Creating your first Live Tab for Adobe LiveMotion™

Before you begin

Install LiveMotion

What is a Live Tab?

LiveMotion SDK allows you to design a Live Tab to extend the functionality of LiveMotion. Live Tabs are custom palettes that use the inbuilt preview engine of LiveMotion to drive the interfaces. Just as you could open a normal .LIV file and preview its animation, scripting, etc., you could load a Live Tab .LIV (by placing it in your Live Tabs folder, launching LM or choosing it from the Live Tabs menu) and see its scripting and animation previewed the same way.

So, to create the interfaces for a Live Tab, you just need to know how to create a user interface: buttons, sliders, text fields, etc. (they work the same way as in a liv file after they're fully implemented). The difference is that when a user interacts with these controls, they're programmed

to send commands to LiveMotion(composition) itself instead. That is, the controls in the Live Tab send out automation/player script commands to another composition or liv file. Since the underlying preview engine is the same, Live Tabs can do the same things as any other liv file would.

A Live Tab is an extension to LiveMotion that you would create to perform certain complex or repetitive tasks. New functionality can be added to LiveMotion that can be controlled through Live Tab palettes. JavaScript is the underlying scripting technology that achieves this functionality. They can also be created to combine a sequence of tasks. They can be used to do the following:

- Create custom dialogs
- Create palettes
- Add your own menus

Live Tabs are basically created with the help of automation scripts as well as player scripts.

Requirements

- Knowledge of automation script
- Knowledge of player script
- Fluency with LiveMotion
- JavaScript

Why Live Tab?

LiveMotion provides you with in-built menu commands, keyboard shortcuts, palettes and tools. But if your requirement is such that you end up repeating certain steps then it would be better to collect those steps and write an automation script. You would then, just have to run that automation script to perform those multiple steps and your job is done. This saves you time. You also don't need to remember the steps.

But lets' say there is some user interaction involved, like choosing a color. Automation script

alone cannot do the job. However, this can be overcome, by using player scripts.

This means here we need the automation script functionality with the player script's user interactivity. Live Tab allows us to use a combination of player script and automation script. Live Tabs are like saved dialogs that can be run whenever required.

Building a Live Tab:

Decide the functionality of the Live Tab

What you would like the Live Tab to do?

Design the UI of the Live Tab

Decide what different objects you would need on your composition like a dynamic text object for user input, or a simple button or a check box etc. What you would like each of the objects to do. What – (if) different states would be required and so on.

Functionality of the individual objects within your Live Tab

What is the purpose of each object you placed on the composition? How and where the scripts will be added.

Tutorials:

Automation Scripts

Let us write an automation script to apply color to selected objects.

1. Open a new script.
2. Add the following code in it.

3. applyColor(43, 129, 24);
   function applyColor(iRed,iGreen,iBlue) //

```
Function to apply color
{
  var selection =
application.currentComposition.selection;
// puts all currently selected objects in the
array selection

application.currentComposition.saveSelecti
on(); // saves the current selection
  for (var i=0; i<selection.length; i++)
  {   // loops through all the selected objects

selection[i].layers[0].colorGradient.startCol
or.red = iRed; // Red value

selection[i].layers[0].colorGradient.startCol
or.green = iGreen; // Green value

selection[i].layers[0].colorGradient.startCol
or.blue = iBlue;  // Blue value
  }

application.currentComposition.restoreSele
ction(); // restore the current selection
}
```

4. Save the file as *applyColor.js*

5. Open a new composition.

6. Create one rectangle and one circle on it with different colors.

7. Automation > Run Automation Script > Navigate to applyColor.js and open it.

8. Result: All the objects are changed to green color.

Now say we want to have different color options to select while applying the color. For this we could convert the above automation

script into a Live Tab.

Live Tabs

Let us now convert the above automation script into a Live Tab that provides you with a palette of different colors. Each color - represented by a button, that the user clicks to apply that color to an object in any LiveMotion composition.

Why you would need this as a Live Tab?

As you know you can create a desired color by experimenting with the RGB (Red, Green, Blue) values in LiveMotion's Color palette. However, each time you need a new color you have to repeat this procedure until you arrive at the color you want. This could be very time consuming. But it would be a lot easier if you had a palette with different colored buttons in it. These could be clicked to apply the color to objects in your composition.

Let us now build a Live Tab with different colored buttons, that can be clicked and as a result that color is applied to the selected objects.

Steps:

1. Open a new composition about the size of the palette with width=190 and height=100. If needed, this can be changed later.

2. Create a small rectangle to represent a color, and name it "button1".

3. Give button1 a color combination from the Color palette, say the RGB values are 43, 129, 24 for Red, Green and Blue respectively.
   If you do not have the Color palette open,

open it using Window > Color.
(You can use any other combination you would like to try. This is the combination for dark green.)

4. Save the composition and name it *Swatches.liv*. Keep saving it at short intervals.

5. Now we would like this button button1 to function in such a way that whenever clicked, it should apply this color to whatever objects are selected. Hence we can say that the script required for this should be written in button1's *down state* or button1's *onButtonPress* event handler.

