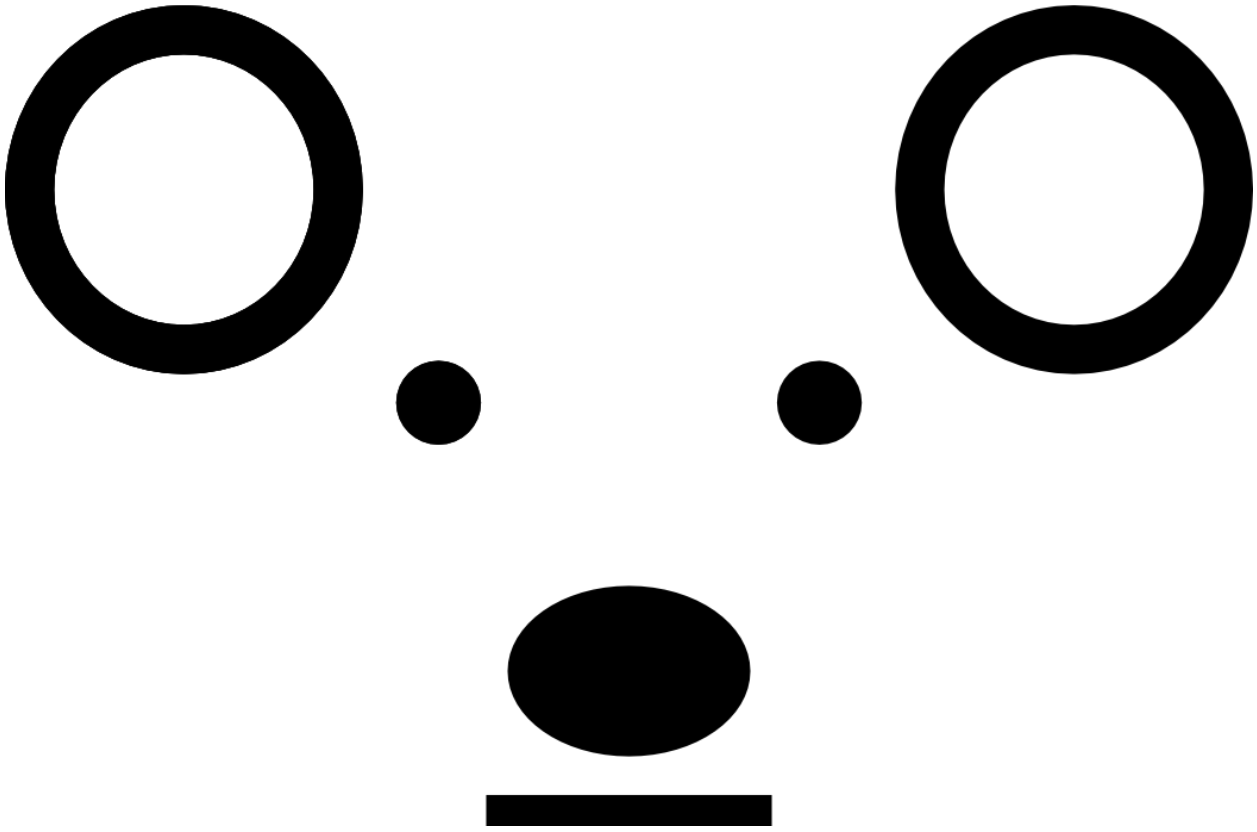
KM_GridPalettes Manual
04/25/21

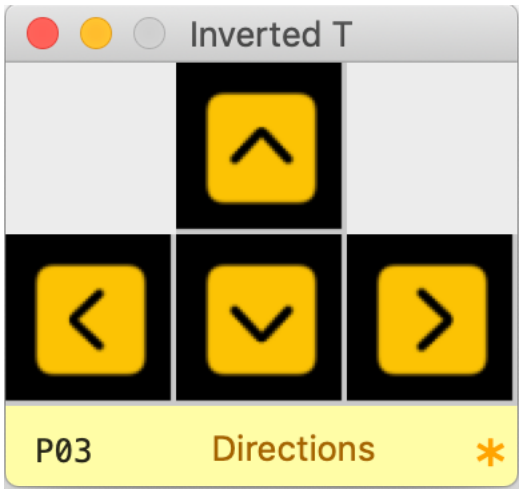


Version: 04/25/21

A new version of the program is released on this date. The videos are not being redone so they will show the look of the old version. The changes that have been made are not deep enough to cause confusion. The program continues to work as it has in the past. The goal has been largely to clean up the user interface.

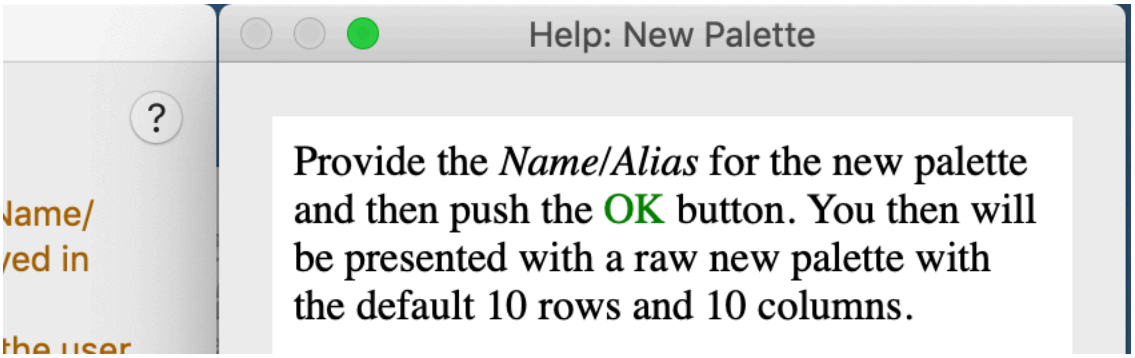
Highlights:

1. The *Name/Alias*, which is the human-friendly name of the palette, now shows up in more locations. One major one is in the yellow information area at the bottom of the palette. If the cursor is over a button, then it shows the *Tip* for that button as before. But if the cursor is elsewhere, it shows the *Name/Alias* of the palette.



Directions is the Name/Alias of this palette and is displayed in the information area

2. Help information is available for many parts of the program. This minimizes the need to refer to the manual when using the program. The help is provided by clicking on a button marked as a circle with an enclosed question mark.



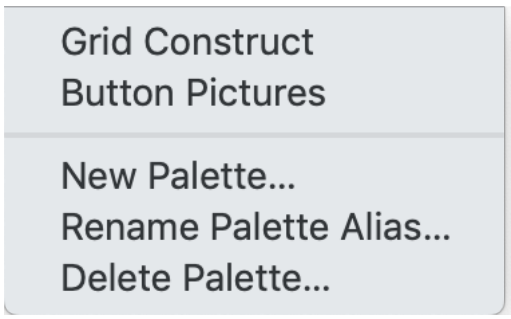
Clicking on the question mark has brought up the Help window for the New Palette window.

3. The original version had a single menu item *Palette Rename/Create/Delete* which grouped together the management of three functions that were not that closely related. This functionality has now been separated into three separate menu items.

New Palette…

Rename Palette Alias…

Delete Palette…



KM_GridPalettes

Purpose

KM_GridPalettes is a program that allows a user to create 2D grids of buttons that can be used to initiate **Keyboard Maestro** macros. This tool exists to supplement or as an alternative to the built-in palettes that already exist in **Keyboard Maestro**.

