

QUANTUM SCRIPTING LUA

MANUEL DE CONFIGURATION

VERSION DU LOGICIEL 1.0.1

SOMMAIRE

1 INTRODUCTION....4
1.1 QU'EST-CE QUE LE SERVEUR QUANTUM....4
1.2 OBJECTIF DE CE MANUEL....4
1.3 EXIGENCES....4
2 APERÇU....5
2.1 SCRIPTS....5
2.2 COUREURS....5
2.3 BIBLIOTHÈQUES....6
2.4 ENTRÉES ET SORTIES....7
2.5 INCLUSION DE SCRIPTS....8
2.6 INSPECTION VARIABLE....8
2.7 TABLEAUX....9
2.7.1 NOMBRE D'ÉLÉMENTS....9
2.7.2 CYCLE ENTRE LES ÉLÉMENTS....9
3 SCRIPTS....10
3.1 INTRODUCTION....10
3.2 LISTE....10
3.3 ÉDITEUR....11
3.4 EXEMPLE....11
4 COUREURS....13
4.1 INTRODUCTION....13
4.2 NOUVEAU COUREUR....13
4.2.1 ENTRÉES....14
4.2.2 RÉSULTATS....15
4.3 BOUCLE....15
4.3.1 EXÉCUTION CYCLIQUE....15
4.3.2 EXÉCUTION SUR INTERRUPTION....16
4.4 DÉBOGAGE....16
5 BIBLIOTHÈQUE D'OBJETS....18
5.1 INTRODUCTION....18
5.2 INCLUSION....18
5.3 CHARGEMENT D'OBJETS....18
5.4 PROPRIÉTÉS DE L'OBJET....19
5.5 COMMANDES....21
5.6 RECHERCHE D'OBJETS....22
5.7 ÉDITION D'OBJETS....23
6 BIBLIOTHÈQUE ENVIRON....24
6.1 INTRODUCTION....24
6.2 INCLUSION....24
6.3 ENTRÉES ET SORTIES....24
6.4 IDENTIFIANT....25
BIBLIOTHÈQUE DE 7 MESSAGES....26
7.1 INTRODUCTION....26
7.2 INCLUSION....26

7.3 NOTIFICATIONS VIDÉO....26
7.4 NOTIFICATIONS PAR COURRIER....26
8 BIBLIOTHÈQUE JSON....28
8.1 INTRODUCTION....28
8.2 INCLUSION....28
8.3 DÉCODAGE....28
8.4 CODAGE....28
8.5 CRÉDITS....28
9 BIBLIOTHÈQUE D'URL....29
9.1 INTRODUCTION....29
9.2 INCLUSION....29
9.3 OBTENIR L'URL....29
9.4 CODAGE....29
10 ANNEXES....31
10.1 TYPES FONCTIONNELS....31
10.2 TYPES WEB (PHPCLASS)....32

1 INTRODUCTION

1.1 QU'EST-CE QUE LE SERVEUR QUANTUM

QUANTUM SERVER est un serveur web de supervision des systèmes domotiques et immotiques basé sur la technologie KNX. Sur le site il est possible de télécharger gratuitement des fiches techniques, des brochures et des manuels pour l'installation, la configuration et l'utilisation du QUANTUM SERVER.

1.2 OBJECTIF DE CE MANUEL

Ce manuel illustre les fonctions offertes par le moteur de script personnalisé du QUANTUM SERVER, grâce auquel il est possible d'étendre les fonctions de base du serveur Web ; Grâce aux scripts, vous pouvez :

- Interagir avec les objets de supervision, lire les états, exécutez la logique et envoyez des commandes
- Envoyer des messages de notification personnalisés
- Interagir avec le système d'exploitation
- Envoyer et recevoir des caractères via le réseau LAN et le port série

Le manuel guide le intégrateur de systèmes étape par étape dans la création d'un script personnalisé et dans son exécution dans l'environnement d'exécution QUANTUM SERVER ; les exemples présents sur le serveur web sont également analysés en détail, qui offrent un aperçu complet des fonctions offertes par cette technologie.

À partir de la version 1.0.1, QUANTUM SERVER vous permet de créer des scripts en langage PHP et LUA. Certains concepts sont communs aux deux environnements de développement, tandis que la syntaxe et les bibliothèques diffèrent ; pour un guide sur les scripts en PHP, reportez-vous au manuel spécifique.

1.3 EXIGENCES

Afin d'exploiter pleinement le potentiel du moteur de script QUANTUM SERVER, il est nécessaire d'avoir :

- QUANTUM SERVER mis à jour vers la version 1.0.1 ou supérieure
- Bonne connaissance de QUANTUM SERVER
- Bonne connaissance de la technologie ETS et KNX
- Connaissance, même à un niveau basique, du langage de script LUA

2 APERÇU

2.1 SCRIPTS

Les SCRIPTS sont des fichiers contenant du code qui peuvent être créés et stockés sur le disque flash QUANTUM SERVER via une section spéciale dans l'administration. espace. Ces fichiers peuvent être créés, modifiés et supprimés via un programme spécial éditeur, grâce auquel il est également possible de les exporter et de les importer sur un autre serveur Web.

Les fichiers restent mémorisés sur le disque QUANTUM SERVER et persistent même après une réinitialisation des paramètres d'usine ; par contre, ils ne sont pas inclus dans le fichier de sauvegarde du QUANTUM SERVER, ils doivent donc être exportés (puis importés) séparément si vous souhaitez transférer le projet vers un autre serveur web.

Les SCRIPTS doivent être créés en suivant les instructions contenues dans ce manuel, en utilisant les bibliothèques et les fonctions mises à disposition par le moteur de script ; tout code allant au-delà de ces indications peut potentiellement compromettre le bon fonctionnement du QUANTUM SERVER.

QUANTUM SERVER est déjà équipé d'une série d'exemples qui offrent un aperçu complet du potentiel offert par le système de script ; vous n'êtes pas censé modifier ou exécuter ces exemples directement, mais vous pouvez les dupliquer et travailler sur la copie sans aucune restriction.

2.2 RUNNERS

Une fois qu'un script personnalisé a été créé (de toutes pièces ou par duplication et modification d'un exemple), il est nécessaire de l'associer à un ou plusieurs objets de type « SCRIPT D'EXÉCUTION » ; ces objets particuliers ont pour tâche d'exécuter le code contenu dans le script auquel ils sont associés, à chaque fois

