fellerLYnk

Création d'applications pour fellerLYnk

10.FLYNKAPP-F.1804/180611

by Schneider Electric

1 Introduction 5
2 Accéder à l'application 5
3 Télécharger l'application 5
4 Bibliothèques/frameworks disponibles 5
5 Structure du répertoire de base 5
6 Mode plein écran 6
7 Structure des applications / widgets 6
8 Structure des répertoires 6
9 Taille des widgets 7
10 Daemons 9
11 Configuration 9
12 Surveillance du bus local côté client 13
13 Surveillance du bus local côté serveur 14
14 Traduction 16
15 Scripts LP 17
16 Fonctions Lua 20
16.1 Fonctions d'objet 20
16.2 Helpers 20
16.3 Demandes de bus 21
16.4 Manipulation des balises 21
16.5 Création et modification d'objets 21
16.6 Fonctions de base de données 22
16.7 Fonctions de base 23
16.8 Helpers INSERT/UPDATE/DELETE 23
16.9 Helpers SELECT 23

1 Introduction

Ce document décrit comment créer des applications pour fellerLYnk. L'app store sera implémenté dans le firmware 2.0. Le framework pour créer et jouer avec les applications est déjà implémenté dans fellerLYnk 1.2.0.

2 Accéder à l'application

Pour lancer l'application, il faut fournir l'url directe.

http://192.168.1.10/data/myapp.lp

http://192.168.1.10/user/index.html

3 Télécharger l'application

SL/HL dispose d'un serveur ftp intégré qui doit être activé dans les services système.

Il y a des utilisateurs 'ftp' et 'applications'. Connectez-vous à l'utilisateur d'applications pour télécharger l'application. Le mot de passe par défaut est 'apps'.

4 Bibliothèques/frameworks disponibles

- jQuery v2 (http://jquery.com/)
  /apps/js/jquery.js.gz
- Bootstrap v3 (http://getbootstrap.com)
  /apps/css/bootstrap.css.gz
  /apps/js/bootstrap.js.gz
- Font Awesome v4 (http://fontawesome.io)
  /apps/css/font-awesome.css.gz
- Windows 10 Icons by Icons8 (http://icons8.github.io/windows-10-icons/)
  /apps/css/icons8-win10.css.gz

Bootstrap est livré sans *Glyphicons*, utiliser *Font Awesome* à la place.

5 Structure du répertoire de base

- /data – les applications et widgets sont stockés ici, accessibles à l'adresse http://IP/apps/data/
- /libs – stockage de la bibliothèque Lua, chargé via *require('custom.lib')* où *lib* est le nom de la bibliothèque.
- /daemon – les daemons des applications sont stockés ici
- /user – permet de stocker les fichiers utilisateurs et les scripts LP, accessibles à l'adresse http://IP/user/

6 Mode plein écran

Pour masquer les icônes de la barre de navigation supérieure, passez la variable GET fs (le contenu de la variable doit être *true*): http://IP/apps/?fs=1

7 Structure des applications / widgets

Le nom de l'application (ID) doit être unique et ne peut contenir que des caractères alphanumériques, des traits d'union et des tirets bas. La longueur maximale du nom est de 64 caractères.

8 Structure des répertoires

- *index.lp* ou *index.html* – requis pour les applications, sauf si *url* est spécifié, un clic sur l'icône de l'application ouvre le répertoire de l'application dans la même fenêtre. Les applications doivent fournir un bouton *Retour* pour permettre à l'utilisateur de revenir à la page d'accueil.
- *icon.svg* ou *icon.png* – requis pour les applications, contient l'icône de l'application, *SVG* est recommandé.
- *widget.lp* ou *widget.js* – requis pour les widgets, peut contenir du code *JavaScript* + *Lua* ou du code source *JavaScript* pur qui affiche le contenu des widgets.
- *title* – optionnel pour les applications, fichier texte avec titre affiché sous l'icône.
- *url* – optionnel pour les applications, fichier texte avec *URL* qui doit s'ouvrir lorsque l'icône est cliquée.
- *style.css* – optionnel pour les widgets, contient une feuille de style CSS personnalisée pour un widget donné.
- *config.lp* ou *config.html* – fichier de configuration optionnel, voir description ci-dessous.

