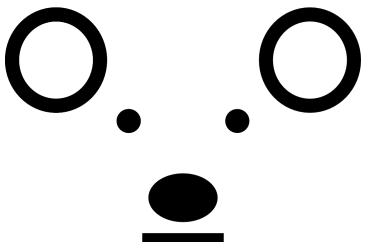
ASK_Palette Manual
04/01/22

Bearboat Software



ASK_Palette Overview

History

ASK_Palette is an application that is a descendant of an older program **KM_GridPalettes**. **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 was developed to supplement or be an alternative to the built-in palettes that already exist in **Keyboard Maestro**.

This newer program has additional functionality although it is very similar in concept to **KM_GridPalettes**. The letters in the name (ASK) refer to:

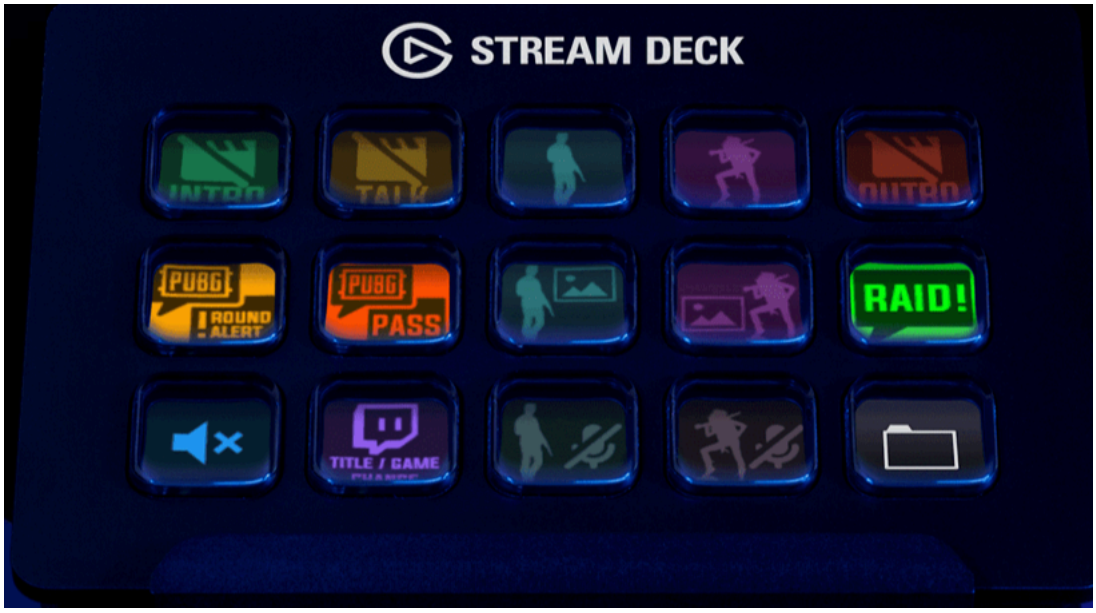
1. A - AppleScript
2. S - Shortcuts
3. K - Keyboard Maestro

It is now possible to initiate an **AppleScript** or a **Shortcut** directly from the buttons of a palette without going through any intermediate steps in **Keyboard Maestro**. The original functionality of launching a Keyboard Maestro script remains and, for most people, likely will remain a primary use of the program.

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.



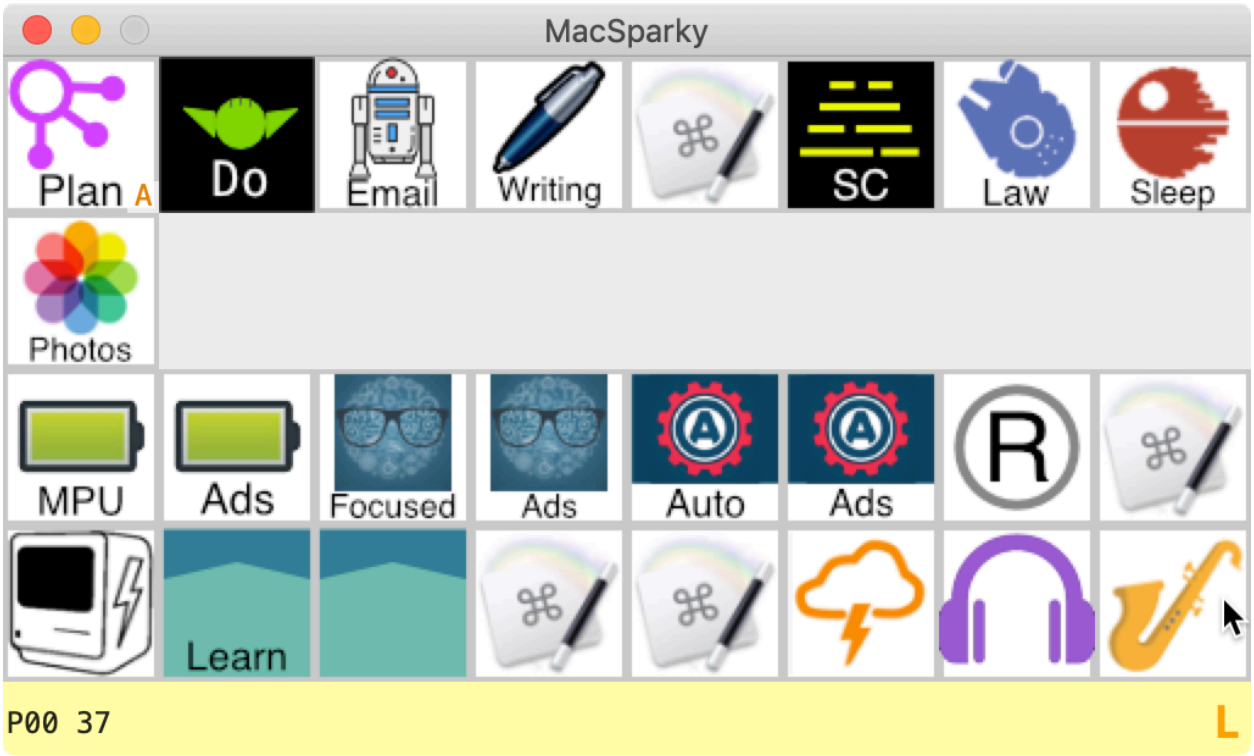
Picture of Elgata Stream Deck from website.

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

Description

ASK_Palette 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

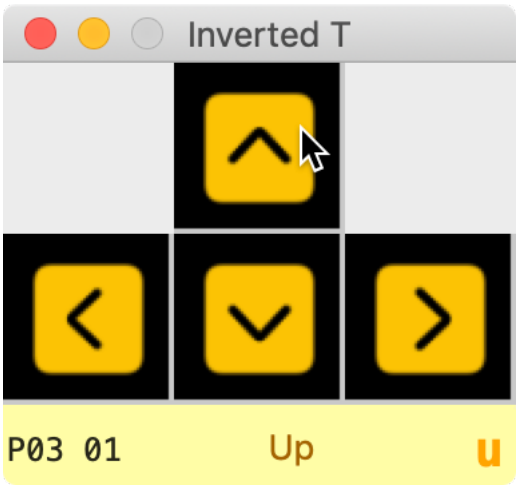
100 buttons. Each button can invoke a separate **Keyboard Maestro** action or an AppleScript or a Shortcut. Some examples are collected below.



