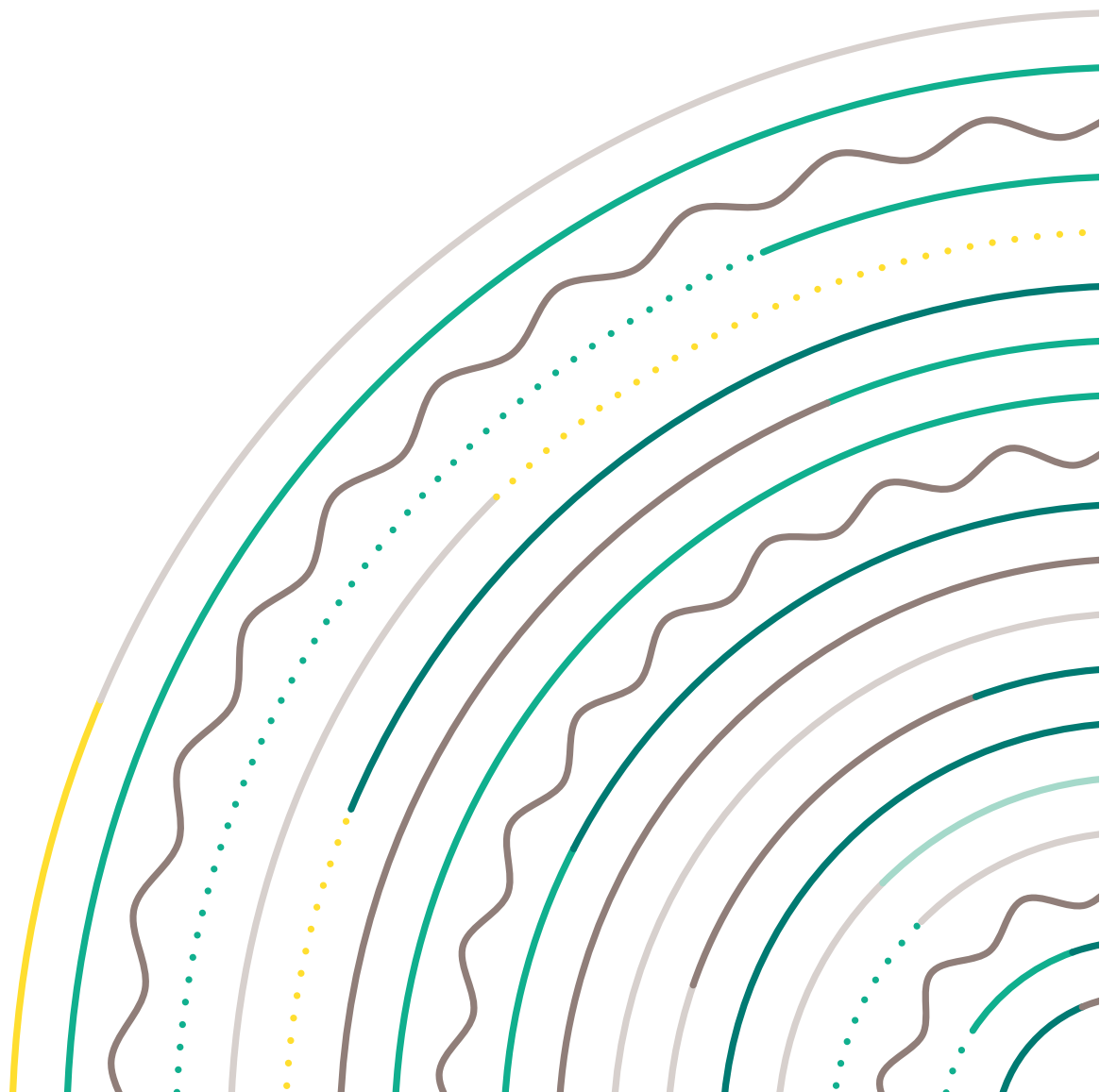


Lifedomus

Universal

29/01/2018

Version 1.2



Universal protocol

Table of contents

1	Operating principle	3
2	Connector creation	3
2.1	Common properties	3
2.1.1	Import/Export	3
2.1.2	Debugger	3
2.1.3	Cyclic readout	3
2.1.4	State feedback persistence	3
2.2	HTTP connector	4
2.3	IP connector	6
3	Javascript	8
3.1	Reserved variables	8
3.2	Useful functions	8
3.3	Numbered types	8
3.4	Type List	9
3.5	Action parameters	9
4	Device properties	10
4.1	Editing actions	11
4.1.1	Specific properties of the HTTP connector	12
4.2	Edit variables	13
5	Widgets	13
5.1	URL	13
5.2	Slidebar	13
5.3	TrackBar	14
5.4	Playlist	15

1 Operating principle

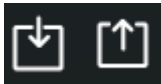
The universal connector can be used to add a customised protocol in Lifemodus, to control an Audio/Video device, a home automation protocol or any other device that can be controlled in IP, RS or HTTP.

It sends ASCII or Hexa commands to a module connected to the network, identified by a specific address.

2 Connector creation

2.1 Common properties

2.1.1 Import/Export




: You can export your module to share it. The export includes connector configuration, JavaScript, devices associated with this connector, as well as the models associated with the devices, if they differ from the system's models.