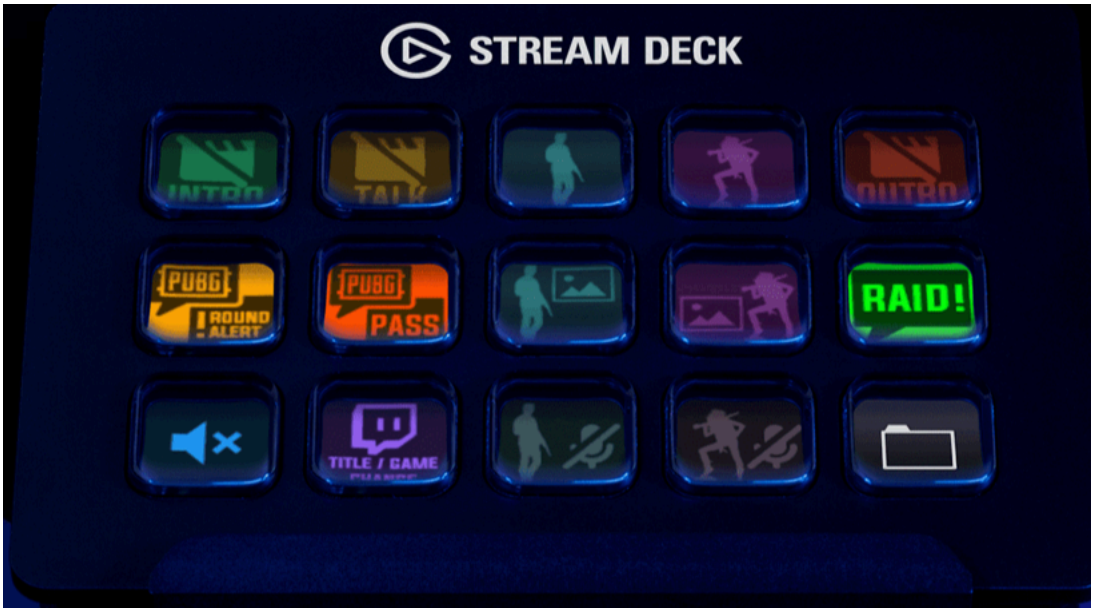
History

I originally primarily used **Keyboard Maestro** to write macro's that allowed me to complete complex tasks commonly involving multiple applications. These tasks were characterized by multiple steps performed repetitively. Interspersed in these steps were points that human intervention/decisions were required. The macros would stop at these times and the actions requiring a human would be done manually and then another macro would be launched, perhaps one among many, to march through a bunch of steps to get to the next decision point.

It was my habit to use hot-keys to launch these various macros. One problem I had was forgetting the appropriate hot-keys. My sessions were rather intensive, but I would go for weeks or months between having to engage again. Then I would have to reteach myself all the required keyboard shortcuts.

I became intrigued by the *Elgato Stream Decks*. They seemed like a clever way to approach the problem. The macros could be started by pushing labeled buttons. But I was reluctant to complicate my life with more hardware. I had more screen space than desktop space, and I thought that

I would try developing a program that would display a grid of buttons on the screen that could be used as you might a hardware Elgato Stream Desk.

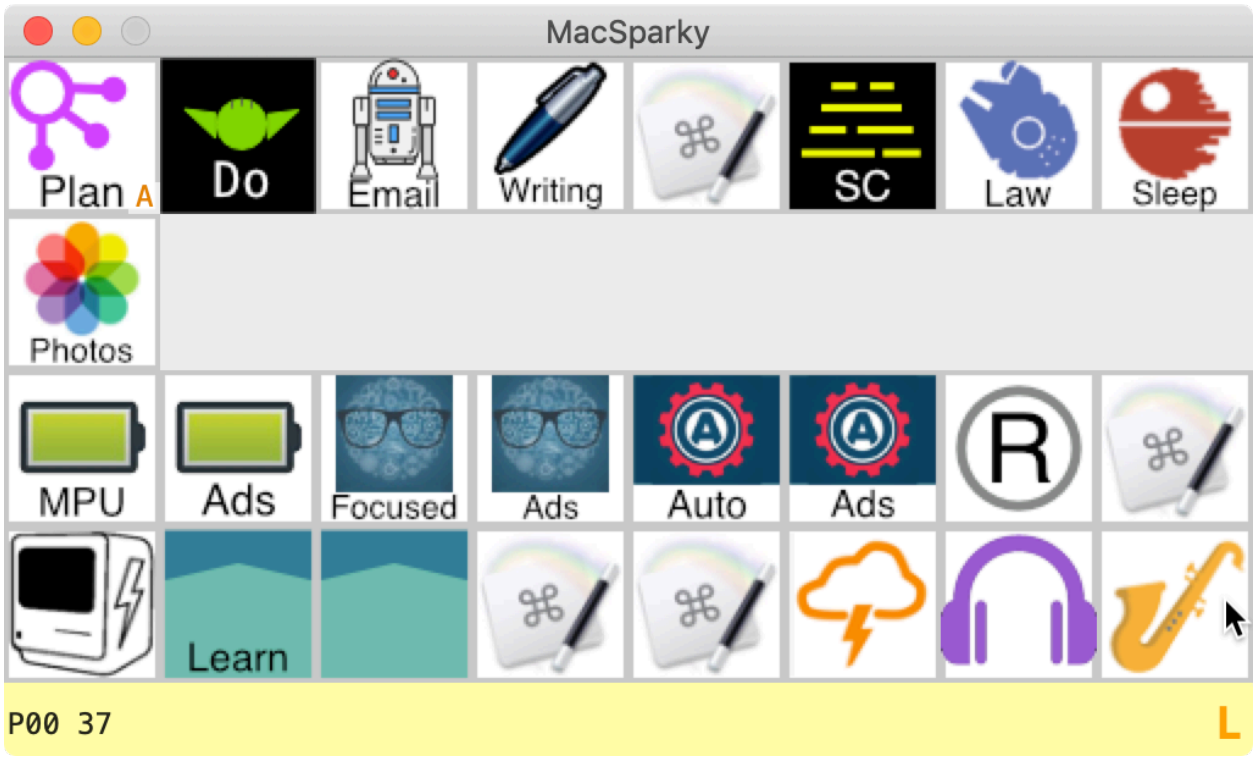


Picture of Elgata Stream Deck from website.

Keyboard Maestro itself has various palettes built-in. Under some circumstances their functionality is superior to what is offered by **KM_GridPalettes**. But there are other situations where I find **KM_GridPalettes** to be easier and more intuitive to use. **Keyboard Maestro** palettes are linear lists of triggers. **KM_GridPalettes** presents a 2D grid of triggers. Each has a place.

Description

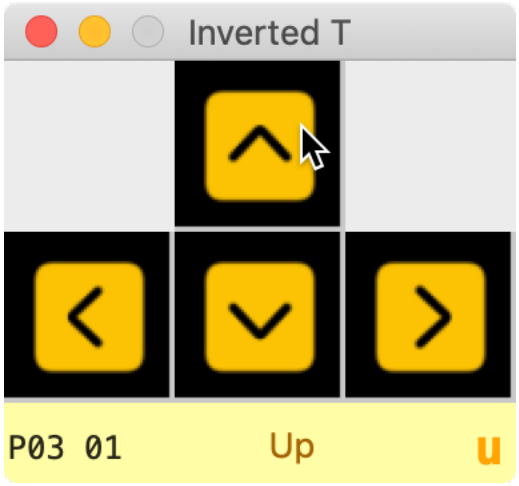
KM_GridPalettes allows the user to design up to 100 different palettes each consisting of a grid of buttons. An individual palette can contain as many as 100 buttons. Each buttons can invoke a separate **Keyboard Maestro** action. Some examples are collected below.



