PROPRIETÀ DEGLI OGGETTI.....21
5.5 COMANDO DI OGGETTI.....23
5.6 RICERCA DI OGGETTI.....24
5.7 MODIFICA DI OGGETTI.....25
6 LIBRERIA "SURROUNDING".....26
6.1 INTRODUZIONE.....26
6.2 INCLUSIONE.....26
6.3 INGRESSI E USCITE.....26
6.4 IDENTIFICATIVI.....27
7 LIBRERIA "MESSAGES".....28
7.1 INTRODUZIONE.....28
7.2 INCLUSIONE.....28
7.3 NOTIFICHE A VIDEO.....28
7.4 NOTIFICHE VIA MAIL.....29
8 LIBRERIA "JSON".....30

8.1 INTRODUZIONE....30
8.2 INCLUSIONE....30
8.3 DECODIFICA....30
8.4 CODIFICA....30
8.5 CREDITS....30
9 LIBRERIA "URL"....31
9.1 INTRODUZIONE....31
9.2 INCLUSIONE....31
9.3 GET URL....31
9.4 CODIFICA....31
10 APPENDICI....33
10.1 TIPI FUNZIONALI....33
10.2 TIPI WEB (PHPCLASS)....34

1 INTRODUZIONE

1.1 COSA E' IKON SERVER

IKON SERVER è un webserver di supervisione per impianti di home & building automation basati su tecnologia KNX, sviluppato e commercializzato da DOMOTICA LABS. Sul sito www.domoticalabs.com è possibile scaricare gratuitamente schede tecniche, brochure e manuali per l'installazione, la configurazione e l'utilizzo di IKON SERVER.

1.2 SCOPO DI QUESTO MANUALE

Questo manuale illustra le funzionalità offerte dal motore di scripting personalizzato di IKON SERVER, mediante il quale è possibile estendere le funzioni base del webserver; attraverso gli script è possibile:

- Interagire con gli oggetti della supervisione, leggendo stati, effettuando logiche ed inviando comandi
- Inviare messaggi di notifica personalizzati
- Interagire con il sistema operativo
- Inviare e ricevere stringhe attraverso la rete LAN e la porta seriale

Il manuale guida l'installatore passo a passo nella realizzazione di uno script personalizzato ed alla sua esecuzione nell'ambiente di runtime di IKON SERVER; sono inoltre analizzati nel dettaglio gli esempi presenti sul webserver, che offrono una panoramica completa delle funzionalità offerte da questa tecnologia.

NEW 2.9.2

A partire dalla versione 2.9.2, IKON SERVER permette di realizzare script sia in linguaggio PHP che LUA. Alcuni concetti sono in comune tra i due ambienti di sviluppo, mentre sintassi e librerie differiscono; per una guida allo scripting in PHP, si rimanda allo specifico manuale.

1.3 REQUISITI

Per poter sfruttare appieno le potenzialità del motore di scripting di IKON SERVER, è necessario disporre di:

- IKON SERVER aggiornato alla versione 2.9.2 o superiore
- Buona conoscenza di IKON SERVER
- Buona conoscenza di ETS e della tecnologia KNX
- Conoscenza, anche a livello base, del linguaggio di scripting LUA

2 PANORAMICA

2.1 SCRIPTS

Gli SCRIPTS sono files contenenti codice che è possibile creare e memorizzare sul disco flash di IKON SERVER attraverso un'apposita sezione nell'ambiente di amministrazione di IKON SERVER. Questi files possono essere creati, modificati e cancellati attraverso un'apposita pagina grafica, mediante la quale è anche possibile esportarli ed importarli su un altro webserver.

I files rimangono memorizzati sul disco di IKON SERVER e persistono anche a seguito di un ripristino dei parametri di fabbrica; d'altro canto, essi non sono compresi nel file di backup di IKON SERVER, pertanto devono essere esportati (e, successivamente, importati) separatamente se di vuole portare il progetto su un altro webserver.

Gli SCRIPTS devono essere realizzati seguendo le indicazioni contenute in questo manuale, facendo uso delle librerie e delle funzioni messe a disposizione dal motore di scripting; qualunque codice che esuli da queste indicazioni può potenzialmente compromettere il corretto funzionamento di IKON SERVER.

IKON SERVER è già dotato di una serie di esempi che offrono una panoramica completa delle potenzialità offerte dal sistema di scripting; non è prevista la modifica o esecuzione diretta di questi esempi, ma è possibile duplicarli e lavorare sulla copia senza alcuna restrizione.

2.2 RUNNERS

Una volta creato uno script personalizzato (ex novo oppure duplicando e modificando un esempio), è necessario associarlo ad uno o più oggetti di tipo "script runner”; questi particolari oggetti hanno il compito di eseguire il codice contenuto nello script a cui sono associati, ogni qualvolta

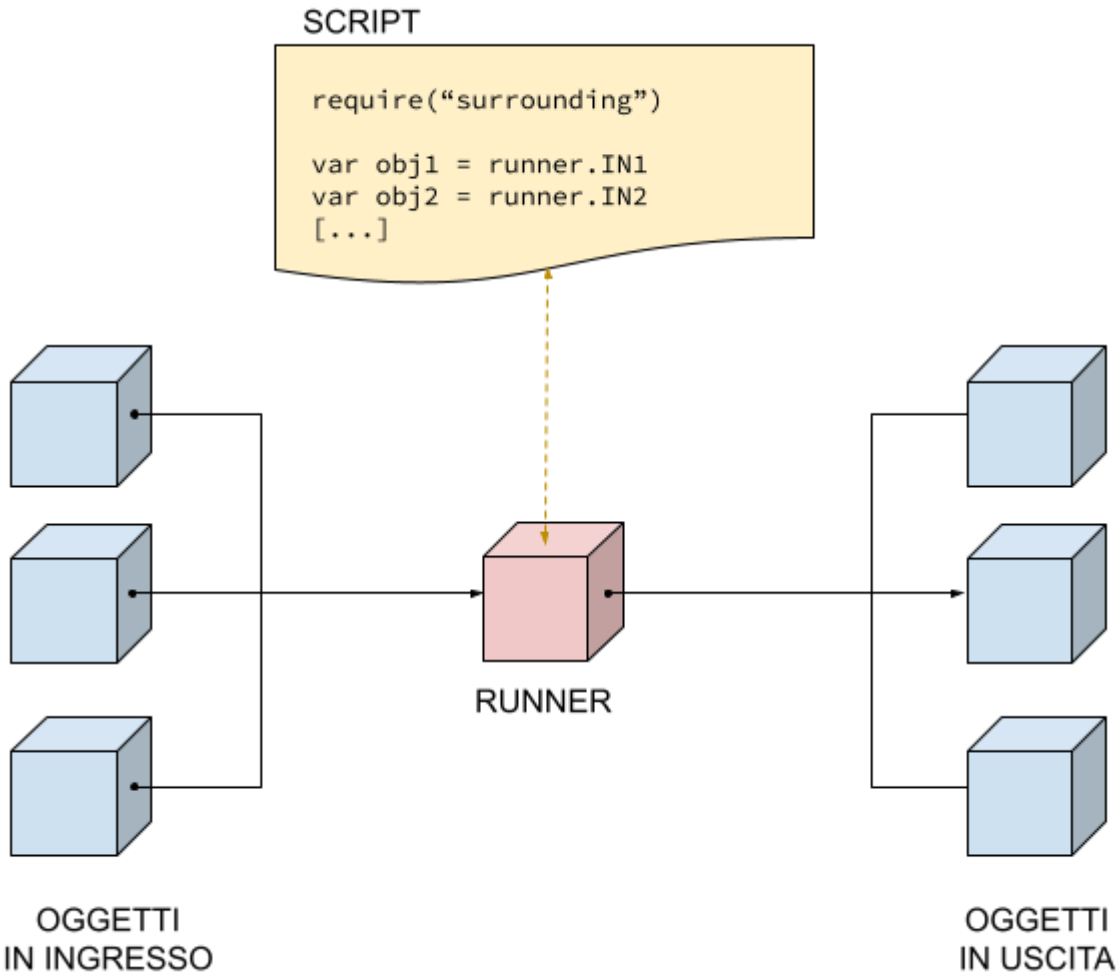
- vengano comandati direttamente dall'utente nel FRONTEND di IKON SERVER, interagendo con il "box" relativo nelle pagine ambiente o funzione
- cambi il valore di uno degli oggetti associati come CONNESSIONI IN INGRESSO, così come accade per qualunque altra tipologia di oggetto in IKON SERVER