En mode widget, l'ID de l'élément icône est le même que le nom du widget, tous les autres ID d'élément HTML doivent être préfixés avec un nom d'application unique pour minimiser les collisions entre les différentes applications. La même règle s'applique aux sélecteurs CSS.

Note: si votre widget a des événements de clics personnalisés, vérifiez si la classe est ***déverrouillée***, ce qui est le cas quand l'utilisateur peut faire glisser le widget. Les événements de clic doivent être ignorés si cette classe est définie.

9 Taille des widgets

La taille par défaut des widgets est 100×100px. La largeur/hauteur peut être augmentée en appelant *setWidgetSize(cols, rows)* sur l'élément widget. Formule de largeur: colonnes * 110–10, formule de hauteur: lignes * 110–10

Exemples

Le widget horloge prend le double de la largeur/hauteur et place l'image SVG qui remplit tout l'espace disponible à l'intérieur du conteneur de widget :

```
(function() {
  // get widget element and set double width/height
  var el = $('#clock').setWidgetSize(2, 2);
  $('<object type="image/svg+xml"></object>') // object allows SVG+JavaScript
    .css('width', '100%') // full width
    .css('height', '100%') // full height
    .attr('data', '/apps/data/clock/clock.svg') // SVG image source
    .appendTo(el); // add to container
})();
```

Basculer entre le mode carré (1, 1) et le mode large (2, 1) après chaque clic.

resize-demo/style.css

```
#resize-demo { overflow: hidden; background: #fd0; color: #333; cursor: pointer; }
#resize-demo small { position: absolute; bottom: 5px; text-align: center; font-size: 17px; line-height: 17px; }
#resize-demo div { width: 100px; height: 100px; float: left; text-align: center; }
#resize-demo div:first-child { font-size: 36px; line-height: 90px; }
#resize-demo div:first-child small { width: 100px; left: 0; }
#resize-demo div:last-child { width: 110px; font-size: 15px; padding-top: 8px; }
#resize-demo div:last-child small { width: 110px; left: 100px; }
```

resize-demo/widget.js

```
(function() {
  var el = $('#resize-demo');
// simple info
$('<div></div>')
  .html('86%<small>Current</small>')
  .appendTo(el);

// extended info to show on click
$('<div></div>')
  .html('Min: 12%<br>Max: 92%<br>Avg: 56%<small>Daily</small>')
  .appendTo(el);

// toggle size on click
el.click(function() {
  var cols = el.hasClass('big') ? 1 : 2;
  el.setWidgetSize(cols, 1);
  el.toggleClass('big');
});
})();
```

10 Daemons

Créez un nouveau répertoire nommé comme votre application dans le répertoire *daemon*. Placez *daemon.lua* à l'intérieur du nouveau répertoire créé. Le daemon est démarré automatiquement lorsque le dispositif démarre et que l'application est installée. Le daemon est automatiquement redémarré en cas d'erreur, les erreurs sont enregistrées dans le journal d'erreurs *fellerLYnk*. On peut forcer le daemon à (re)démarrer ou à s'arrêter via une requête HTTP :

http://IP/apps/request.lp?password=ADMINPASSWORD&action=restart&name=YOURAPPNAME

http://IP/apps/request.lp?password=ADMINPASSWORD&action=stop&name=YOURAPPNAME

Exemple (daemon.lua)

Exemple minimal qui enregistre l'horodatage actuel toutes les 5 secondes

```
while true do
  alert(os.time())
  os.sleep(5)
end
```

11 Configuration

- Le répertoire de l'application doit contenir le fichier *config.lp* ou *config.html.*
- Ce fichier doit contenir un élément *form*, l'identifiant doit être au format *myapp-config*, où *myapp* est le nom unique de l'application.
- L'échange de données se fait via des événements déclenchés sur l'élément *form* :

  config-load – (vers l'application) fournit un objet avec toutes les paires clé/valeur de configuration.