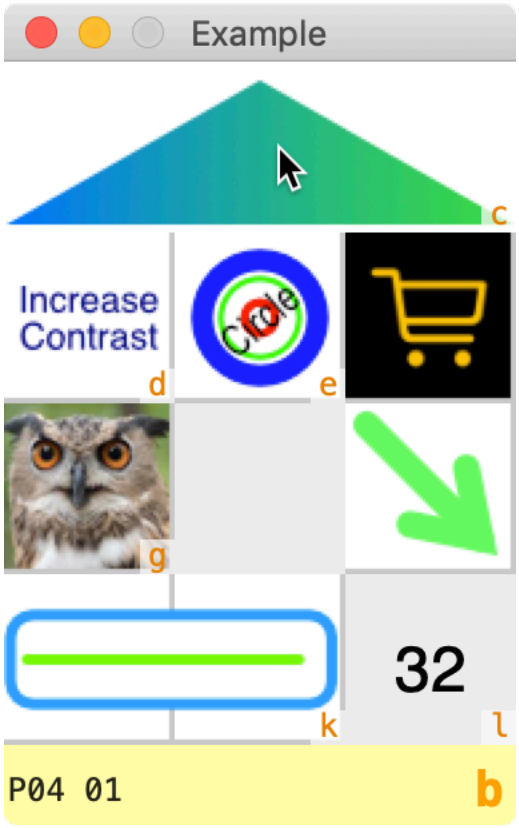
Mockup of an Elagato Setup used by David Sparks in his Field Guide to Keyboard Maestro.

4 by 4 grid of buttons using emoji for the button graphics

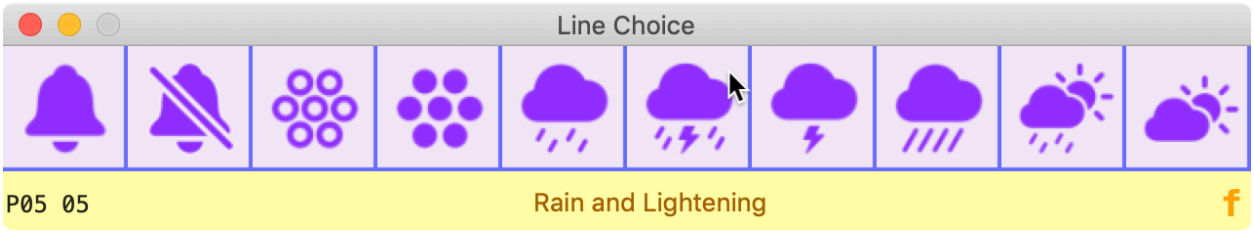
4 by 4 grid with hint revealed in yellow information area



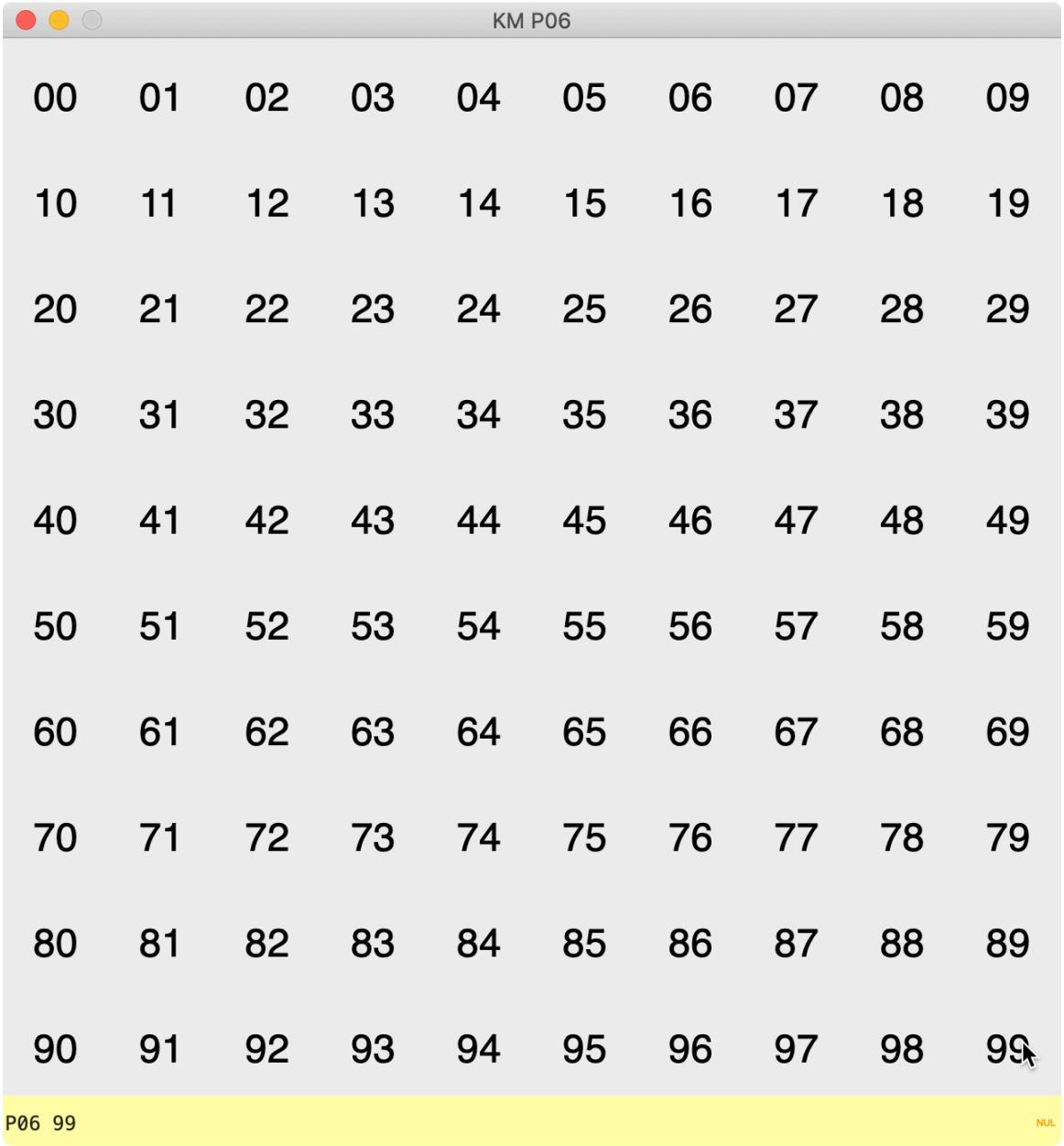
Simple Inverted T distribution of buttons



Graphic variety in button picture design



Single row of buttons



Initial grid of buttons: Design starts here.

KM_GridPalettes is a simple application with its own main window that can be transformed one of a multitude of palettes as seen above. It is easy to move from one palette to another. Each palette has a unique identify specified as P00, P01, P02, …, P99.

Link to Keyboard Maestro

The mechanism by which **KM_GridPalettes** links to **Keyboard Maestro** is via an AppleScript that is imbedded in the program. When any button is clicked, the unique identifying information about that button is passed to **Keyboard Maestro** by the AppleScript. Each palette should be paired to a **Keyboard Maestro** script that shares the same unique name (P00 or P01 or P02 etc.)

POO

No triggers specified.

Will execute the following actions:

- Set Variable “whichButton” to Text
  > %TriggerValue%
- Comment “POO”
  > POO contains all the actions relating to the POO palette. When this Keyboard Maestro Macro is activated, the particular button that was pressed is passed as a parameter (%TriggerValue%). With this knowledge, the Macro can decide what to do or possibly pass on the task by activating a specific Macro associated with the button.
- Switch of Variable “whichButton”
  - If it is “00”, Execute the Following Actions:
    - Display Text Briefly
      > The button 00 was pressed.
  - If it is “01”, Execute the Following Actions:
    - Display Text Briefly
      > The user pushed or invoked the button 01.
  - If it is “02”, Execute the Following Actions:
    - Display Text Briefly
      > Oh!. You pushed button 02
  - Otherwise, Execute the Following Actions:
    - Display Text Briefly
      > %Variable%whichButton%

Typical Keyboard Maestro macro. Expanded version below

P00

Triggered by any of the following (when macro group is active):

New Trigger

Or by script.