: You can import a Script in JavaScript to run data parsing. When you import your script, it will be compiled and you will be notified in the event of a compilation error. If an error occurs, the old script will be stored. The connector must be restarted to take the JS into account

2.1.2 Debugger

The  icon opens a web browser displaying your connector's debugger. The display includes incoming frames, as well as messages that you can write in your JS using the `ldprint()` method.

If your JavaScript has an infinite loop, Lifedomus will stop the connector and display 'Break BI' in the debugger

2.1.3 Cyclic readout

In the connector's properties, you can determine the update time for some states. This period of time corresponds to the repeated sending of commands whose cyclic readout is activated (see Device Properties)

2.1.4 State feedback persistence

By default, all state feedbacks are reset when the server starts. The persistence of these states can be stored every time the server starts.

2.2 HTTP connector

In the connector management window, select 'Universal' then 'Universal TCP/IP' in the top toolbar and create a 'Universal HTTP' connector.

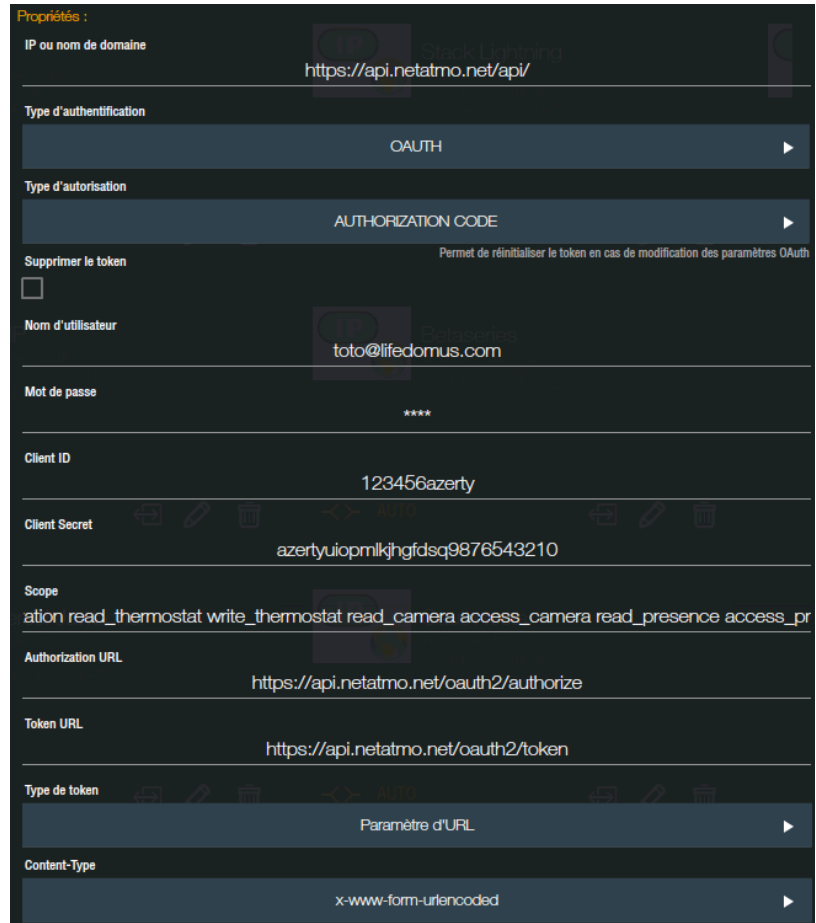
First enter the API's or device's basic URL then enter the authentication parameters.

OAUTH authentication:

If the authentication type is OAUTH (only V2 is accepted) you must enter the following properties:

- Authorization type: 'grant_type' parameter generally specified at API level
- Remove token: Resets the token if the parameters have been edited (scope, client ID, etc.)
- User name, password, client ID, client secret: Account login details
- Scope: Specifies the scopes (separated with a space) required to run the commands
- Token type:
 - Bearer: Sends the token via an 'Authorization: Bearer {token}' header
 - URL parameter: Adds an 'access_token={token}' parameter in the URL
- Content-type: The method by which the OAuth parameters will be posted during a call to 'Token URL'

On first login, your browser will open, asking you for access to the API data by the Lifemodus application. Confirm and a message notifying you of the successful or unsuccessful login will appear.



The screenshot shows the configuration window for a Universal HTTP connector. The 'Propriétés' tab is active. The 'IP ou nom de domaine' field contains 'https://api.netatmo.net/api/'. The 'Type d'authentification' is set to 'OAUTH'. The 'Type d'autorisation' is set to 'AUTHORIZATION CODE'. The 'Supprimer le token' checkbox is unchecked, with a note 'Permet de réinitialiser le token en cas de modification des paramètres OAuth'. The 'Nom d'utilisateur' field contains 'toto@lifedomus.com'. The 'Mot de passe' field is masked with '****'. The 'Client ID' field contains '123456azerty'. The 'Client Secret' field contains 'azertyuiopmlkjngfidsq9876543210'. The 'Scope' field contains 'ation read_thermostat write_thermostat read_camera access_camera read_presence access_pr'. The 'Authorization URL' field contains 'https://api.netatmo.net/oauth2/authorize'. The 'Token URL' field contains 'https://api.netatmo.net/oauth2/token'. The 'Type de token' is set to 'Paramètre d'URL'. The 'Content-Type' is set to 'x-www-form-urlencoded'.

Initialisation frames:

To control them or before interfacing with them, some modules require authentication or a specific mode. You will be able to enter one or several frames sent when the connector starts. The frames will be sent separately.