  config-check – (vers l'application) déclenché lorsque le bouton Enregistrer est cliqué, la configuration de l'application doit soit afficher un message d'erreur si la configuration n'est pas valide, soit déclencher *config-save.*

  config-save – (de l'application) enregistre la configuration côté serveur et ferme la fenêtre modale, l'application doit passer les paramètres de configuration en tant qu'objet.
- Les fonctions suivantes permettent d'accéder à la configuration à partir de Lua :

  config.get(app, key, default) – retourne une valeur unique pour un nom d'application donné, une valeur par défaut ou *nil* si la clé n'est pas trouvée.

  config.getall(app) – retourne une table avec toutes les valeurs de configuration pour un nom d'application donné ou *nil* si la configuration est vide.

  config.set(app, key, value) – ajoute une nouvelle paire clé/valeur ou écrase une paire clé/valeur existante.

  config.setall(app, cfg) – écrase la configuration existante avec une table *cfg* donnée contenant des clés/valeurs.

  config.delete(app, key) – supprime la paire clé/valeur existante.
- Les applications non publiées qui ont un fichier de configuration apparaissent sous *Dev apps* sur la page d'administration.

Exemples (config.html)

Accéder à une valeur de configuration à partir du daemon *Lua* :

```
-- get delay value from config, use 15 as default when delay is not set
delay = config.get('myapp', 'delay', 15)
-- main daemon loop
while true do
  alert(os.time())
  os.sleep(delay)
end
```

Définir une valeur dans un script *lp* publié par l'utilisateur :

```
<?
delay = getvar('delay') -- GET/POST variable
delay = tonumber(delay) or 0 -- convert to number
-- set to default value if empty or invalid
if delay < 5 or 100 < delay then
  delay = 15
end
config.set('myapp', 'delay', 15)
```

Créer un script simple pour un élément à une seule entrée numérique qui accepte des valeurs dans la plage 5...100.

```
<form id="myapp-config">
  <div class="form-group">
    <label for="myapp-input">Delay (seconds)</label>
    <input type="number" name="delay" id="myapp-delay" class="form-control" min="5"
max="100">
  </div>
</form>
```

```
<script>
(function() {
  var el = $('#myapp-config') // form element
    , input = $('#myapp-delay'); // input element
  // set element values when config is loaded
  el.on('config-load', function(event, data) {
    $.each(data, function(key, value) {
      $('#myapp-' + key).val(value);
  });
});

// runs when Save button is clicked
el.on('config-check', function() {
  var val = parseInt(input.val(), 10) // input value
  , min = parseInt(input.attr('min'), 10) // minimum value
  , max = parseInt(input.attr('max'), 10); // maximum value

// invalid value
if (isNaN(val) || val < min || max < val) {
  alert('Please enter a value between ' + min + ' and ' + max);
}
// all good, save configuration
else {
  el.triggerHandler('config-save', { delay: val });
}
});
})();
</script>
```

Fonctions wrapper localStorage

localStorage permet d'enregistrer la configuration côté client. Plusieurs fonctions sont fournies pour exécuter de façon sûre des fonctions *localStorage* car elles peuvent échouer dans certains cas comme en mode privé sur *iOS*. Elles permettent également de stocker toutes les valeurs qui peuvent être sérialisées à l'aide de *JSON.stringify*.

- *storeSet(key, value)* – définit la paire clé/valeur
- *storeGet(key)* – récupère la valeur de la clé, retourne *null* si la clé n'est pas trouvée
- *storeRemove(key)* – supprime la clé du stockage

Les clés de stockage doivent être préfixées avec un nom d'application unique pour minimiser les collisions entre différentes applications.