Or via the web server but all remote access is disabled.

Will execute the following actions:

Comment "00"

Set Variable "whichButton" to Text "%TriggerValue%"

Set variable whichButton Insert Token

to %TriggerValue%

00 → Not Available in Editor

Switch of Variable "whichButton"

If **Variable** whichButton

is 00

execute the following actions

Display Text "The button 00 was pressed" Briefly

Display text briefly Insert Token

The button 00 was pressed

is 01

execute the following actions

Display Text "The user pressed or invoked the button 01." Briefly

Display text briefly Insert Token

The user pressed or invoked the button 01.

is 02

execute the following actions

Display Text "Oh! You pushed button 02" Briefly

Display text briefly Insert Token

Oh! You pushed button 02

otherwise

execute the following actions

Display Text "%Variable%whichButton%" Briefly

Display text briefly Insert Token

%Variable%whichButton%

Illustrated here is a **Keyboard Maestro** macro that is invoked when any button on the P00 palette is activated. The user is responsible for creating these macros and making them accessible. It makes sense to place all the macros (P00, P01, P02 etc.) in a Group called something like *My Palettes* and have that group only activated by the actions of the **KM_GridPalettes** application.

The above can serve as a model. The first step captures the particular button that was pushed to the variable *whichButton*. Note the use of %TriggerValue%. That variable contains the ID of the button that was pressed. The user can then decide how to handle this. In the example above, this value is passed to a Switch statement so each individual button can be handled in an independent way.

If there is no **Keyboard Maestro** macro for a given palette, then nothing happens. The application cannot detect that the operation has failed. It passes the information to the embedded AppleScript and considers that the end of its responsibility.

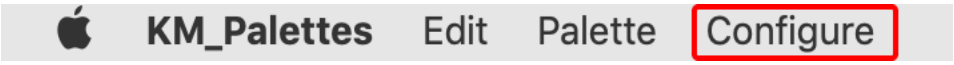
If you have 5 palettes in your application, you should have 5 **Keyboard Maestro** macros to deal with their output.

KM_GridPalettes Functionality

You can consider the functionality of **KM_GridPalettes** to be divided into two distinct areas.

1. Design of the palettes.
2. Use of the palettes.

The design part of the program is accessed through the menu **Configure**



The KM_GridPalettes menubar

Normally, when initially setting up your copy of **KM_GridPalettes** you would spend your time here designing your palettes. Whenever you wanted to create a new palette, you would return to this menu item.

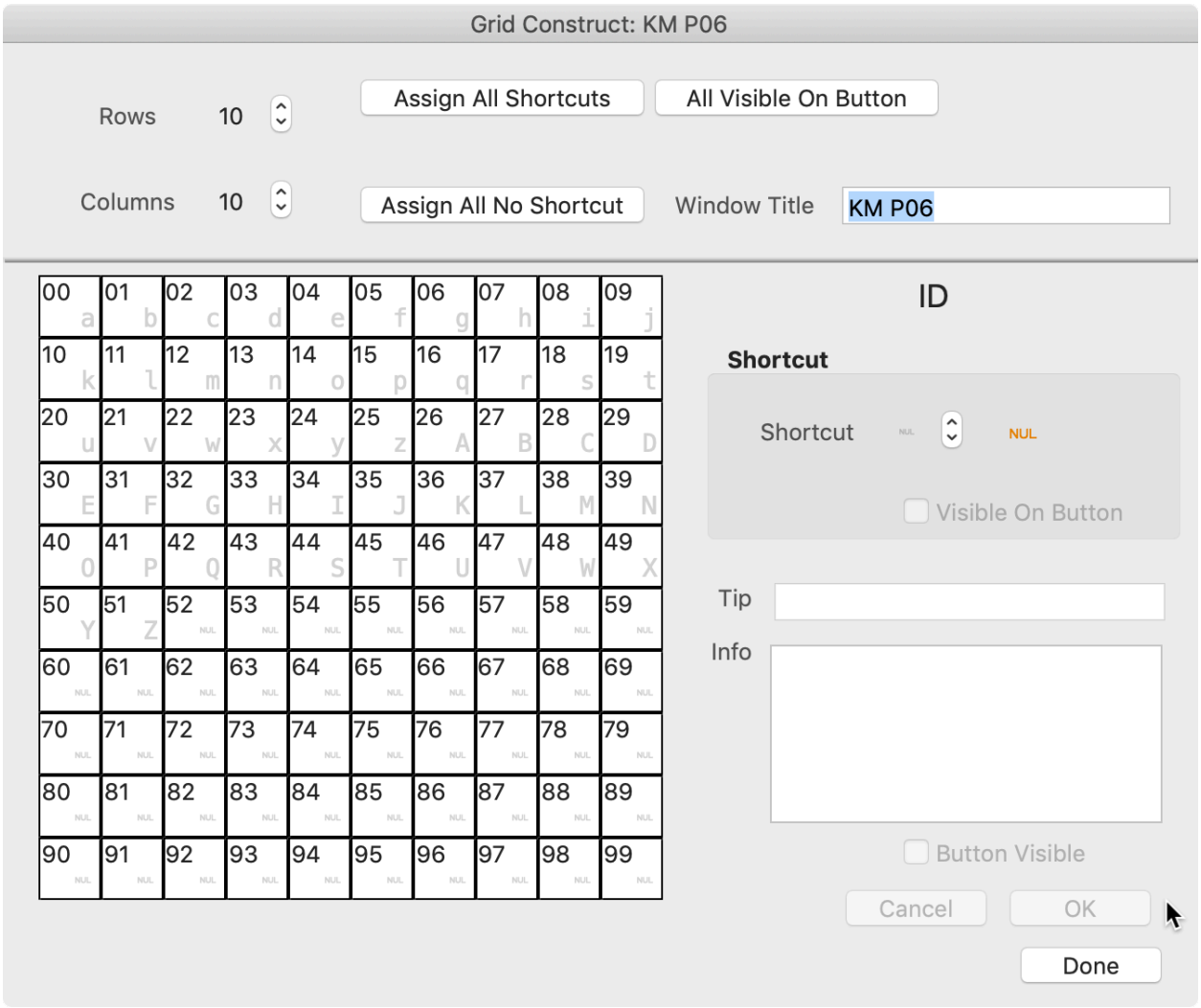
The menu **Palette** provides the basic functionality that allows you to change which palette that you want to use at any given time. Once your palettes have been designed, this is the only menu that you would need to be using.

Quitting the program is done simply by clicking on the red close window button.

Configuring Your Palettes

Under the **Configure** menu there are five menu items: *Grid Construct*, *Button Pictures, New Palette…, Rename Palette Alias…,* and *Delete Palette….*

Grid Construct Window



Here is where the major design of the palette takes place. Commonly, the first decision is how many buttons are required. Click on the up/down control to specify how many rows and how many columns are needed. Once that has been done, decide whether you want the buttons to have shortcuts. With shortcuts, the user can simply type a letter when the **KM_GridPalettes** application is foremost to activate any specific button. There are 52 shortcut characters available (a-z and A-Z). The shortcut can be made visible on the button to help the user remember their existence. You can also provide a name that will be displayed in the window title.

All of these decisions above the horizontal line are applied globally to the palette. Below the horizontal line, you can make decisions at the level of the individual button. These will override the global settings that have been applied previously.

In the grid on the left, click on an individual button to brings its properties up for editing.

Shortcut

You can individually choose the shortcut that you want to connect with any button. Obviously, two buttons cannot share the same shortcut. If you chose a shortcut for a button that is already assigned to another button, that button will loose it shortcut and that character will be assigned to the button currently being edited.