Click on '+' to configure the frames:

Get Datas
Init status

Label :

Init status

Comande :

/init_status

Méthode :

POST

Paramètres :

```
{
  "on" : true,
  "xy" : [0.123, 0.456]
}
```

En-tête :

Nom	Valeur
Accept	application/json

Enter the label, command to be sent and the HTTP communication method (GET, POST, PUT, DELETE). If the send method is not GET, you can enter the command parameters, either in raw form (e.g. *on=true&value=1234*), or in JSON format.

Finally, enter one or more http headers then confirm.

2.3 IP connector

Enter the module's IP address and communication port and specify the protocol used for sending commands.

The following fields are optional:

Escape characters at the end of outgoing frames:

Characters used at the end of each frame sent to the module. '\r' and/or '\n' are the most commonly used. These characters are represented in various forms in the documentations.

In Lifedomus, the hexa code for the characters to send must be entered:

- 0A for \n or <LF>
- 0D for \r or <CR>

If for example, both are required, enter 0A0D for the outgoing escape command value to be 0A then 0D.

If you must send a different code, enter the associated code in hexadecimal.

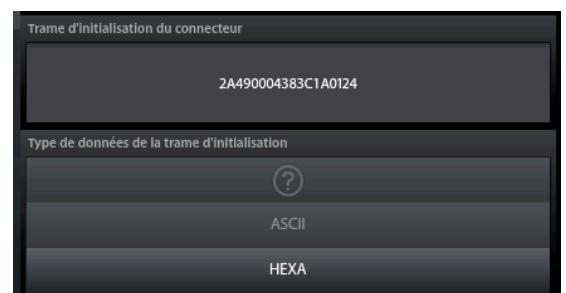
Waiting time between outgoing commands:

Some modules have problems sending commands too close together. You can therefore specify the minimum waiting time between the sending of the commands. The time is expressed in ms.

Connector initialisation frame(s):

To control them or before interfacing with them, some modules require authentication or a specific mode. You will be able to enter one or several frames sent when the connector starts.

If the connector does not run properly, a message will notify the user of the associated error.



Enter the type of outgoing frames: ASCII or HEXA.

In HEXA, commands are made up of bonded hexadecimal codes.

Example: frame sent to an lport

Sent from Controller

RequestIPodName	
Value	Meaning
0x2A	Start character
0x49	Device type (iPort)
0x00	Packet Length High Byte
0x03	Packet Length Low Byte
0x38	Packet Command Type
0x3C	Packet Command High Byte
0x28	Packet Command Low Byte
0x18	Checksum

The frame will be written in Lifemodus as follows: 2A490003383C2818

Escape characters at the end of incoming frames:

This parameter is used to activate the recovery of information via JavaScript. If you only want to control, you can leave '?' in the field.

Characters used for parsing data received by the connector. You can select <CR>, <LF> or both.

If there is no escape character and/or if you want to calculate the frame length yourself, enter 'None'.

If it is another character, select 'Others' in the drop-down list then enter the characters directly in the following field.

Example: frame ends with a star



Finally, the last case scenario, if the frame size is variable and can only be seen by reading part of the frame, you can specify how to calculate it.

To do this, select 'Size' in the combo then complete the following fields:

Value to be added to the size to calculate: if the frame size does not include the header, add the size of this header.

Size calculation type: you can select 'byte' or 'bit'. If the size is calculated with bytes in the frame, select 'byte' (most cases). If the size is calculated using X first or last bits in a byte, specify 'bit'.

First Byte/bit and Last Byte/bit: Specify the first and last byte or bit for the calculation

E.g. lport:



Packet Structure

The iPort communication packet structure is constructed using a combination of ASCII characters and hex data. A packet consists of seven elements described below.

Byte Number	Value	Meaning
0x00	0x2A	Start character
0x01	0x49	Device type (iPort)
0x02	0xNN	Packet Length High Byte
0x03	0xNN	Packet Length Low Byte
0x04	0xNN	Packet Command type
0x05	0xNN	Packet Command High Byte
0x06	0xNN	Packet Command Low Byte
0x07 – 0xNN	0xNN	Packet Data Bytes
0xNN	0xNN	Checksum = 2's compliment of the sum of all bytes excluding the start character and checksum

3 Javascript

For data parsing, you can decide on the development method and can use all JavaScript functions. Here is a list of useful features and reserved variables:

3.1 Reserved variables

- **frame:** Contains the response sent by the module, and thus the base of each parser. Use this variable to update your state feedbacks.
Depending on the type of escape characters at the end of a frame, 'frame' will appear as a character string, byte or byte table:
 - **None:** Frame will contain a byte
 - **Others, \r, \n, \r\n:** Frame will contain a character string
 - **Size:** Frame will contain a byte table
- **command:** As in the JS type device commands, detailed subsequently, there is a variable exclusive to Lifedomus, namely the 'command' variable. If you write something in this variable, Lifedomus will send it at the end of the execution of your JavaScript. This way, you can create a sequence of commands.
- **ip:** For a Universal IP connector, the IP variable is reserved by Lifedomus to provide you with the IP address entered in the connector's properties.

3.2 Useful functions

- **ldprint(message):** This method displays a message in the debugger
- **utf8_decode(utf8_text):** Decrypts a string in UTF-8 format
- **JSON.parse(str):** parses a character string in JSON and builds the JavaScript value or the object described by this string.
- **eval(str):** Evaluates the JavaScript code represented in character string form. Very useful to assign variables dynamically.