Gli SCRIPTS non possono essere pertanto eseguiti direttamente, ma solo attraverso uno SCRIPT RUNNER, che si comporta in modo del tutto analogo alle altre tipologie di oggetto: ha un'icona ed un valore associati, può essere inserito negli AMBIENTI, nelle FUNZIONI o nei PREFERITI, può essere associato ad altri oggetti mediante EVENTI ATTIVI o PASSIVI, far parte di uno SCENARIO etc...

Il vantaggio di avere una struttura basata su SCRIPTS e SCRIPT RUNNERS distinti è duplice:

- Possibilità di associare lo stesso SCRIPT a più di un oggetto RUNNER, a sua volta inserito in logiche, eventi, scenari etc... differenti e quindi in grado di manipolare dati differenti
- Possibilità di esportare, importare e condividere gli script a prescindere dal progetto realizzato

Lo schema seguente illustra a livello di principio le interazioni tra uno SCRIPT RUNNER e gli oggetti associati tramite EVENTI, oppure il suo comando attraverso le pagine di RUNTIME del webserver:



2.3 LIBRERIE

IKON SERVER mette a disposizione una serie di librerie, che offrono strumenti già pronti per l'interazione con i principali aspetti coperti dal motore di scripting. Le librerie non sono automaticamente incluse all'interno degli scripts, ma devono essere richiamate mediante il comando LUA di inclusione

```
require(<nome della libreria>)
```

Esempio:

```
require("messages")
```

Giacché l'inclusione di ogni libreria comporta un tempo di elaborazione, si consiglia l'inclusione delle sole librerie strettamente necessarie al compito che deve svolgere lo script. Le librerie possono a loro volta dipendere da altre librerie; in questo caso, non è necessario includere anche le librerie correlate, in quanto l'inclusione avviene automaticamente richiamando la libreria che presenta le dipendenze.

La tabella seguente riporta un elenco delle librerie LUA disponibili, una sintetica descrizione e le eventuali dipendenze; nei capitoli successivi saranno esaminate nel dettaglio.

Libreria	Descrizione
object	Permette di caricare ed interagire con oggetti del progetto
surrounding	Offre un insieme di oggetti per interagire comodamente con l'oggetto SCRIPT RUNNER che sta eseguendo lo script ed i relativi EVENTI ATTIVI e PASSIVI
messages	Permette di inviare notifiche a video, email e push
json	Offre metodi di codifica e decodifica di stringhe JSON
url	Permette di richiamare un URL esterno e gestire la risposta

2.4 INGRESSI E USCITE

Quando l'oggetto SCRIPT RUNNER viene eseguito, esso riceve sempre un valore di INPUT; in particolare:

- Nel caso di comando diretto dall'interfaccia del FRONTEND di IKON SERVER, il valore è dato dall'input dell'utente (es: digitando un valore in una casella di testo, oppure premendo un pulsante – dipende dall'aspetto grafico scelto per lo SCRIPT RUNNER)
- Nel caso di comando a seguito della variazione di stato di uno degli oggetti "padri" associati come CONNESSIONI IN INGRESSO allo SCRIPT RUNNER, il valore di input è dato dalle proprietà dell'evento stesso, così come definito in amministrazione. In particolare, è possibile passare come valore:
    - Il valore dell'oggetto che cambia di stato, e che determina l'esecuzione dell'evento
    - Un valore costante

In entrambi i casi, il valore di input è disponibile all'interno dello script richiamando la funzione

```
input()
```

Esempio:

```
val = input()
```

Al termine dell'esecuzione, lo script può restituire un valore di OUTPUT all'ambiente di esecuzione; questo valore viene impostato come valore dello SCRIPT RUNNER, e può essere visualizzato nel FRONTEND oppure utilizzato in esecuzioni successive dello SCRIPT RUNNER stesso. Inoltre, se è presente almeno una CONNESSIONE IN USCITA, questo valore viene passato come valore di ingresso agli oggetti "figli" correlati.

Per restituire un valore all'ambiente di esecuzione è sufficiente richiamare la funzione

```
output()
```

Esempio:

```
output(1)
```



La chiamata alla funzione output() determina la fine dell'esecuzione dello script; ogni eventuale riga di codice successiva viene ignorata.

Qualora non si desideri passare alcun valore all'ambiente chiamante, è sufficiente richiamare la funzione senza argomenti:

```
output();
```

In questo caso:

- Non viene impostato alcun valore allo SCRIPT RUNNER
- Non vengono eseguite le CONNESSIONI IN USCITA eventualmente associati allo SCRIPT RUNNER



La mancata esecuzione delle CONNESSIONI IN USCITA in caso di chiamata alla funzione output() senza argomenti non deve essere vista come un limite, bensì come una maggiore flessibilità.

E' possibile infatti eseguire manualmente all'interno dello script gli eventi passivi, passando valori differenti ad ognuno di essi (cosa altrimenti impossibile attraverso il valore di output, uguale per tutti), prima di terminare l'esecuzione dello script; in questo caso non si desidera ovviamente che l'ambiente chiamante esegua nuovamente tali eventi. Per ulteriori dettagli si rimanda alla sezione relativa alla libreria "**surrounding**".

2.5 INCLUSIONE DI SCRIPTS

E' possibile includere altri SCRIPT all'interno di uno SCRIPT; questa funzionalità permette di utilizzare SCRIPT realizzati in precedenza come una sorta di "libreria" personalizzata. Per includere uno SCRIPT è sufficiente richiamare la stessa funzione di inclusione vista per le librerie:

```
require("nomeScript")
```

Dove **nomeScript** è il nome dello SCRIPT da eseguire; ad esempio:

```
require("prova")
```

2.6 ISPEZIONE VARIABILI

E' possibile ispezionare il valore di variabili e oggetti del codice attraverso la seguente funzione:

```
printall
```

Esempio:

```
printall(variabile)
```

Questa funzione restituisce in output il contenuto della variabile o dell'oggetto; se strutturato, il suo contenuto viene restituito su più righe, visualizzate nella sezione DEBUG della pagina dei runner (vedere capitoli successivi per dettagli).

Per le sole stringhe di testo, è possibile anche utilizzare a tale scopo la funzione di sistema

```
print(stringa)
```

2.7 TABLES

Le *tables* sono strutture molto flessibili nel linguaggio LUA, che corrispondono indicativamente agli *array* di altri linguaggi; le *tables* tuttavia possono avere come indici anche stringhe o chiavi non numeriche, e come tali quindi sono assimilabili agli *oggetti* di altri linguaggi di programmazione (es: JavaScript).

Molte librerie dello scripting LUA fanno uso delle tabelle per manipolare e restituire liste di elementi; le seguenti sintassi possono essere utili per operare con esse.