The up/down control will allow you to select any alphabetic character. If you click on the down button, you will see “a” as the first available character. Subsequent clicks move you down the alphabet. To make reaching a distant character less onerous, you can hold the SHIFT key down while clicking and move more quickly through the alphabet.

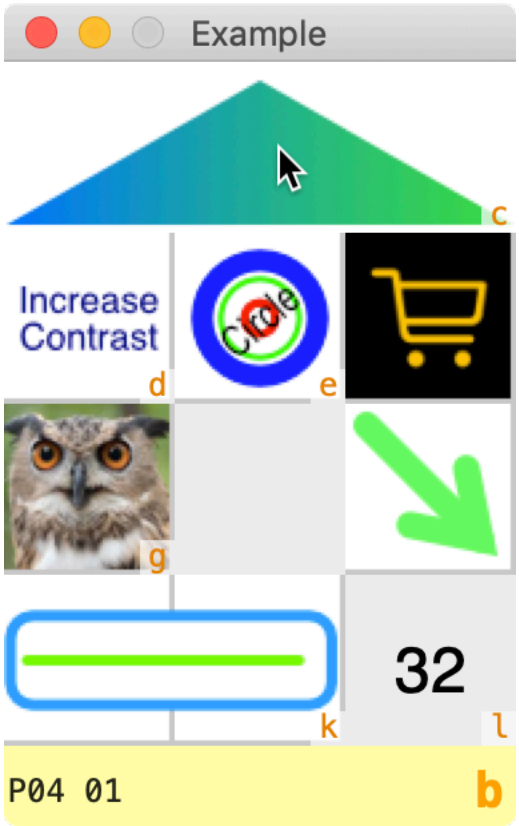
The second decision about shortcuts is whether you want them to be visible in the corner of their button. In the example palette below, you can see that some of the buttons have been configured to have their shortcut visible. It is seen as a small orange letter in the right lower corner of the button.

This example also shows the utility of the information bar that is seen in yellow at the bottom of the palette. This always shows information about the button over which the cursor lies. On the left side of the information bar,

you can see the palette ID (here P04) and the button underneath the cursor (here 01). Remember that the button in the left upper corner is 00.

On the right side of the information bar, you can see the shortcut for the button underneath the cursor. In this case, that shortcut *b* is not actually visible on the button itself. But the information bar can serve to remind the user of the shortcut even if, by design, the user decided not to show the shortcut over the button itself.

This example also serves to show a design option. By clever assignment of pictures to the individual buttons of the top row, they have been selected to meld into what looks like a single picture. But actually “underneath” the picture the identity of the three distinct buttons is preserved.

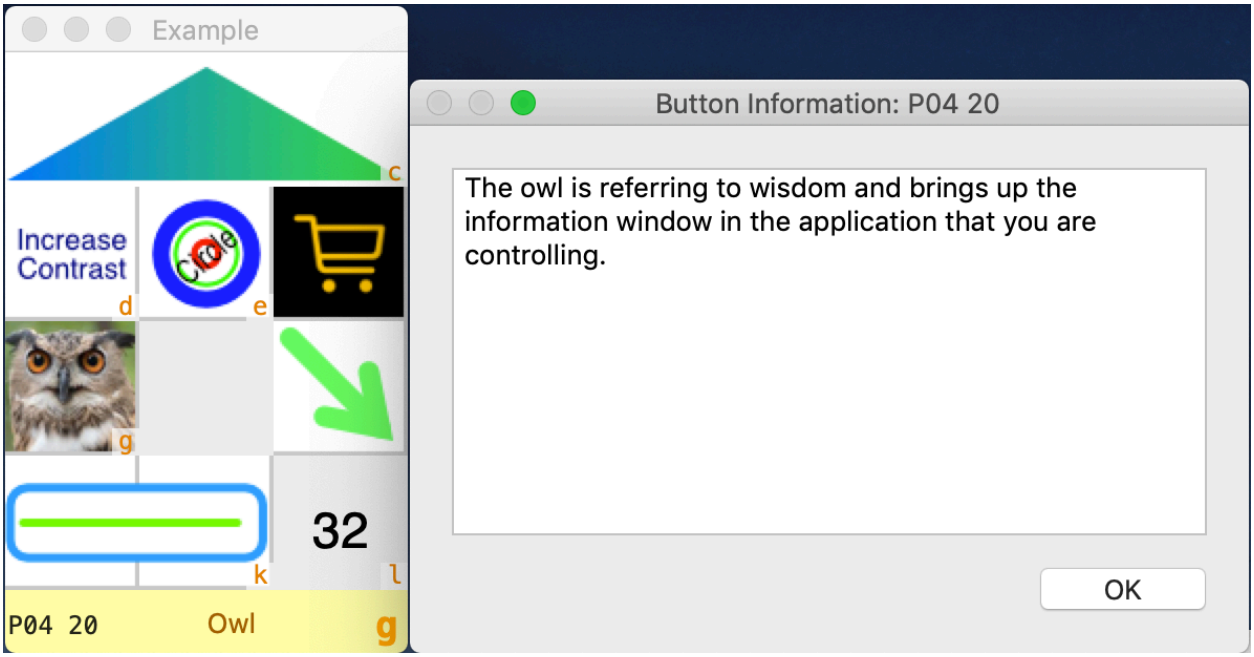


Tip and Info

It is possible to assign a *Tip* to any individual button. The text of the **Tip** will show up in the middle of the yellow information bar when the cursor is over the button. This can help remind the user of the function of any given button if there is concern that the graphic of the button might be a little too abstract. If a button has been assigned a *Tip*, it shows a pale purple color in the selection grid

In rare cases where you might need more room to explain (remind) the user of the functionality of button, you can write a paragraph in the **Info** area. The user can access any information that might have been entered here while using the program by OPTION clicking on the individual button. With the OPTION key held down, a click brings up a small explanatory window rather than actually activating the window. If a button has been assigned some Info, it shows a pale orange color in the selection grid.

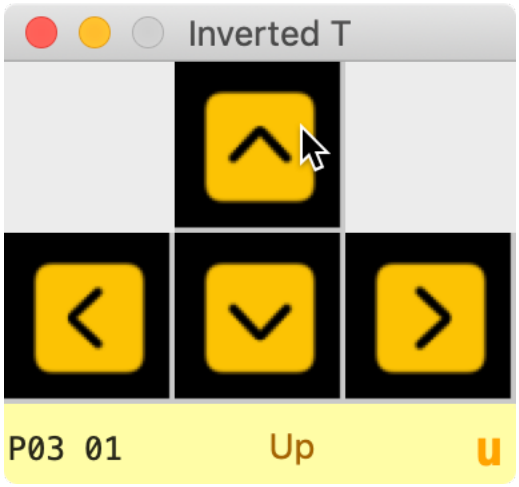
This is shown in the example below. Here the user has OPTION-clicked on the button with the picture of the owl. A window with explanatory text has appeared. Also noticed that during configuration, the word “Owl” was entered as a Tip and that is visible in the information Bar.