Exemples

Obtenir le thème actuellement sélectionné (clair/sombre)

```
var theme = storeGet('myapp_theme') || 'light';
```

Stocker des objets JavaScript

```
var user = { name: 'John', surname: 'Doe', age: 42 };
storeSet('myapp_user', user);
```

Fonctions de cookies

Utilisez l'objet global *Cookie* pour accéder aux cookies depuis le côté client.

- *Cookie.set(key, value[, attributes])* – définit la paire clé/valeur du cookie. Si la valeur est un tableau *JavaScript* ou un objet, elle sera automatiquement codée en *JSON* pour les appels set et décodée en *JSON* pour les appels *get*.
- *Cookie.get()* – retourne un objet contenant toutes les paires clé/valeur du cookie.
- *Cookie.get(key)* – retourne la valeur du cookie.
- *Cookie.remove(key[, attributes])* – supprime le cookie du stockage.
- Attributs des cookies (optionnel) :

  path – chemin du cookie, par défaut à "/"

  expires – nombre (jours) ou objet Date, contenant la durée d'expiration du cookie. Par défaut, le cookie expire lorsque la fenêtre du navigateur est fermée.

  secure – n'autoriser l'envoi du cookie que par HTTPS, désactivé par défaut.
- Les cookies doivent être préfixés avec un nom d'application unique pour minimiser les collisions entre différentes applications.

Exemples

Obtenir le thème actuellement sélectionné (clair/sombre)

```
var theme = Cookie.get('myapp_theme') || 'light';
```

Définir l'expiration du cookie dans 1 an

```
var config = { theme: 'dark', language: 'Italian' };
Cookie.set('myapp_config', config, { expires: 365 });
```

Stocker des objets *JavaScript*

```
var user = { name: 'John', surname: 'Doe', age: 42 };
Cookie.set('myapp_user', user);
```

12 Surveillance du bus local côté client

localbus permet de surveiller les changements d'adresses de groupe et de variables de stockage. En cas d'utilisation séparée, inclure
/apps/js/localbus.js.gz dans votre application et appeler *localbus.init()*. Cela n'est pas nécessaire pour les widgets.

localbus.listen('object', groupaddr, callback)

Exécute la fonction de rappel lorsque la valeur de l'adresse de groupe change, callback reçoit un argument – la nouvelle valeur de l'objet. Ne fonctionne pas si l'objet n'a pas un type de données connu.

localbus.listen('storage', key, callback)

Exécute la fonction de rappel lorsque la valeur de la clé de stockage change, callback reçoit un argument – la nouvelle valeur de stockage.

localbus.listen('groupwrite', callback)

localbus.listen('groupread', callback)

localbus.listen('groupresponse', callback)

Exécute la fonction de rappel lorsqu'un nouveau télégramme de groupe est reçu, callback reçoit un argument – l'objet événement.

Objet événement :

- *event.dst* – adresse de groupe de destination
- *event.src* – adresse source, vide pour les télégrammes locaux
- *event.tsec* – horodatage UNIX du télégramme (partie secondes)
- *event.usec* – horodatage UNIX du télégramme (partie microsecondes)
- *event.type* – type d'événement: groupread, groupwrite ou groupresponse
- *event.value* – nouvelle valeur, valable uniquement pour les types groupwrite et groupresponse et lorsque l'objet de destination a un type de données connu

Exemple de widget

```
(function() {
  // get widget element and add data div
  var el = $('#mywidget'), inner = $('<div></div>').addClass('data').appendTo(el);
  // create title element and add to widget
  $('<div></div>').addClass('title').text('My value').appendTo(el);
  // listen for mystoragevar changes
  localbus.listen('storage', 'mystoragevar', function(res) {
    // check if value is a valid number
    res = parseFloat(res);
    if (!isNaN(res)) {
      // update data div
      inner.text(res.toFixed(2));
    }
  });
})();
```

13 Surveillance du bus local côté serveur

lb = require('localbus').new([timeout])

Charge la bibliothèque localbus et crée une nouvelle connexion, timeout est une valeur optionnelle en secondes et est fixé à 1 seconde par défaut.

lb:step()

Attendre un message ou timeout. Retourne true en cas de nouveau message, nil en cas de timeout.

lb:sethandler('groupwrite', groupcallback)

lb:sethandler('groupread', groupcallback)

lb:sethandler('groupresponse', groupcallback)

Définit une fonction de rappel pour chacun des types de message de groupe.