2.7.1 NUMERO DI ELEMENTI

Per conoscere il numero di elementi di una *table* richiamare la seguente funzione:

```
tablelength(<TABLE>)
```

2.7.2 CICLO TRA GLI ELEMENTI

Per ciclare tra gli elementi di una *table* utilizzare la seguente sintassi:

```
for k,v in pairs(<TABLE>) do
   -- k is the key of element
   -- v is the element itself
end
```

Ad esempio, per scorrere una lista di oggetti, filtrarli in base ad una determinata tipologia ed interagire con essi, utilizzare il seguente codice:

```
require("objects")


-- Searching for all lights in project
```

```
OBJECTS = objM.findObjs("Light","NAME",false)
for k,v in pairs(OBJECTS) do
   -- Considering only KNX lights
   if(v.TYPE == "EIBOBJECT") then
      -- Printing a debug message with info about current object
      -- k is the ID of object, v is the object itself
      system.debug("Light with ID " .. k .. " has name " . v.DB.NAME)
   end
end
```

Per ulteriori informazioni, si rimanda ai capitoli successivi dedicati alle librerie di gestione degli oggetti.

3 SCRIPTS

3.1 INTRODUZIONE

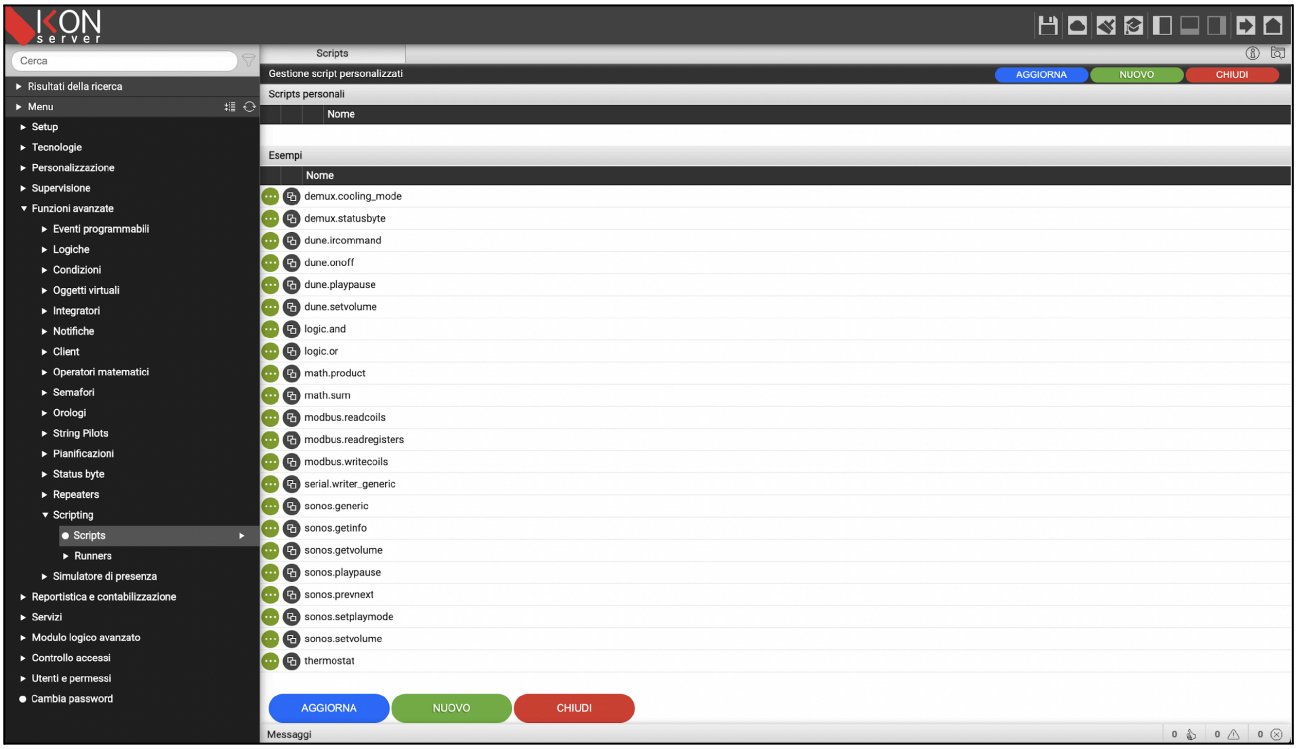
Questo capitolo illustra come creare, modificare, cancellare ed esportare gli script personalizzati su IKON SERVER. Le informazioni contenute in questo capitolo valgono sia per lo scripting in PHP che in LUA, ad eccezione di eventuali istruzioni o esempi specifici per il linguaggio PHP.

3.2 LISTA

Per accedere alla pagina di gestione degli scripts è sufficiente:

- Effettuare l'accesso ad IKON SERVER con un utente dotato dei permessi amministrativi
- Accedere all'ambiente di CONFIGURAZIONE
- Selezionare "FUNZIONI AVANZATE" e quindi "SCRIPTING" dal MENU LATERALE
- Selezionare "SCRIPTS"

Viene mostrata una pagina simile all'esempio seguente:



La parte in alto della pagina contiene una serie di script di esempio precaricati in IKON SERVER; essi non possono essere cancellati o modificati, ma solo consultati o duplicati (per realizzare, a partire da essi, un proprio script).

La parte in basso mostra invece gli script personalizzati presenti sul disco di IKON SERVER. Per creare un nuovo script è sufficiente premere il pulsante "NUOVO" al fondo della pagina: viene chiesto di indicare il nome da assegnare al nuovo script e, dopo alcuni secondi, data conferma dell'avvenuta creazione del nuovo file.

Volendo creare uno script partendo da un esempio esistente, viceversa, è sufficiente premere il corrispondente pulsante "DUPLICA" (grigio); anche in questo caso viene chiesto di specificare un nome da assegnare al nuovo script.

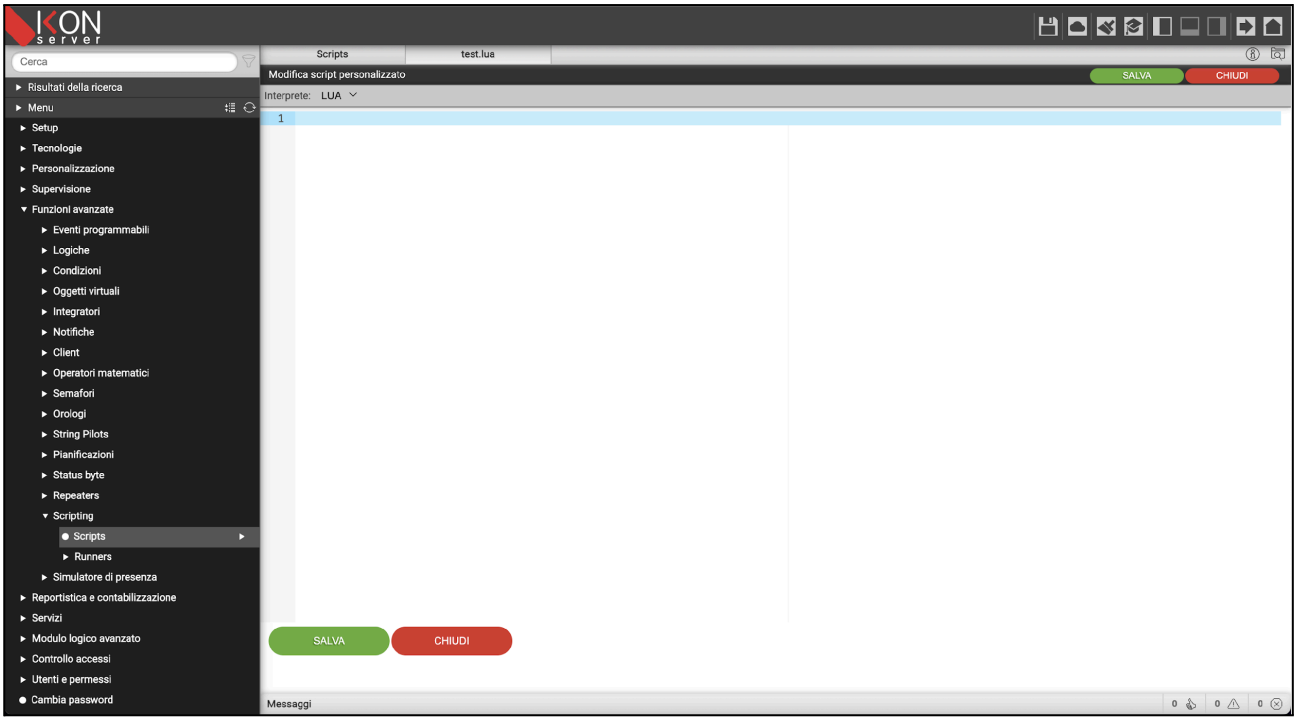
Per eliminare uno script è sufficiente premere il corrispondente pulsante "ELIMINA" (rosso); una volta data conferma, il file viene rimosso dal disco di IKON SERVER (in modo irreversibile).