   Here we will be writing script on button1's down state. So first create a down state for button1. To create a down state, open the *States* palette (Window > States or Press F11). At the bottom of the States palette click on the *New State* button. Choose down from the dropdown menu. This also converts button1 to a Movie Clip.

6. You can give different effects – depth/softness etc. as you like. This is the advantage of creating states as opposed to using event handlers.

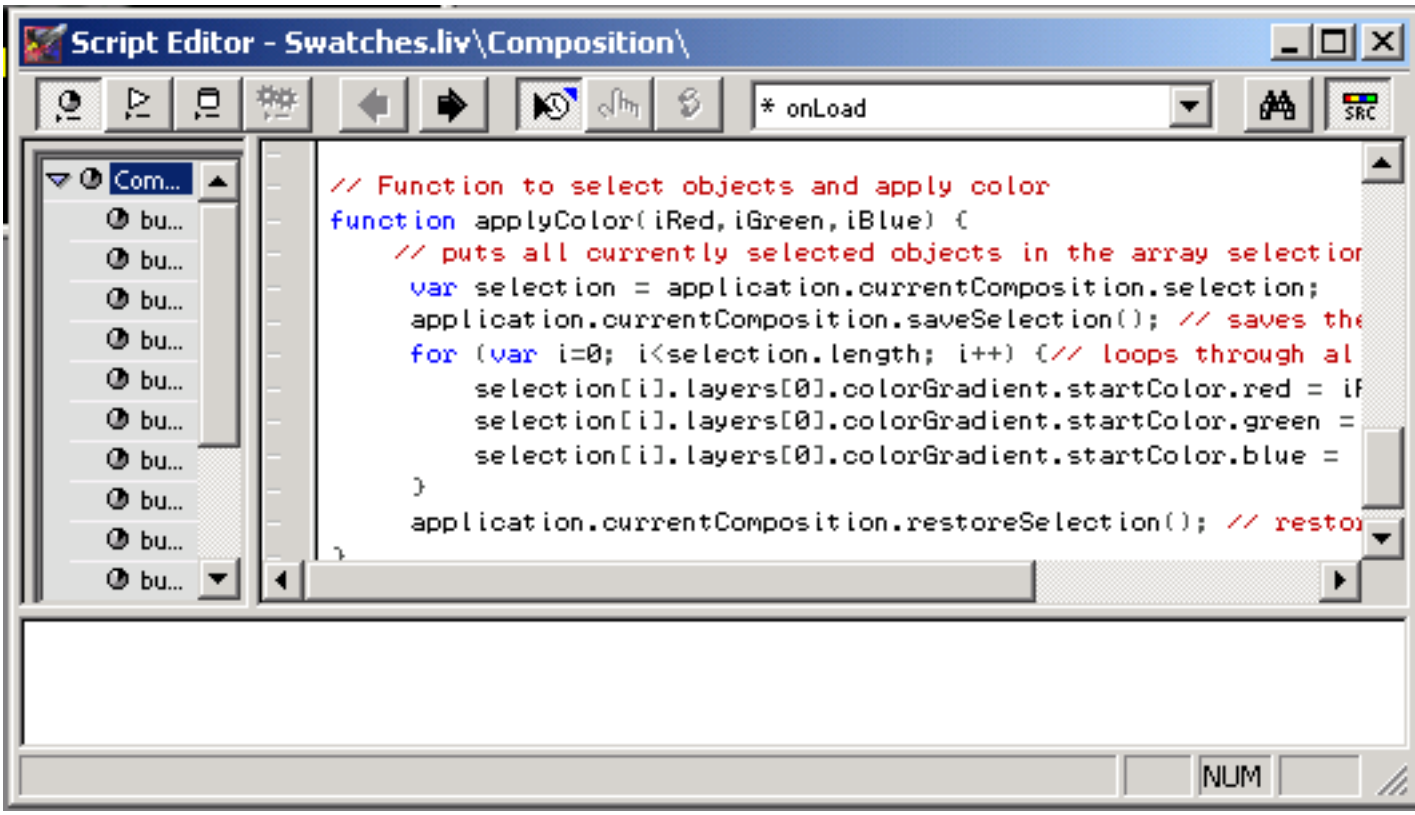
7. Now we need to write a function that takes as argument the Red, Green and Blue color values and applies the color to the selected objects in the current composition.

   It is a good practice to write functions on the *onLoad* event handler of the movie clip. This way they can then be accessed from anywhere by a function call statement. Here we are going to have multiple color buttons and they all have to use the same function. Hence it would be ideal to write it

on the onLoad event of the Composition, which is also a movie clip. The advantage of this is -Functions written on the Composition become global functions that can be accessed by all child movie clips.

Add the following code snippet (the same we had in the automation script we wrote above) in your Composition's onLoad event handler. Open the script editor using Scripts > Editor or Ctrl/Cmd + J. The script editor's title bar will show *Script Editor - Swatches.liv\Composition* and the handler box dropdown menu shows onLoad. If you do not see onLoad event handler click on Handler Scripts button and click on the dropdown menu and choose onLoad.