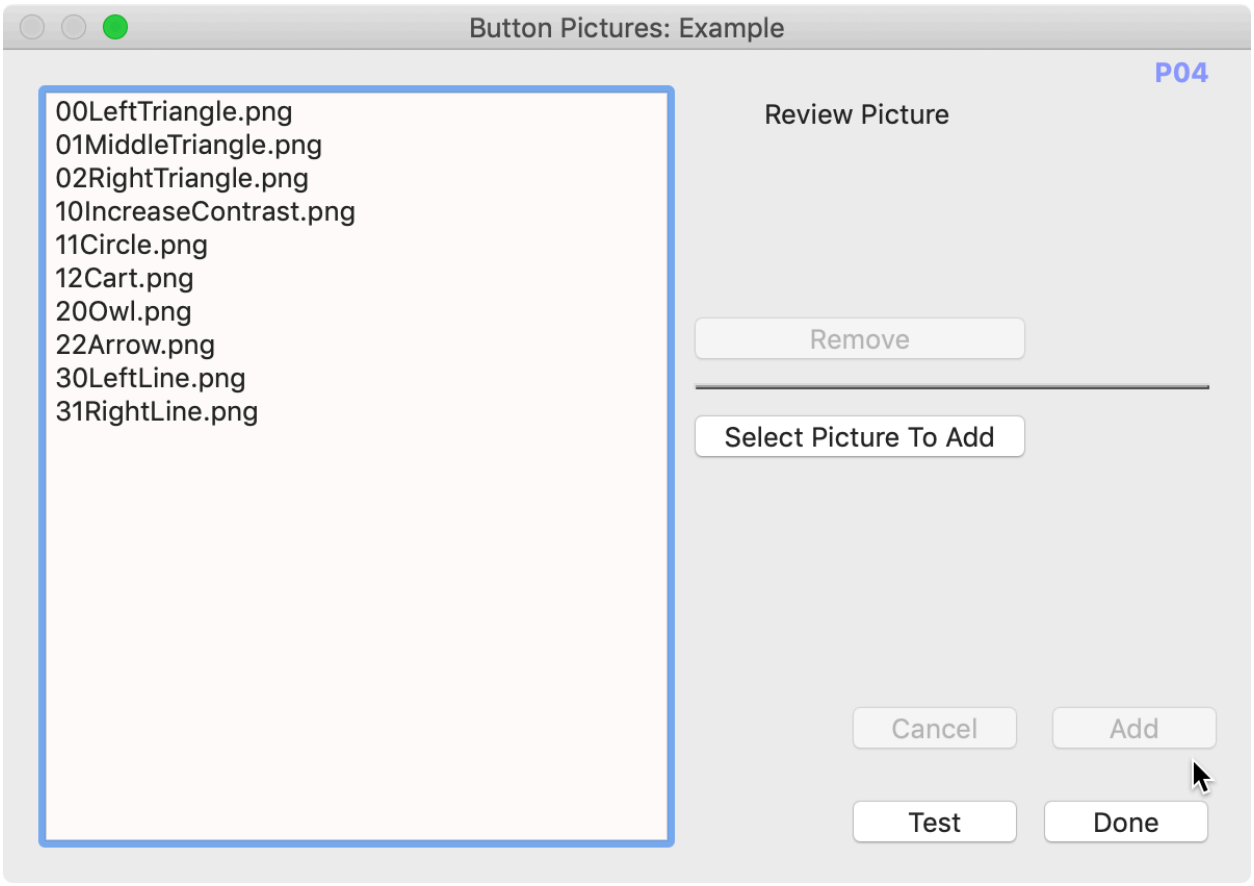
Button Visible Checkbox

Any individual button can be made “invisible”. When you uncheck the Button Visible, the selection grid will show that button as a light blue. This means that it shows up in the palette as a bland gray area and is impervious to clicks. This allows additional flexibility in the design of a palette. In the example below, the button areas on the right and left side of the top row have been made “invisible” which helps create the design of the inverted T. Clicks in those areas will be ignored. Notice also, in this example, the data revealed information bar. The cursor is over the functioning button 01 and this is the palette P03. This is all visible on the left side of the information bar. This button has been assigned the shortcut *u.* Although that does not actually appear on the graphic itself, it does appear in the information bar. The **Tip** *Up* has been assigned to the button 01 and this is also revealed in the information bar.

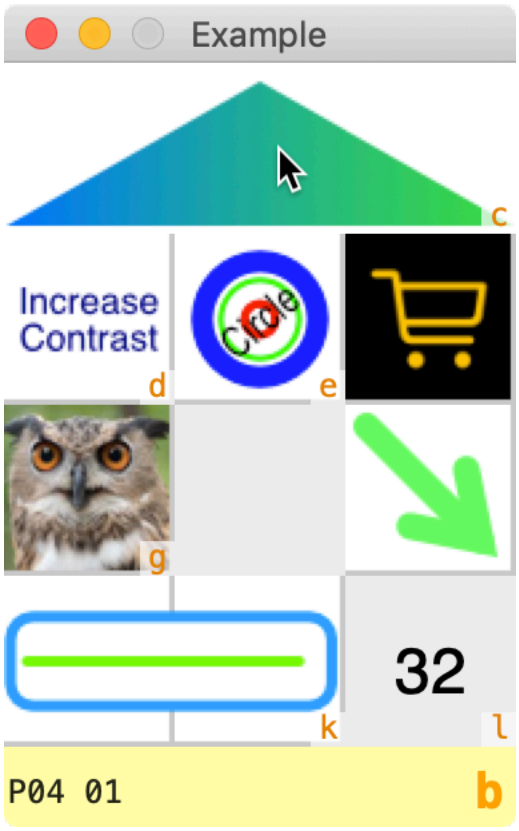
Clicking on the button under the cursor will activate the P03 **Keyboard Maestro** macro and the button ID, 01, will be passed as a parameter.



Picture Buttons



The Button Pictures window showing the windows associated with the P04 palette



The example palette above is P04. The pictures that appear on the buttons are handled in this program by grouping them in a folder given the name of the palette. In this case, the folder would be *P04*. This folder is a subfolder of a folder called *ButtonPictures* which is itself in the *ApplicationSupport* folder for *com.bearboat.KM-GridPalettes*.

While the Mac user familiar with accessing the *~Library* world could directly access this folder, Apple discourages mucking around in this part of the Finder.

KM_GridPalettes provides the tools to move pictures in and out of this folder without any specialized knowledge of where the folder resides. When pictures are properly located in the Finder hierarchy, they will appear over their owner button and allow the customized look that you can see in the example palettes.

The Button Pictures windows shows on the left side a list of all the picture files that are associated with the palette. The association with a specific button is controlled by the name of the file. The first two characters have to be unique (in their folder) and have to be digits. 00LeftTriangle

means that that picture will be pasted on the button 00. After the first two digits, the files can be named anything that you want for your convenience.

Use any graphic program that you want to create the pictures. The correct size is 64 by 64 pixels. I have a starting template that has a one pixel line along the right and lower margin when I design a button. Those lines help delineate the buttons one from another in the final palette. Commonly, this provides the effect I want. But there are exceptions.

In the example above, in the first row, I have designed the button pictures without any delineation, deliberately allowing the three images to flow into a single image. The same technique is seen involving the left two buttons of the third row.

I generally set up my graphics so that they are produced at 144 DPI. I think this improves the look of the images on retina Mac screen. I almost always use PNG files.

One you have produced the images that you want, name them with the guidelines outlined above. The first two characters of the name should match the name of the button you are trying to label.

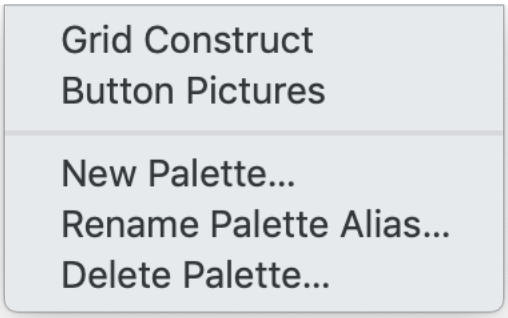
In the example above, I have included images that have only text (left second row), vector graphics with some text (middle second row), icons from Apple's SF Symbols (right second row). The owl is just a picture reduced to 64 pixels in size. (left third row). The middle of the third row is a button that was declared as invisible in the Grid Construct window. The right button of the fourth row simply displays its number ID. This is the appearance of a button that has had no picture associated with it. It will still function just fine.

The *Select Picture To Add* button will open a Finder choose file dialog. You can select any picture that you want that is appropriately named and sized, and it will end up being copied and moved to the correct location so that subsequently the palette will show this image over its button.