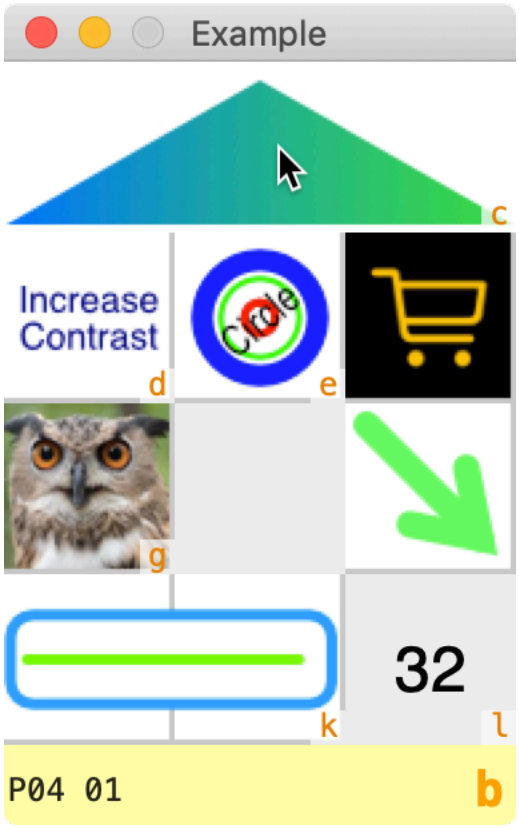
Mockup of an Elagato Setup used by David Sparks in his Field Guide to Keyboard Maestro.

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

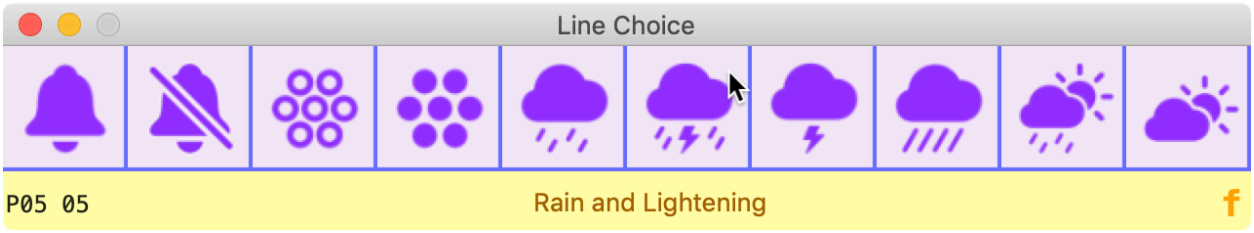
4 by 4 grid with hint revealed in yellow information area



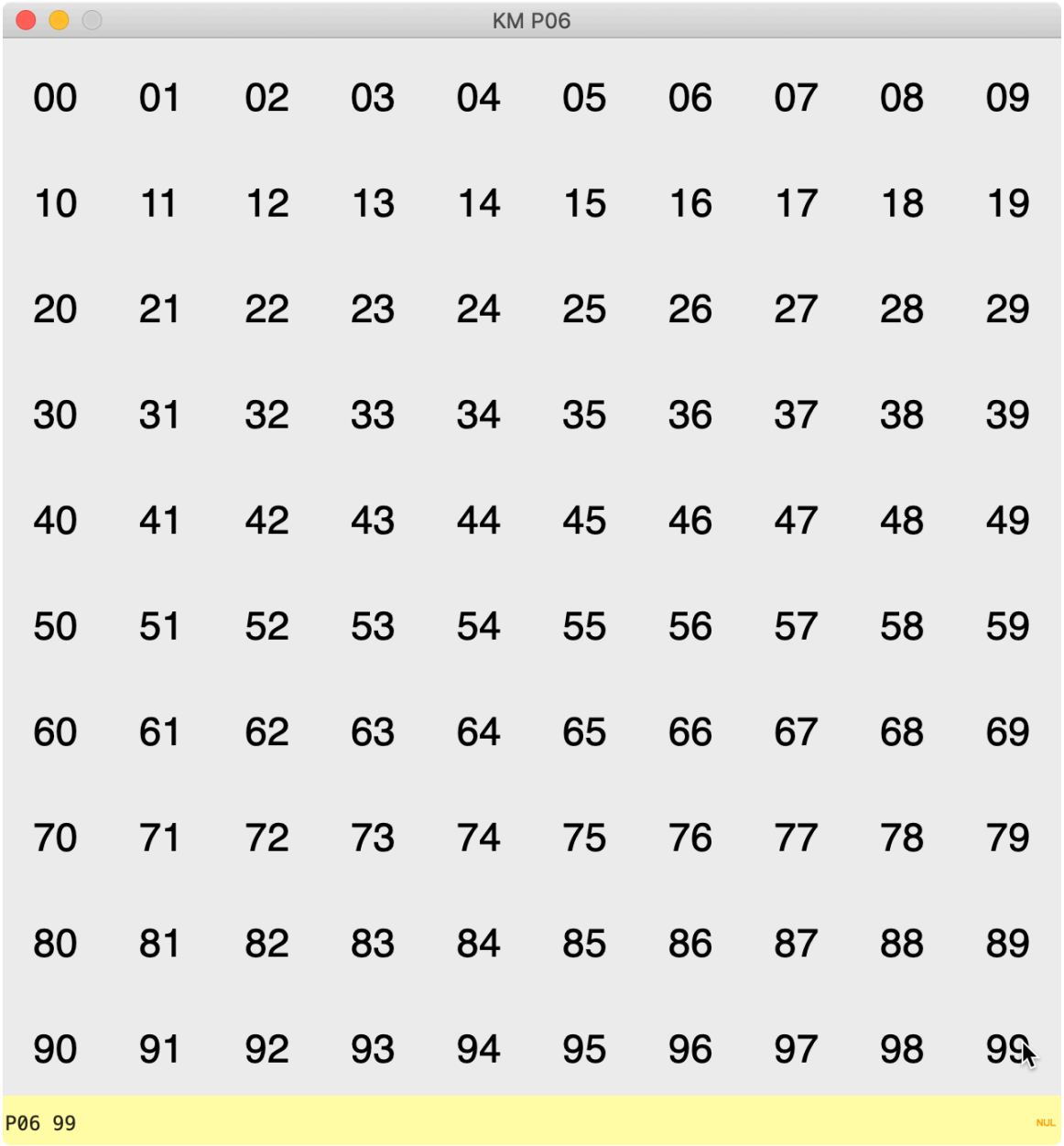
Simple Inverted T distribution of buttons



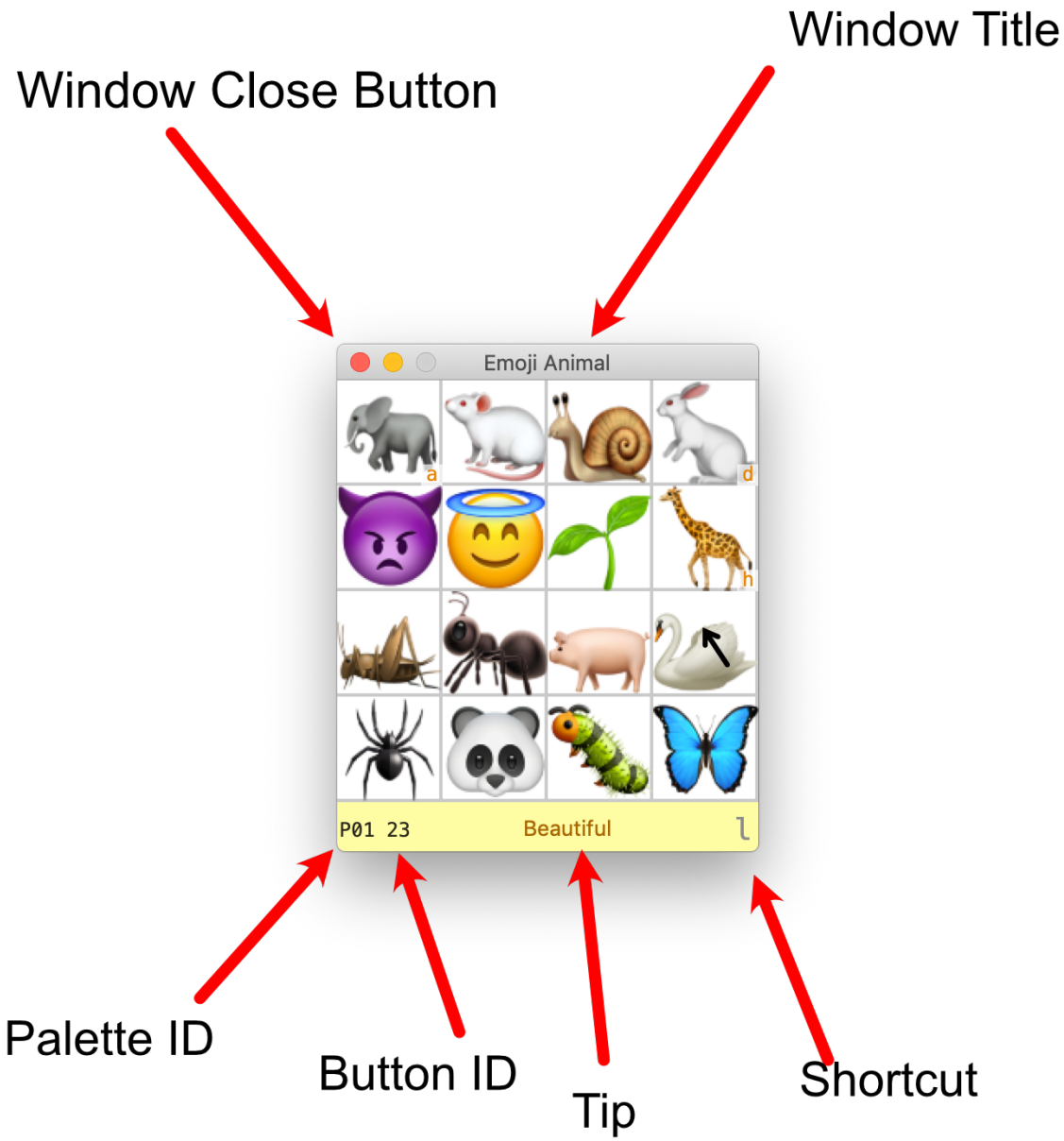
Graphic variety in button picture design