3.3 Numbered types

For non generic devices (e.g. light, roller shutter, etc.), a number of actions and state feedbacks are available by default. These states can be simple (BOOLEAN, STRING, DOUBLE), or numbered as set in the Lifedomus system. These types are used in JS with the following values:

```
CLIM_MODE =
{
    AUTO: 0,
    HEAT: 1,
    COOL: 3,
    FAN: 9,
    DRY: 14,
    OFF: 99
}

BATTERY_LEVEL =
{
    NO_STATUS_AVAILABLE: 0,
    LOW: 1,
    MEDIUM: 2,
    FULL: 3
}

FAN_SPEED =
{
    LOW: 0,
```



```
HIGH: 1,  
MEDIUM: 2,  
AUTO: 3,  
TOP: 4  
}  
  
THERMOSTAT_HEAT_MODE =  
{  
    OFF: 0,  
    HEAT: 1,  
    COOL: 2,  
    AUTO: 3  
}  
  
THERMOSTAT_MODE =  
{  
    OFF: 0,  
    FREEZEOUT: 1,  
    ECO: 2,  
    CONFORT_MINUS_1: 3,  
    CONFORT_MINUS_2: 4,  
    CONFORT: 5,  
    MANUAL: 6,  
    PROGRAM: 7,  
    AWAY: 8,  
    REDUCED: 9,  
    PROTECT: 10  
}
```

For example, if you use a 'mode' variable to manage the mode of a universal air-conditioning unit, you can thus assign the variable as follows in your parser:

```
mode = CLIM_MODE.COOL;
```

3.4 Type List

A last specific type exists in universal type, namely List type. This type is exclusively used to manage the widget list in Design Studio and appears in two-dimensional table form as follows:

```
var list = new Array();  
list[0] = new Array();  
list[0][0] = '1';  
list[0][1] = 'Element 1';  
list[1] = new Array();  
list[1][0] = '2';  
list[1][1] = 'Element 2';  
...
```

3.5 Action parameters

Prior to executing a universal action to a device, you may want to recover the value or a variable to edit or use it as action parameter. All you need to do is enter the name of the variable between braces. The usable parameters can either be an action parameter, or any javascript variable set or not as device variable.

Example with a 'value' variable:

```
value = value + 1;  
command = '/setCpt={value}';
```

Important: When such a feature is used with HTTP parameters in JSON format (see Device properties), the 'value' parameter must be written as follows: $\$(value)$

4 Device properties

The universal connector is available for all generic devices, as well as for all the following devices:

Audio/Video	DVD player Television Satellite Amp Tuner Video Proj
Heating/Cooling	Room thermostat Radiator Towel warmer Boiler Cumulus Air-conditioning Underfloor heating
Lighting and sockets	Lighting Sockets 230V dimmer 1-10V dimmer RVB LED
Motors	Cinema screen Electric gate Electric door Garage door Awning Pool protection Roller shutter Blind Skylight Horizontal curtain Vertical curtain
Ventilation	CMV
Consumption	Gas consumption Water consumption Fuel consumption Liquid consumption Electric meter Electric power Intensity Voltage Electric frequency
Sensors	Anemometer Air humidity Ground humidity Brightness Noise level CO2 level Rainfall Atmospheric pressure Thermometer
Detectors	Broken window Closing Door bottom Gas leak Water leak Fuel leak Liquid leak Smoke/Fire Motion Opening Presence Fall Emergency calls Rain detector Snow detector


















Create the required device and edit its properties. If the device is not generic, then it will show a default list of actions and/or default state feedbacks to ensure the good working order of the associated widget. All default commands cannot be removed from the device.




The device properties are compiled in 2 separate tabs: commands and variables

4.1 Editing actions

By default, the list of set actions is displayed, indicating the associated frame type on each one (ASCII, JS or HEXA), as well as the quick action buttons (edit, delete and execute action).

Commands created by the user appear in a yellow frame and can be deleted

Commandes	Variables
LED allumée / ASCII	 
LED éteinte / ASCII	 
Choix de la couleur RVB / JS	 
Read / ASCII	  
Commuter la LED	 
Variation de la LED en %	 
Positionner la valeur du blanc froid	 
Positionner la valeur du blanc chaud	 

 Execute action
 Edit action
 Delete action (only those created by the user)

LED allumée
LED éteinte
Variation de la LED en %
Choix de la couleur RVB
Read
Commuter la LED
Positionner la valeur du blanc fr...
Positionner la valeur du blanc c...

Label :
Variation de la LED en %

Type :


Ascii

JS

Commande :

```
bri = parseInt({percentage} * 2.5, 10);
command = '/api/Lifedomus1234/lights/13/state';
```

Lecture cyclique :
☐

For each action, enter the label, command type (ASCII, JS, or HEXA), as well as the command you want to execute. If the action has settings, you can see the names of these settings by clicking on the  icon.

In the above example, the 'LED dimming' action has a parameter called 'percentage', ranging between 0 and 100. This parameter value can be reused to convert it from 0 to 255 that can be allocated to the 'bri' variable. Finally, edit the value of the reserved 'command' value with the url to call up.

It is also possible to enhance an action of additional parameters by adding parameters between braces in the command.