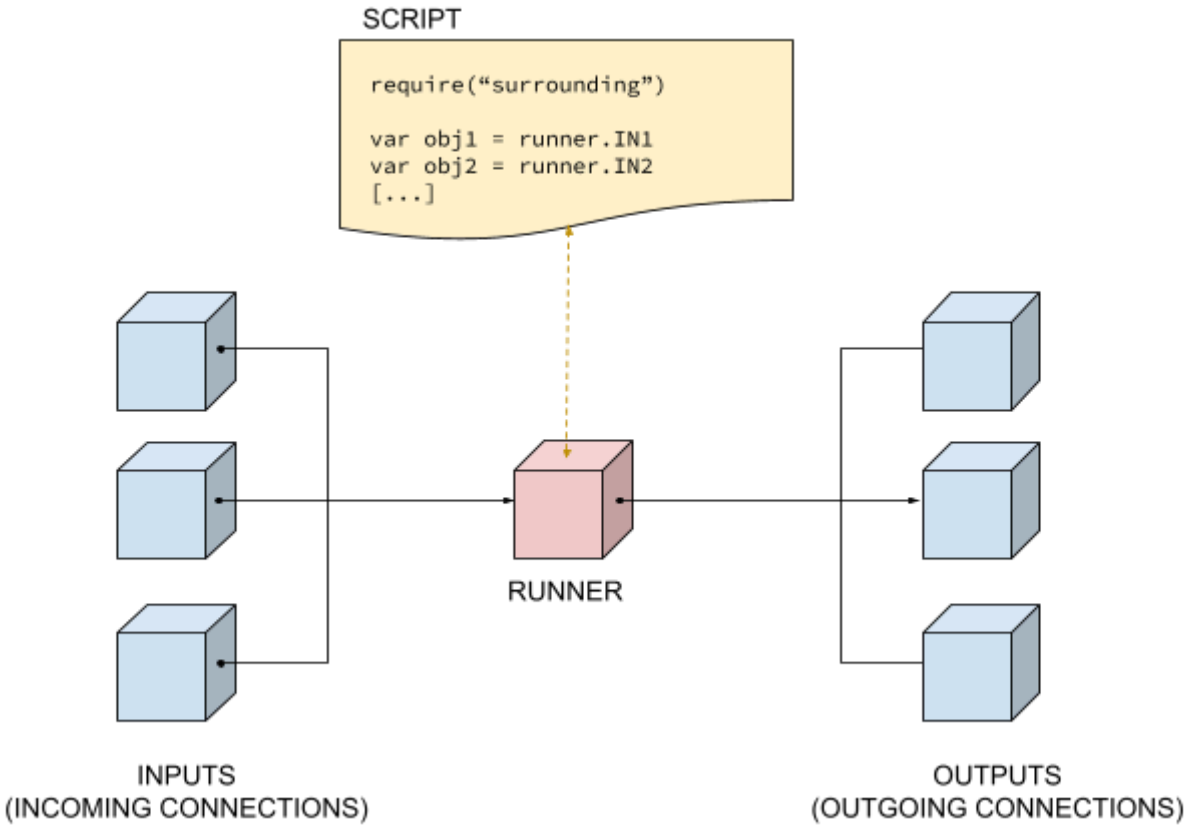
- sont contrôlés directement par l'utilisateur dans l'interface graphique du QUANTUM SERVER, en interagissant avec la "box" relative dans l'environnement ou les pages de fonctions
- changer la valeur d'un des objets associés comme CONNEXIONS ENTRANTES, de la même manière cela se produit pour tout autre type d'objet dans QUANTUM SERVER

Les SCRIPTS ne peuvent donc pas être exécutés directement, mais uniquement via un SCRIPT D'EXÉCUTION, qui se comporte de manière tout à fait analogue aux autres types d'objets : il possède une icône et une valeur associée, il peut être inséré dans des ENVIRONNEMENTS, des FONCTIONS ou des FAVORIS, il peut être associé à d'autres objets via ENTRANT ou SORTANT RELATIONS, faire partie d'un SCÉNARIO etc...

L'avantage d'avoir une structure basée sur des SCRIPTS et des SCRIPT D'EXÉCUTIONS distincts est double :

- Possibilité d'associer le même SCRIPT à plusieurs objets RUNNER, eux-mêmes insérés dans différentes logiques, événements, scénarios, etc... et donc capables de manipuler différentes données
- Possibilité d'exporter, d'importer et de partager des scripts quel que soit le projet créé

Le schéma suivant illustre en principe les interactions entre un SCRIPT D'EXÉCUTION et les objets associés via ÉVÉNEMENTS, ou sa commande via les pages RUNTIME du serveur web :



2.3 BIBLIOTHÈQUES

QUANTUM SERVER fournit une série de bibliothèques qui offrent des outils prêts à l'emploi pour interagir avec les principaux aspects couverts par le moteur de script. Les bibliothèques ne sont pas automatiquement incluses dans les scripts, mais doivent être appelées à l'aide de la commande LUA include

```
require(<nom de la bibliothèque>)
```

Exemple:

```
require("serial")
```

L'inclusion de chaque bibliothèque impliquant un temps de traitement, il est recommandé de n'inclure que les bibliothèques strictement nécessaires à la tâche que doit réaliser le script. Les bibliothèques peuvent à leur tour dépendre d'autres bibliothèques ; dans ce cas, il n'est pas nécessaire d'inclure également les bibliothèques associées, car l'inclusion se fait automatiquement en appelant la bibliothèque qui possède les dépendances.

Le tableau suivant présente une liste des bibliothèques LUA disponibles, une brève description et toutes les dépendances ; dans les chapitres suivants, ils seront examinés en détail.

Bibliothèque	Description
object	Si propose des méthodes pour charger et interagir avec les objets

surrounding	Il propose un ensemble d'objets pour interagir avec l'objet SCRIPT D'EXÉCUTION qui exécute le script et les éléments associés. ENTRÉES et RÉSULTATS
messages	Il vous permet d'envoyer des vidéos, des e-mails et des notifications push
json	If propose des méthodes pour encoder et décoder les chaînes JSON
url	Il permet d'appeler une URL externe et de gérer la réponse

2.4 ENTRÉES ET SORTIES

Lorsque l'objet SCRIPT D'EXÉCUTION est exécuté, il reçoit toujours une valeur INPUT ; en particulier:

- Dans le cas d'une commande directe depuis l'interface FRONTEND, la valeur est donnée par la saisie de l'utilisateur (par exemple en tapant une valeur dans une zone de texte, ou en appuyant sur un bouton - cela dépend de l'aspect graphique choisi pour le SCRIPT D'EXÉCUTION)
- Dans le cas d'une commande suite au changement d'état d'un des objets "parents" associés comme INPUT CONNECTIONS au SCRIPT D'EXÉCUTION, la valeur d'entrée est donnée par les propriétés de l'événement lui-même, telles que définies dans l'administration. En particulier, il est possible de passer comme valeur :
    - La valeur de l'objet qui change d'état, provoquant l'exécution de l'événement
    - Une valeur constante

Dans les deux cas, la valeur d'entrée est disponible dans le script en appelant la fonction

```
input()
```

Exemple:

```
val = input()
```

A la fin de l'exécution, le script peut renvoyer une valeur OUTPUT à l'environnement d'exécution ; cette valeur est définie comme la valeur de SCRIPT D'EXÉCUTION et peut être affichée dans le FRONTEND ou utilisée dans des exécutions ultérieures de SCRIPT D'EXÉCUTION lui-même. De plus, s'il existe au moins une CONNEXION DE SORTIE, cette valeur est transmise comme valeur d'entrée aux objets « enfants » associés.

Pour renvoyer une valeur à l'environnement d'exécution, appelez simplement la fonction

```
output()
```

Exemple:

```
output(1)
```



L’appel de la fonction output() termine l’exécution du script ; toute ligne de code suivante est ignorée.

Si vous ne souhaitez transmettre aucune valeur à l'environnement appelant, appelez simplement la fonction sans argument :

```
output();
```

Dans ce cas:

- Aucune valeur n'est définie pour SCRIPT D'EXÉCUTION
- Ils ne sont pas exécutés les CONNEXIONS DE SORTIE éventuellement associé au SCRIPT D'EXÉCUTION



Le fait de ne pas exécuter CONNEXIONS SORTANTES lors de l'appel de la fonction output() sans argument ne doit pas être considéré comme une limitation, mais comme plus de flexibilité.

