fellerLYnk

Creation of apps for fellerLYnk

10.FLYNKAPP-E.1804/180406

by Schneider Electric

1 Introduction

This document describes how to create apps for fellerLYnk. The app store will be imple- mented in firmware 2.0. Framework to create and play with apps is already implemented in fellerLYnk 1.2.0.

2 Accessing app

To launch app direct url must be provided.

http://192.168.1.10/data/myapp.lp

http://192.168.1.10/user/index.html

3 Uploading app

SL/HL has build in ftp server which has to be enabled in the System/Services

There is 'ftp' and 'apps' user. Connect to apps user to download app. Default password is 'apps'.

4 Available libraries/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 comes without *Glyphicons*, use *Font Awesome* instead.

5 Base directory structure

- /data – apps and widgets are stored here, accessible at http://IP/apps/data/
- /libs – Lua library storage, loaded via *require('custom.lib')* where *lib* is library name.
- /daemon – app daemon are stored here
- /user – allows storing user files and LP scripts, accessible at http://IP/user/

6 Full-screen mode

In order to hide the top navigation bar icons pass *fs* GET variable (variable contents must evaluate to *true*): http://IP/apps/?fs=1

7 App / Widget structure

Application name (ID) must be unique and can only contain alphanumeric characters, hyphens and un- derscores. Maximum name length is 64 characters.

8 Directory structure

- *index.lp* or *index.html* – required for apps, unless *url* is specified, clicking app icon will open app di- rectory in the same window. Applications must provide a *Back* button so user can return to starting page
- *icon.svg* or *icon.png* – required for apps, contains application icon, *SVG* is recommended
- *widget.lp* or *widget.js* – required for widgets, can contain *JavaScript + Lua* code or pure *JavaScript* source which displays widget contents
- *title* – optional for apps, text file with title that is shown beneath the icon
- *url* – optional for apps, text file with *URL* that should be open when icon is clicked
- *style.css* – optional for widget, contains custom *CSS* stylesheet for given widget
- *config.lp* or *config.html* – optional configuration file, see description below

In widget mode icon element ID is the same as widget name, all other HTML element IDs must be pre- fixed with unique application name to minimize collisions between different applications. The same rule applies to CSS selectors.

Note: if your widget has any custom click events, make sure to check for ***unlocked*** class which is set when widget can be dragged by the end user. Click events must be ignored if this class is set.

9 Widget sizing

Default widget size is 100×100px. Width/height can be increased by calling *setWidgetSize(cols, rows)* on widget element. Width formula: cols * 110–10, height formula: rows * 110–10

Examples

Clock widget which takes double width/height and places SVG image which fills all available space inside of widget container:

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

Toggle between square (1, 1) and wide (2, 1) mode after each click.

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

Create new directory named as your application in *daemon* directory. Place *daemon.lua* inside your newly created directory. Daemon is started automatically when device boots up and when application is installed. Daemon is automatically restarted in case of an error, errors are logged to *fellerLYnk* error log. Daemon can be forced to (re)start or to stop via HTTP request:

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

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

Example (daemon.lua)

Minimal example which logs current timestamp every 5 seconds

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

11 Configuration

- Application directory must contain either *config.lp* or *config.html* file
- This file must contain *form* element, id must be set in *myapp-config* format, where *myapp* is unique application name
- Data exchange is done via events triggered on *form* element:

  config-load – (to app) provides an object with all configuration key/value pairs

  config-check – (to app) triggered when Save button is clicked, app configuration must either show an error message if configuration is invalid or trigger *config-save*

  config-save – (from app) saves configuration on server side and closed modal window, application must pass configuration parameters as an object
- Configuration can be accessed from Lua using these functions:

  config.get(app, key, default) – returns single value for given application name, default value or *nil* if key is not found

  config.getall(app) – return table with all configuration values for given application name or *nil* if configuration is empty

  config.set(app, key, value) – adds a new key/value pair or overwrites an existing one

  config.setall(app, cfg) – overwrites existing config with given *cfg table* with keys/values

  config.delete(app, key) – deletes existing key/value pair
- Unpublished apps that have configuration file present will appear under *Dev apps* in admin page