Example:

Label :

Set Volume

Type :

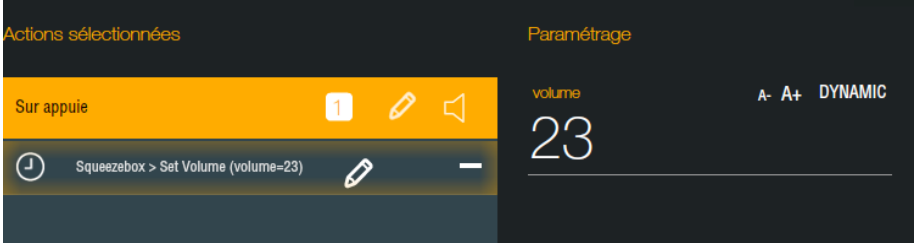
Ascii

JS

Commande :

{zone} mixer volume {volume}

Here, {zone} refers to a global variable, and is thus not interpreted as an action parameter, only {volume} will be used as a parameter. The following will appear in Design Studio:



Finally, it is also possible to repeat a command cyclically (e.g. to update all state feedbacks).

The time for sending cyclic commands is specified in the connector's properties.

4.1.1 Specific properties of the HTTP connector

Méthode :

PUT

Paramètres :

```
{
  "on" : true,
  "bri" : $(bri)
}
```

En-tête :

Ajout d'une en-tête

Parseur JS de la commande :

☒

If the device is associated with an HTTP connector, other settings must be entered:

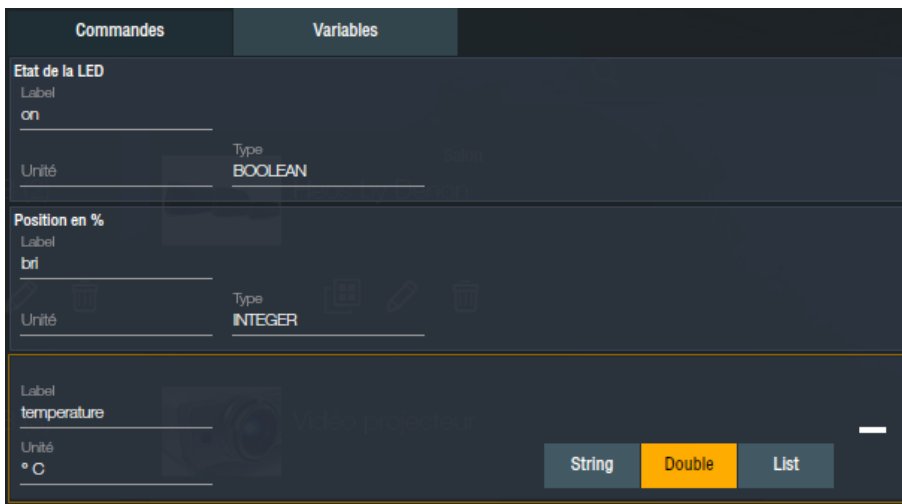
As with the connector's other initialisation frames, specify the HTTP communication method (GET, POST, PUT, DELETE). If the send method is not GET, you can enter the command parameters, either in raw form (e.g. *on=true&value=1234*), or in JSON format

In this case, the 'bri' variable is used in the JSON settings, recovered as follows: *\$(bri)*

You can also indicate headers as well as a specific parser associated with the command.

If no command parser is set, the connector's parser will be called up to evaluate the query's response.

4.2 Edit variables



For a non generic device, it has preset default state feedbacks, whose type cannot be edited. You can associate a state feedback with a JS variable, specifying the label of the variable.

The unit can also be edited (free text field).

For the user variable, the type of variable must be specified (only String, Double and List available). Variables are automatically saved when a field is entered.

5 Widgets

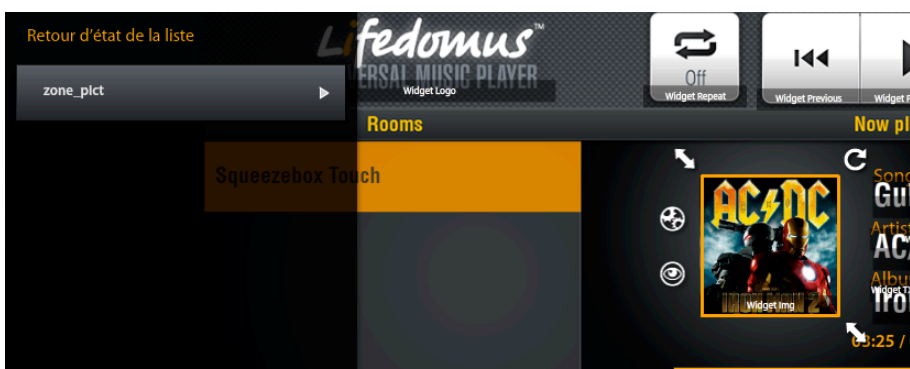
For non generic devices, you will see all preset widgets used for other protocols.

For others, there are widgets designed for using the bidirectional universal module:

5.1 URL

You can create a widget with an URL state feedback to an image.