En fait, il est possible d'exécuter manuellement les événements passifs à l'intérieur du script, en passant des valeurs différentes à chacun d'eux (ce qui serait autrement impossible via la valeur de sortie, la même pour tous), avant de terminer l'exécution du script ; dans ce cas, vous ne souhaitez évidemment pas que l'environnement appelant exécute à nouveau ces événements. Pour plus de détails, reportez-vous à la section sur la bibliothèque "**environnant**".

2.5 INCLUSION DE SCRIPTS

Il est possible d'inclure d'autres SCRIPT au sein d'un SCRIPT ; cette fonctionnalité vous permet d'utiliser des SCRIPT précédemment créés comme une sorte de « bibliothèque » personnalisée. Pour inclure un SCRIPT, appelez simplement la même fonction d'inclusion que celle utilisée pour les bibliothèques :

```
require("Nom du script")
```

Où **Nom du script** est le nom du SCRIPT à exécuter ; Par exemple:

```
require("procès")
```

2.6 INSPECTION VARIABLE

Il est possible d'inspecter la valeur des variables et des objets de code grâce à la fonction suivante :

```
printall
```

Exemple:

```
printall(variable)
```

Cette fonction affiche le contenu de la variable ou de l'objet ; s'il est structuré, son contenu est renvoyé sur plusieurs lignes, affichées dans la section DEBUG de la page des coureurs (voir les chapitres suivants pour plus de détails).

Pour les chaînes de texte uniquement, vous pouvez également utiliser la fonction système à cet effet

```
print(string)
```

2.7 TABLEAUX

Le *tableaux* sont des structures très flexibles dans le langage LUA, qui correspondent à peu près au *tableau* d'autres langues; *tableaux* cependant, ils peuvent également avoir des chaînes ou des clés non numériques comme indices, et sont donc assez similaires à *objets* d'autres langages de programmation (par exemple JavaScript).

De nombreuses bibliothèques de scripts LUA utilisent des tables pour manipuler et renvoyer des listes d'éléments ; les syntaxes suivantes peuvent être utiles pour travailler avec elles.

2.7.1 NOMBRE D'ÉLÉMENTS

Connaître le nombre d'éléments d'un *tableau* appelez la fonction suivante :

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

2.7.2 CYCLE ENTRE LES ÉLÉMENTS

Pour parcourir les éléments d'un *tableau* utilisez la syntaxe suivante :

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

Par exemple, pour faire défiler une liste d'objets, les filtrer en fonction d'un certain type et interagir avec eux, utilisez le code suivant :

```
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
```

Pour plus d'informations, reportez-vous aux chapitres suivants dédiés aux bibliothèques de gestion d'objets.

3 SCRIPTS

3.1 INTRODUCTION

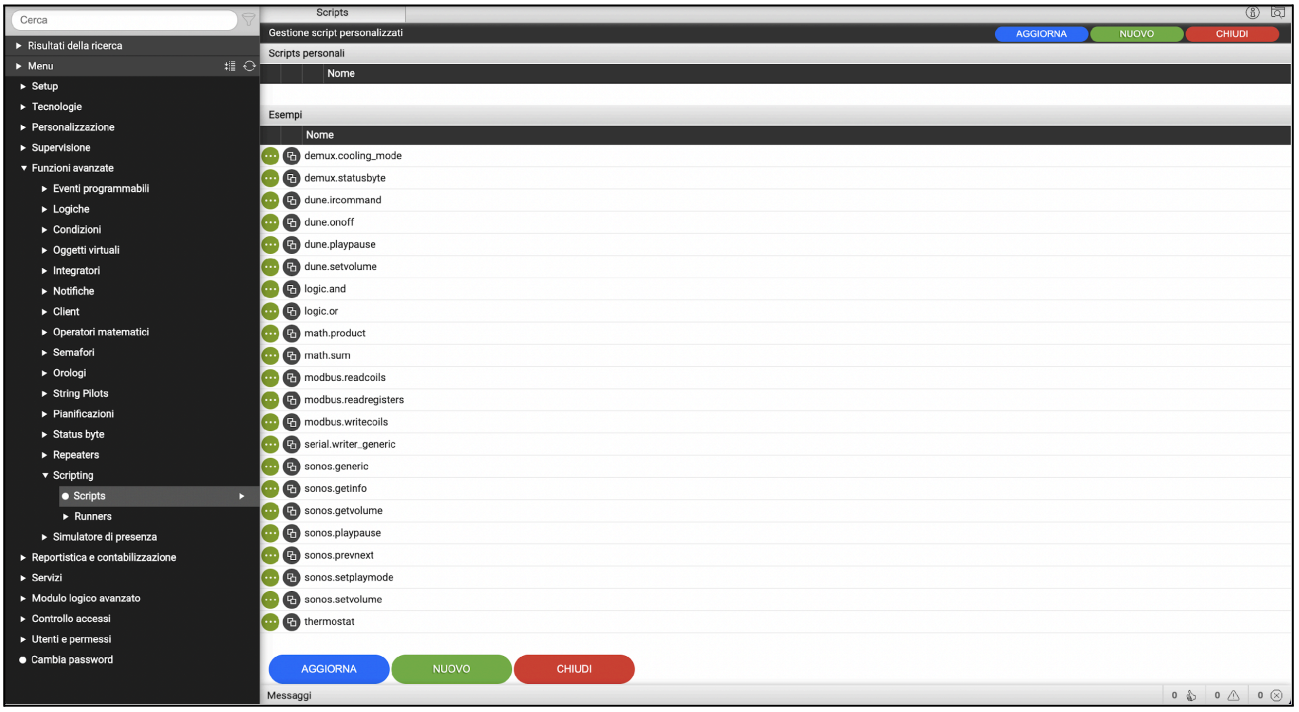
Ce chapitre explique comment créer, modifier, supprimer et exporter des scripts personnalisés sur QUANTUM SERVER. Les informations contenues dans ce chapitre s'appliquent aux scripts PHP et LUA, à l'exception des instructions ou exemples spécifiques au langage PHP.

3.2 LISTE

Pour accéder à la page de gestion des scripts, il suffit de :

- Connectez-vous à QUANTUM SERVER avec un utilisateur disposant des autorisations administratives
- Accéder à l'environnement CONFIGURATION
- Sélectionnez « FONCTIONS AVANCÉES » puis « SCRIPTING » dans le MENU LATÉRAL
- Sélectionnez "SCRIPTS"

Une page similaire à l'exemple suivant s'affiche :



La partie supérieure de la page contient une série d'exemples de scripts préchargés dans QUANTUM SERVER ; ils ne peuvent pas être supprimés ou modifiés, mais seulement consultés ou dupliqués (pour créer votre propre script à partir d'eux).

La partie inférieure montre à la place les scripts personnalisés présents sur le disque QUANTUM SERVER. Pour créer un nouveau script, appuyez simplement sur le bouton "NOUVEAU" en bas de page : il vous est demandé d'indiquer le nom à attribuer au nouveau script et, après quelques secondes, vous recevrez la confirmation que le nouveau fichier a été créé.

Si vous souhaitez créer un script à partir d'un exemple existant, vice versa, appuyez simplement sur la touche "correspondante “COPIER" bouton (gris) ; dans ce cas également, il vous est demandé de spécifier un nom à attribuer au nouveau script.