Single row of buttons



Initial grid of buttons: Design starts here.

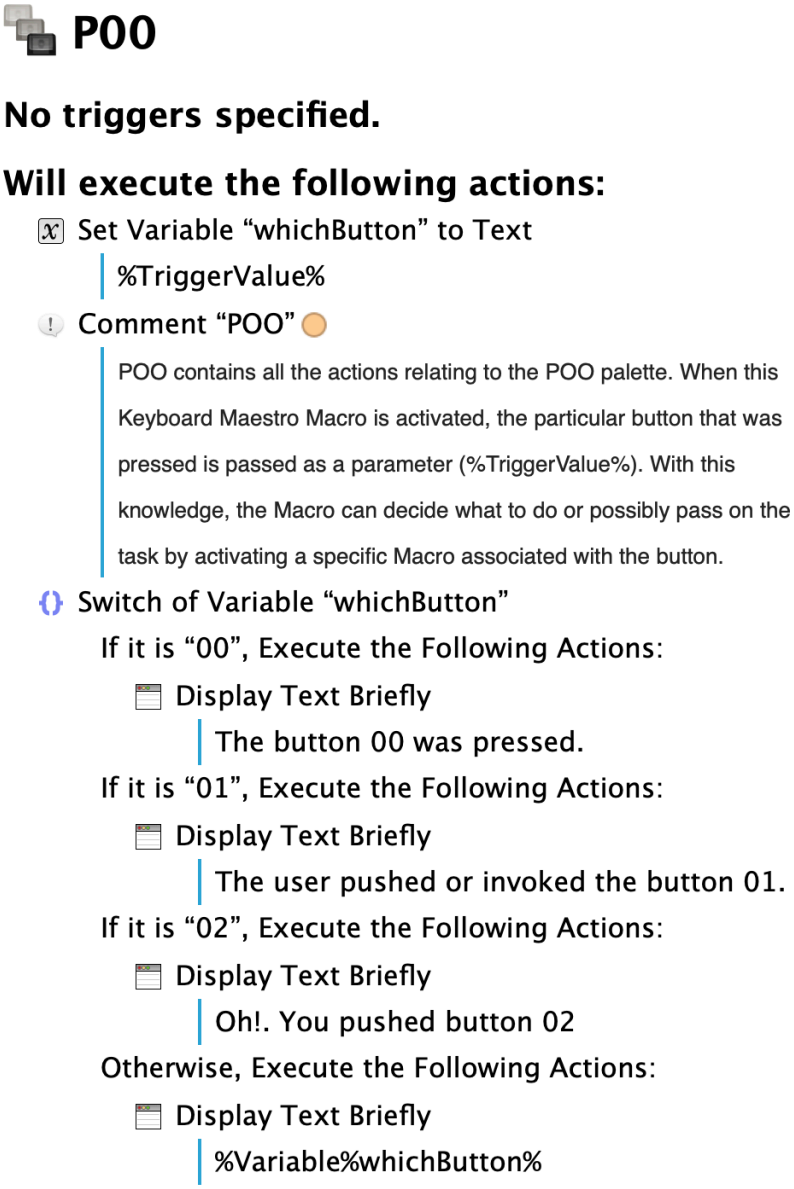


The yellow bar at the inferior margin of the palette is the information bar. The cursor is overlying the fourth button in the third row. Since the numbering always starts with zero, this is overlying button 23. That is displayed in the left part of the information bar, just to the right of the palette ID which is at the extreme left of the information bar. The middle of the information bar displays Beautiful which is the *Tip* that the user created for this button. The right side of the information bar shows the keyboard shortcut associated with the button 23 (lowercase l). During the palette design, the user decided not to display the shortcut on the button. The button above with the giraffe has its keyboard shortcut (lowercase h) actually displayed on the graphic. The window title is chosen by the user, in this case it is *Emoji Animal*.

ASK_Palette 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 **ASK_Palette** 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.) if its buttons are to successfully launch **Keyboard Maestro actions.**



Typical Keyboard Maestro macro.

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 **ASK_Palette** 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 most commonly have 5 **Keyboard Maestro** macros to deal with their output. Any palette that contains some buttons that link to a **Keyboard Maestro** macro should have its own macro so you can define what actions those button initiate.

Link to AppleScript or Shortcuts

AppleScript is a scripting language created by Apple Inc. that facilitates automated control over scriptable Mac applications. *Shortcuts* is a recently created application on the Mac (2022) that lets you create your own shortcuts with multiple steps. When designing a palette, a button can be assigned to launch an AppleScript or a Shortcut simply by providing the button the name of the AppleScript or the individual Shortcut. If such a name is provided, this will override launching a **Keyboard Maestro** macros. All the AppleScripts that are addressed by the application need be stored in a single specified folder. The individual shortcuts are all stored within the Shortcuts application. The predecessor of **ASK_Palette**, **KM_GridPalettes**, only allowed initiation of **Keyboard Maestro** macros. Of course, **Keyboard Maestro** macros themselves can launch AppleScripts. But **ASK_Palette** can easily be configured so buttons directly launch AppleScripts or Shortcuts.

ASK_Palette Functionality

You can consider the functionality of **ASK_Palette** 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 ASK_Palette menubar

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

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 frequently need to be using.

The menu **Misc** can be used to provide Help information about the program while in use.

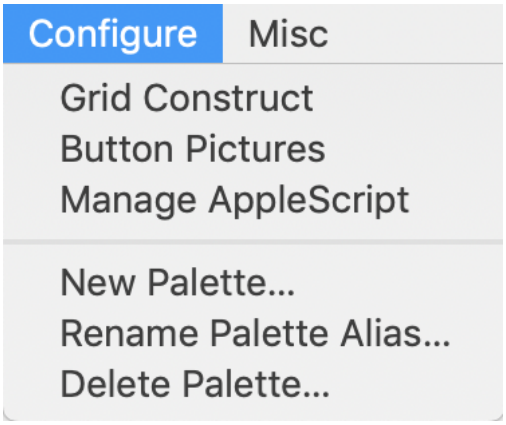
Quitting the program is done simply by clicking on the red close window button or the *Quit ASK_Palette* menu item under the menu **ASK_Palette**.

Configuring Your Palettes

New Palette…

This is a menu item under the **Configure** menu that allows you to initiate the process to create a new palette. This is the first step in the building of a palette. A new 100 button palette appears. You get a chance to specify a human-friendly name for the new palette. The user is expected to now go to the **Configure** menu begin the process of specifying the palette.