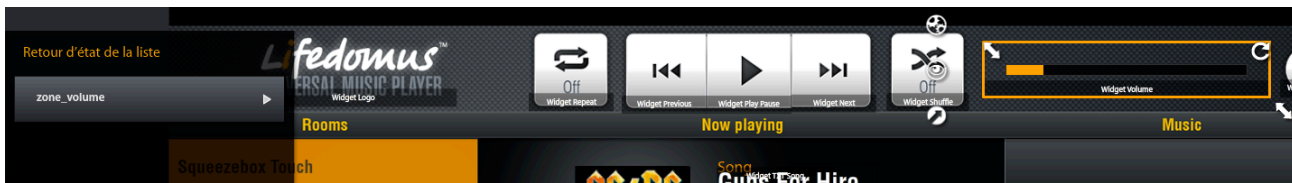
Example: a sleeve for the Audio device medium



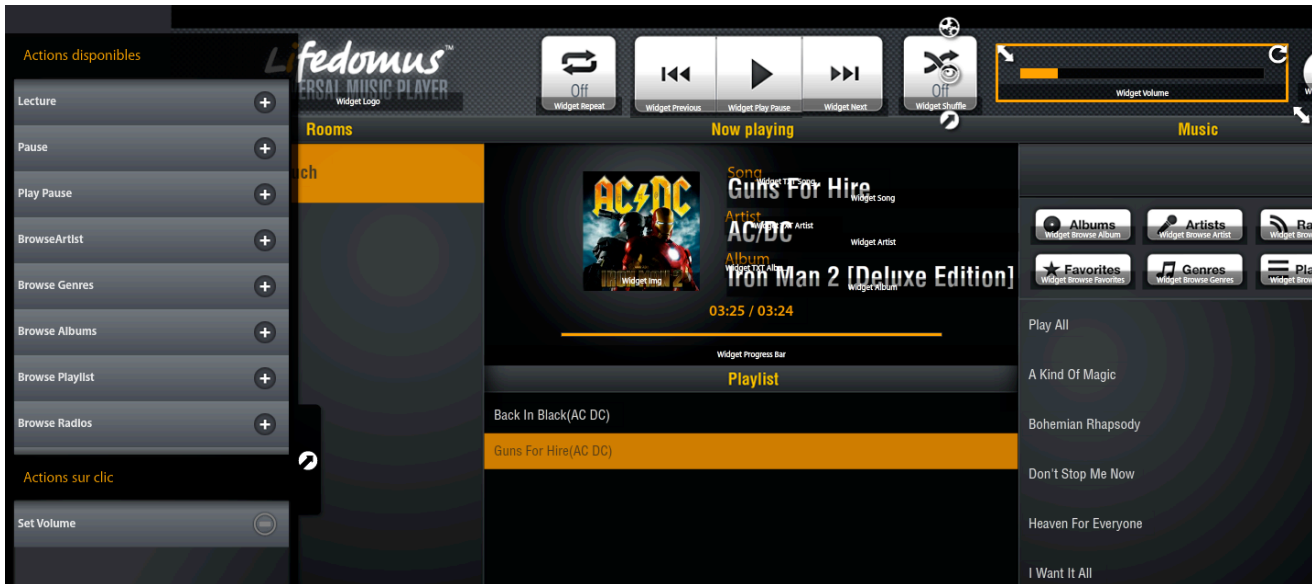
5.2 Slidebar

You can create a progress bar to manage the volume, bass, treble, etc.

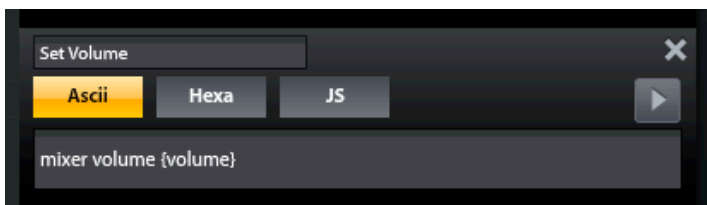
This widget is associated with a numeric state feedback as a percentage



and one or more configured actions that will replace the command setting with the entered percentage:



The SetVolume action is declared as follows in Config Studio:

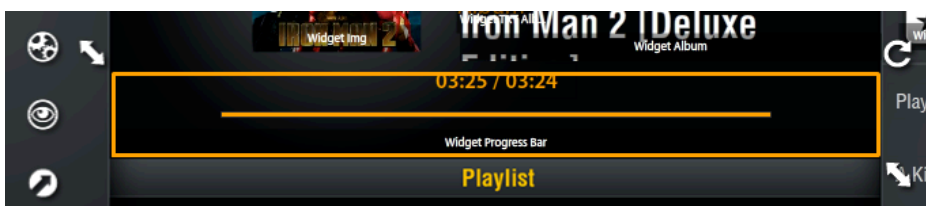


In this example, when the user places the progress bar in the middle, {volume} will be replaced by 50 in the command.

The command sent to the module will be: 'mixer volume 50'.

5.3 TrackBar

Operates like the SlideBar widget to which several states are associated.



The first state is the length of a 'number' track to see the maximum size of the trackbar.

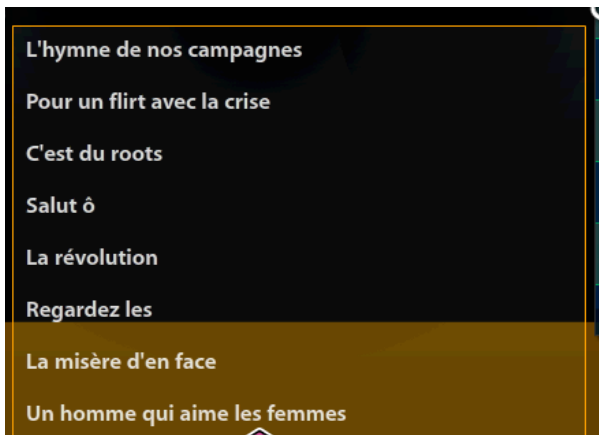
The second state is the current position (in time) of the track, to synchronise the progress of the trackbar.