Examples (config.html)

Access config value from *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
```

Set value in *lp* script posted from user:

```
<?
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)
```

Create a simple for element with single numeric input which accepts values in 5..100 range

```
<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 functions

localStorage allows saving client-side configuration. Several functions are provided to safely execute *localStorage* functions, as they might fail is some cases like private mode on *iOS*. It also allows storing any values that can be serialized using *JSON.stringify*.

- *storeSet(key, value)* – sets key/value pair
- *storeGet(key)* – retrieves key value, returns *null* when key is not found
- *storeRemove(key)* – removes key from storage

Storage keys must be prefixed with unique application name to minimize collisions between different applications.

Examples

Get currently selected theme (light/dark)

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

Store JavaScript objects

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

Cookie functions

Use *Cookie* global object to access cookies from client-side.

- *Cookie.set(key, value[, attributes])* – sets key/value cookie pair. If value is a *JavaScript* array or an object, it will be automatically *JSON*-encoded for set calls and *JSON*-decoded for *get* calls.
- *Cookie.get()* – returns object containing all key/value cookie pairs
- *Cookie.get(key)* – returns cookie value
- *Cookie.remove(key[, attributes])* – removes cookie from storage
- Cookie attributes (optional):

  path – cookie path, defaults to "/"

  expires – number (days) or Date object, containing cookie expiration time. By default cookie expires when browser window is closed

  secure – only allow cookie to be sent via HTTPS, disabled by default
- Cookie keys must be prefixed with unique application name to minimize collisions between differ- ent applications.

Examples

Get currently selected theme (light/dark)

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

Set cookie to expire in 1 year

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

Store *JavaScript* objects

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

12 Client-side local bus monitoring

localbus allows monitoring group address and storage variable changes. If used separately, include */apps/js/localbus.js.gz* in your app and call *localbus.init()*. This is not needed for widgets.

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

Runs callback function when group address value changes, callback receives one argument – new object value. Will not work if object does not have a known data type.

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

Runs callback function when storage key value changes, callback receives one argument – new storage value.

localbus.listen('groupwrite', callback)

localbus.listen('groupread', callback)

localbus.listen('groupresponse', callback)

Runs callback function when new group telegram is received, callback receives one argument – event object.

Event object:

- *event.dst* – destination group address
- *event.src* – source address, empty for local telegrams
- *event.tsec* – UNIX timestamp of the telegram (seconds part)
- *event.usec* – UNIX timestamp of the telegram (microseconds part)
- *event.type* – event type, either groupread, groupwrite or groupresponse
- *event.value* – new value, only valid for groupwrite and groupresponse types and when destination object has known data type

Widget example

```
(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 Server-side local bus monitoring

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

Loads localbus library and creates new connection, timeout is an optional value in seconds and is set to 1 second by default.

lb:step()

Wait for a single message or timeout. Returns true on new message, nil on timeout.

lb:sethandler('groupwrite', groupcallback)

lb:sethandler('groupread', groupcallback)

lb:sethandler('groupresponse', groupcallback)

Sets a callback function for each of the group message types.

Callback receives one argument – *event* table:

- *event.dst* – destination group address
- *event.src* – source address, empty for local telegrams
- *event.type* – event type, either *groupread*, *groupwrite* or *groupresponse*
- *event.datahex* – HEX-encoded raw data

lb:sethandler('storage', storagecallback)

Sets a callback function for storage changes.

Callback receives three arguments: *action*, *key*, *value*:

- *action* – either "set" or "delete"
- *key* – storage item key
- *value* – new storage item value, *nil* for "delete" action

Example (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 Translation

- *$.i18n.lang* – current language or *undefined* if default language is used
- *$.i18n.add(ns, dictionary)* – adds translations to current dictionary, *ns* must be a unique application name
- *$.i18n.translate(key, default, vars)* or *$.tr(key, default, vars)* – translates a given *key* or uses *default* value if translation is not found for current language. Additional *vars* object can be passed to replace vari- ables inside of translation text

Example 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);
```