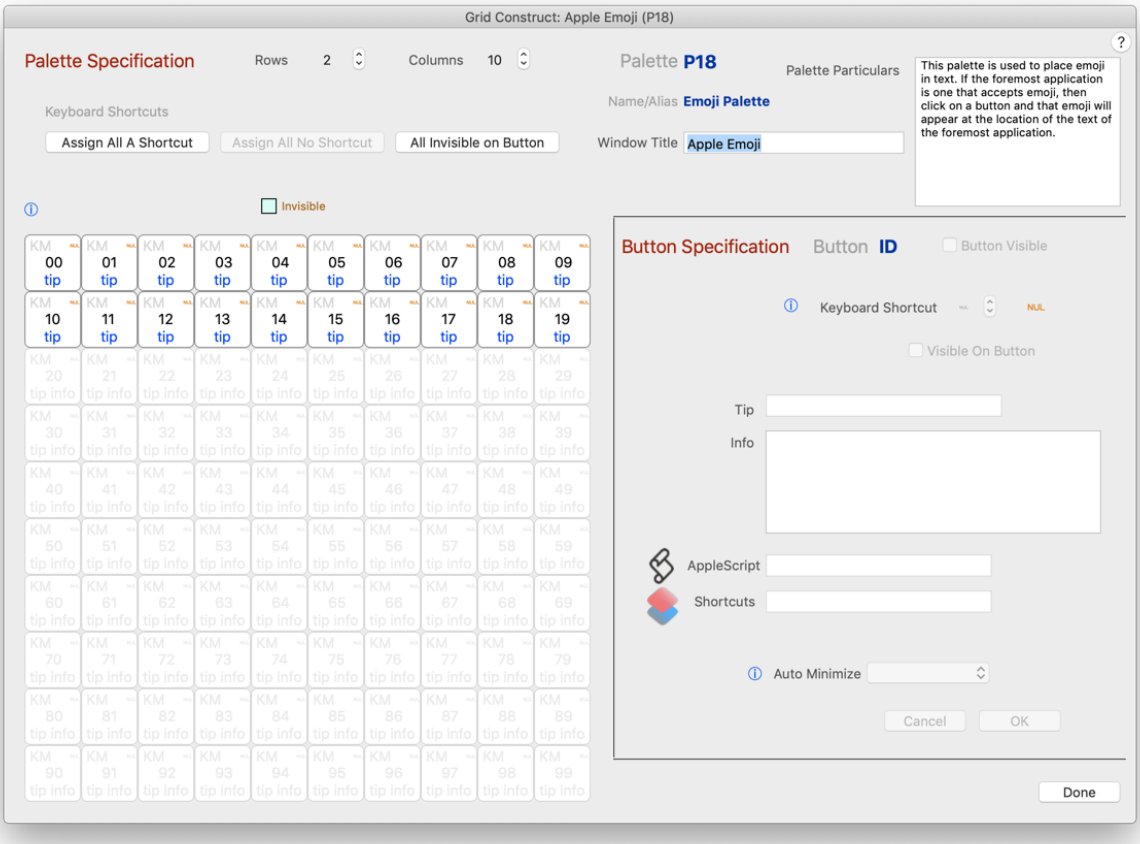
The **Configure** menu items provide that actions that allow a palette to be designed and made functional. Under the **Configure** menu there are six menu items: *Grid Construct*, *Button Pictures*, *Manage AppleScript*, *New Palette…*, *Rename Palette Alias…*, and *Delete Palette….*



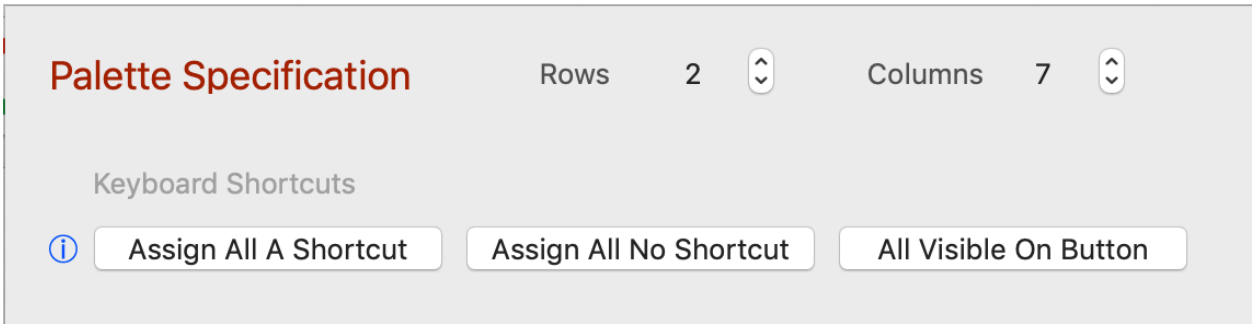
Now that a new palette exists, open up *Grid Construct*.

Grid Construct

This is the window that appears when selecting the *Grid Construct* menu item.



Global Palette Specification



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 in general you want the buttons to have keyboard shortcuts. With keyboard shortcuts, the user can simply type a letter when the **ASK_Palette** is the foremost application 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.

In the Palette Specification part of the window, you set up the default pattern of the use of keyboard shortcuts. Later, you can override these defaults for any given button. So you might decide that all the buttons are assigned a keyboard shortcut, but subsequently the shortcut can be removed from a given button etc.

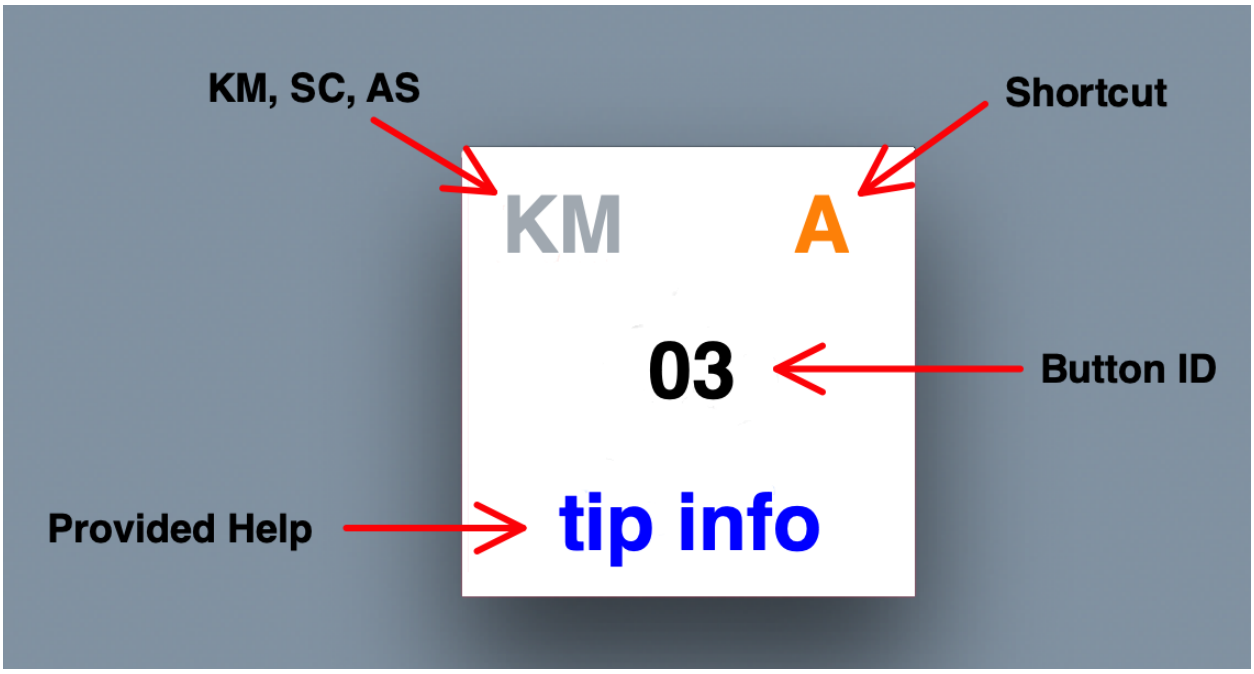
Button keyboard shortcuts can be useful under a number of circumstances. Commonly, the user simply wants to be able to hit a keyboard key instead of having to use the mouse. There are other potential reasons to want keyboard shortcuts. **ASK_Palette**, it should be realized, is an application like any other on you Mac. Therefore, it can be accessed by **Keyboard Maestro**. There are situations in which it can be useful to be able to include commands in a **Keyboard Maestro** macro that make **ASK_Palette** foremost then invoke the action of any button on that palette by having **Keyboard Maestro** simply type the appropriate keystroke.

