# DMX Scenes

A fundamental requirement which is necessary to control the colour lighting is storing the favourite colours of the RGB lamps. In order to meet the customers' expectations we have created a solution that allows you to store any number of DMX light scenes.

**What can you learn from this tutorial?**

After reading this tutorial you will be able to create interfaces which enable saving/invoking any number of DMX scenes using the menu of the **Remote** application and visualization. You do not need programming skills to perform this functionality. The whole necessary code is included in the `rgbmem.lua` script, which is available in the update of the `resources` component in version `130308` for the **Base** module. The script code can be seen in the following tab: **Ressources** > **Scripts** > **dmxscene.lua**. If for some reason there is no script in your module, you can add it by clicking on the **Add** button and then indicating the file on the local drive. The script is available as an attachment to this tutorial.

After reading this tutorial you will be able to create interfaces which enable saving/invoking any number of DMX scenes using the menu of the **Remote** application and visualization.

## 1. Groups and scenes declaration

In order to start creating light scenes, first of all import the `dmxscene.lua` script to the **Logic** tab. To import the scripts use `import '<script>'`, in this case `import 'dmxscene'`.

In order to make scenes management more intuitive, the script will group them automatically according to the given name.Thanks to this you will be able to create many scenes connected with e.g. the given room or a particular source of light. You can add an unlimited number of scenes to the group. It is very easy to create a new group - just add a single code line in the **Logic** tab as in the example below:

`<variable> = dmxscene('<group>',...)`, where `<variable>` is any name of the variable, by which you refer to the group. In the following part of the tutorial the variables will be called objects. `<Group>` stands for any group name. Instead of `...` enter numbers of the DMX channels, which are to be assigned to a given group. The script enables you to assign any number of the DMX channels to the group.

**Examples of the ready group declarations:**

`living room = dmxscene('living room',2,3,4,5,6,7,8,9,10)` – group connected with the room

`living room ceiling = dmxscene('ceiling',2,3,4)` – scene connected with a given lamp.

Please bear in mind to save the **Logic** tab.

The script enables saving many scenes referring to the same set of DMX channels (to the same group). The `save` function is used to perform such action. Pass the scene name you want to save as an argument to the `save` function. The syntax is as follows: ob-

`ject:save('name')`, e.g. `living room:save('relax')`. If this function is invoked, the DMX channel values given in the group declaration will be saved to the `'relax'` scene. In this example these are 2 and 10.

The scenes are saved in the non-volatile memory of the **Base** module. Each time the `save` function is invoked, the scene is overwritten with the current values of the DMX channels which are assigned to a given scene. The scene values can be seen in the **State** tab. Each scene is a separate MEM variable with the same name as the name of the group and scene. The value of a given MEM variable includes all the values of channels expressed in the hexadecimal code. It should be written as follows: `MEM.dmxs.<name>.<scene>=<value>`.

In order to invoke the scenes, use the `restore` function. The `restore` function takes the scene name you want to invoke as an argument. The syntax is as follows: `object:restore('name')`, e.g. `living room:restore('relax')`.

Having declared the groups you can create the interface to control scenes.

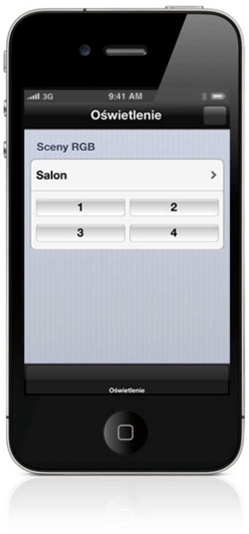
## 2. Remote

Interface in the **Remote** application should consist of at least two elements: **RGB** light- selection of the colour and brightness of the light and **Button** – button to save and invoke the scene. Short pressing the button invokes the scene, while long pressing saves the current values of the DMX channels specified in the group declaration. If there are several scenes declared for a given set of the DMX channels, add the appropriate amount of **Buttons**.

The following procedure refers to the declaration of the `'living room'` group shown in the previous section.

1. Add new **RGB Light** element and then double click on it. Fill in its properties. In the next fields enter numbers of the DMX channels. In next fields enter numbers of the DMX channels which are assigned to the given light colour, e.g. `DMX.2`, `DMX.3`, `DMX.4`. Channels from 2 to 10 belong to this group so this step should be repeated three times.
2. Add **Button**, double click on it and fill in its properties:
   - In the field **Label** enter the button description.
   - Choose the **Pressing** tab.
   - Click on **Add command** and in the displayed window in the **Name** field enter: `C.LOGIC` and in the **Value** field enter: `<name>:restore('<scene>')`, in this case `living room:restore('relax')`.
   - Go to the **Pressing** tab and repeat the previous step changing the contents of the **Value** field to `<name>:save('<scene>')`, in this case `living room:save('relax')`.

Below we present an example of the interface:



## 3. Visualization

Invoking of the DMX scenes can be easily implemented on the visualization. As control elements use controls like **Button**. The definition of the control is limited to assigning the command and label (optional) to the **Buttons.**

The command should have the syntax as follows: `LOGIC=object:restore(’sce-ne’)`, in this case: `LOGIC=living room:restore(’relax’)`.

```
--
-- DMX Scenes
--
-- Copyright 2013 DOMIQ Sp. z o.o.
--
function dmxscene(group,...)
     assert(group, "Group name required")
     for _,v in ipairs(arg) do
          assert(v >= 0 and v <= 255, "Invalid slot value "..v)
     end
local t = {}
function t:save(name)
     local val = {}
     for _,v in ipairs(arg) do
          table.insert(val,
          string.format('%02x',
          math.ceil((get('DMX.'..v))*2.55)))
     end
     set(string.format('MEM.dmxs.%s.%s',group,name),
     table.concat(val))
end
function t:restore(name)
     local val = get(string.format('MEM.dmxs.%s.%s',group,name))
     assert(val,"Missing scene "..name)
     local c = 0
     for v in string.gmatch(val,"(%x%x)") do
          c = c + 1
          command('C.DMX.'..arg[c],
          math.floor(tonumber(v,16)/2.55+0,5))
     end
end
     return t
end
```