Infine, per accedere alla scheda di consultazione e/o modifica di uno script (o di un esempio), utilizzare il pulsante "MODIFICA" (verde).

3.3 EDITOR

Selezionando il pulsante di MODIFICA di uno script o di uno esempio, viene mostrata una pagina contenente il codice che costituisce lo script stesso; nel caso degli esempi, il codice non può essere modificato. Nel caso di uno script nuovo, il codice presenta alcuni commenti (identificati dal prefisso "//") che forniscono alcune iniziali indicazioni su come realizzare lo script, come più dettagliatamente riportato in questo manuale.

La figura seguente mostra la pagina di modifica di uno script nuovo:



Per ulteriori informazioni su come realizzare uno script, sfruttando in particolare le librerie a disposizione, si rimanda ai capitoli successivi.

Una volta terminate le modifiche al codice, premere il pulsante "SALVA" per renderle persistenti; premendo viceversa il pulsante "CHIUDI" prima di aver salvato, ogni eventuale modifica andrà persa.

3.4 ESEMPIO

A titolo di esempio, nel proseguo di questo manuale, salvo diversamente indicato, si utilizzerà il seguente codice di esempio, che semplicemente restituisce in uscita il valore ricevuto come VALORE DI INGRESSO, dopo aver riportato il valore stesso in un messaggio di DEBUG (visualizzabile solo durante la simulazione di esecuzione, come evidenziato in seguito).

```
value = input()
system.debug("Ricevuto come ingresso il valore " .. value)
output(value)
```

4 RUNNERS

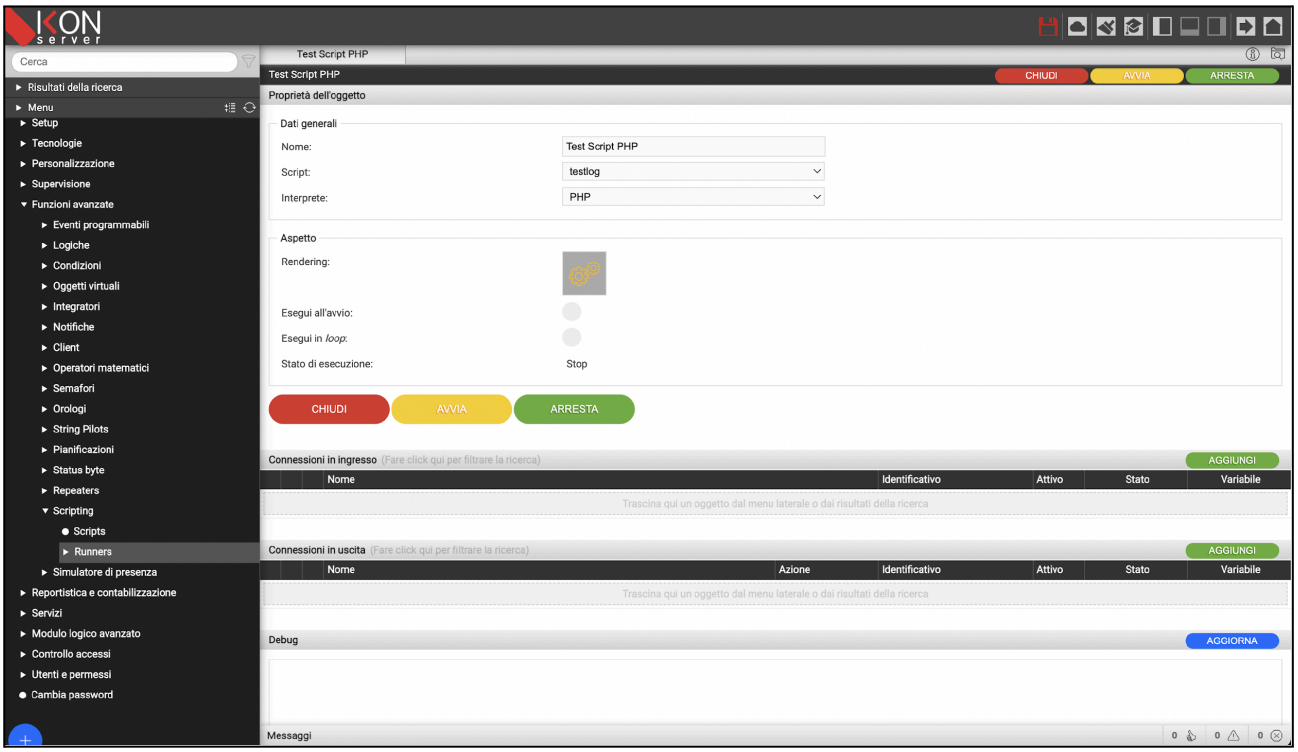
4.1 INTRODUZIONE

Questo capitolo illustra come creare e gestire oggetti di tipo SCRIPT RUNNER, che permettono di eseguire SCRIPTS all'interno della supervisione di IKON SERVER. Le informazioni contenute in questo capitolo valgono sia per lo scripting in PHP che in LUA, ad eccezione di eventuali istruzioni o esempi specifici per il linguaggio PHP.

4.2 NUOVO RUNNER

Per creare un nuovo SCRIPT RUNNER è sufficiente:

- Effettuare l'accesso ad IKON SERVER con un utente amministrativo, ed accedere alla sezione di configurazione
- Identificare la voce "SCRIPTING" all'interno di "FUNZIONI AVANZATE", quindi selezionare "SCRIPT RUNNERS"
- Premere il pulsante "AGGIUNGI" nella toolbar in basso
- Evidenziare il nuovo oggetto creato e premere i "puntini" a lato, oppure – in alternativa – il pulsante "MODIFICA" nella toolbar in basso, per accedere alla sua scheda



Oltre ad assegnare un nome allo SCRIPT RUNNER, è necessario selezionare anche quale script sarà messo in esecuzione, scegliendolo nell'apposito menu a tendina. Inoltre, è possibile personalizzare l'aspetto grafico con cui lo SCRIPT RUNNER, se inserito in una pagina grafica del FRONTEND, viene visualizzato e permette all'utente di interagire con lo SCRIPT associato.

E' possibile inoltre specificare le seguenti opzioni per lo SCRIPT RUNNER:

ESEGUI ALL'AVVIO	Selezionando questo flag, lo SCRIPT RUNNER viene eseguito automaticamente all'avvio della supervisione
ESEGUI IN LOOP	Selezionando questo flag, lo SCRIPT RUNNER viene eseguito iterativamente per un numero indefinito di volte, fintanto che non viene arrestato manualmente oppure mediante un EVENTO. Per ulteriori dettagli, si rimanda alla apposita sezione "esecuzione in background"
TEMPO DI LOOP	Nel caso di esecuzione in loop, specificare la durata in millisecondi di ogni iterazione.