In the *Palette Specification* part of the window (the upper ⅓), you can also provide a *Window Title* for the palette. That may or may not be the same as the *Name/Alias* that you earlier created. It is up to you.

Finally, free text can be entered or pasted into the *Palette Particulars* text area. If you use a particular palette infrequently and your memory is imperfect, it can be useful to write a description of the palette and its purpose and anything else you want to remember. The information here can subsequently be accessed for reference under the Menu Item **Misc** >> *Palette Particulars* when this particular palette is in use.

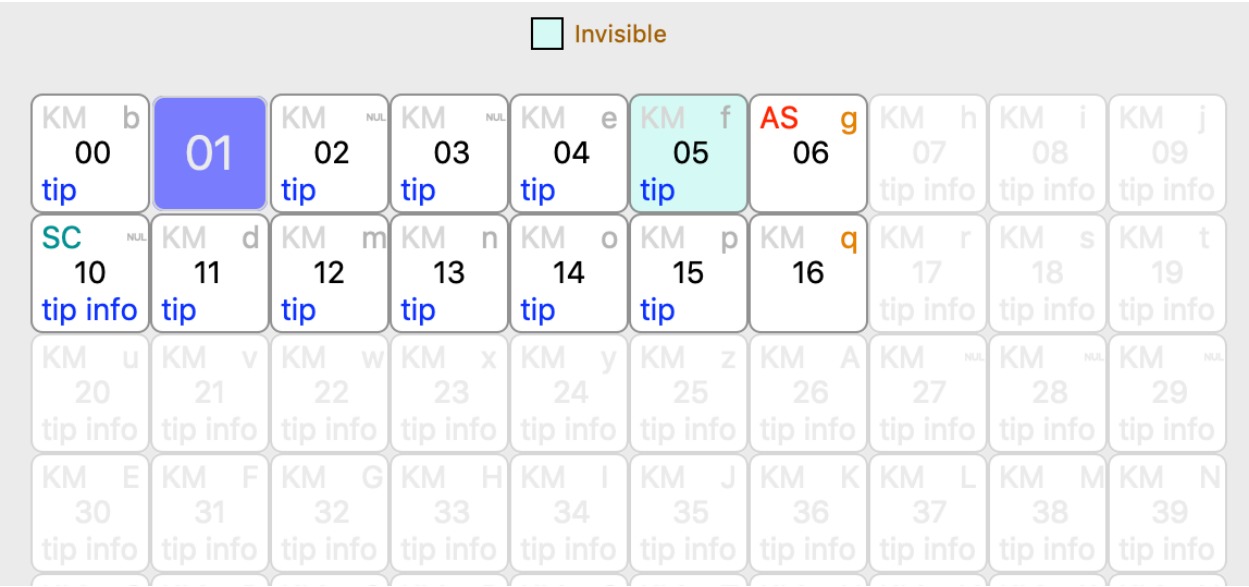
Button Specification

There is a grid in the lower left part of the window which is an abstract representation of the grid of buttons that you are creating for your palette.

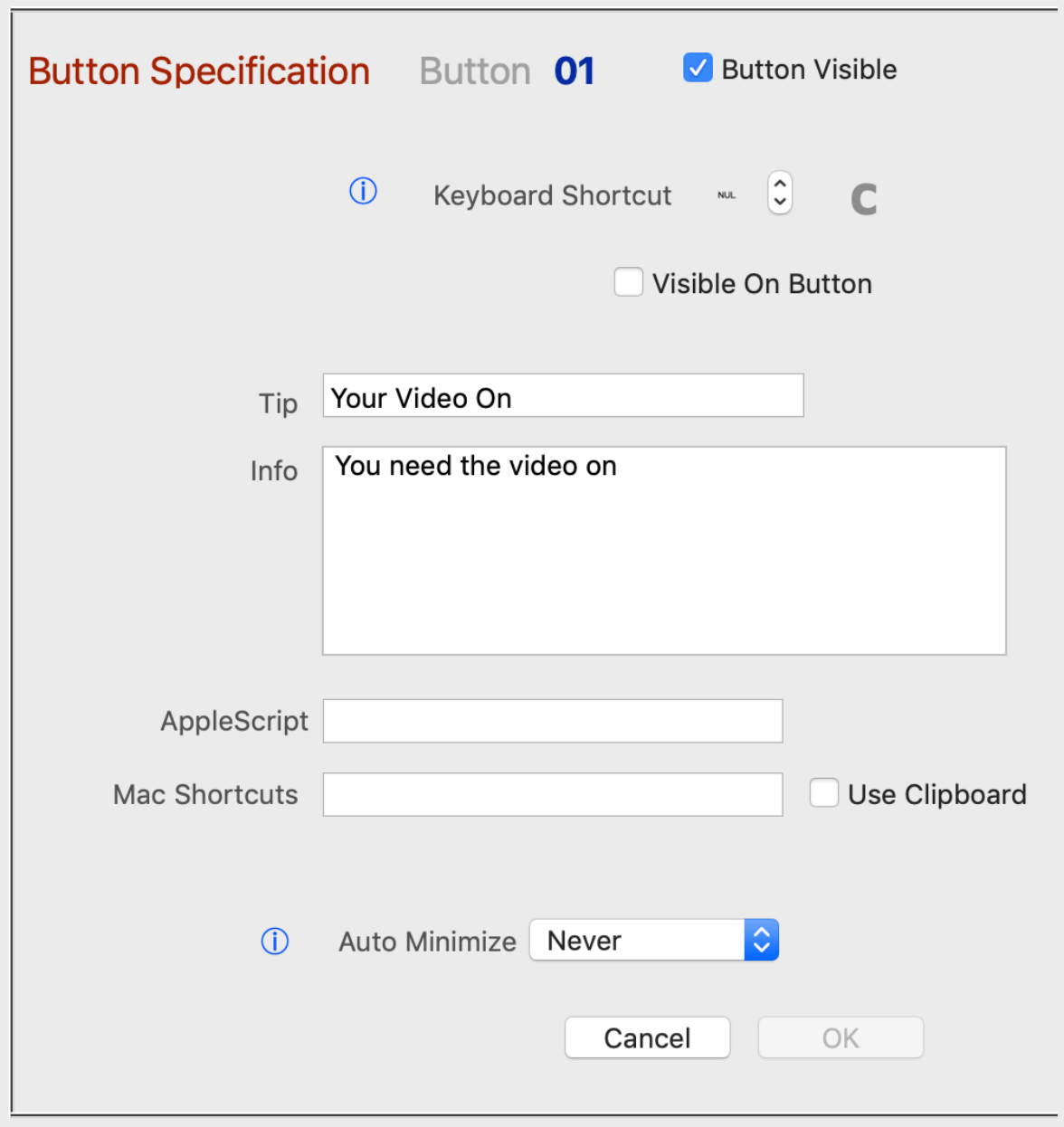


On each grid cell a representation of its properties is seen. Does it invoke Keyboard Maestro (KM) or Shortcut (SC) or AppleScript(AS)? The keyboard shortcut associated with the button is in the right upper corner,. The button ID in the center. And, on the bottom, is there a tip and/or information associated with that particular button?