Callback reçoit un argument – une table *event* :

- *event.dst* – adresse de groupe de destination
- *event.src* – adresse source, vide pour les télégrammes locaux
- *event.type* – type d'événement: *groupread*, *groupwrite* ou *groupresponse*
- *event.datahex* – données brutes codées HEX

lb:sethandler('storage', storagecallback)

Définit une fonction de rappel pour les changements de stockage.

Callback reçoit trois arguments: *action*, *key*, *value* :

- *action* – soit "set", soit "delete"
- *key* – clé de l'élément de stockage
- *value* – nouvelle valeur de l'élément de stockage, *nil* pour action "delete"

Exemple (daemon)

```
function groupcallback(event)
  if event.dst == '1/1/1' then
    local value = knxdatatype.decode(event.datahex, dt.uint16)
    dosomethingwithvalue(value)
  end
end

function storagecallback(action, key, value)
  if action == 'set' and key == 'mystoragekey' then
    dosomethingwithstorage(value)
  end
end

lb = require('localbus').new(0.5) -- timeout is 0.5 seconds
lb:sethandler('groupwrite', groupcallback)
lb:sethandler('storage', storagecallback)

while true do
  lb:step()
  handledaemonstuff()
end
```

14 Traduction

- *$.i18n.lang* – langue courante ou *undefined* si la langue par défaut est utilisée
- *$.i18n.add(ns, dictionary)* – ajoute des traductions au dictionnaire courant, *ns* doit être un nom d'application unique
- *$.i18n.translate(key, default, vars)* ou *$.tr(key, default, vars)* – traduit une *key* donnée ou utilise la valeur *default* si la traduction n'est pas trouvée pour la langue courante. Des objets *vars* supplémentaires peuvent être passés pour remplacer des variables à l'intérieur du texte de la traduction.

Exemple 1

```
// register translation for application "myapp"
$.i18n.add('myapp', {
  // translation for
  mylang mylang: {
    hello: 'Hello %{username}, current temperature is %{temperature}',
    goodbye: 'Goodbye %{username}'
  }
})
var text = $.tr('myapp.hello', 'No translation', { username: 'John', temperature: 21 });
// alerts “Hello John, current temperature is 21” if current language is "mylang"
// otherwise alerts "No translation"
alert(text);
```

Exemple 2

Vous pouvez appliquer une traduction à des sélecteurs *jQuery* en utilisant la fonction *tr*: tous les éléments HTML qui ont la classe *tr* et l'attribut *data-tr-key* auront leur contenu remplacé par la version traduite.

HTML

```
<div id="myapp-container">
  <span class="tr" data-tr-key="myapp.hello">Hello!</span>
</div>
```

JavaScript

```
// register french translation
$.i18n.add('myapp', {
  fr: {
    hello: 'Bonjour!'
  }
});
// apply translation to all elements inside of myapp-container
$('#myapp-container').tr();
```

15 Scripts LP

Permettent de mélanger HTML et Lua à l'intérieur d'un seul fichier, les partie en Lua doivent être placées entre des balises < ? ?>, il n'est pas nécessaire de fermer la balise à la fin du document.

Fonctions disponibles :