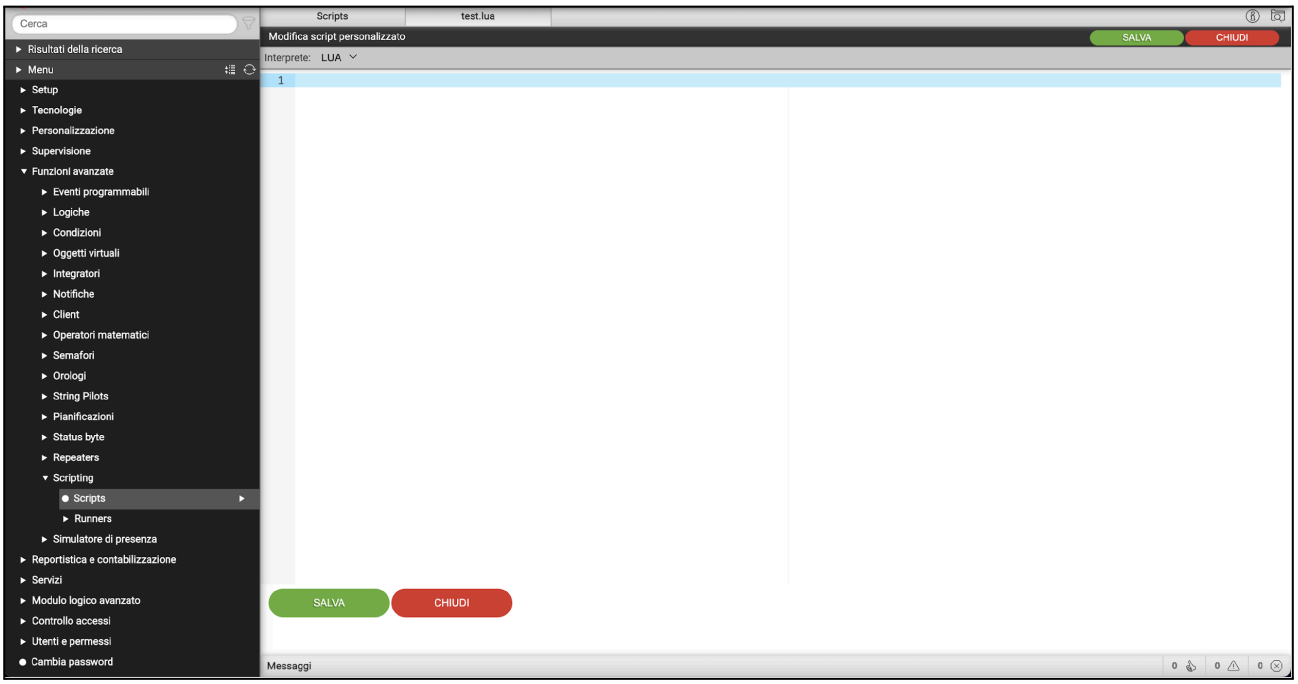
Pour supprimer un script, appuyez simplement sur le bouton "SUPPRIMER" (rouge) correspondant ; une fois la confirmation donnée, le fichier est supprimé du disque QUANTUM SERVER (de manière irréversible).

Enfin, pour accéder au formulaire de consultation et/ou de modification d'un script (ou d'un exemple), utilisez le bouton "EDIT" (vert).

3.3 ÉDITEUR

En sélectionnant le bouton MODIFIER d'un script ou d'un exemple, une page contenant le code qui constitue le script lui-même s'affiche ; dans le cas des exemples, le code n'est pas modifiable. Dans le cas d'un nouveau script, le code comporte des commentaires (identifiés par le préfixe "//") qui fournissent quelques premières indications sur la manière de créer le script, comme décrit plus en détail dans ce manuel.

La figure suivante montre la page d'édition d'un nouveau script :



Pour plus d'informations sur la création d'un script, notamment en exploitant les bibliothèques disponibles, veuillez vous référer aux chapitres suivants.

Une fois les modifications du code terminées, appuyez sur le bouton « ENREGISTRER » pour les rendre persistantes ; vice versa, en appuyant sur le bouton "FERMER" avant de sauvegarder, toutes les modifications seront perdues.

3.4 EXEMPLE

A titre d'exemple, dans la suite de ce manuel, sauf indication contraire, l'exemple de code suivant sera utilisé, qui renvoie simplement la valeur reçue comme INPUT VALUE, après avoir rapporté la même valeur dans un message DEBUG (visible uniquement pendant l'exécution simulation, comme souligné ci-dessous).

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

4 RUNNERS

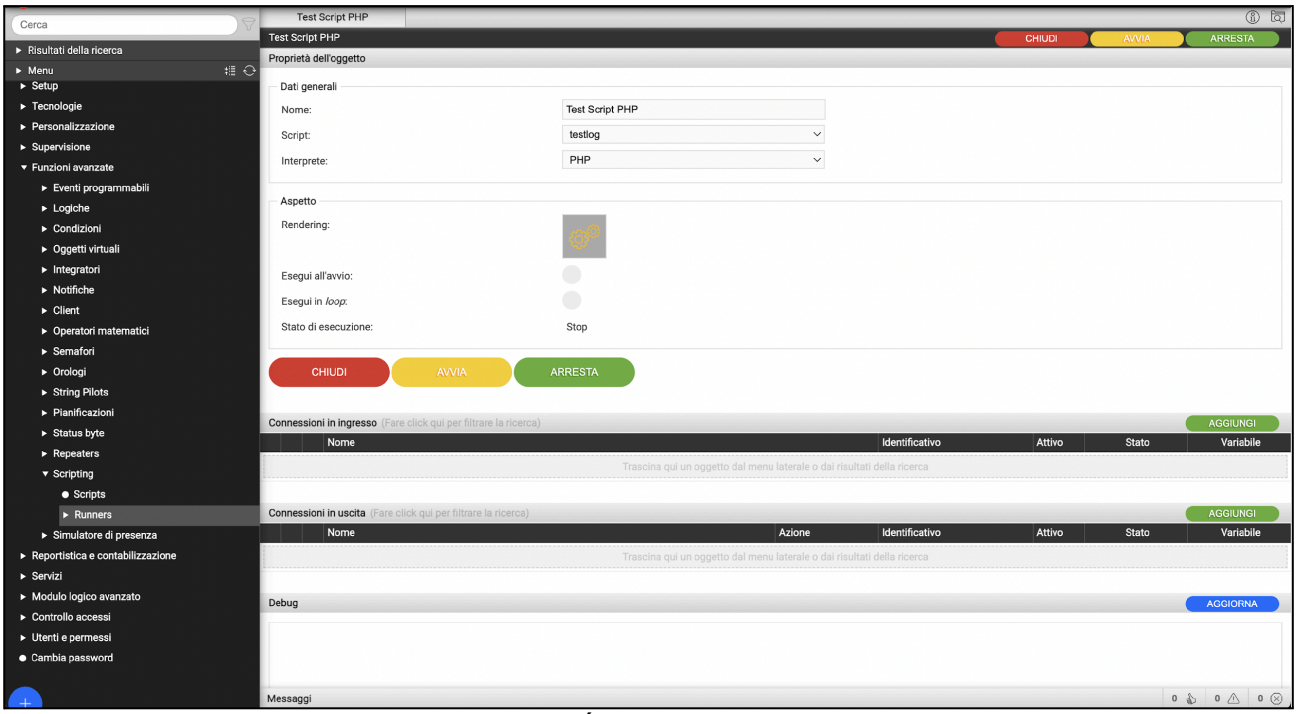
4.1 INTRODUCTION

Ce chapitre illustre comment créer et gérer des objets de type SCRIPT D'EXÉCUTION, qui permettent d'exécuter des SCRIPTS sous la supervision du QUANTUM SERVER. Les informations contenues dans ce chapitre s'appliquent aux scripts PHP et LUA, à l'exception des instructions ou exemples spécifiques au langage PHP.