Example 2

You can apply translation to *jQuery* selectors by using *tr* function: all HTML elements that have *tr* class and *data-tr-key* attribute will have contents replaced with translated version

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

Allows mixing HTML and Lua inside a single file, Lua chunks must be enclosed in <? ?> tags, closing tag at the end of the document is not required.

Available functions:

- *header(hdr)* – adds a custom header to the output
- *getvar(name)* – returns named *GET/POST* variable or *nil* when variable is not set
- *getvars()* - returns all *GET/POST* variables as *Lua table*
- *getcookie(name)* – returns named *cookie* contents or *nil* when cookie is not set
- *getheader(header)* – returns named *header* contents or *nil* when header is not set, might return *table* when header with the same name has been passed several times. *Header* case does not matter, it is normalized to a pure lowercase form with all underscores converted to dashes in case of a lookup miss
- *setcookie(name, value [, expires [, path]])* – sets named *cookie* contents, must be called before any output is sent, *expires* must be a number (days) or a table, containing cookie expiration time. By default cookie expires when browser window is closed. Default *path* value is "/"
- *print(...)* – outputs any number of variables, ending output with *CRLF*
- *write(...)* – similar to *print* but does not output **CRLF** at the end
- *escape(val)* – escape single/double quotes, less than/greater than characters to HTML entities
- *include(file)* – include and run another lp file, note that parent local variables are not visible inside of the included file (see example below for more info)

request table contains information on current file and path:

- *request.file* – full path to the requested lp file
- *request.path* – full path to the directory containing the requested lp file

Library package is loaded via *require('apps')* and provides access to these functions:

- all built-in HL functions: *alert*, *log*, *grp*, *storage* etc if core HL package is installed
- *config* library
- *vprint(...)* and *vprinthex(...)* functions to view variable contents in human-readable form
- *json* library
- *getlanguage([default])* – returns selected anguage for current user or default value (*nil* when not specified) when language is not set