If you want to remove a picture (one common reason is that you want to replace it with another version), select the picture in the listing and then click on the *Remove* button. You cannot have two pictures with the same initial two digit characters, so this has to be done before adding its replacement.

When you *Remove* a picture, it is not actually simply trashed. The picture file that is no longer needed will show up in a *DeletesPalettePictures* folder on the Desktop. You can then decide what its ultimate fate should be. I do not want the user to lose carelessly a picture that they actually want.

Palette Create/Rename/Delete



The **Configure** menu provides three different additional functions.

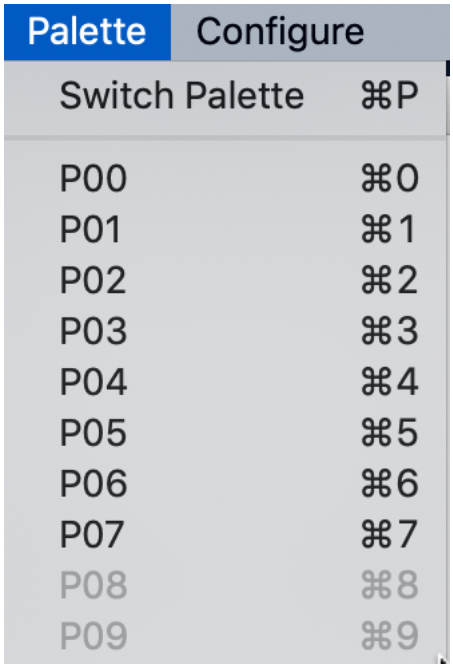
1. **New Palette…** This is used to create a new palette, the initial step in that process. The initial palette that is created will be a ten row by ten column palette with no pictures. After it has been created here, use the **Configure** menu with its *Grid Construct* and *Button Pictures* menu items to customize its appearance.
2. **Rename Palette Alias…** This offers an opportunity to rename the current palette. The most definitive name that a palette has is its name ID of the form P00 or P01 or P02 etc. That is the name that is used to communicate with **Keyboard Maestro**. That name is set in stone. But it is not that friendly a name for a human. Here you can assign a name that is a little more evocative, and it appears in various locations like palette listings. It is, as it were, an alias to the P05 style names.
3. **Delete Palette…** This provides is a way to delete an already constructed palette. This seems, on the face of it, to be fairly simple. That palette disappears. When a new palette is created, it will recycle the name (P04, for example) of the palette that was removed. In this way, you will not “use up” all the available palette names prematurely. The possible palettes names are restricted to the 100 names in the range (P00, P01, P02, …, P98, P99). When a palette is removed, the

associated **Keyboard Maestro** macro will remain. (P04, for example). You would have to deal with this manually.

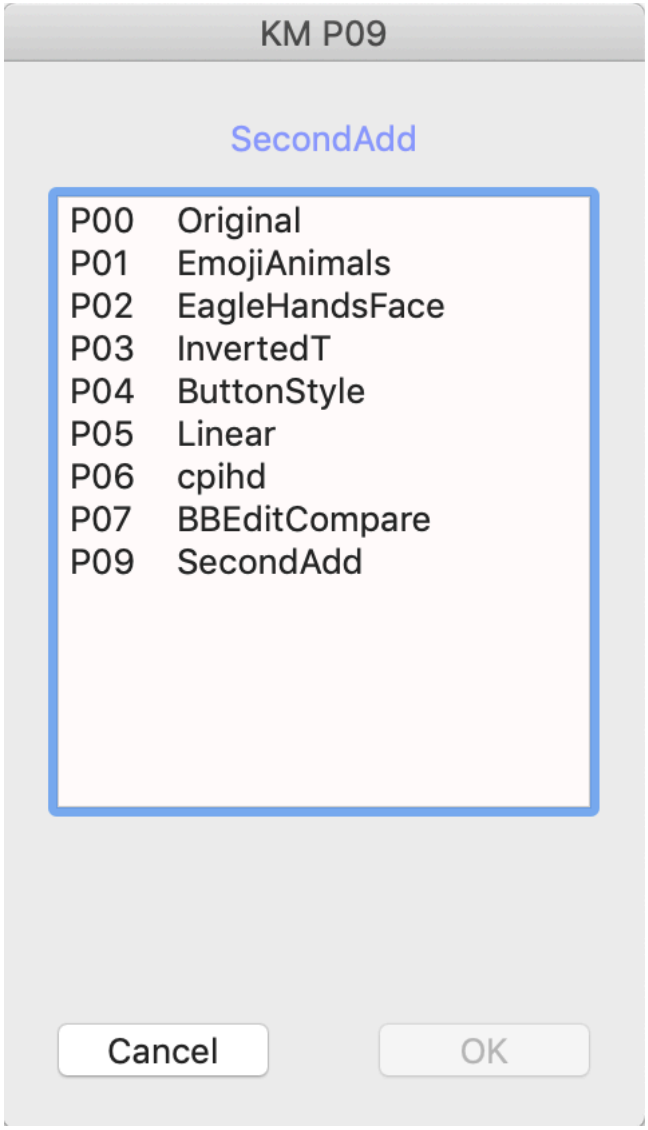
Using Your Palettes

One the palettes and their associated windows have been constructed, you need not bother with the **Configure** menu item unless something needs further editing.

In the day to day use of the program, the **Palette** menu allows you to change what is the currently active palette. Theoretically, you could have up to 100 different palettes. There are specific menu items allowing you to directly select one of the first 10 palettes. If you have more than that in your quiver, you can select the *Switch Palette* menu item which will bring up a window to allow you to select any palette you have created.



The Switch Palette window has some automation features. If you type two digits then it will directly select the corresponding palette. You type “4” “2” then, if the P42 palette exists, it will immediately come to the fore.



Remember that you can actually use **Keyboard Maestro** itself to change the palette of the application **KM_GridPalettes**. That is occasionally useful. Your own workflow might include moving among different palettes. For example, a **Keyboard Maestro** macro with the consecutive actions to type Command-P and then “4” and “2” would automate bringing up the P42 palette (if it existed).

Moving Palettes to a Different Computer

There is no syncing mechanism built into the program. All the design work is saved in a specific location:

~/Library/ApplicationSupport/com.bearboat.KM-GridPalettes

That folder (*com.bearboat.KM-GridPalettes*) will contain two things

1. A folder called: *ButtonPictures*
2. A file called: *PreserveState.txt*

ButtonPictures contains folders with names like *P00*, *P01*, *P02*, *P03* etc. Those folders contain the pictures that are associated with each of the palettes that have been created.

It takes some familiarity with advanced features of the Finder to access the *~/Library* folder. You can Google the topic to learn how. It varies slightly depending on the macOS version.

If you wanted to duplicate your **KM_GridPalettes** work on a different machine, you would have to copy the *com.bearboat.KM-GridPalettes* folder with its sub-folders and place it into the *ApplicationSupport* folder of the different machine.

Final Notes

Click On a Button and Nothing Happens

There are several potential causes for this problem and the program cannot tell you which one is responsible. Look for the following possibilities.

1. There is no **Keyboard Maestro** macro whose name corresponds to the name of the palette on which the button resides. Remember that every palette has a name ID like P00, P01, or P02. There has to be a **Keyboard Maestro** macro that you have created that has this identical name.
2. The **Keyboard Maestro** macro is in a Group that the **KM_GridPalettes** program does not have access to. It is probably best to create a Group that **is** accessible by **KM_GridPalettes** and put all the macros you have written in that Group.
3. The **Keyboard Maestro** macro corresponding to the palette is passed a parameter that refers to the specific button that was pushed. That needs to be handled in the macro. It will reside in %TriggerValue% at the start of the script. (See the example above in the manual) Make sure that your macro makes use of this value.