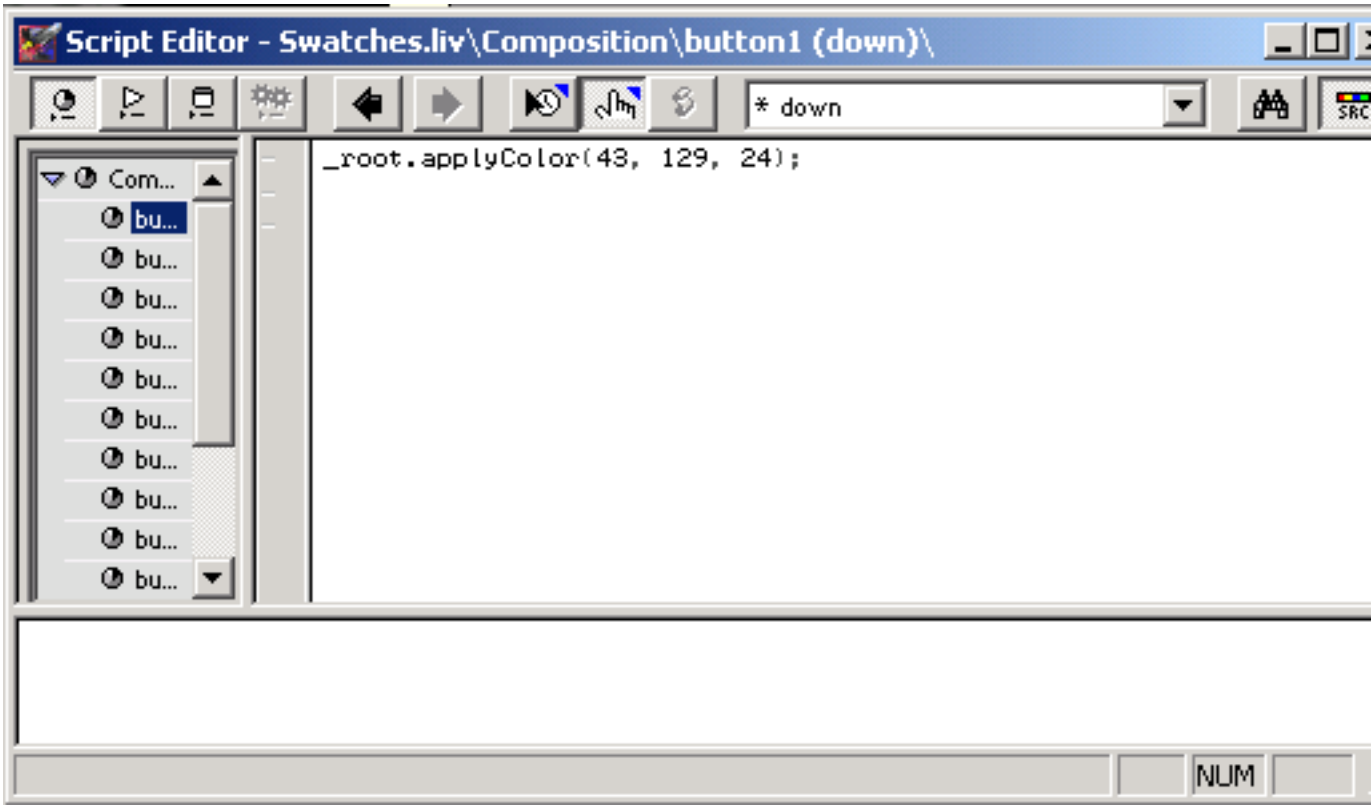
```
// Function to select objects and apply color
function applyColor(iRed,iGreen,iBlue) {
    var selection =
application.currentComposition.selection;
// puts all currently selected objects in the
array selection

application.currentComposition.saveSelecti
on(); // saves the current selection
    for (var i=0; i<selection.length; i++) {//
loops through all the selected objects

selection[i].layers[0].colorGradient.startCol
or.red = iRed; // Red value

selection[i].layers[0].colorGradient.startCol
or.green = iGreen; // Green value

selection[i].layers[0].colorGradient.startCol
or.blue = iBlue;  // Blue value
    }

application.currentComposition.restoreSele
ction(); // restore the current selection
}
```



8. Now this function is to be called when the button1 is clicked. Hence the following function call statement can be place in the down state of button1. In the script editor double click *button1* Movie Clip in the left pane. Click on the *state scripts* button. From the drop down menu select *down*. Add the following function call statement there.

   _root.applyColor(43, 129, 24);

   Make sure the sequence of Red, Green and Blue values is the same as the respective RGB combination in the Color palette.



9. Similarly you can create other buttons by duplicating this one. This saves you the time to add states, script to states, event handlers etc. Give another color to it from the Color palette. Add the same RGB combination as in the palette and also in the function call statement in the down state. You can create as many duplicates as you want the same way.

Testing your Live Tab

1. Open a new composition.
2. Create at least one object on it.
3. Select the objects you want to apply the color to.
4. Automation > Load Live Tab – Navigate to your Live Tab and open it.
5. Choose any color from the buttons.
6. The color is applied to the selected objects.