Examples

Print current date

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

Output multiplication table. Size can be a *GET/POST* variable in 1..20 range (defaults to 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>
```

Include document (*row.lp*), must be in the same directory as the parent document (*index.lp*)

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

Parent document (*index.lp*). Note: *ival* global is used because local counter variable *i* is not visible in *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 Lua functions

Full Lua function referenceanual is available at http://openrb.com/docs/lua.htm

16.1 Object functions

Most functions use *alias* parameter — either object group address or object name. (e.g. '1/1/1' or 'My ob- ject')

Finding single/multiple objects

grp.find(alias)

Returns single object for given alias. Object value will be decoded if data type is set.

Returns *nil* when object cannot be found, otherwise it returns *table* with the following items:

- address — object group address
- updatetime — latest update time in *UNIX timestamp* format. Use os.date() to convert to readable date formats
- name — unique object name datatype — object data type
- decoded — set to *true* when decoded value is available
- value — decoded object value

grp.tag(tags [, mode])

Returns a *table* containing objects with given tag. Tags parameter can be either table or a string. Mode parameter can be either 'or' (default — returns objects that have any of given tags) or 'and' (return objects that have all of given tags). You can use object functions on the returned table.

grp.dpt(dpt, [strict])

Find all objects with matching data type. *dpt* can be either a *string* ("bool", "scale", "uint32" etc) or a field from *dt* table (dt.bool, dt.scale, dt.uint32). For example, if *dpt* is set to *dt.uint8*, in normal mode all sub-datatypes like *dt.scale* and *dt.angle* will be included. If exact data type match is re- quired, set *strict* to *true*.

grp.all()

Returns a table with all known objects.

16.2 Helpers

grp.alias(alias)

Converts group address to object name or name to address. Returns *nil* when object cannot be found.

grp.getvalue(alias)

Returns value for given alias or *nil* when object cannot be found.

16.3 Bus requests

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

Sends group write request to given alias. Data type is taken from the database if not specified as third parameter. Returns boolean as the result.

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

Similar to grp.write. Sends group response request to given alias.

grp.read(alias)

Sends group read request to given alias. Note: this function returns immediately and cannot be used to return the result of read request. Use event-based script instead.

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

Similar to grp.write, but does not send new value to the bus. Useful for objects that are used only in visualization.

16.4 Tag manipulation

grp.gettags(alias)

Returns a *table* with all tags that are set for given alias.

grp.addtags(alias, tags)

Adds single or multiple tags to given alias. *Tags* parameter can be either a *string* (single tags) or *Lua table* consisting of strings (multiple tags).

grp.removetags(alias, tags)

Removes single or multiple tags from given alias. *Tags* parameter can be either a *string* (single tags) or *Lua table* consisting of strings (multiple tags).

grp.removealltags(alias)

Removes all tags for given alias.

grp.settags(alias, tags)

Overwrites all tags for given alias. *Tags* parameter can be either a *string* (single tags) or *Lua table* consisting of strings (multiple tags)

16.5 Object creation and modification

grp.setcomment(alias, comment)

Sets *comment* field for given alias

grp.create(config)

Creates a new or overwrites an existing object based on provided *config*, which must be a Lua table. Returns object ID on success, nil plus error message otherwise.

config fields:

- datatype – *required*, object data type. Can be either a string ("bool", "scale", "uint32" etc) or a field from *dt* table (dt.bool, dt.scale, dt.uint32)
- name – *optional*, unique object name. If an object with the same name already exists, nu- meric prefix will be added
- comment – *optional*, object comment (*string*)
- units – *optional*, object units/suffix (*string*)
- address – *optional*, object group address (*string*). If not set the first free address from con- figured range will be used
- tags – *optional*, object tags, can be either a string (*single* tags) or *Lua table* consisting of strings (multiple tags)

If an object with the same group address already exists, only *units*, *datatype* and *comment fields* will be changed. All other properties will be kept unchanged.

Examples

Create new object with known address

```
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' },
})
```

Create new object with automatic address assignment

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

16.6 Database functions

SQLite v3 is used as the database engine.

Note: Database tables must be prefixed with unique application name to minimize collisions between different applications.

16.7 Core functions

- *db:execute(query)* – executes given query, return value can be either a database cursor or query result
- *db:escape(value)* – escapes given *string* value so it can be safely used in a query
- *db:query(query, …)* – executes given query, question marks in the query are replaced by additional parameters (see examples below)

16.8 INSERT/UPDATE/DELETE helpers

Note: *Lua tables* passed as *values* and *where* parameters must not have fields that are not present in given database table. Otherwise query will fail

- *db:insert(tablename, values)* – performs *INSERT* query based on given *values*
- *db:update(tablename, values, where)* – performs *UPDATE* query based on given *values* and *where* parameters
- *db:delete(tablename, where)* – performs *DELETE* query based on *where* parameter

16.9 SELECT helpers

Note: parameters must be passed in the same way as for *db:query()* function

- *db:getone(query, …)* – returns first field value from the first matching row from given query
- *db:getrow(query, …)* – returns first matching row from given query
- *db:getlist(query, …)* – returns complete query result as *Lua table*, where each table item is first field from each row
- *db:getall(query, …)* – returns complete query result as *Lua table*, where each table item is Lua table with field → value mapping

Examples

```
-- 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
Téléphone +41 21 653 24 45 | Telefax +41 21 653 24 51

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

by Schneider Electric