The button, 01, has been highlighted



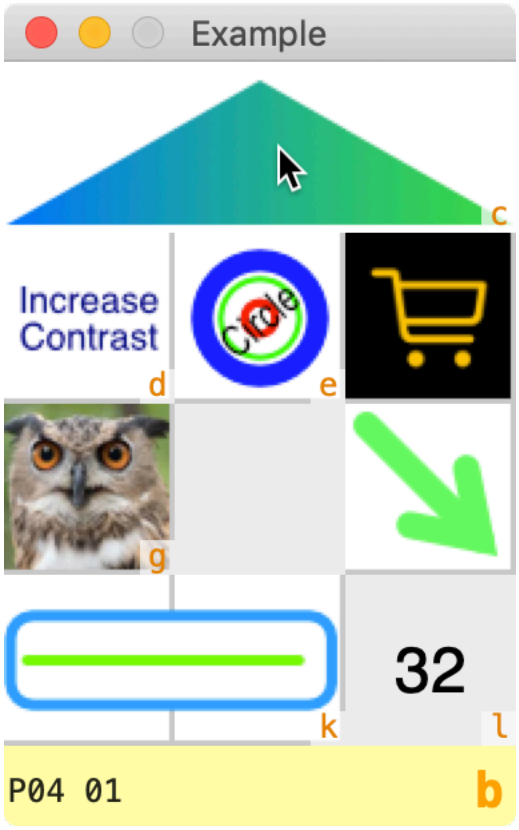
In this example, the user has chosen to make a palette with 2 rows and 7 columns. The grid displays most of the properties associated with the individual buttons. When you click on an individual grid cell, it highlights. In the right lower part of the window, the properties of that individual button are visible in greater detail and can be modified. The 01 grid cell has been highlighted.



These are the properties associated with the 01 button of the palette.

Button Visibility

It is possible to make a given button invisible. This is for design purposes. In the final palette, such buttons have no function and are not seen — there is just a blank space. Some of the examples of palettes show such blank areas within a palette. In the example below, the button to the right of the Owl has been made invisible; this means that it shows up in the palette as a bland gray area and is impervious to clicks.. When you uncheck the Button Visible, the selection grid will show that button in a light blue color.



Keyboard 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. Lowercase is seen before the uppercase characters.

The second decision about shortcuts is whether you want them to be visible in the corner of their button. In the example palette above, 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.

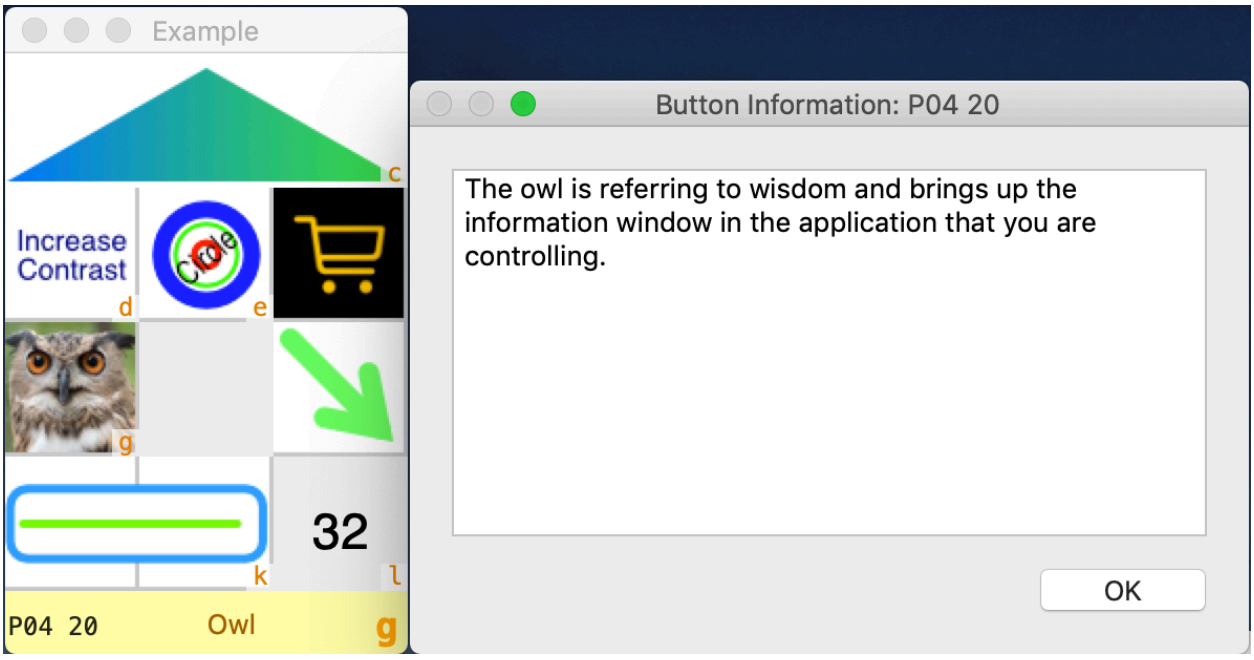
This example also serves to show a design option. By clever assignment of pictures to the individual buttons of the top row, they have been selected to meld into what looks like a single picture. But actually “underneath” the picture the identity of the three distinct buttons is preserved.

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.

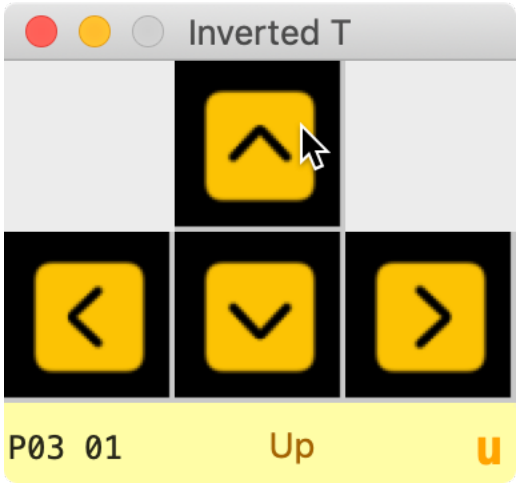
In rare cases where you might need more room to explain (remind) the user of the functionality of button, you can write a paragraph in the *Info* area. The user can access any information that might have been entered here while using the program by Option-clicking on the individual button. With the Option key held down, a click brings up a small explanatory window rather than actually activating the clicked button.

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 notice that during configuration, the word “Owl” was entered as a *Tip* and that is visible in the middle of the information bar.



In the next example below, again the *Tip* data is 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.



Apple Script and Shortcuts

If text information is provided in the AppleScript text area or the Shortcuts text area, then when the user clicks on that button, rather than launching a Keyboard Maestro macro, an AppleScript or a Shortcut will be activated. That text information is the name of the AppleScript or the name of the Shortcut. Text can only be entered in one of these two areas.

If there is no corresponding AppleScript of Shortcut present, a subtle reminder is seen in that the name will be displayed in gray. This issue need not be addressed here immediately. You can write the AppleScript or the Shortcut at a later time. Obviously, if you never attend to this issue, this button will not work because there will be no AppleScript or Shortcut to activate.

Auto Minimize

This is an advanced feature which only rarely will be of any use. The default option is *Never* and that is what should normally be the setting. Auto Minimize refers to minimizing the **ASK_Palette** application to the Dock. This is equivalent to manually clicking on the yellow button in the left upper corner of the palette window.

It is possible to have the application minimize itself to the dock so many seconds after a button is pressed. This can be set to occur 2 seconds, 10 seconds or 50 seconds after the button is clicked. I have used this feature occasionally. Unless you are enamored of this possibility in some workflow, I would not bother with it.

Button Pictures

ASK_Palette will work without any pictures being assigned to the individual buttons, but it is not ideal. Pictures make using the palette more pleasant in that appropriate pictures make the purpose of each button clear at a glance. They do require some work to create; work that need only be done once. PNG or JPG pictures are accepted. Any graphic program can be used to design the pictures.

A button picture should be 64 by 64 points in size. It makes sense to create a 64 by 64 template in whatever program you prefer that you can use as the starting point for each picture. I happen to primarily use *AffinityDesign*. Most of my pictures are created with a gray 1 pixel border along the right and bottom edge. Below is my template. I can paste or draw my picture on this template and easily prepare a Button Picture. The gray lines act to visually differentiate the different buttons on the palette.