If the module does not automatically return the state of the track every second, you can add a third state as a number to increment the bar automatically:

0: stop
 1: playback
 2: playback X2

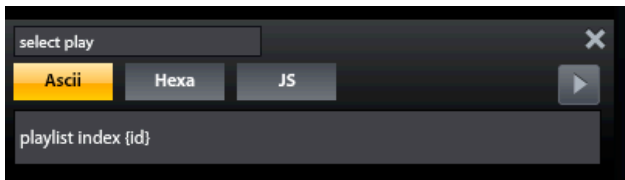
5.4 Playlist

The playlist widget to which only 'list' type variables can be associated for a playlist (see item 4, device management). When clicking on an item in the playlist, you can associate an action that can have a parameter that will be replaced by the ID part of your playlist:



```
var playlist = new Array();
playlist[0] = new Array();
playlist[0][0] = '1';
playlist[0][1] = 'L'hymne de nos campagnes';
playlist[1] = new Array();
playlist[1][0] = '2';
playlist[1][1] = 'Pour un flirt avec la crise';
...
```

The associated action is the following:



Like with the volume, {id} will be replaced with the item 0 of your list, here 1 or 2 ...

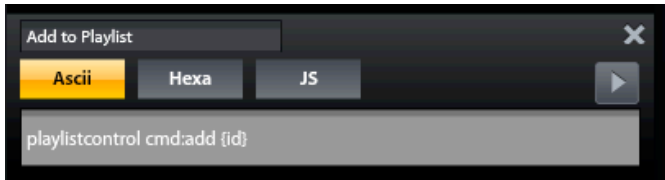
This creates the following frame sent to the module: 'playlist index 2' if you click on 'Pour un flirt avec la crise'.

Item 0 can be an ID like in the previous example, though it can also be part of the command sent like in the example below.



```
var playlist = new Array();
playlist[0] = new Array();
playlist[0][0] = 'track_id:112';
playlist[0][1] = 'Around the World';
playlist[1] = new Array();
playlist[1][0] = 'track_id:114';
playlist[1][1] = 'Californication';
...
```

For the following command:

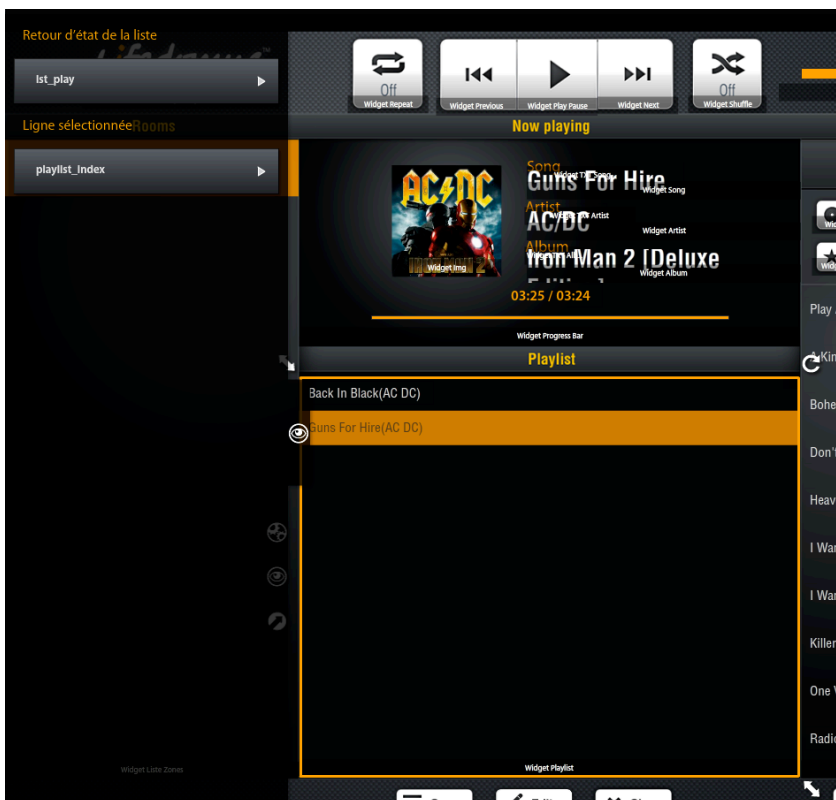


The frame sent to the module will be: "playlistcontrol cmd:add track_id:112".

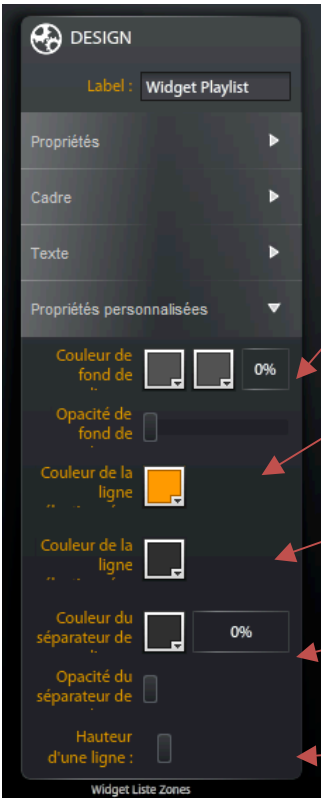
Highlighting a line in the list:

To highlight a line in your table, you can associate a state feedback in the 'selected line' property. The state feedback must contain one of the IDs on the list.