- *header(hdr)* – ajoute un en-tête personnalisé à la sortie
- *getvar(name)* – retourne la variable *GET/POST* nommée ou *nil* si la variable n'est pas définie
- *getvars()* - retourne toutes les variables *GET/POST* sous la forme d'une *table Lua*
- *getcookie(name)* – retourne le contenu du cookie nommé ou *nil* si le cookie n'est pas défini
- *getheader(header)* – retourne le contenu du *header* nommé ou *nil* si l'en-tête n'est pas défini, peut retourner une *table* si un en-tête de même nom a été passé plusieurs fois. La casse de *header* n'a pas d'importance, elle est normalisée en minuscules pures avec tous les tirets bas convertis en tirets en cas d'erreur de recherche.
- *setcookie(name, value [, expires [, path]])* – définit le contenu du *cookie* nommé, doit être appelé avant l'envoi de toute sortie, *expires* doit être un nombre (jours) ou une table contenant le temps d'expiration du cookie. Par défaut, le cookie expire lorsque la fenêtre du navigateur est fermée. La valeur par défaut de *path* est "/".
- *print(...)* – imprime un nombre quelconque de variables, terminant la sortie par *CRLF*
- *write(...)* – similaire à *print* mais n'imprime pas **CRLF** à la fin
- *escape(val)* – échappe les caractères guillemets simples/doubles, inférieur à/supérieur à vers des entités HTML
- *include(file)* – inclut et exécute un autre fichier lp, notez que les variables locales parent ne sont pas visibles à l'intérieur du fichier inclus (voir l'exemple ci-dessous pour plus d'informations)

La table *request* contient des informations sur le fichier actuel et le chemin d'accès :

- *request.file* – chemin d'accès complet du fichier lp demandé
- *request.path* – chemin d'accès complet du répertoire contenant le fichier lp demandé

Le package de bibliothèque est chargé via *require('apps')* et donne accès aux fonctions suivantes :

- toutes les fonctions HL intégrées: *alert*, *log*, *grp*, *storage* etc. si le package HL de base est installé
- bibliothèque *config*
- fonctions *vprint(...)* et *vprinthex(...)* pour visualiser le contenu des variables sous une forme lisible
- bibliothèque *json*
- *getlanguage([default])* – retourne la langue sélectionnée pour l'utilisateur courant ou la valeur par défaut (*nil* si non spécifiée) si la langue n'est pas définie

Exemples

Imprimer la date du jour

```
<!DOCTYPE html>
<html>
  <body>Current date is <? write(os.date()) ?></body>
</html>
```

Sortir une table de multiplication. Size peut être une variable *GET/POST* dans la plage 1..20 (par défaut 10).

```
<!DOCTYPE html>
<html><body>
<?
size = getvar('size') -- GET/POST variable
size = tonumber(size) or 0 -- convert to number
if size < 1 or 20 < size then
  size = 10 -- set to default value if empty or invalid
end
?>
<table border="1" cellpadding="3">
<? for i = 1, size do ?>
  <tr>
    <? for j = 1, size do ?>
      <td><? write(i * j) ?></td>
    <? end ?>
  </tr>
<? end ?>
</table>
</body></html>
```

Inclure le document (*row.lp*), doit être dans le même répertoire que le document parent (*index.lp*)

```
<tr>
  <? for j = 1, size do ?>
    <td><? write(ival * j) ?></td>
  <? end ?>
</tr>
```

Document parent (*index.lp*). Note: la variable globale *ival* est utilisée parce que la variable de compteur locale *i* n'est pas visible dans *row.lp*

```
<!DOCTYPE html>
<html>
<body>
<?
size = getvar('size') -- GET/POST variable
size = tonumber(size) or 0 -- convert to number
if size < 1 or 20 < size then
 size = 10 -- set to default value if empty or invalid end
?>
<table border="1" cellpadding="3">
<? for i = 1, size do ?>
  <? ival = i ?>
  <? include('row.lp') ?>
<? end ?>
</table>
</body>
</html>
```

16 Fonctions Lua

Le manuel de référence complet des fonctions Lua est disponible à l'adresse http://openrb.co/docs/lua.htm

16.1 Fonctions d'objet

La plupart des fonctions utilisent le paramètre *alias* — soit l'adresse de groupe de l'objet, soit le nom de l'objet. (p. ex. '1/1/1' ou 'My object')

Trouver des objets simples/multiples

grp.find(alias)

Retourne un objet simple pour un alias donné. La valeur de l'objet est décodée si le type de données est défini.

