fellerLYnk

Erstellen von Apps für fellerLYnk

10.FLYNKAPP-D.1804/180611

by Schneider Electric

1 Einleitung

In diesem Dokument wird beschrieben, wie sich Apps für fellerLYnk erstellen lassen. Der Appstore wird in Firmware 2.0 integriert. Framework zum Erstellen und Spielen mit Apps ist bereits in fellerLYnk 1.2.0 enthalten.

2 Zugriff auf die App

Zum Starten der App muss eine direkte URL angegeben werden.

http://192.168.1.10/data/myapp.lp

http://192.168.1.10/user/index.html

3 App hochladen

Der von SL/HL integrierte FTP-Server eingebaut muss in System/Dienste aktiviert werden

Dort gibt es „ftp“ und „Apps“-Benutzer. Mit Apps-Benutzer verbinden und App herunterladen. Das Standard-Passwort lautet „Apps“.

4 Verfügbare Bibliotheken/Frameworks

- 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 wird ohne *Glyphicons* ausgeliefert, stattdessen *Font Awesome* verwenden.

5 Basisverzeichnisstruktur

- /data – Apps und Widgets sind hier gespeichert und unter http://IP/apps/data/elektronisch abrufbar
- /libs – Lua Bibliothek Hinterlegen, geladen über *require('custom.lib')*, wobei der Name der Bibliothek *lib* ist.
- /daemon – Appdaemon ist hier gespeichert
- /user – ermöglicht das Speichern von Benutzerdateien und LP-Scripts, abrufbar unter http://IP/user/

6 Vollbildmodus

Zum Ausblenden der Symbole in der oberen Navigationsleiste, fs GET Variable eingeben (die Variableninhalte müssen als *wahr* ausgewertet werden): http://IP/apps/?fs=1

7 App / Widget-Struktur

Der Name der Anwendung (ID) muss eindeutig sein und darf nur alphanumerische Zeichen, Bindestriche und Unterstriche enthalten. Die maximale Länge des Namens ist 64 Zeichen.

8 Verzeichnisstruktur

- index.lp oder index.html - für Apps erforderlich, sofern *URL* nicht angegeben ist. Wenn Sie auf das App-Symbol klicken, wird das App-Verzeichnis in demselben Fenster geöffnet. Anwendungen müssen über eine *Zurück*-Schaltfläche verfügen, damit der Nutzer zur Startseite zurückkehren kann
- *icon.svg* oder icon.png – für Apps erforderlich, enthält App-Symbol, *SVG* ist empfohlen
- *widget.lp* oder *widget.js* – für Widgets erforderlich, enthält unter Umständen *JavaScript* + *Lua* Code oder die reine *JavaScript*-Quelle, wodurch Widget-Inhalte angezeigt werden
- *Titel* – optional für Apps, Textdatei mit Titel, der unter dem Symbol angezeigt wird
- *url* – optional für Apps, Textdatei mit *URL*, die durch Klicken auf das Symbol geöffnet werden sollte
- *style.css* – optional für Widget, enthält ein benutzerdefiniertes *CSS*-Stylesheet für ein bestimmtes Widget
- *config.lp* oder *config.html* – optionale Konfigurationsdatei, siehe Beschreibung unten

Im Widget-Modus ist die Symbolelement-ID dem Widget-Namen identisch. Alle anderen HTML-Element-IDs müssen mit einem eindeutigen Anwendungsnamen vordefiniert sein, um Kollisionen zwischen verschiedenen Anwendungen zu minimieren. Dieselbe Regel gilt für CSS-Selektoren.

Hinweis: Wenn Ihr Widget über benutzerdefinierte Klickereignisse verfügt, überprüfen Sie, ob die **entsperrte** Klasse aktiviert ist, wenn das Widget vom Endnutzer hingezogen werden kann. Klick-Ereignisse müssen ignoriert werden, wenn diese Klasse festgelegt ist.