Data for the program, and that includes the pictures, have to be stored someplace on your computer. This program shores this data in the application support folder for the program. The full path to this folder is:

/Users/UserName/Library/Application Support/com.bearboat.ASK-Palette/

A subfolder of the *Application Support* folder is *ButtonPictures* and all the Button Pictures need to end up in subfolders of this folder. Each subfolder is named for its palette (P00, P01, P02 etc.) Each button picture starts with the two digits that identify its button. Any text after the first two characters can be whatever you find convenient and memorable.

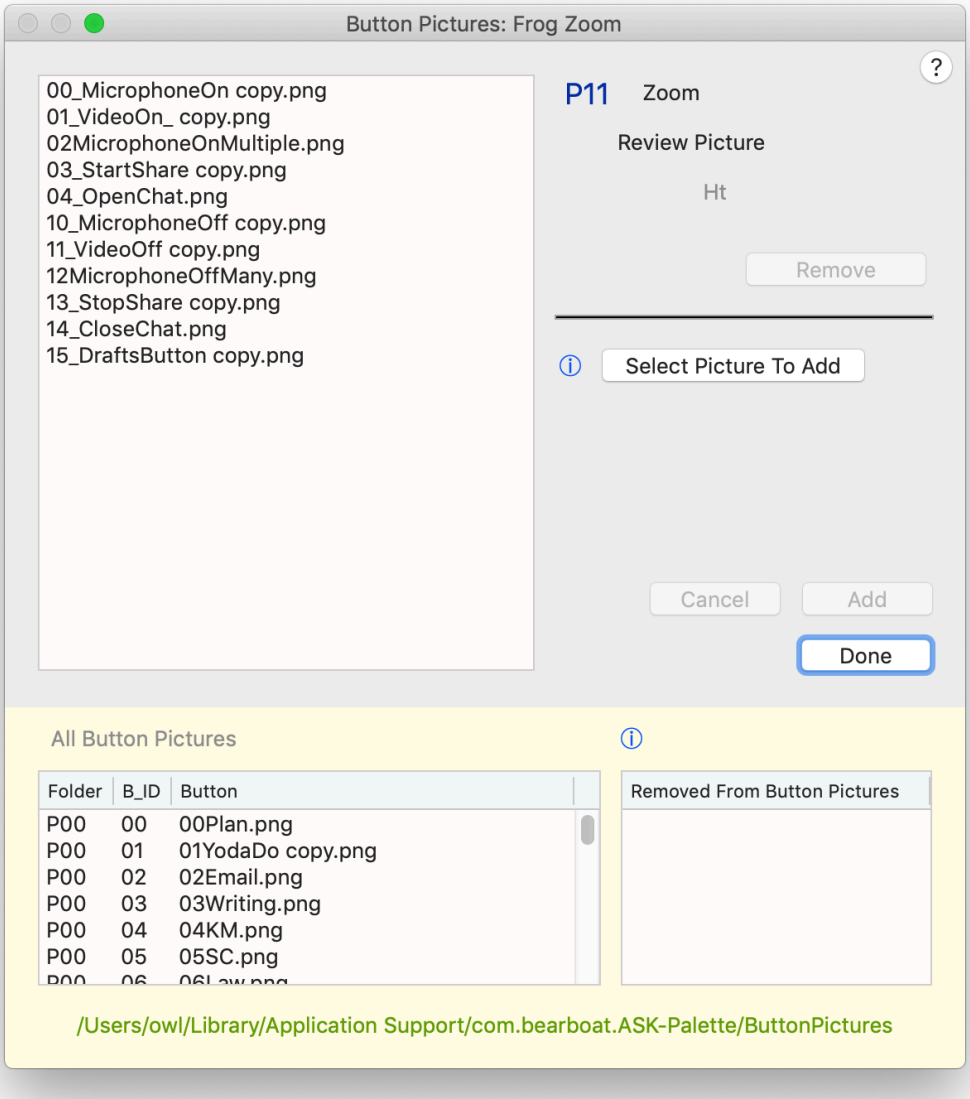
Let's say that the picture for the sixth button of the fourth palette is an image of a grasshopper. Remember that all the numbering starts with zero. The name of this picture has to start with 05. Let's call it 05Grasshopper.

So the correct location for this button, 05Grasshopper, would be in the folder *P03* in the folder *ButtonPictures*. The complete path:

/Users/UserName/Library/Application Support/com.bearboat.ASK-Palette/ButtonPictures/P03/05Grasshopper

There is nothing that prevents a savvy Mac user from placing these pictures in the correct location by hand. But Apple discourages people from mucking around in the Library folder and its subfolders. You can inadvertently stomp on things that will cause you trouble. So **ASK_Palette** includes functionality that helps manage the placement and labeling of button pictures that you create.

When you go to the menu item, *Button Pictures*, under the **Configure** menu, the window below appears.



The Window information is linked to the current palette. The current palette is P11 which also goes by the name Zoom. On the left side is a list of all the buttons associated with the palette that are in their correct location. Click on any one of them to see the picture. The user is given an opportunity to remove the picture. When you click on the Remove button, a picture 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.

Any picture on the computer can be selected using the Select Picture To Add button. If it is the correct size and has an appropriate name, it can be added to the button pictures that belong to this palette.

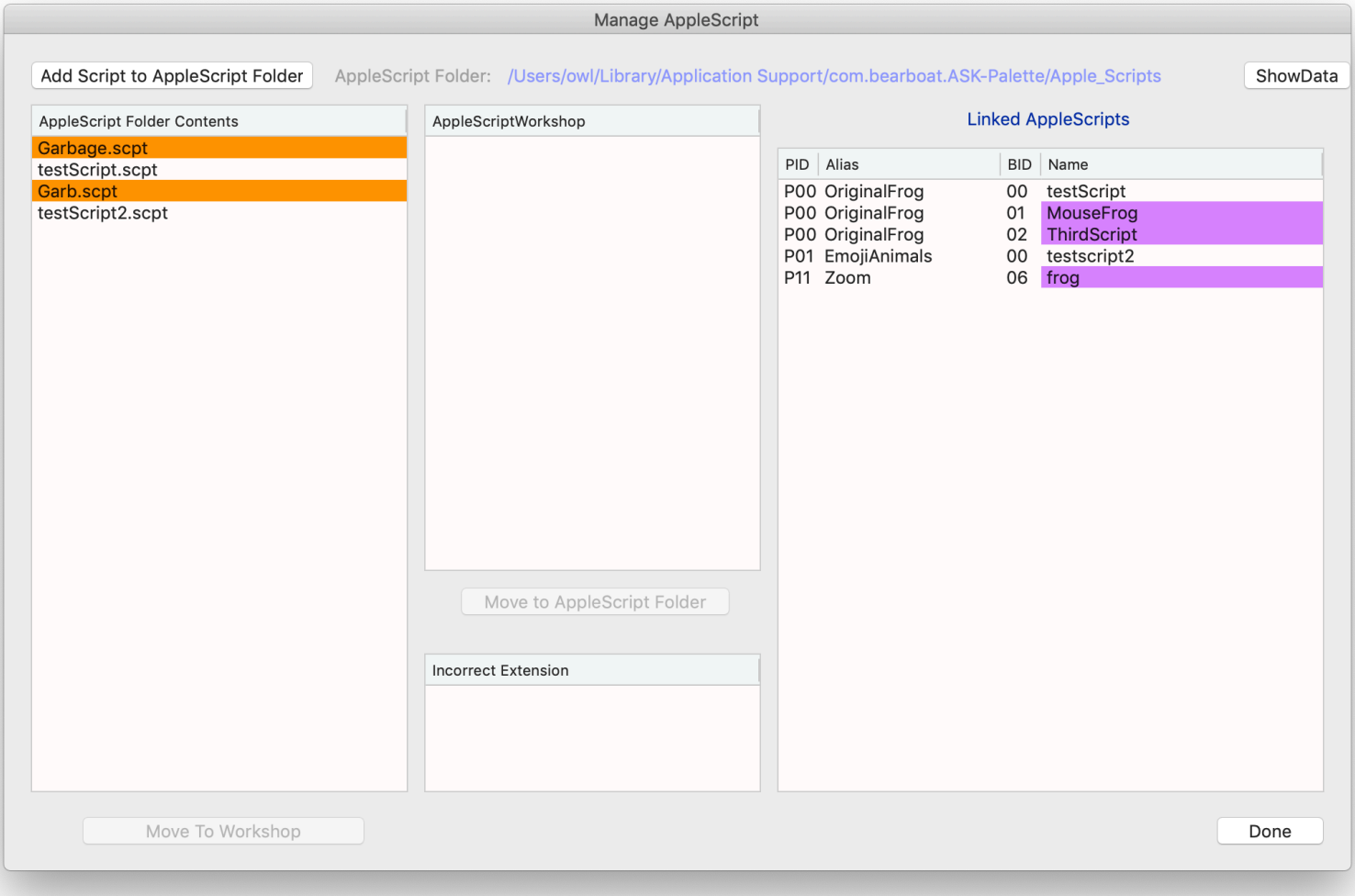
The yellow area in the bottom third is only informative. It lists ALL the pictures in the *ButtonPicture* folder. These live in the corresponding subfolders within the *ButtonPicture* folder. There is also a list of pictures that were automatically removed from the *ButtonPicture* folder because the program detected an incorrect name or some such thing. This listing will generally be empty. Any such picture that is removed automatically in this fashion will end up in a *RemovedFromButtonPictures* folder on the Desktop.