4.2 NOUVEAU RUNNER

Pour créer un nouveau SCRIPT D'EXÉCUTION simplement :

- Connectez-vous à QUANTUM SERVER avec un utilisateur administrateur et accédez à la section de configuration
- Localisez l'élément "SCRIPTING" sous "FONCTIONNALITÉS AVANCÉES", puis sélectionnez "SCRIPT D'EXÉCUTIONS".
- Appuyez sur le bouton "AJOUTER" dans la barre d'outils ci-dessous
- Mettez en surbrillance le nouvel objet créé et appuyez sur les "points" sur le côté, ou - alternativement - sur le bouton "EDIT" dans la barre d'outils en bas, pour accéder à sa fiche



En plus d'attribuer un nom au SCRIPT D'EXÉCUTION, il est également nécessaire de sélectionner quel script sera exécuté, en le choisissant dans le menu déroulant approprié. De plus, il est possible de personnaliser l'aspect graphique avec lequel le SCRIPT D'EXÉCUTION, s'il est inséré dans une page graphique du FRONTEND, est affiché et permet à l'utilisateur d'interagir avec le SCRIPT associé.

Il est également possible de spécifier les options suivantes pour SCRIPT D'EXÉCUTION :

EXÉCUTER AU DÉMARRAGE	En sélectionnant ce flag, le SCRIPT D'EXÉCUTION est automatiquement exécuté au démarrage de la supervision
COURIR EN BOUCLE	En sélectionnant ce flag, le SCRIPT D'EXÉCUTION est exécuté de manière itérative un nombre de fois indéfini, jusqu'à son arrêt manuel ou via un EVENT. Pour plus de détails, veuillez vous référer au suivant "BOUCLE" section
TEMPS DE BOUCLE	En cas d'exécution en boucle, spécifiez la durée en millisecondes de chaque itération.

Il est également possible (à partir de la version 1.1.0 du logiciel) d'exécuter ou d'arrêter un SCRIPT D'EXÉCUTION à l'aide des boutons « START » et « STOP » appropriés.

4.2.1 ENTRÉES

La rubrique **ENTRANT RELATIONS** permet de glisser un ou plusieurs objets qui, lorsque l'état change, déterminent l'exécution de SCRIPT D'EXÉCUTION.

Une fois que vous avez glissé un élément dans cette liste, vous devez décider lequel **VALEUR** doit être transmis au SCRIPT D'EXÉCUTION chaque fois que l'objet lui-même change d'état.

Contrairement à d'autres parties du logiciel où ce choix est guidé par des menus déroulants spéciaux, dans ce cas, la valeur doit être spécifiée manuellement. Ce champ (visible uniquement si vous activez la vue AVANCÉE via la barre d'outils en haut à droite) peut contenir les valeurs suivantes :

$VAL	La valeur actuelle de l'objet est transmise au SCRIPT D'EXÉCUTION en tant que valeur d'entrée. Option par défaut.
$NVAL	La valeur inversé de l'objet est transmis au SCRIPT D'EXÉCUTION en tant que valeur d'entrée (il a du sens dans le cas de valeurs booléennes)
Toute valeur numérique ou chaîne	La valeur est transmise au SCRIPT D'EXÉCUTION en tant que valeur d'entrée, quel que soit l'état de l'objet lui-même.

Dans tous les cas, la valeur est disponible dans le script via l'appel

```
input()
```

déjà décrit ci-dessus. De plus, dans le cas de scripts associés à plusieurs objets d'entrée, il est possible d'obtenir l'ID de l'objet qui a changé (et qui transmet la valeur d'entrée) via l'appel de fonction.

```
callerid()
```



La valeur indiquée dans ce champ n'est pas le seul moyen d'accéder, au sein du SCRIPT, à la valeur de l'objet associé en entrée. À travers le **environnant** bibliothèque en effet, comme mieux décrit ci-dessous, il est possible de connaître

l'état de toutes les entrées et d'effectuer des opérations sur elles ; pour plus de détails, reportez-vous au chapitre dédié à cette bibliothèque.

4.2.2 RÉSULTATS

De même, faire glisser des objets dans la section **CONNEXIONS SORTANTES** il est possible de déterminer son exécution à la fin du SCRIPT. Pour chaque objet, il est nécessaire de spécifier quelle action exécuter et (si l'objet le prévoit) quelle valeur transmettre à la fin de l'exécution de SCRIPT D'EXÉCUTION ; dans ce dernier cas, choisissez "VALUE OF THE SCRIPT D'EXÉCUTION" pour passer à l'objet la valeur renvoyée en sortie du SCRIPT avec la commande

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



Comme souligné précédemment, il est possible d'inhiber l'exécution automatique des objets associés en sorties en fin de SCRIPT, en ne transmettant aucune valeur à la fonction. **sortir()**. Dans ce cas, la commande éventuelle des objets associés en sorties est confiée au code SCRIPT lui-même, comme mieux détaillé dans le chapitre dédié à la bibliothèque « environnante ».

Également pour les sorties, il est possible d'associer un identifiant et affichez le nom de la variable avec laquelle faire référence à la sortie dans le SCRIPT.

4.3 BOUCLE

Un script peut être exécuté en continu en "arrière-plan" en activant l'option *boucle* option dans l'onglet SCRIPT D'EXÉCUTION. Généralement, lorsque vous activez cette option, vous activez également le *exécuter au démarrage* option, afin que le code soit réexécuté si le système redémarre.

4.3.1 EXÉCUTION CYCLIQUE

Dans *boucle* mode, la fonction suivante est appelée cycliquement :

```
main_loop()
```

que vous devez définir dans le script, par exemple :

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

Cette fonction est appelée périodiquement avec un temps d'attente, entre une exécution et la suivante, à définir dans le *temps de boucle* option (préciser une valeur exprimée en millisecondes) ; plus le temps réglé est bas, plus la réactivité du RUNNER est élevée, mais aussi la charge sur le CPU du serveur web.

4.3.2 EXÉCUTION À L'INTERRUPTION

Il est possible de définir une autre fonction dans le script, qui est appelée à chaque fois qu'une des entrées change d'état : par exemple...

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

L'argument de la fonction est un *tableau* contenant les informations suivantes :

demandeur	ID de l'objet entrant qui a changé d'état
valeur	Valeur reçue de l'objet
code de hachage	Tout code PIN transmis par le sujet Alternativement, la chaîne "NO-HASHCODE" est transmise
options	Concaténation des attributs de la relation entre l'objet en question et le RUNNER Si un IDENTIFIANT a été spécifié pour l'objet d'entrée, il est fourni via l'attribut "tag_complex_element"

Exemple de sortie du *imprimer tout* sur l'argument de la fonction :

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

Où

123	ID de l'objet appelant
1	Valeur reçue
CUISINE	Identifiant associé à l'objet dans la carte RUNNER

Cette fonction peut être mieux exploitée en incluant la bibliothèque SURROUNDING présentée plus loin dans ce manuel, qui fournit des méthodes pour interagir plus confortablement avec les ENTRÉES du RUNNER.