9 Widget-Grössenanpassung

Die Widget-Standardgrösse beträgt 100×100px. Zum Vergrössern von Breite/Höhe *setWidgetSize(Spalten, Zeilen) am Widget-Element aufrufen.* Breitenberechnung: Spalten * 110–10, Höhenberechnung: Zeilen * 110–10

Beispiele

Uhr-Widget mit doppelter Breite / Höhe und einem SVG-Bild, das den gesamten verfügbaren Platz innerhalb des Widget-Containers ausfüllt:

```
(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
})();
```

Nach jedem Klick zwischen dem Modus „quadratisch“ (1, 1) und „breit“ (2, 1) wechseln.

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 Daemon

Erstellen Sie ein neues Verzeichnis, dass im Daemon-Verzeichnis als Ihre Anwendung bezeichnet wird. Legen Sie *daemon.lua* in Ihr neu erstelltes Verzeichnis ab. Daemon wird automatisch gestartet, wenn das Gerät hochfährt und die Anwendung installiert ist. Daemon wird im Falle eines Fehlers automatisch neu gestartet, Fehler werden im Fehlerprotokoll *fellerLYnk* protokolliert. Daemon kann über eine HTTP-Anfrage gezwungen werden, neu zu starten oder zu stoppen:

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

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

Beispiel (daemon.lua)

Minimales Beispiel, bei dem der aktuelle Zeitstempel alle 5 Sekunden protokolliert wird

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

11 Konfiguration

- Das Anwendungsverzeichnis muss entweder die Datei *config.lp* oder die Datei *config.html* enthalten
- Diese Datei muss das *Formular*-Element enthalten. Die ID muss im Format *myapp-config* festgelegt werden, wobei *myapp* der eindeutige Name der Anwendung ist
- Der Datenaustausch erfolgt über Ereignisse, die im *Formular*-Element ausgelöst werden:

  config-load – (an die App) stellt ein Objekt mit allen Konfigurationsschlüssel- / Wert-Paaren bereit

  config-check – (an die App) wird ausgelöst, wenn auf die Schaltfläche „Speichern“ geklickt wird. Die App-Konfiguration muss entweder eine Fehlermeldung anzeigen, wenn die Konfiguration ungültig ist oder *config-save* auslösen

  config-save – (von der App) speichert die Konfiguration auf dem Server, bei geschlossenem modalen Fenster. Die Anwendung muss die Konfigurationsparameter als Objekt übergeben
- Das Konfigurationsmenü ist über Lua mit Hilfe der folgenden Funktionen aufrufbar:

  config.get(app, key, default) – gibt einen einzelnen Wert für den angegebenen Anwendungsnamen, den Standardwert oder <*nil*> zurück, wenn der Schlüssel nicht gefunden wird

  config.getall(app) – gibt die Tabelle mit allen Konfigurationswerten für den angegebenen Anwendungsnamen oder <*nil*> zurück, wenn die Konfiguration leer ist

  config.set(app, key, value) – fügt ein neues Schlüssel-/Werte-Paar hinzu oder überschreibt ein bereits Vorhandenes

  config.setall(app, cfg) – überschreibt eine bestehende Konfiguration mit gegebenem *cfg table* mit Schlüsseln/Werten

  config.delete(app, key) – löscht ein vorhandenes Schlüssel-/Werte-Paar
- Nicht veröffentlichte Apps mit einer vorhandenen Konfigurationsdatei werden auf der Admin-Seite unter *Dev apps* angezeigt

Beispiele (config.html)

Zugriff auf den Config-Wert über Lua daemon:

```
-- 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
```

Wert in *lp*-Skript einstellen, das durch den Nutzer gesendet wurde:

```
<?
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)
```

Ein einfaches Element mit einer einzelnen numerischen Eingabe erstellen, die Werte im Bereich 5..100 akzeptiert

```
<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>
```

localStorage Wrapper-Funktionen

localStorage ermöglicht eine clientseitige Konfiguration. Es gibt mehrere Funktionen, um lokale Speicherfunktionen sicher auszuführen, da sie in einigen Fällen, wie dem Privat-Modus auf iOS, fehlschlagen können. Es ermöglicht auch das Speichern von Werten, die mit JSON.stringify serialisiert werden können.