E' possibile inoltre (a partire dalla versione software 1.1.0) mettere in esecuzione o arrestare uno SCRIPT RUNNER attraverso gli appositi pulsanti "AVVIA" e "ARRESTA".

4.2.1 INGRESSI

La sezione **CONNESSIONI IN INGRESSO** permette di trascinare uno o più oggetti che, al cambio di stato, determinano l'esecuzione dello SCRIPT RUNNER.

Una volta trascinato un oggetto all'interno di questo elenco, è necessario stabilire quale **VALORE** debba essere passato allo SCRIPT RUNNER ogni qualvolta l'oggetto stesso cambia di stato.

A differenza di altri punti del software in cui questa scelta è guidata attraverso appositi menu a tendina, in questo caso il valore deve essere specificato manualmente. Questo campo (visibile solo se si attiva la visualizzazione AVANZATA attraverso la toolbar in alto a destra) può contenere i seguenti valori:

$VAL	Il valore corrente dell'oggetto viene passato allo SCRIPT RUNNER come valore di ingresso. Opzione predefinita.
$NVAL	Il valore negato dell'oggetto viene passato allo SCRIPT RUNNER come valore di ingresso
Un qualunque valore numerico o stringa	Il valore viene passato allo SCRIPT RUNNER come valore di ingresso, a prescindere dallo stato dell'oggetto stesso

In ogni caso, il valore è disponibile all'interno dello script attraverso la chiamata

```
input()
```

già descritta in precedenza. Inoltre, nel caso di script associati a più di un oggetto in ingresso, è possibile ottenere l'ID dell'oggetto che è cambiato (e che sta passando il valore di ingresso) attraverso la chiamata alla funzione

```
callerid()
```



Il valore indicato in questo campo non è l'unico modo per accedere, all'interno dello SCRIPT, al valore dell'oggetto relazionato come ingresso. Attraverso la libreria surrounding, infatti, come meglio descritto in seguito, è possibile conoscere lo stato di tutti gli ingressi ed effettuare operazioni su di essi; per ulteriori dettagli si rimanda al capitolo dedicato a questa libreria.

4.2.2 USCITE

Analogamente, trascinando oggetti all'interno della sezione **CONNESSIONI IN USCITA** è possibile determinarne l'esecuzione al termine dello SCRIPT. Per ogni oggetto è necessario specificare quale azione eseguire e (se l'oggetto lo prevede) quale valore passare al termine dell'esecuzione dello SCRIPT RUNNER; in quest'ultimo caso, scegliere "VALORE DELLO SCRIPT RUNNER" per passare all'oggetto il valore restituito in uscita dallo SCRIPT con il comando

```
output(...)
```



Come già evidenziato in precedenza, è possibile inibire l'esecuzione automatica degli oggetti associati come uscite al termine dello SCRIPT, non passando alcun valore alla funzione output(). In questo caso, viene affidato al codice dello SCRIPT stesso l'eventuale comando degli oggetti associati come uscite, come meglio dettagliato nel capitolo dedicato alla libreria "surrounding".

Anche per le uscite è possibile associare un **identificativo** e visualizzare il nome della variabile con cui fare riferimento all'uscita nello SCRIPT.

4.3 LOOP

Uno script può essere seguito in modo continuo in "background" attivando l'opzione *loop* nella scheda dello SCRIPT RUNNER. Solitamente, quando si abilita questa opzione, si attiva anche l'opzione *esegui all'avvio*, in modo che il codice venga nuovamente eseguito, se il sistema si riavvia.

4.3.1 ESECUZIONE CICLICA

In modalità *loop*, viene ciclicamente chiamata la funzione

```
main_loop()
```

che è necessario definire all'interno dello script, ad esempio:

```
function main_loop()  -- Programma principale che va in loop ciclico
  system.debug("Esecuzione ciclica" )
end
```

Questa funzione viene chiamata periodicamente con un tempo di attesa, tra una esecuzione e la successiva, da impostare nel campo *tempo di loop* (specificare un valore espresso in millisecondi); minore è il tempo impostato, maggiore è la reattività del RUNNER, ma anche il carico sulla CPU del webserver.

4.3.2 ESECUZIONE AD INTERRUPT

E' possibile definire un'altra funzione nello script, che viene richiamata ogni qualvolta uno degli ingressi cambia di stato: ad esempio...

```
function event_loop(event)  -- Funzione richiamata al cambio di stato di un ingresso
  printall(event)
end
```

L'argomento della funzione è una *table* contenente le seguenti informazioni:

caller	ID dell'oggetto in ingresso che è cambiato di stato
value	Valore ricevuto dall'oggetto
hashcode	Eventuale PIN passato dall'oggetto In alternativa viene passata la stringa "NO-HASHCODE"
optionals	Concatenazione di attributi della relazione tra l'oggetto in questione ed il RUNNER Se è stato specificato un IDENTIFICATIVO per l'oggetto in ingresso, esso viene fornito attraverso l'attributo "tag_complex_element"

Esempio di output della *printall* sull'argomento della funzione:

```
(TABLE)
___[optionals]= tag_complex_element='CUCINA'
___[hashcode]= NO-HASHCODE
___[value]= 1
___[caller]= 123
```

dove

123 ID dell'oggetto chiamante

1 Valore ricevuto

CUCINA Identificativo associato all'oggetto nella scheda del RUNNER

Questa funzione può essere maggiormente sfruttata includendo la libreria SURROUNDING presentata più avanti in questo manuale, la quale mette a disposizione metodi per interagire in modo più comodo con gli INGRESSI del RUNNER.

4.4 DEBUGGING

E' possibile inserire all'interno degli script messaggi rispettivamente di DEBUG (finalizzati al monitoraggio delle attività svolte dallo script in fase di realizzazione), di WARN (utili per notificare all'ambiente di esecuzione messaggi di avviso) e di ERRORE (necessari per segnalare e tracciare eventuali anomalie durante l'esecuzione dello script che ne compromettono il corretto completamento).

Per inserire questi messaggi è sufficiente richiamare, rispettivamente, una delle seguenti funzioni:

```
system.debug()
system.info()
system.warn()
system.err()
```

Esempio:

```
system.debug("Questo è un messaggio di debug")
system.info("Questo è un messaggio di info")
system.warn("Questo è un messaggio di warning")
system.err("Questo è un messaggio di errore")
```

Questi messaggi vengono visualizzati nella sezione DEBUG, colorati in modo differenziato a seconda del livello.

5 LIBRERIA "OBJECTS"

5.1 INTRODUZIONE

La libreria LUA "objects" mette a disposizione una classe con una dotazione di metodi per interagire con gli oggetti del progetto di supervisione.

Questa libreria è alla base della libreria "SURROUNDING", che permette una agevole interazione con gli INGRESSI e le USCITE del RUNNER, materia del capitolo successivo di questo manuale.

5.2 INCLUSIONE

Per includere la libreria all'interno di uno SCRIPT è necessario inserire all'inizio la seguente riga:

```
require("objects")
```

L'inclusione della libreria mette a disposizione una classe denominata "object manager" accessibile tramite la parola chiave

```
objM
```

i cui metodi sono descritti a seguire.