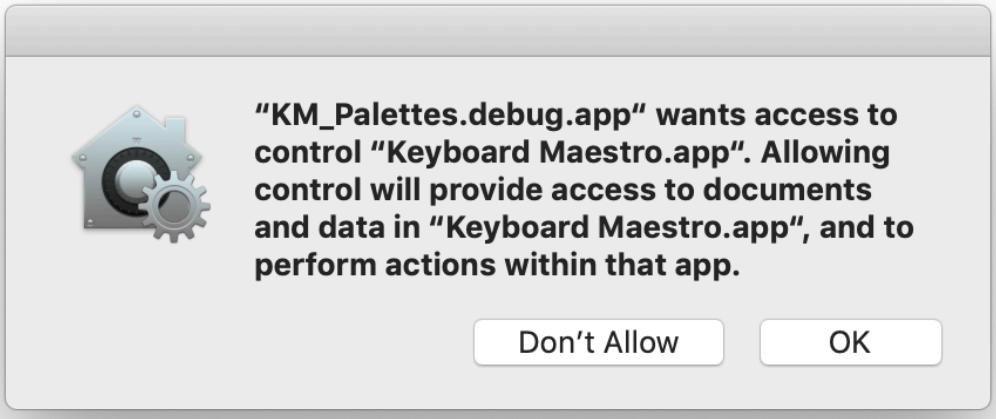
Security

I am not a registered developer with Apple. When you first try to open the program after downloading it, you will have to give it permission to run on your device. You are probable familiar with the dialogue boxes that appear related to this issue. Right-click on the program and select Open from the Pop-Up.

The first time that you run the program, a dialog button will appear asking for permission for the application **KM_GridPalettes** to access the application **Keyboard Maestro**. Apple is wary of applications accessing

other applications. Of course, **Keyboard Maestro** main purpose in life is to do just that.

Once you overcome these two hurdles, the program should just “work” in future use.



Hints for Macro Design

I have included an actual **Keyboard Maestro** Macro below to illustrate a common pattern. First of all, the macro is named for the palette that controls it (P07). Secondly, information in the form of a string is passed to the variable %TriggerValue% which contains the name of the button that invoked the macro. (00, 01, 02 etc).

Using a Switch/Case is a relatively easy way to deal with differentiating the actions launched by different buttons. Each button is given its own "Case" and it Actions can be placed here easily enough if they are short. You could also use several If Then Else Actions if you preferred. (or some combination.) If the Actions prompted by the various bottoms were complex, it might make sense to populate the "Cases" with *Execute Macro* and then dedicate that macro to the intended Actions of a single button.

Note the last Action. It makes **KM_GridPalettes** the foremost application after the macro has run. It is often convenient to include this last step. If you want to use shortcuts, it really makes sense. For a keyboard shortcut to work, **KM_GridPalettes** has to be foremost. If you end up having to use the mouse to click on the **KM_GridPalettes** window to activate it, the usefulness of the shortcut is diminished. Fortunately, it is not so important if you are primarily using the mouse to click on the various buttons. The **initial** click will fire off the **Keyboard Maestro** request. This is to say that you do not have to click first to "activate" **KM_GridPalettes** and then click again to choose a button. A single click will do.

P07

No triggers specified.

Will execute the following actions:

- Set Variable “whichButton” to Text %TriggerValue%

- Comment “PO7”

  POO contains all the actions relating to the PO7 palette. When this Keyboard Maestro Macro is activated, the particular button that was pressed is passed as a parameter (%TriggerValue%). With this knowledge, the Macro can decide what to do or possibly pass on the task by activating a specific Macro associated with the button.
  This palette is for the BBEdit Compare function. Written by Robert Livingston author of KM_GridPalettes

- Switch of Variable “whichButton”

  If it is “00”, Execute the Following Actions:

  - Activate BBEdit
    Notify on failure.
  -
  - Pause for .1 Seconds
    Notify on failure.
  -
  - Move and Resize Front Window
    To: (SCREENVISIBLE(Main,Left),SCREENVISIBLE(Main,Top),

SCREENVISIBLE(Main,Width),SCREENVISIBLE(Main,Height)-140)

- Notify on failure.
- Pause for .1 Seconds
  Notify on failure.
- Type the ⌘T Keystroke

- If it is “01”, Execute the Following Actions:
  - Display Text Briefly
    Wrap Text
  - Activate BBEdit
    Notify on failure.
  - Pause for .1 Seconds
    Notify on failure.
  - Select Menu Item in BBEdit
    Select: View ⇢ Text Display ⇢ Soft Wrap Text
  - Stop macro and notify on failure.

- If it is “02”, Execute the Following Actions:
  - Display Text Briefly
    Invisible

- If it is “03”, Execute the Following Actions:
  - Activate BBEdit
    Notify on failure.
  -
  - Type the ⌘Left Arrow Keystroke
  - Pause for .1 Seconds
    Notify on failure.
  -
  - Type the Down Arrow Keystroke
- If it is “04”, Execute the Following Actions:
  - Activate BBEdit
    Notify on failure.
  -
  - Type the ⌘Right Arrow Keystroke
  - Pause for .1 Seconds
    Notify on failure.
  -
  - Type the Down Arrow Keystroke
- If it is “05”, Execute the Following Actions:

- Activate BBEdit
  Notify on failure.
-
- Type the Right Arrow Keystroke
- Pause for .1 Seconds
  Notify on failure.
-
- Type the Down Arrow Keystroke

- If it is “06”, Execute the Following Actions:
  - Activate BBEdit
    Notify on failure.
  -
  - Type the ⌘Left Arrow Keystroke
  - Pause for .1 Seconds
    Notify on failure.
  -
  - Type the Down Arrow Keystroke

- If it is “07”, Execute the Following Actions:
  - Activate BBEdit
    Notify on failure.

-
- Type the ⌘Right Arrow Keystroke
- Pause for .1 Seconds
  Notify on failure.
-
- Type the Down Arrow Keystroke

- If it is “08”, Execute the Following Actions:
  - Activate BBEdit
    Notify on failure.
  -
  - Pause for .1 Seconds
    Notify on failure.
  -
  - Move and Click
    At (0,0) from the center of the found image in all screens.
  - 

    (Unique). Fuzz: 15%
  - Stop macro and notify on failure.
  -

- If it is “09”, Execute the Following Actions:
  - Activate BBEdit
    Notify on failure.
  -

- Type the Down Arrow Keystroke

- Otherwise, Execute the Following Actions:
  - Display Text Briefly
    %Variable%whichButton%

-

- Activate KM_GridPalettes
  Notify on failure.

Written with Xojo

www.xojo.com.

Xojo is a useful tool for interacting with **Keyboard Maestro**. This application grew out of more highly customized apps that I created in Xojo to deal with specific problems. It is an excellent tool for creating GIU interfaces, unlike other programming tools like Python.

KM_GridPalettes is strictly a one way street. **KM_GridPalettes** is almost entirely a GUI. The user interacts with the program only to specify actions that will be performed by **Keyboard Maestro** and the apps that it in turn is directing.

In a broader context, Xojo is capable to creating apps that can handle data that is passed back to it in a useful way. Not only can it control **Keyboard Maestro**, it can accept and work on information that applications return. On of my more useful applications for personal use in one that has a Xojo app controlling data that is being massaged and analyzed within Xojo as well as being communicated out to **Keyboard Maestro** and the applications it is directing. The Clipboard and text files are ways of communicating data between the players.

Xojo is a commercial app that requires a license for compiled apps. However, it is free to explore with non-compiled apps that run within its own environment. It you are interested in building applications with a quickly and easily contracted GUI, it is worth checking out.

Contact

The best way to contact me is email.

Email: rlivingston@me.com.

Twitter: @_rlivingston or @bearboat

Web: www.bearboat.net.

Any kind of feedback is welcome. If you run into bugs, tell me and I will try to help.

LICENSE / DISCLAIMER