4.4 DÉBOGAGE

Il est possible d'insérer dans les scripts des messages DEBUG (destinés à surveiller les activités réalisées par le script en construction), WARN (utile pour notifier à l'environnement d'exécution des messages d'avertissement) et des messages ERROR (nécessaires pour signaler et tracer

d'éventuelles anomalies au cours de l'exécution). exécution du script qui compromettent son achèvement correct).

Pour insérer ces messages, il suffit d'appeler respectivement une des fonctions suivantes :

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

Exemple:

```
system.debug("Ceci est un message de débogage")
system.info("Ceci est un message d'information")
system.warn("Ceci est un message d'avertissement")
system.err("Ceci est un message d'erreur")
```

Ces messages sont affichés dans la section DEBUG, colorés différemment selon le niveau.

5 BIBLIOTHÈQUE D'OBJETS

5.1 INTRODUCTION

La bibliothèque LUA « objets » met à disposition une classe avec un ensemble de méthodes pour interagir avec les objets du projet de supervision.

Cette bibliothèque est la base de la bibliothèque "SURROUNDING", qui permet une interaction facile avec les ENTRÉES et SORTIES du RUNNER, sujet du prochain chapitre de ce manuel.

5.2 INCLUSION

Pour inclure la bibliothèque dans un SCRIPT il faut insérer la ligne suivante au début :

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

L'inclusion de la bibliothèque fournit une classe appelée "gestionnaire d'objets" accessible via le mot-clé

```
objM
```

dont les méthodes sont décrites ci-dessous.

5.3 CHARGEMENT D'OBJETS

Pour charger un seul objet de supervision dans une variable LUA il faut appeler la méthode suivante :

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

où XXX est l'ID numérique de l'objet à charger et <OBJ> est n'importe quel nom de variable auquel attribuer l'objet. Exemple:

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

Si l'objet a déjà été appelé auparavant, cette méthode vous donne un *exemple*; sinon, une requête est faite à la base de données du projet, et en plus de l'attribuer à la variable, elle est enregistrée dans un *collection* interne au script, pour les requêtes ultérieures.

Si vous souhaitez charger plusieurs objets simultanément, vous pouvez appeler la méthode

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

auquel il faut passer un tableau (*tableau*) d'identifiants d'autant d'objets, par exemple :

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

La fonction renvoie un tableau contenant les objets demandés, dans lequel l'index est représenté par l'ID ; dans l'exemple précédent il est donc possible d'accéder ultérieurement à un seul objet de la collection grâce à la syntaxe

```
CUISINE_LIGHT = OBJECTS[123]
```

Les objets obtenus par chargement multiple, comme méthode alternative, peut être ensuite récupéré grâce à la méthode de chargement unique vue précédemment...

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

… sans autres lectures de la base de données du projet (étant déjà dans le *collection* interne au script).

5.4 PROPRIÉTÉS DE L'OBJET

Les objets chargés avec les fonctions vues précédemment mettent à disposition une série d'attributs primaires, présentés dans le tableau suivant.

Les variables qui représentent les objets du projet de supervision ont les propriétés suivantes, qui peuvent être utilisées dans le SCRIPT (attention à la casse qui doit être respectée pour une syntaxe correcte) :

PROPRIÉTÉ	DESCRIPTION	EXEMPLE
IDENTIFIANT	ID unique de l'objet	`<OBJ>.ID`
TAPER	Objet de type « fonctionnel », utilisé par les services de communication. Quelques exemples : Objet KNX : "EIBOBJECT" Variable : "OBJET VIRTUEL" Scène: "SCÉNARIO" Exécuteurs de script : "CSCMD" *REMARQUE : Pour une liste complète des types, voir l'annexe 10.1.*	`<OBJ>.TYPE`
PHPCLASS	Type d'objet graphique, utilisé par l'interface web. Quelques exemples : Objets KNX : "dpadEibObject" Variables : "dpadVirtualObject" Scénarios : "dpadScenario" Exécuteurs de script : "dpadScriptRunner" *REMARQUE : il n'y a pas toujours de correspondance unique entre « type » et « phpclass » : dans certains cas, plusieurs « phpclass » partagent le même « type ». Pour une liste complète, veuillez vous référer à l'Annexe 10.2*	`<OBJ>.PHPCLASS`

VALUES_TYPE	Type de données associé à l'objet. Le codage de ce champ dépend du TYPE.	<OBJ>.VALUES_TYPE
Base de données	Collection de tous les champs de la base de données liés à l'objet en question. Voir ci-dessous pour plus de détails	<OBJ>.DB
statut	État actuel de l'objet. REMARQUE : cette propriété n'a de signification que pour les scénarios ; dans ce cas: 1 → scénario arrêté 2 → scénario d'exécution	<OBJ>.status
valeur	Valeur actuelle de l'objet REMARQUE : cette propriété a une signification pour tous les types d'objets qui ont un statut, scénarios exclus	<OBJ>.value

La collection DB contient la liste de tous les champs de la base de données relatifs à l'objet ; ils peuvent être lus via la syntaxe

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

Par exemple:

```
CUISINE_LIGHT =objM.getObj(123)
LIGHT_NAME =CUISINE_LIGHT.DB.NAME        -- Renvoie par exemple "Kitchen Light"
```

Ci-dessous une liste des principaux champs :

CHAMP	DESCRIPTION
NAME	Nom de l'objet
DESCRIPTION	Description de l'article
MSP	Attribut principal de l'objet ; la signification de ce champ dépend du TYPE, il contient généralement le *adresse* (par exemple adresse de groupe KNX)
UNIT	Lorsqu'il est disponible, il représente l'unité de mesure
IS_WRITABLE IS_READABLE	Indicateur d'autorisation respectivement pour la lecture de commandes et d'objets
IS_VISIBLE	Indicateur de visibilité des objets
IS_STOPPABLE	Indicateur d'autorisation Object STOP (valable uniquement pour les SCÉNARIOS)
IS_BOOLEAN	Indicateur qui établit si l'objet est de type BOOLEAN (c'est-à-dire qu'il n'admet que les valeurs 1 ou 0) ou non
IS_PROTECTED	Indicateur de protection des objets contre les modifications apportées par l'interface utilisateur

ENABLE_FLAG	Indicateur d'activation générale de l'objet pour l'opération et l'interaction avec d'autres objets
TRACK_FLAG	Indicateur de suivi d'objet dans la base de données de journalisation

Enfin, une sous-collection de *choix* est accessible avec ce qui suit syntaxe

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

Par exemple:

```
CUISINE_LIGHT =objM.getObj(123)
CATÉGORIE =CUISINE_LIGHT.DB.options.category     -- Renvoie par exemple "éclairage"
```

Pour une ventilation complète de tous les champs, il est recommandé d'inspecter l'objet via la méthode *imprimer tout* documenté ci-dessus, car la liste des options disponibles et leurs valeurs dépendent fortement du type d'objet.