5.3 CARICAMENTO DI OGGETTI

Per caricare un singolo oggetto della supervisione all'interno di una variabile LUA è necessario richiamare il seguente metodo:

```
<OBJ> = objM.getObj(XXX)
```

dove XXX è l'ID numerico dell'oggetto che si intende caricare, e <OBJ> è un qualunque nome di variabile alla quale assegnare l'oggetto. Esempio:

```
LUCE_CUCINA = objM.getObj(123)
```

Se l'oggetto è già stato richiamato in precedenza, attraverso questo metodo si ottiene una *istanza* ad esso; in caso contrario, viene effettuata una interrogazione al database di progetto, e oltre ad assegnarlo alla variabile, viene salvato in una *collection* interna allo script, per successive interrogazioni.

Se si desidera caricare più oggetti simultaneamente, è possibile richiamare il metodo

```
<ARRAY> = objM.getObjs(<array of IDs>)
```

al quale deve essere passato un array (*table*) di ID di altrettanti oggetti, ad esempio:

```
OBJECTS = objM.getObjs({123,456,789})
```

La funzione restituisce un array contenente gli oggetti richiesti, nel quale l'indice è rappresentato dall'ID; nell'esempio precedente è possibile quindi successivamente accedere ad un singolo oggetti della collezione attraverso la sintassi

```
LUCE_CUCINA = OBJECTS[123]
```

Gli oggetti ottenuti per caricamento multiplo, in alternativa, possono essere successivamente recuperati attraverso il metodo di caricamento singolo visto in precedenza...

```
LUCE_CUCINA = objM.getObj(123)
```

… senza che vengano effettuate ulteriori letture dal database di progetto (essendo già nella *collection* interna dello script).

5.4 PROPRIETÀ DEGLI OGGETTI

Gli oggetti caricati con le funzioni viste in precedenza, mettono a disposizione una serie di attributi primari, riportati nella seguente tabella.

Le variabili che rappresentano oggetti del progetto di supervisione presentano le seguenti proprietà, che possono essere utilizzate nello SCRIPT (si presti attenzione alle maiuscole / minuscole che devono essere rispettate per una corretta sintassi):

PROPRIETÀ	DESCRIZIONE	ESEMPIO
ID	ID univoco dell'oggetto	<OBJ>.ID
TYPE	Tipo "funzionale" di oggetto, utilizzato dai servizi di comunicazione. Alcuni esempi: Oggetti KNX: "EIBOBJECT" Variabili: "VIRTUALOBJECT" Scenari: "SCENARIO" Script runners: "CSCMD" *NOTA: per un elenco completo dei tipi, si rimanda all'appendice 10.1*	<OBJ>.TYPE
PHPCLASS	Tipologia grafica di oggetto, utilizzata dall'interfaccia web. Alcuni esempi: Oggetti KNX: "dpadEibObject" Variabili: "dpadVirtualObject" Scenari: "dpadScenario" Logiche: "dpadLogic" Condizioni: "dpadCondition" Script runners: "dpadScriptRunner" *NOTA: non sempre esiste una corrispondenza univoca tra "type" e "phpclass": in alcuni casi, più "phpclass"*	<OBJ>.PHPCLASS

	condividono il medesimo "type". Per un elenco completo, si rimanda all'appendice 10.2	
VALUES_TYPE	Tipo di dato associato all'oggetto. La codifica di questo campo dipende dal TYPE.	<OBJ>.VALUES_TYPE
DB	Collezione di tutti i campi del database relativi all'oggetto in questione. Vedere sotto per dettagli	<OBJ>.DB
status	Stato corrente dell'oggetto. NOTA: questa proprietà ha significato solo per gli scenari; in questo caso: 1 → scenario fermo 2 → scenario in esecuzione	<OBJ>.status
value	Valore corrente dell'oggetto NOTA: questa proprietà ha significato per tutte le tipologie di oggetto che hanno uno stato, scenari esclusi	<OBJ>.value

La collezione DB contiene la lista di tutti i campi del database relativa all'oggetto; essi possono essere letti tramite la sintassi

```
<OBJ>.DB.<FIELD>
```

ad esempio:

```
LUCE_CUCINA = objM.getObj(123)
NOME_LUCE = LUCE_CUCINA.DB.NAME        -- Restituisce ad esempio "Luce Cucina"
```

Segue una lista dei principali campi:

CAMPO	DESCRIZIONE
NAME	Nome dell'oggetto
DESCRIPTION	Descrizione dell'oggetto
MSP	Principale attributo dell'oggetto; il significato di questo campo dipende dal TYPE, solitamente contiene l' *indirizzo* (es: indirizzo di gruppo KNX)
UNIT	Laddove disponibile, rappresenta l'unità di misura
IS_WRITABLE IS_READABLE	Flag di permesso rispettivamente per il comando e la lettura dell'oggetto
IS_VISIBLE	Flag di visibilità dell'oggetto
IS_STOPPABLE	Flag di permesso per lo STOP dell'oggetto (ha valenza solo per gli SCENARI)
IS_BOOLEAN	Flag che stabilisce se l'oggetto è di tipo BOOLEAN (cioè ammette solo valori 1 o 0) oppure no

IS_PROTECTED	Flag di protezione dell'oggetto da modifiche effettuate dall'interfaccia utente
ENABLE_FLAG	Flag di abilitazione generale dell'oggetto per il funzionamento e l'interazione con altri oggetti
TRACK_FLAG	Flag di tracciamento dell'oggetto nel database di storicizzazione

Infine, è disponibile una sotto-collezione di *opzioni* accessibile tramite la sintassi

```
<OBJ>.DB.options.<FIELD>
```

ad esempio:

```
LUCE_CUCINA = objM.getObj(123)
CATEGORIA = LUCE_CUCINA.DB.options.category     -- Restituisce ad esempio "lighting"
```

Per un dettaglio completo di tutti i campi, si consiglia di ispezionare l'oggetto tramite il metodo *printall* documentato in precedenza, in quanto la lista di opzioni disponibili ed i rispettivi valori dipendono fortemente dal tipo di oggetto.

5.5 COMANDO DI OGGETTI

E' possibile inviare un comando ad un oggetto, impostando un nuovo valore, tramite il metodo

```
<OBJ>:set(<VALUE>,<ACTION>)
```

dove

<VALUE> Nuovo valore passato all'oggetto

<ACTION> Azione da richiamare (opzionale, default: "SETVALUE")

Ad esempio:

```
LUCE_CUCINA = objM.getObj(123)
LUCE_CUCINA:set(1)      -- Accende la luce
```

Le azioni previste, e gli eventuali valori, per le principali tipologie di oggetto sono riportate nella tabella seguente:

TIPO	ACTION	DESCRIZIONE	VALORE
Oggetto KNX	SETVALUE	Invia un comando al gruppo KNX corrispondente	Dipende dal tipo di oggetto KNX Es: 0 → spento 1 → acceso 50 → imposta al 50%

	GETVALUE	Invia una richiesta di ritrasmissione stato	Nessun valore
Scenario	EXECUTE	Esegue lo scenario	Nessun valore
	STOPEXECUTION	Arresta lo scenario (se in esecuzione)	Nessun valore
Logica	EVALUATE	Valuta la logica ed esegue le uscite corrispondenti	Nessun valore
Condizione	CALCULATE	Calcola la condizione ed esegue le uscite corrispondenti	Nessun valore
Notifica a video	INSERT	Inserisce il messaggio nella tabella di log, in modo che venga visualizzato nel centro messaggi	Nessun valore
Notifica via email	SENDMAIL	Invia il messaggio via posta elettronica	Nessun valore
Integratore	INTEGRATE	Ricalcola il valore aggiornato dell'integratore	Nessun valore
	RESET	Riporta a 0 il valore dell'integratore	Nessun valore
Variabile	SETVALUE	Imposta il valore della variabile	In base al tipo di variabile creata

5.6 RICERCA DI OGGETTI

E' possibile ricercare uno o più oggetti nel progetto a partire da uno dei sui campi del database, elencati in precedenza; per ottenere un singolo oggetto utilizzare il metodo

```
<OBJ> = objM.findObj(<KEY>,<FIELD>,<EXACT_MATCH>)
```

Dove

<KEY> Parola chiave, da specificare sotto forma di stringa

<FIELD> Nome del campo del database su cui effettuare la ricerca

<EXACT_MATCH> Flag che specifica se la parola chiave debba coincidere esattamente con il campo del database, oppure sia sufficiente che una parte del valore a database risponda alla parola chiave.