Example current Playlist:



The widget is fully configurable graphically using the property panel:



DESIGN

Label : Widget Playlist

Propriétés

Cadre

Texte

Propriétés personnalisées

Couleur de fond de .. 0%

Opacité de fond de ..

Couleur de la ligne ..

Couleur de la ligne ..

Couleur du séparateur de .. 0%

Opacité du séparateur de ..

Hauteur d'une ligne : ..

Widget Liste Zones

Colours and opacity of the items in the list.

You can use different colours: a yellow line and a red line

Colours of the item in the list when you click on it.

Colours of the item in the list when it is selected via the Selected line state feedback

Colours and opacity of the delimiter between the lines

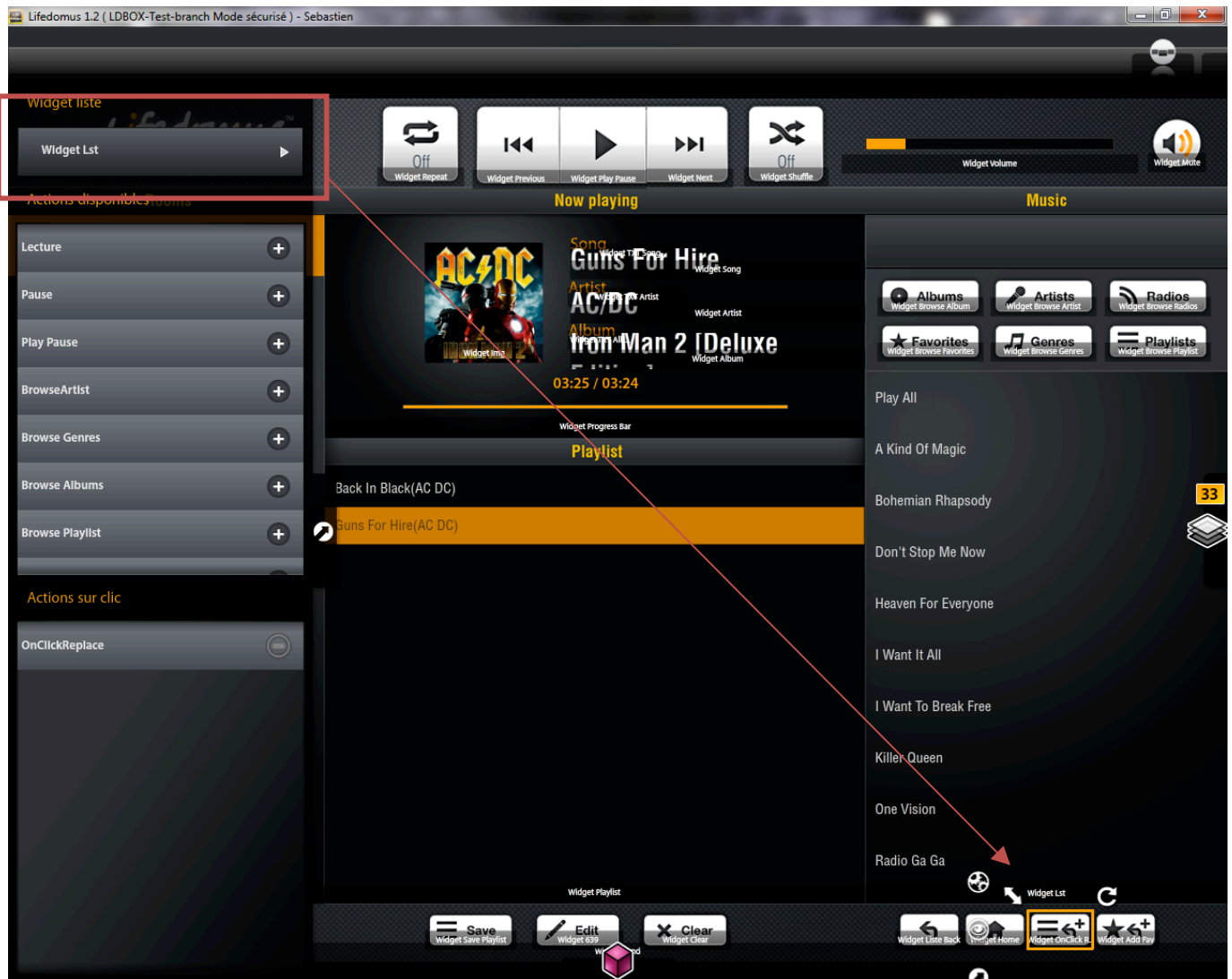
Height of a line in the list

Playlist Widget Switch action

This widget is used to perform a different action on the list than the default action.

For example, click on the playlist to execute the associated command: browse command in the artists, albums, tracks.

However, clicking on the widget will execute the command associated with this widget, taking into account the ID of the item selected from the playlist. For example, adding the artist, album or track to the current playlist.



Associate the widget action switch to the playlist widget with which it must interface and add the commands you want to execute (in the above example: OnClickReplace).