Retourne *nil* si l'objet ne peut pas être trouvé, sinon retourne une *table* avec les éléments suivants :

- address — adresse de groupe de l'objet
- updatetime — dernière heure de mise à jour au format *horodatage UNIX.* Utiliser os.date() pour convertir en formats de date lisibles.
- name — type de données de nom d'objet unique — type de données de l'objet
- decoded — mis à *true* lorsque la valeur décodée est disponible
- value — valeur de l'objet décodé

grp.tag(tags [, mode])

Retourne une *table* contenant des objets avec une balise donnée. Le paramètre tags peut être une table ou une chaîne de caractères. Le paramètre mode peut être soit 'or' (par défaut — retourne les objets qui ont une des balises données), soit 'and' (retourne les objets qui ont toutes les balises données). Vous pouvez utiliser les fonctions d'objet sur la table retournée.

grp.dpt(dpt, [strict])

Trouver tous les objets dont le type de données correspond. *dpt* peut être soit une chaîne de caractères ("bool", "scale", "uint32", etc.), soit un champ de la table *dt* (dt.bool, dt.scale, dt.uint32). Par exemple, si *dpt* est défini à *dt.uint8*, en mode normal, tous les sous-types de données comme *dt.scale* et *dt.angle* seront inclus. Si une correspondance exacte du type de données est requise, mettre *strict* à *true*.

grp.all()

Retourne une table avec tous les objets connus.

16.2 Helpers

grp.alias(alias)

Convertit l'adresse de groupe en nom d'objet ou le nom en adresse. Retourne *nil* si l'objet ne peut pas être trouvé.

grp.getvalue(alias)

Retourne la valeur pour un alias donné ou *nil* si l'objet ne peut pas être trouvé.

16.3 Demandes de bus

grp.write(alias, value [, datatype])

Envoie une demande d'écriture de groupe à un alias donné. Le type de données est tiré de la base de données s'il n'est pas spécifié comme troisième paramètre. Retourne un booléen comme résultat.

grp.response(alias, value [, datatype])

Similaire à grp.write. Envoie une demande de réponse de groupe à un alias donné.

grp.read(alias)

Envoie une demande de lecture de groupe à un alias donné. Note: cette fonction retourne immédiatement et ne peut pas être utilisée pour retourner le résultat de la demande de lecture. Utilisez un script basé sur un événement à la place.

grp.update (alias, value [, datatype])

Similaire à grp.write, mais n'envoie pas de nouvelle valeur au bus. Utile pour les objets qui ne sont utilisés qu'en visualisation.

16.4 Manipulation des balises

grp.gettags(alias)

Retourne une *table* avec toutes les balises définies pour un alias donné.

grp.addtags(alias, tags)

Ajoute une ou plusieurs balises à un alias donné. Le paramètre *tags* peut être soit une chaîne de caractères (balises simples), soit une *table Lua* composée de chaînes de caractères (balises multiples).

grp.removetags(alias, tags)

Supprime une ou plusieurs balises d'un alias donné. Le paramètre *tags* peut être soit une chaîne de caractères (balises simples), soit une *table Lua* composée de chaînes de caractères (balises multiples).

grp.removealltags(alias)

Supprime toutes les balises pour un alias donné.

grp.settags(alias, tags)

Écrase toutes les balises pour un alias donné. Le paramètre *tags* peut être soit une chaîne de caractères (balises simples), soit une *table Lua* composée de chaînes de caractères (balises multiples).

16.5 Création et modification d'objets

grp.setcomment(alias, comment)

Définit un champ *comment* pour un alias donné.

grp.create(config)

Crée un nouvel objet ou écrase un objet existant en fonction de la *config* fournie, qui doit être une table Lua. Retourne l'ID de l'objet en cas de succès, nil plus un message d'erreur sinon.