Specificare questo flag nel formato booleano *true* o *false*

Esempio:

```
LUCE_CUCINA = objM.findObj("Luce cucina","NAME",true
```

Anche nel caso della ricerca, è possibile richiedere il caricamento di più oggetti che rispondano al criterio di ricercare, attraverso il metodo

```
<ARRAY> = objM.findObjs(<KEYS>,<FIELD>,<EXACT_MATCH>)
```

Il primo argomento <KEYS> può essere una singola parola chiave (nel qual caso deve essere passata come stringa) oppure un array di parole chiave; nel secondo caso, in base al campo <EXACT_MATCH> vengono effettuate rispettivamente le seguenti tipologie di ricerca nel database:

<EXACT_MATCH> = true	Vengono caricati tutti gli oggetti che hanno esattamente nel campo <FIELD> i valori contenuti nell'array <KEYS>
<EXACT_MATCH> = false	Vengono cercati tutti gli oggetti che contengono nel campo <FIELD> almeno una delle parole chiave specificate in <KEYS>

Esempi:

```
-- Ricerca di tutti gli oggetti che contengono una parola chiave nel nome
<ARRAY> = objM.findObjs("Luce","NAME",false)
```

```
-- Ricerca di una specifica lista di oggetti in base al nome
<ARRAY> = objM.findObjs({"Luce cucina","Luce salotto"},"NAME",true)
```

5.7 MODIFICA DI OGGETTI

E' possibile programmaticamente modificare il valore di un campo nel database relativo ad un oggetto, attraverso il metodo

```
<RESULT> = objM.updatFieldObj(<ID>,<FIELD>,<VALUE>)
```

dove

<ID>	ID dell'oggetto da modificare
<FIELD>	Nome del campo del database a cui apportare la modifica
<VALUE>	Nuovo valore da assegnare al campo. Passare questo argomento sotto forma di stringa.

Il valore di ritorno del metodo informa sulla modifica avvenuta con successo (*true*) o con errore (*false*).

6 LIBRERIA "SURROUNDING"

6.1 INTRODUZIONE

La libreria LUA "SURROUNDING" permette facilmente di interagire con gli INGRESSI e le USCITE del RUNNER, estendendo e specializzando i metodi già visti in precedenza della libreria "OBJECTS".

6.2 INCLUSIONE

Per includere la libreria all'interno di uno SCRIPT è necessario inserire all'inizio la seguente riga:

```
require("surrounding")
```

6.3 INGRESSI E USCITE

A partire dalla riga immediatamente successiva all'inclusione della libreria, risulta disponibile allo SCRIPT il seguente oggetto:

```
runner
```

Questo oggetto rappresenta lo SCRIPT RUNNER che sta eseguendo lo SCRIPT, e permette di interagire con esso all'interno dello SCRIPT. L'oggetto mette a disposizione anche le proprie eventuali relazioni con gli INGRESSI ("padri") e le USCITE ("figli") configurati nella scheda dello SCRIPT RUNNER (vv. capitolo 4.2) attraverso i rispettivi array

```
runner.INPUTS   //Oggetti in INGRESSO
runner.OUTPUTS  //Oggetti in USCITA
```

All'interno degli array, i diversi oggetti sono rappresentati dalla variabile "OBJ_XXX" dove XXX è l'ID dell'oggetto, ad esempio:

```
runner.INPUTS.OBJ_123   //Oggetto in INGRESSO con ID 123
```

Questi oggetti mettono a disposizione le proprietà ed i metodi descritti nel capitolo precedente.

Inoltre, mettono a disposizione una ulteriore *collection* di informazioni relative alla RELAZIONE con il RUNNER:

PROPRIETÀ	DESCRIZIONE	ESEMPIO
RELATION	Collezione di attributi della relazione con il RUNNER	<OBJ>.RELATION

I campi della relazione sono elencati nella tabella seguente:

<table>
<tr><th>CAMPO</th><th>DESCRIZIONE</th></tr>
<tr><td>ACTION</td><td>Azione eseguita sull'oggetto di destinazione della relazione</td></tr>
<tr><td>ACTION_VALUE</td><td>Valore passato all'oggetto di destinazione</td></tr>
<tr><td>options</td><td>Collezione di valori opzionali della relazione, tra cui:<table><tr><td>tag_complex_element</td><td>IDENTIFICATIVO assegnato all'INGRESSO / USCITA</td></tr></table></td></tr>
</table>

6.4 IDENTIFICATIVI

E' possibile associare ad ogni oggetto un **identificativo** (che deve essere una stringa priva di spazi o caratteri speciali) che ne agevoli l'identificazione all'interno dello SCRIPT, quando si fa uso della libreria SURROUNDING. Se ad esempio si associa l'identificativo "TEST" ad un oggetto, esso risulta disponibile nel codice scrivendo

```
runner.TEST
```

Per conoscere l'identificativo di un oggetto in INGRESSO o in USCITA ad un RUNNER noto il suo ID, è possibile richiamare le seguenti funzioni:

```
runner.getIdentifierById(id)          -- Ricerca identificativo tra INGRESSI e USCITE
runner.getInputIdentifierById(id)     -- Ricerca identificativo solo tra gli INGRESSI
runner.getOutputIdentifierById(id)    -- Ricerca identificativo solo tra le USCITE
```

Ad esempio, all'interno di un RUNNER in *loop* è possibile intercettare il cambio di stato di un ingresso, identificarlo e gestirlo con il seguente codice di esempio:

```
function event_loop(event)
   local IDENTIFIER = runner.getInputIdentifierById(event.caller)
   system.debug("Input " .. IDENTIFIER .. " changed to value " .. tostring(event.value))


   local OBJ = objM.getObj(event.caller)
   system.debug("Object name: " .. OBJ.NAME)
end
```

7 LIBRERIA "MESSAGES"

7.1 INTRODUZIONE

La libreria LUA "messages" permette di inviare notifiche a video e via email direttamente all'interno di uno SCRIPT, senza dover necessariamente creare oggetti all'interno del progetto di supervisione.

7.2 INCLUSIONE

Per includere la libreria all'interno di uno SCRIPT è sufficiente inserire all'inizio la seguente riga:

```
require("messages")
```

A partire dalla riga immediatamente successiva, risulta disponibile allo SCRIPT la seguente classe:

```
msgM
```

Questa classe ("manager dei messaggi") mette a disposizione una serie di funzioni (o "metodi") per caricare oggetti della supervisione ed effettuare operazioni su di essi.

7.3 NOTIFICHE A VIDEO

Per inviare una nuova notifica a video "on the fly" durante l'esecuzione di uno script è sufficiente richiamare la funzione

```
msgM.sendVideoMsg(msg,level)
```

Dove

msg	Messaggio da inviare
level	Livello di priorità del messaggio. Valori possibili:
	0 Allarme (predefinito)
	1 Avvertimento
	2 Informazione

Esempio:

```
msgM.sendVideoMsg("Ricevuto valore " .. input() .. " in ingresso",2);
```

La funzione restituisce 0 in caso di successo, un codice di errore in caso contrario.