- *storeSet(key, value)* – stellt das Schlüssel-/Werte-Paar ein
- *storeGet(key)* – ruft den Schlüsselwert ab, gibt *null* zurück, wenn der Schlüssel nicht gefunden wird
- *storeRemove(key)* – entfernt den Schlüssel aus dem Speicher

Speicherschlüsseln muss ein eindeutiger Anwendungsname vorangestellt sein, um Kollisionen zwischen verschiedenen Anwendungen zu minimieren.

Beispiele

Das aktuell ausgewählte Thema erhalten (hell/dunkel)

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

JavaScript-Objekte speichern

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

Cookie-Funktionen

Über das globale Cookie-Objekt können Sie auf clientseitige Cookies zugreifen.

- *Cookie.set(key, value[, attributes])* – stellt das Schlüssel-/Werte-Cookie-Paar ein. Wenn der Wert ein *JavaScript*-Array oder ein Objekt ist, wird er automatisch zum Einstellen von Anrufen *JSON*-codiert sowie zum Empfangen von Anrufen *JSON*-decodiert.
- *Cookie.get()* – gibt Objekte zurück, die alle Schlüssel-/Werte-Cookie-Paare enthalten
- *Cookie.get(key)* – gibt den Cookie-Wert zurück
- *Cookie.remove(key[, attributes])* – entfernt Cookie aus dem Speicher
- Cookie-Attribute (optional):

  path – Cookie-Pfad, standardmässig "/"

  expires – Anzahl (Tage) oder Datum Objekt, welche die Cookie-Verfallszeit enthalten. Standardmässig verfällt ein Cookie, wenn das Browser-Fenster geschlossen wird

  secure – Cookies nur über HTTPS senden lassen, standardmässig deaktiviert
- Cookie-Schlüsseln muss ein eindeutiger Anwendungsname vorangestellt sein, um Kollisionen zwischen verschiedenen Anwendungen zu minimieren.

Beispiele

Das aktuell ausgewählte Thema erhalten (hell/dunkel)

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

Cookie auf Ablauf in 1 Jahr einstellen

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

JavaScript-Objekte speichern

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

12 Clientseitige lokale Bus-Überwachung

localbus ermöglicht die Überwachung von Gruppenadressen und Speichervariablen-Änderungen. Bei einer getrennten Verwendung, bitte */apps/js/localbus.js.gz* in Ihre App einfügen und *localbus.init()* aufrufen. Dies ist für Widgets nicht erforderlich.

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

> Führt die Rückruf-Funktion aus, wenn sich der Gruppenadresswert ändert. Rückruf erhält ein Argument - neuer Objektwert. Funktioniert nicht, wenn das Objekt keinen bekannten Datentyp hat.

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

> Führt die Rückruf-Funktion aus, wenn sich der Speicherschlüssel-Wert ändert.
> Rückruf erhält ein Argument - neuer Objektwert.

localbus.listen('groupwrite', callback)

localbus.listen('groupread', callback)

localbus.listen('groupresponse', callback)

Führt die Rückruf-Funktion aus, wenn ein neues Gruppentelegramm empfangen wird.
Rückruf erhält ein Argument - Ereignisobjekt.