Champs de *config* :

- datatype – *requis*, type de données de l'objet. Peut être soit une chaîne de caractères ("bool", "scale", "uint32", etc.), soit un champ de la table *dt* (dt.bool, dt.scale, dt.uint32).
- name – *optionnel*, nom unique de l'objet. Si un objet de même nom existe déjà, un préfixe numérique sera ajouté.
- comment – *optionnel*, commentaire de l'objet (*chaîne de caractères*)
- units – *optionnel*, unités/suffixe de l'objet (*chaîne de caractères*)
- address – *optionnel*, adresse de groupe de l'objet (*chaîne de caractères*). Si elle n'est pas définie, la première adresse libre de la plage configurée sera utilisée.
- tags – *optionnel*, balises de l'objet, peut être soit une chaîne de caractères (balises simples), soit une *table Lua* composée de chaînes de caractères (balises multiples).

Si un objet de même adresse de groupe existe déjà, seuls les champs *units*, *datatype* et *comment* seront modifiés. Toutes les autres propriétés resteront inchangées.

Exemples

Créer un nouvel objet d'adresse connue

```
address = grp.create({
  datatype = dt.float16,
  address = '1/1/1',
  name = 'My first object',
  comment = 'This is my new object', units = 'W',
  tags = { 'My tag A', 'My tag B' },
})
```

Créer un nouvel objet avec attribution automatique d'adresse

```
address = grp.create({
  datatype = dt.bool,
  name = 'My second object',
})
```

16.6 Fonctions de base de données

SQLite v3 est utilisé comme moteur de base de données.

Note: Les tables de la base de données doivent être préfixées avec un nom d'application unique pour minimiser les collisions entre différentes applications.

16.7 Fonctions de base

- *db:execute(query)* – exécute la requête donnée, la valeur de retour peut être soit un curseur de base de données, soit le résultat de la requête.
- *db:escape(value)* – échappe une valeur de chaîne de caractères donnée pour pouvoir l'utiliser sans risque dans une requête.
- *db:query(query, …)* – exécute une requête donnée, les points d'interrogation dans la requête sont remplacés par des paramètres supplémentaires (voir les exemples ci-dessous).

16.8 Helpers INSERT/UPDATE/DELETE

Note: Les *tables Lua* passées comme *values* et les paramètres *where* ne doivent pas avoir de champs qui ne sont pas présents dans la table de base de données. Sinon, la requête échouera.

- *db:insert(tablename, values)* – effectue une requête *INSERT* basée sur les valeurs données.
- *db:update(tablename, values, where)* – effectue une requête *UPDATE* basée sur les *values* et les paramètres *where* donnés.
- *db:delete(tablename, where)* – effectue une requête *DELETE* basée sur les paramètres *where*.

16.9 Helpers SELECT

Note: les paramètres doivent être passés de la même manière que pour la fonction *db:query()*.

- *db:getone(query, …)* – retourne la première valeur de champ de la première ligne correspondante de la requête donnée.
- *db:getrow(query, …)* – retourne la première ligne correspondante de la requête donnée.
- *db:getlist(query, …)* – retourne le résultat complet de la requête sous forme de *table Lua*, chaque élément de la table étant le premier champ de chaque ligne.
- *db:getall(query, …)* – retourne le résultat complet de la requête sous forme de *table Lua*, chaque élément de la table étant une table Lua avec correspondance des valeurs de champ.

Exemples

```
-- Query parameter replacement
db:query('UPDATE table SET field=? WHERE id=?', 'test', 42)
-- Same as INSERT INTO table (id, value) VALUES (42, 'test')
db:insert('table', { id = 42, value = 'test' })
-- Same as UPDATE table SET value='test' WHERE id=42
db:update('table', { value = 'test' }, { id = 42 })
-- Same as DELETE FROM table WHERE id=42
db:delete('table', { id = 42 })
```

Feller AG | Postfach | CH-8810 Horgen
Téléphone +41 44 728 72 72 | Téléfax +41 44 728 72 99

Feller SA | Caudray 6 | CH-1020 Renens
Téléphone +41 21 653 24 45 | Téléfax +41 21 653 24 51

Ligne de service | Téléphone +41 44 728 74 74 | info@feller.ch | www.feller.ch

by Schneider Electric