7.4 NOTIFICHE VIA MAIL

Analogamente è possibile inviare "on the fly" un messaggio di posta elettronica attraverso la chiamata

```
msgM.sendEMailMsg(msg,to,subject,cc,bcc)
```

Dove

msg	Messaggio da inviare
to	Indirizzo/i email a cui inviare il messaggio Nel caso di indirizzo multipli, separarli con ";"
subject	Oggetto del messaggio [Facoltativo]
cc	Indirizzo/i email in copia [Facoltativo]
bcc	Indirizzo/i email in copia nascosta [Facoltativo]

La funzione restituisce 0 in caso di successo, un codice di errore in caso contrario.

8 LIBRERIA "JSON"

8.1 INTRODUZIONE

Questa libreria permette di manipolare stringhe in formato JSON trasformandole in strutture dati di tipo *table* di LUA.

8.2 INCLUSIONE

Per includere la libreria all'interno di uno SCRIPT è sufficiente inserire all'inizio la seguente riga:

```
JSON = require("json")
```

8.3 DECODIFICA

Una volta inclusa la libreria ed assegnata alla classe JSON nel codice, è possibile decodificare una stringa in formato JSON attraverso il metodo

```
<TABLE> = JSON.decode(<JSON_STRING>)
```

Esempio:

```
JSON.decode('[1,2,3,{"x":10}]') -- Returns { 1, 2, 3, { x = 10 } }
```

8.4 CODIFICA

In modo analogo, è possibile parterndo da una struttura *table* LUA ottenere una rappresentazione JSON all'interno di una stringa con il metodo

```
<JSON_STRING> = JSON.decode(<TABLE>)
```

Esempio:

```
JSON.encode({ 1, 2, 3, { x = 10 } }) -- Returns '[1,2,3,{"x":10}]'
```

8.5 CREDITS

Questa libreria è un progetto OPEN-SOURCE non sviluppato da DOMOTICA LABS; per ulteriori dettagli si rimanda a:

https://github.com/rxi/json.lua

9 LIBRERIA "URL"

9.1 INTRODUZIONE

Questa libreria permette di richiamare un URL e gestire la risposta. Può essere utilizzata per richiamare API HTTP(S) tramite chiamate GET, e decodificare il risultato (ad esempio dal formato JSON).

9.2 INCLUSIONE

Per includere la libreria all'interno di uno SCRIPT è sufficiente inserire all'inizio la seguente riga:

```
require("url")
```

La libreria mette a disposizione una classe "URL manager" richiamabile con la parola chiave

```
urlM
```

9.3 GET URL

Per richiamare un URL tramite GET, ed assegnare la risposta ad una variabile, utilizzare la seguente sintassi:

```
<RESPONSE> = urlM.get(<URL>)
```

Esempio:

```
JSON = require("json")
require("url")


-- Restituisce il proprio IP pubblico in formato JSON
-- Es: {"ip": "150.60.100.80"}
response = urlM.get("http://ip.jsontest.com")
my_address = JSON.decode(response)["ip"]
system.debug("My IP public address is " .. my_address)
```

10 APPENDICI

10.1 TIPI FUNZIONALI

La tabella seguente riporta un elenco completo di tutti i tipi funzionali di oggetto previsti da IKON SERVER, utilizzati dai servizi di comunicazione per lo svolgimento delle operazioni "in background" del webserver. Solitamente si fa riferimento a questo tipo di oggetto come "TYPE".

Per ognuno di essi è riportato, oltre al tipo e la descrizione, le azioni possibili; queste ultime possono essere richiamate con i metodi visti in precedenza per il comando degli oggetti.

TIPO (TYPE)	DESCRIZIONE	AZIONI
VIRTUALOBJECT	Variabile	SETVALUE → Imposta un valore all'oggetto
SCENARIO	Scenario	EXECUTE → Esegue lo scenario STOPEXECUTION → Arresta l'esecuzione
LOGIC	Logica	EVALUATE → Valuta la logica
CONDITION	Condizione	CALCULATE → Calcola il confronto di valori
CLIENTBROWSER	Client	REDIRECT → Effettua un salto pagina sul client VOIPCALL → Effettua una chiamata VOIP all'indirizzo IP del client
NOTIFY_VIDEO	Notifica a video	INSERT → Crea un messaggio a video
NOTIFY_EMAIL	Notifica via email	SENDMAIL → Invia un messaggio email
INTEGRATOR	Integratore	INTEGRATE → Aggiorna il calcolo dell'integratore RESET → Riporta a 0 il valore dell'integratore
EIBOBJECT	Oggetto KNX	SETVALUE → Imposta un valore sul corrispondente indirizzo di gruppo KNX GETVALUE → Emette una richiesta sul bus KNX di lettura stato aggiornato
CSCMD	Script runner	RUN → Esegue lo script QUIT → Termina l'esecuzione dello script
USER	Utente	REDIRECT → Effettua un salto pagina su tutti i client in cui è connesso l'utente VOIPCALL → Effettua una chiamata VOIP a tutti i clienti in cui è connesso l'utente
USERGROUP	Gruppo utenti	
USERPERMISSION	Permesso	

CONTAINER	Oggetto composito	
CAMERA	Telecamera IP	
PBXELEMENT	Telefono IP Videocitofono IP Gruppo di chiamata VOIP	VOIPCALL → Effettua una chiamata VOIP destinata all'oggetto

10.2 TIPI WEB (PHPCLASS)

La tabella seguente riporta un elenco completo di tutti i tipi di oggetto previsti da IKON SERVER per quanto riguarda l'interfaccia web. Solitamente, si fa riferimento a questo tipo di oggetto come "PHPCLASS".

Per ognuno di essi è riportato – laddove presente - il tipo funzionale (o "TYPE") corrispondente; è possibile che più di una PHPCLASS facciano riferimento al medesimo TYPE. Se non è presente, la PHPCLASS non ha una controparte funzionale per i servizi di comunicazione; questo accade tipicamente per tutte le tipologie web che hanno una funzione puramente grafica (ambienti, collegamenti etc...)

TIPO (PHPCLASS)	**DESCRIZIONE**	**TYPE DI RIFERIMENTO**
dpadObject	Oggetto generico *	
dpadGroup	Ambiente (o "cartella" in senso generale)	
dpadUrl	Collegamento	
dpadSysCmd	Comando di sistema *	SYSCMD
dpadSysCmdWait	Comando di attesa di uno scenario	SYSCMD
dpadUser	Utente	USER
dpadUserGroup	Gruppo utenti	USERGROUP
dpadUserPermission	Permesso di accesso per gruppi utente	USERPERMISSION
dpadTriggerObject	Trigger di sistema su cambiamento DB *	TRIGGEROBJECT
dpadContainer	Oggetto composito	CONTAINER
dpadVirtualObject	Variabile	VIRTUALOBJECT
dpadScenario	Scenario	SCENARIO
dpadPbxExtension	Telefono IP	PBXELEMENT
dpadPbxTrunk	Linea telefonica VOIP	

dpadPbxQueue	Gruppo di chiamata VOIP	
dpadPbxDoorOpener	Videocitofono IP	
dpadLogic	Logica	LOGIC
dpadCondition	Condizione	CONDITION
dpadClientBrowser	Client	CLIENTBROWSER
dpadNotifyVideo	Notifica a video	NOTIFY_VIDEO
dpadNotifyEmail	Notifica via email	NOTIFY_EMAIL
dpadIntegrator	Integratore	INTEGRATOR
dpadEibObject	Oggetto KNX	EIBOBJECT
dpadCamera	Telecamera IP	CAMERA
dpadScriptRunner	Script runner	CSCMD
dpadLoadControl	Controllo carichi	CSCMD

** : tipologie di oggetto solitamente non utilizzate o disponibili in interfaccia web*