Ereignisobjekt:

- *event.dst* – Zielgruppen-Adresse
- *event.src* – Quelladresse, leer für lokale Telegramme
- *event.tsec* – UNIX-Zeitstempel für das Telegramm (Sekunden-Teil)
- *event.usec* – UNIX-Zeitstempel für das Telegramm (Mikrosekunden-Teil)
- *event.type* – Ereignis-Typ, entweder Groupread, Groupwrite oder Groupresponse
- *event.value* – neuer Wert, nur gültig für Groupwrite- und Groupresponse-Typen und wenn das Zielobjekt einen bekannten Datentyp hat

Widget-Beispiel

```
(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 Serverseitige lokale Bus-Überwachung

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

Lädt die Localbus-Bibliothek und erstellt eine neue Verbindung. Timeout ist ein optionaler Wert in Sekunden und standardmässig auf 1 Sekunde eingestellt.

lb:step()

Auf eine einzelne Nachricht warten oder Timeout. Gibt bei neuer Nachricht den Wert <true> zurück sowie <nil> bei Timeout.

lb:sethandler('groupwrite', groupcallback)

lb:sethandler('groupread', groupcallback)

lb:sethandler('groupresponse', groupcallback)

Legt eine Rückruf-Funktion für jeden Gruppennachricht-Typ fest.

Rückruf erhält ein Argument - Ereignis-Tabelle:

- *event.dst* – Zielgruppen-Adresse
- *event.src* – Quelladresse, leer für lokale Telegramme
- *event.type* – Ereignis-Typ, entweder *Groupread, Groupwrite* oder *Groupresponse*
- *event.datahex* – HEX-codierte Rohdaten

lb:sethandler('storage', storagecallback)

Legt eine Rückruf-Funktion für Speicheränderungen fest.

Rückruf erhält drei Argumente: *action*, *key*, *value*:

- *action* – entweder "festlegen" oder "löschen"
- *key* – Speicherelemente-Schlüssel
- *value* – neuer Speicherelement-Wert, <*nil*> für den „Lösch“-Vorgang

Beispiel (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 Übersetzung

- *$.i18n.lang* – aktuelle Sprache oder nicht festgelegt bei Verwendung der Standardsprache
- *$.i18n.add(ns, dictionary)* – fügt dem aktuellen Wörterbuch Übersetzungen hinzu, *ns* muss ein eindeutiger Anwendungsname sein
- *$.i18n.translate(key, default, vars)* oder *$.tr(key, default, vars)* – übersetzt einen gegebenen Schlüssel oder benutzt einen Standardwert, wenn für die aktuelle Sprache keine Übersetzung gefunden wird. Zusätzliches *vars*-Objekt kann übergeben werden, um Variablen im Übersetzungstext zu ersetzen

Beispiel 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);
```

Beispiel 2

Über die *tr*-Funktion können Sie die Übersetzung auf *jQuery*-Selektoren anwenden: alle HTML-Elemente der Kategorie *tr* und data-tr-key Attribut enthält Inhalte, die durch die übersetzte Version ersetzt wurden

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 LP-Scripts

Ermöglicht das Mischen von HTML und Lua in einer einzigen Datei, Lua-Chunks müssen enthalten sein in <? ?> Tags, schliessender Tag am Dokument-Ende wird nicht benötigt.

Verfügbare Funktionen:

- *header(hdr)* – fügt der Ausgabe einen benutzerdefinierten Header hinzu
- x*getvar(name)* –gibt die Variable GET/POST oder <*nil*> zurück, wenn die Variable nicht festgelegt ist
- *getvars()* - gibt alle *GET/POST*-Variablen als *Lua-Tabelle zurück*
- *getcookie(name)* – gibt alle *Cookie*-Inhalte zurück oder <*nil*>, wenn Cookie nicht festgelegt ist
- *getheader(header)* – gibt den Inhalt des benannten Headers zurück oder <*nil*>, wenn der Header nicht festgelegt ist. Die Tabelle könnte zurückgegeben werden, wenn der Header mit demselben Namen mehrmals übergeben wurde. Die *Header*-Schrift ist nicht relevant, da sie ausschliesslich auf kleine Buchstaben standardisiert ist, zuzüglich aller Unterstriche, die bei einem Suchfehler in Bindestriche umgewandelt werden
- *setcookie(name, value [, expires [, path]])* – legt benannte Cookie-Inhalte fest, muss aufgerufen werden, bevor eine Ausgabe gesendet wird, *expires* muss eine Zahl (Tage) oder eine Tabelle sein, die die Cookie-Ablaufzeit enthält. Standardmässig verfällt ein Cookie, wenn das Browser-Fenster geschlossen wird. Der *path*-Wert ist standardmässig "/"
- *print(...)* – gibt eine beliebige Anzahl von Variablen aus und beendet die Ausgabe mit *CRLF*
- *write(...)* – ähnlich wie *print*, gibt aber am Ende keine **CRLF** aus
- *escape(val)* – *Escape* für einzelne/doppelte Anführungszeichen, weniger/grösser als Zeichen in HTML-Entities
- *include(file)* – Eine weitere lp-Datei einfügen und starten. Achtung: die lokalen Stammvariablen sind nicht innerhalb der enthaltenen Datei sichtbar (für weitere Informationen siehe Beispiel unten)

Die *request*-Tabelle enthält Informationen über aktuelle Datei und Pfad:

- *request.file* – vollständiger Pfad zur angeforderten lp-Datei
- *request.path* – vollständiger Pfad zu dem Verzeichnis, das die angeforderte lp-Datei enthält

Das Bibliothekspaket wird über *require('apps')* geladen und bietet Zugriff auf diese Funktionen:

- alle eingebauten HL-Funktionen: *alert*, *log*, *grp*, *storage* usw., wenn HL-Hauptpaket installiert ist
- *config* library
- *vprint(...)* und *vprinthex(...)* Funktionen zum Anzeigen variabler Inhalte in allgemein lesbarer Form
- *json*-Bibliothek
- *getlanguage([default])* – gibt die ausgewählte Sprache für den aktuellen Nutzer oder Standardwert zurück (<*nil*> wenn nicht angegeben), wenn die Sprache nicht eingestellt ist

Beispiele

Aktuelles Datum drucken

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

Ausgabe-Multiplikationstabelle. Die Grösse kann eine *GET/POST*-Variable im Bereich von 1..20 sein (Standardwert 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>
```

Dokument einfügen (*row.lp*), muss im selben Verzeichnis wie das Hauptdokument sein (*index.lp*)

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

Hauptdokument (*index.lp*). Hinweis: *ival* global findet Verwendung, da die lokale Zählervariable *i* in *row.lp* nicht sichtbar ist

```
<!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 Lua-Funktionen

Das vollständige Referenzhandbuch für die Funktionen ist über folgende Adresse elektronisch abrufbar
http://openrb.co/docs/lua.htm

16.1 Objekt-Funktionen

Die meisten Funktionen verwenden einen Alias-Parameter — entweder Objektgruppenadresse oder Objektname. (z.B. '1/1/1' oder 'My object')

Einzelne/Mehrere Objekte finden

grp.find(alias)

Gibt ein einzelnes Objekt für einen gegebenen Alias zurück. Der Objektwert wird decodiert, wenn der Datentyp festgelegt ist.

Gibt <*nil*> zurück, wenn das Objekt nicht gefunden werden kann. Andernfalls gibt es eine *Tabelle* mit den folgenden Elementen zurück:

- address — Objektgruppen-Adresse
- updatetime — die letzte Aktualisierungszeit im *UNIX*-Zeitstempelformat. Zur Umwandlung in lesbare Dateiformate os.date() verwenden
- name — eindeutiger Objektname datatype — Objektdatentyp
- decoded — eingestellt auf <*true*>, wenn der decodierte Wert vorhanden ist
- value — decodierter Objektwert

grp.tag(tags [, mode])

Gibt eine *Tabelle* mit darin enthaltenen Objekten mit dem angegebenen Tag zurück. Tags-Parameter können entweder eine Tabelle oder eine Zeichenfolge sein. Der Modus-Parameter kann entweder „oder“ (Standard - gibt Objekte mit bestimmten Tags zurück) oder „und“ sein (gibt Objekte zurück, die alle angegebenen Tags haben). An der zurückgegebenen Tabelle lassen sich die Objektfunktionen nutzen.