5.5 COMMANDES

Il est possible d'envoyer une commande à un objet, en définissant une nouvelle valeur, via la méthode

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

Où

<VALUE>	Nouvelle valeur transmise à l'objet
<ACTION>	Action à appeler (facultatif, par défaut : "SETVALUE")

Par exemple:

```
CUISINE_LIGHT =objM.getObj(123)
CUISINE_LIGHT:set(1)      -- Allume la lumière
```

Les actions envisagées, et leurs éventuelles valeurs, pour les principaux types d'objets sont présentées dans le tableau suivant :

TAPER	ACTION	DESCRIPTION	VALEUR
Objet KNX	SETVALUE	Envoyer une commande au groupe KNX correspondant	Cela dépend du type d'objet KNX Est: 0 → éteint 1 → activé 50 → taxe à 50%
	GETVALUE	Soumettre une demande de retransmission de statut	Aucune valeur

Scénario	EXECUTE	Exécuter le scénario	Aucune valeur
	STOPEXECUTION	Arrêter le scénario (s'il est en cours d'exécution)	Aucune valeur
Notification vidéo	EVALUATE	Insère le message dans la table du journal, afin qu'il apparaisse dans le centre de messagerie	Aucune valeur
Notification par courrier électronique	CALCULATE	Envoyer le message par e-mail	Aucune valeur
Supplément	INSERT	Recalcule la valeur mise à jour de l'intégrateur	Aucune valeur
	SENDMAIL	Réinitialise la valeur de l'intégrateur à 0	Aucune valeur
Variable	INTEGRATE	Définir la valeur de la variable	Basé sur le type de variable créée
	RESET		
	SETVALUE		

5.6 RECHERCHE D'OBJETS

Il est possible de rechercher un ou plusieurs objets du projet à partir d'un des champs de la base de données listés ci-dessus ; pour obtenir un seul objet, utilisez la méthode

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

Où

<KEY> Mot-clé, à spécifier sous forme de chaîne

<FIELD> Nom du champ de la base de données à rechercher

<EXACT_MATCH> Indicateur qui spécifie si le mot-clé doit correspondre exactement au champ de la base de données ou si une partie de la valeur de la base de données est suffisante pour correspondre au mot-clé.

Spécifiez cet indicateur au format booléen *vrai* ou *FAUX*

Exemple:

```
CUISINE_LIGHT = objM.findObj("Lumière de cuisine","NAME",true)
```

Également dans le cas de la recherche, il est possible de demander le chargement de plus d'objets répondant au critère de recherche, via la méthode

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

Le premier argument <KEYS> peut être un seul mot-clé (auquel cas il doit être passé sous forme de chaîne) ou un tableau de mots-clés ; dans le second cas, à partir du champ <EXACT_MATCH>, les types de recherches suivants sont effectués respectivement dans la base de données :

<EXACT_MATCH> = vrai	Tous les objets qui ont exactement dans le champ <FIELD> les valeurs contenues dans le tableau <KEYS> sont chargés
<EXACT_MATCH> = faux	Tous les objets qui contiennent dans le champ <FIELD> au moins un des mots-clés spécifiés dans <KEYS> sont recherchés.

Exemples :

```
- Rechercher tous les objets contenant un mot-clé dans le nom
<TABLEAU> = objM.findObjs("Lumière","NAME",false)
```

```
- Rechercher une liste spécifique d'objets par nom
<TABLEAU> = objM.findObjs({"Lumière de cuisine","Lumière de salon"},"NAME",true)
```

5.7 ÉDITION D'OBJETS

Il est possible de modifier par programmation la valeur d'un champ dans la base de données relatif à un objet, grâce à la méthode

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

Où

<ID>	ID de l'objet à modifier
<FIELD>	Nom du champ de la base de données à modifier
<VALUE>	Nouvelle valeur à attribuer au champ. Passez cet argument sous forme de chaîne.

La valeur de retour de la méthode informe de la modification réussie (*true*) ou avec une erreur (*false*).

6 BIBLIOTHÈQUE SURROUNDING

6.1 INTRODUCTION

La bibliothèque LUA "SURROUNDING" permet d'interagir facilement avec les ENTRÉES et SORTIES du RUNNER, en étendant et en spécialisant les méthodes vues précédemment de la bibliothèque "OBJETS".

6.2 INCLUSION

Pour inclure la bibliothèque dans un SCRIPT il faut insérer la ligne suivante au début :

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

6.3 ENTRÉES ET SORTIES

A partir de la ligne suivant immédiatement l'inclusion de la bibliothèque, l'objet suivant est disponible pour le SCRIPT :

```
runner
```

Cet objet représente le SCRIPT D'EXÉCUTION qui exécute le SCRIPT et vous permet d'interagir avec lui au sein du SCRIPT. L'objet met également à disposition ses relations possibles avec les INPUTS ("parents") et OUTPUTS ("enfants") configurés dans le formulaire SCRIPT D'EXÉCUTION (voir chapitre 4.2) à travers les tableaux respectifs

```
runner.INPUTS   //Entrées
runner.OUTPUTS  //Sorties
```

Au sein des tableaux, les différents objets sont représentés par la variable "OBJ_XXX" où XXX est l'ID de l'objet, par exemple :

```
runner.INPUTS.OBJ_123   //Objet entrant avec l'ID 123
```

Ces objets fournissent les propriétés et méthodes décrites dans le chapitre précédent.

De plus, ils fournissent un *collection* contenant des informations à propos la RELATION avec le COUREUR :

PROPRIÉTÉ	DESCRIPTION	EXEMPLE
RELATION	Collection d'attributs de la relation avec le RUNNER	<OBJ>.RELATION

Les champs du rapport sont répertoriés dans le tableau suivant :

CHAMP	DESCRIPTION
ACTION	Action effectuée sur l'objet cible de la relation
ACTION_VALUE	Valeur transmise à l'objet cible
options	Collection de valeurs de relation facultatives, notamment : tag_complex_element : ID attribué à l'ENTRÉE / SORTIE

6.4 IDENTIFIANT

Il est possible d'associer à chaque objet un **identifiant** (qui doit être une chaîne sans espaces ni caractères spéciaux) ce qui facilite son identification au sein du SCRIPT, lors de l'utilisation de la bibliothèque SURROUNDING. Par exemple, si l'identifiant « TEST » est associé à un objet, il est disponible dans le code en écrivant

```
runner.TEST
```

Pour connaître l'ID d'un objet INPUT ou OUTPUT à un RUNNER connu son ID, il est possible d'appeler les fonctions suivantes :

```
runner.getIdentifierById(id)              -- Recherche d'identification entre ENTRÉES et SORTIES
runner.getInputIdentifierById(id)      -- Recherche d'identification uniquement parmi les ENTRÉES
runner.getOutputIdentifierById(id)     -- Recherche d'identification uniquement parmi les SORTIES
```

Par exemple, à l'intérieur d'un RUNNER dans *boucle* il est possible d'intercepter le changement d'état d'une entrée, de l'identifier et de la gérer avec l'exemple de code suivant :

```
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 BIBLIOTHÈQUE DE MESSAGES

7.1 INTRODUCTION

La bibliothèque LUA « messages » permet d'envoyer des notifications vidéo et email directement au sein d'un SCRIPT, sans nécessairement avoir à créer des objets au sein du projet de supervision.

7.2 INCLUSION

Pour inclure la bibliothèque dans un SCRIPT, insérez simplement la ligne suivante au début :

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

À partir de la ligne immédiatement suivante, la classe suivante est disponible pour le SCRIPT :