grp.dpt(dpt, [strict])

Alle Objekte mit übereinstimmendem Datentyp suchen. *dpt* kann entweder eine Zeichenfolge ("bool", "scale", "uint32" usw.) sein oder ein Feld aus der *dt*-Tabelle (dt.bool, dt.scale, dt.uint32). Wenn beispielsweise *dpt* auf *dt.uint8* eingestellt ist, werden im Normalmodus alle Unter-Datentypen wie *dt.scale* und *dt.angle* mit eingeschlossen. Wenn eine exakte Übereinstimmung des Datentyps verlangt wird*, [strict]* auf <*true*> einstellen.

grp.all()

Gibt eine Tabelle mit allen bekannten Objekten zurück.

16.2 Helfer

grp.alias(alias)

Wandelt Gruppenadresse in Objektname um oder Name in Adresse. Gibt <*nil*> zurück, wenn das Objekt nicht gefunden werden kann.

grp.getvalue(alias)

Gibt den Wert für gegebene Alias zurück oder <*nil*>, wenn das Objekt nicht gefunden werden kann.

16.3 Bus-Anforderungen

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

Sendet Gruppen-Schreibaufforderungen an vorhandene Alias. Der Datentyp wird aus der Datenbank entnommen, wenn er nicht als dritter Parameter angegeben wurde. Gibt als Ergebnis einen booleschen Ausdruck zurück.

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

Ähnlich wie bei grp.write. Sendet Gruppen-Antwortaufforderungen an vorhandene Alias.

grp.read(alias)

Sendet Gruppen-Leseaufforderungen an vorhandene Alias. Hinweis: diese Funktion gibt sofort zurück und ist nicht dazu geeignet, um das Ergebnis einer Leseaufforderung zurückzugeben. Stattdessen ein ereignisbasiertes Skript verwenden.

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

Ähnlich wie grp.write, sendet aber keinen neuen Wert an den Bus. Geeignet für Objekte, die nur für die Ansicht verwendet werden.

16.4 Tag-Änderung

grp.gettags(alias)

Gibt eine Tabelle mit allen Tags zurück, die für vorhandene Alias festgelegt wurden.

grp.addtags(alias, tags)

Fügt einzelne oder mehrere Tags zu vorhandenen Alias hinzu. *Tags*-Parameter können entweder eine Zeichenfolge (einzelne Tags) sein oder eine Lua-Tabelle mit Zeichenfolgen (mehrere Tags).

grp.removetags(alias, tags)

Entfernt einzelne oder mehrere Tags aus vorhandenen Alias. *Tags*-Parameter können entweder eine Zeichenfolge (einzelne Tags) sein oder eine Lua-Tabelle mit Zeichenfolgen (mehrere Tags).

grp.removealltags(alias)

Entfernt alle Tags für vorhandene Alias.

grp.settags(alias, tags)

Überschreibt alle Tags für vorhandene Alias. *Tags*-Parameter können entweder eine Zeichenfolge (einzelne Tags) sein oder eine Lua-Tabelle mit Zeichenfolgen (mehrere Tags)

16.5 Erstellen und Ändern von Objekten

grp.setcomment(alias, comment)

Legt das *Kommentar*-Feld für vorhandene Alias fest

grp.create(config)

Erstellt ein neues Objekt oder überschreibt ein vorhandenes Objekt, basierend auf dem bereitgestellten *config*, das eine Lua-Tabelle sein muss. Gibt die Objekt-ID bei Erfolg zurück, ansonsten <*nil*> plus Fehlermeldung.

config-Felder:

- datatype – *erforderlich*, Objektdatentyp. Kann entweder eine Zeichenfolge ("bool", "scale", "uint32" usw.) sein oder ein Feld aus der *dt*-Tabelle (dt.bool, dt.scale, dt.uint32)
- name – *optional*, eindeutiger Objektname. Wenn bereits ein Objekt mit demselben Namen vorhanden ist, wird ein numerisches Präfix hinzugefügt
- comment – *optional*, Objekt-Kommentar (*Zeichenfolge*)
- units – *optional*, Objekt-Einheiten/Suffix (*Zeichenfolge*)
- address – *optional*, Objektgruppen-Adresse (*Zeichenfolge*). Falls nicht festgelegt, wird die erste freie Adresse aus dem konfigurierten Bereich verwendet
- tags – *optional*, Objekt-Tags, können entweder eine Zeichenfolge (*einzelne Ta*gs) sein oder eine *Lua-Tabelle* mit Zeichenfolgen (mehrere Tags)

Wenn bereits ein Objekt mit der gleichen Gruppenadresse vorhanden ist, werden nur die Felder *Einheiten, Datentyp* und *Kommentar* geändert. Alle anderen Eigenschaften werden unverändert beibehalten.

Beispiele

Neues Objekt mit bekannter Adresse erstellen

```
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' },
})
```

Neues Objekt mit automatischer Adressen-Zuweisung erstellen

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

16.6 Datenbank-Funktionen

SQLite v3 wird als Datenbank-Engine genutzt.

Hinweis: Datenbank-Tabellen muss ein eindeutiger Anwendungsname vorangestellt sein, um Kollisionen zwischen verschiedenen Anwendungen zu minimieren.

16.7 Kernfunktionen

- *db:execute(query)* – führt die angegebene Abfrage aus. Der Rückgabewert kann entweder ein Datenbank-Cursor oder ein Abfrageergebnis sein
- *db:escape(value)* – Escape-Vorgang für vorhandene Zeichenfolge (string)-Werte, um bei einer Abfrage sicher verwendet werden zu können
- *db:query(query, …)* – führt die angegebene Abfrage aus. Fragezeichen in der Abfrage werden durch zusätzliche Parameter ersetzt (siehe Beispiele unten)

16.8 INSERT/UPDATE/DELETE - Helfer

Hinweis: Lua-Tabellen werden als Werte übergeben, und *Where*-Parameter dürfen keine Felder enthalten, die in der angegebenen Datenbanktabelle nicht enthalten sind. Andernfalls wird die Abfrage fehlschlagen

- *db:insert(tablename, values)* – fürt eine *INSERT*-Abfrage durch, basierend auf vorhandenen Werten
- *db:update(tablename, values, where)* – fürt eine *UPDATE* -Abfrage durch, basierend auf vorhandenen Werten und *Where*-Parametern
- *db:delete(tablename, where)* – führt eine *DELETE*-Abfrage durch, basierend auf *Where*-Parametern

16.9 SELECT-Helfer

Hinweis: Parameter müssen auf die gleiche Weise wie bei der Funktion *db:query()* übergeben werden

- *db:getone(query, ...)* – gibt den ersten Feldwert aus der ersten Treffer-Zeile der angegebenen Abfrage zurück
- *db:getrow(query, ...)* – gibt die erste Treffer-Zeile aus der angegebenen Abfrage zurück
- *db:getlist(query, ...)* – gibt das vollständige Abfrage-Ergebnis als *Lua-Tabelle* zurück, wobei jedes Tabellenelement das erste Feld aus jeder Zeile ist
- *db:getall(query, ...)* – gibt das vollständige Abfrage-Ergebnis als *Lua-Tabelle* zurück, wobei jedes Tabellenelement eine Lua-Tabelle mit Feld ist → Werte-Mapping

Beispiele

```
-- 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
Telefon +41 44 728 72 72 | Telefax +41 44 728 72 99

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

Service-Hotline | Telefon +41 44 728 74 74 | info@feller.ch | www.feller.ch

by Schneider Electric