```
msgM
```

Cette classe (« gestionnaire de messages ») fournit une série de fonctions (ou « méthodes ») permettant de charger des objets de supervision et d'effectuer des opérations sur ceux-ci.

7.3 NOTIFICATIONS VIDÉO

Pour envoyer une nouvelle notification vidéo "à la volée" pendant qu'un script est en cours d'exécution, il suffit d'appeler la fonction

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

Où

message	Message à envoyer
level	Niveau de priorité des messages. Valeurs possibles :
	0 Alarme (par défaut)
	1 Avertissement
	2 Information

Exemple:

```
msgM.sendVideoMsg("Valeur d'entrée reçue " .. input() .. ",2);
```

La fonction renvoie 0 en cas de succès, un code d'erreur dans le cas contraire.

7.4 NOTIFICATIONS PAR COURRIER

De même, un message électronique peut être envoyé « à la volée » via l'appel

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

Où

message	Message à envoyer
to	Adresse(s) e-mail à laquelle envoyer le message En cas de plusieurs adresses, séparez-les par ";"
subject	Objet du message [Facultatif]
cc	Adresse(s) e-mail en copie [Facultatif]
bcc	Adresse(s) e-mail en copie aveugle [Facultatif]

La fonction renvoie 0 en cas de succès, un code d'erreur dans le cas contraire.

8 BIBLIOTHÈQUE JSON

8.1 INTRODUCTION

Cette bibliothèque permet de manipuler des chaînes au format JSON en les transformant en structures de données de type *tableau* et LUA.

8.2 INCLUSION

Pour inclure la bibliothèque dans un SCRIPT, insérez simplement la ligne suivante au début :

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

8.3 DÉCODER

Une fois la bibliothèque incluse et affectée à la classe JSON dans le code, il est possible de décoder une chaîne au format JSON grâce à la méthode

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

Exemple:

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

8.4 CODAGE

De même, il est possible à partir d'un *tableau* Structure LUA pour obtenir une représentation JSON dans une chaîne avec la méthode

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

Exemple:

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

8.5 CRÉDITS

Cette bibliothèque est un projet OPEN-SOURCE ; pour plus de détails, veuillez vous référer à :

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

9 BIBLIOTHÈQUE D'URL

9.1 INTRODUCTION

Cette bibliothèque vous permet d'appeler une URL et de gérer la réponse. Il peut être utilisé pour invoquer l'API HTTP(S) via des appels GET et décoder le résultat (par exemple à partir du format JSON).

9.2 INCLUSION

Pour inclure la bibliothèque dans un SCRIPT, insérez simplement la ligne suivante au début :

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

La bibliothèque fournit une classe "Gestionnaire d'URL" qui peut être appelée avec le mot-clé

```
urlM
```

9.3 OBTENIR L'URL

Pour récupérer une URL via GET, et affecter la réponse à une variable, utilisez la syntaxe suivante :

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

Exemple:

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


-- Returns your own IP address
-- 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 ANNEXES

10.1 TYPES FONCTIONNELS

Le tableau suivant présente une liste complète de tous les types fonctionnels de objets fourni par QUANTUM SERVER, utilisé par les services de communication pour effectuer les opérations « en arrière-plan » du serveur web. Nous appelons généralement ce type d'objet « TYPE ».

Pour chacun d'eux, outre le type et la description, les actions possibles sont rapportées ; cette dernière peut être appelée avec les méthodes vues précédemment pour la commande des objets.

TYPE (TYPE)	DESCRIPTION	ACTES
VIRTUALOBJECT	Variable	SETVALUE → Définir une valeur pour l'objet
SCENARIO	Scénario	EXECUTE → Exécuter le scénario STOPEXECUTION → Arrêter l'exécution
LOGIC	Logique	EVALUATE → Devise à la logique
CONDITION	Condition	CALCULATE → Calculer la comparaison des valeurs
CLIENTBROWSER	Client	REDIRECT → Faire un saut de page sur le client VOIPCALL → Passer un appel VOIP vers l'adresse IP du client
NOTIFY_VIDEO	Notification vidéo	INSERT → Créer un message vidéo
NOTIFY_EMAIL	Notification par courrier électronique	SENDMAIL → Envoyer un e-mail
INTEGRATOR	Supplément	INTEGRATE → Mettre à jour le calcul de l'intégrateur RESET → Réinitialise la valeur de l'intégrateur à 0
EIBOBJECT	Objet KNX	SETVALUE → Définir une valeur sur l'adresse de groupe KNX correspondante GETVALUE → Emet une requête sur le bus KNX pour lire l'état mis à jour
CSCMD	Exécuteur de script	RUN → Exécuter le script QUIT → Terminer l'exécution du script
USER	Utilisateur	REDIRECT → Faire un saut de page sur tous les clients où l'utilisateur est connecté VOIPCALL → Passer un appel VOIP vers tous les clients auxquels l'utilisateur est connecté
USERGROUP	Groupe d'utilisateurs	
USERPERMISSION	Permis	

CONTAINER	Objet composé	
CAMERA	Caméra IP	
PBXELEMENT	téléphone IP Interphone vidéo IP Groupe d'appel VOIP	VOIPCALL → Passer un appel VOIP à destination de l'objet

10.2 TYPES DE WEB (CLASSE PHP)

Le tableau suivant présente une liste complète de tous les types d'objets attendus par QUANTUM SERVER concernant l'interface Web. Généralement, ce type d'objet est appelé « PHPCLASS ».

Pour chacun d'eux, le type fonctionnel (ou "TYPE") correspondant est rapporté ; il est possible que plusieurs PHPCLASS fassent référence au même TYPE. S'il n'est pas présent, le PHPCLASS n'a pas d'équivalent fonctionnel pour les services de communication ; cela se produit généralement pour tous les types de sites Web ayant une fonction purement graphique (environnements, liens, etc...)

TYPE (PHPCLASS)	**DESCRIPTION**	**TYPE DE RÉFÉRENCE**
dpadObject	objet générique *	
dpadGroup	Environnement (ou « dossier » au sens général)	
dpadUrl	Connexion	
dpadSysCmd	Commande système *	SYSCMD
dpadSysCmdWait	Commande d'attente de scénario	SYSCMD
dpadUser	Utilisateur	USER
dpadUserGroup	Groupe d'utilisateurs	USERGROUP
dpadUserPermission	Autorisation d'accès pour les groupes d'utilisateurs	USERPERMISSION
dpadTriggerObject	Déclencheur système lors du changement de base de données *	TRIGGEROBJECT
dpadContainer	Objet composé	CONTAINER
dpadVirtualObject	Variable	VIRTUALOBJECT
dpadScenario	Scénario	SCENARIO
dpadPbxExtension	téléphone IP	PBXELEMENT
dpadPbxTrunk	Ligne téléphonique VOIP	

dpadPbxQueue	Groupe d'appel VOIP	
dpadPbxDoorOpener	Interphone vidéo IP	
dpadLogic	Logique	LOGIC
dpadCondition	Condition	CONDITION
dpadClientBrowser	Client	CLIENTBROWSER
dpadNotifyVideo	Notification vidéo	NOTIFY_VIDEO
dpadNotifyEmail	Notification par courrier électronique	NOTIFY_EMAIL
dpadIntegrator	supplément	INTEGRATOR
dpadEibObject	Objet KNX	EIBOBJECT
dpadCamera	Caméra IP	CAMERA
dpadScriptRunner	Exécuteur de script	CSCMD
dpadLoadControl	Contrôle de charge	CSCMD

** : types d'objets habituellement peu utilisés ou disponibles dans l'interface web*