Manage AppleScript

AppleScripts are files that the user creates or obtains from others. The AppleScript app compiles these into .scpt files that can be run. These files can exist in any location on the Mac, but to be addressed by the **ASK_Palette** program, they need to live in a specific folder, *Apple_Scripts* which is a subfolder of *com.bearboat.ASK-Palette*. The full path is:

/Users/UserName/Library/Application Support/com.bearboat.ASK-Palette/Apple_Scripts.

When configuring a palette, buttons that are to initiate AppleScripts are associated with the name of the AppleScript file. In order for this to work, the file has to be in the proper location. While it is possible for an experienced Mac user to place the file using the Finder, it is better to take advantage of the Window that appears when you go to the menu item, *Manage AppleScript*, under the **Configure** menu.



In this window, on the left is a list of all the AppleScripts that exist in the relevant folder. If the item is highlighted in orange it means that at this time no palette button actually references this script. There is a button, Add Script to Apple_Scripts Folder that allows the user to choose any AppleScript on the computer and add it to this special folder.

To edit an AppleScript that lives in this folder, it is best to move it into a folder on the Desktop called *AppleScriptWorkshop*. Here you can open the script with the ScriptEditor and make any necessary changes. You will see it listed in the middle of the window as currently residing in the *AppleScriptWorkshop*. Once any necessary corrections have been made, it can be returned to the AppleScript folder — Move to Apple_Scripts Folder button.

On the right side of the window is another list which contains all the palette buttons that have been associated with an AppleScript name. If the name is not presently associated with any script in the *Apple_Scripts* folder, it will be highlighted in purple.

Rename Palette Alias…

This is a menu item under the **Configure** menu that 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.

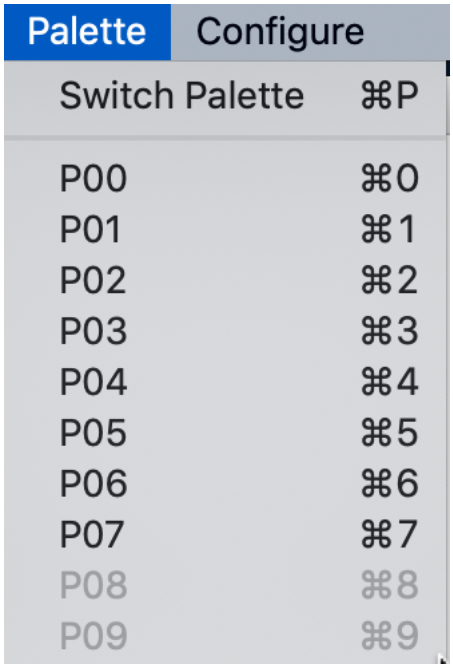
Delete Palette…

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

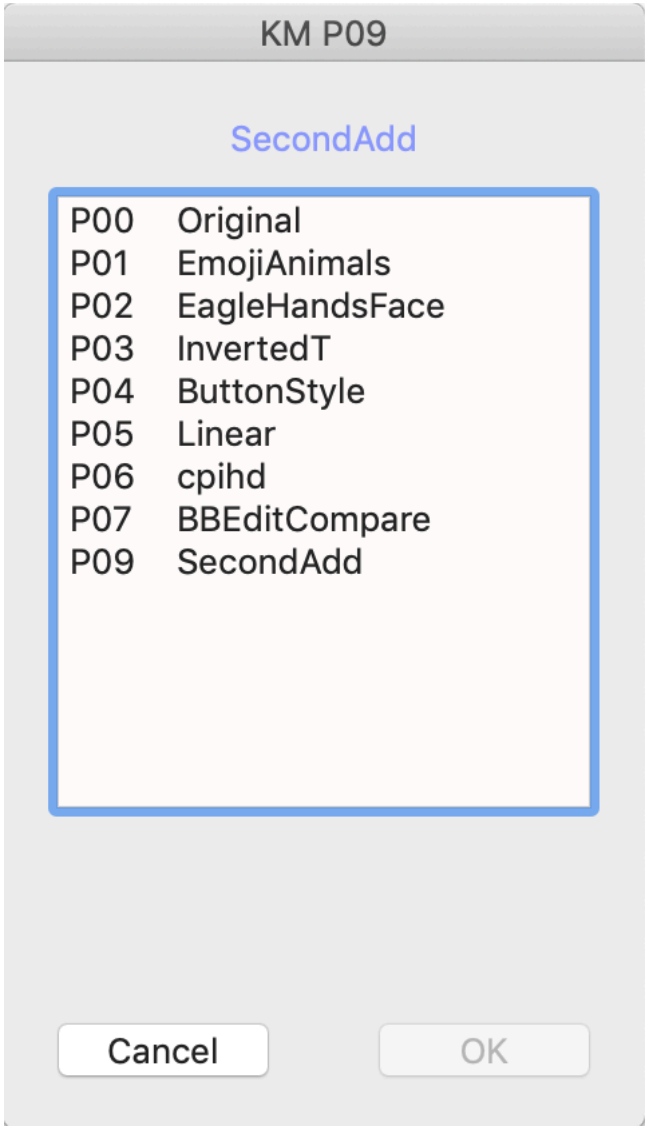
Using Your Palettes

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 **App_Palette**. 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).

Designing Button Pictures

In general the goal of creating pictures for your palette buttons is to have pictures that declare or, at least hint at, the purpose of the particular button.

In this example, both text and an image have been used to convey the purpose of the button

Designing buttons can be done in any graphics program capable of creating png or jpg images that are 64 points in size.

Sources of images are myriad. Emoji and Apple's SF_Symbols are one resource that I use frequently. For the super-creative, you can create you images from scratch. As mentioned above, I use *AffinityDesigner* as my primary graphics program. It is a vector-based program well suited to the task. (*Adobe Illustrator* is a well-known app in this genre).

SF Symbols

This is my favorite source/inspiration. It is possible to export as an svg file which is a vector format that can be easily modified. I will mention here the steps required using *AffinityDesigner*. I would assume that other vector graphics programs would be able to do similar things.

The Apple program, *SF Symbols*, allows you to choose one of thousands of images (icons) and export it as an svg file (*Export Custom Symbol Template*). You can open the svg file with *AffinityPhoto* and can see that icon is provided in various weighs and sizes. Select the one that you want to start with and open that particular icon in a new *AffinityPhoto* document. You will find that the various pieces (perhaps 3 to 9 for these simple graphics) are labeled Curves. To be able to work with them individually, you need to be able to break them up into the components. In *AffinityPhoto*, you select the Curves and then go to menu item *Layer >> Geometry >> Separate curves*.

Now you have access to the *individual* curves with their various controls points. In this example, the icon is made of four curves (The handle/ structure; the basket; the back wheel; the front wheel). You can be a creative as you want. Change the color. Apply gradients. Distort the element. Ultimately, you can export your creation as a 64 point png image and use for a palette button. Here I have selected and then modified the basket curve.

Original and Modified

Help

There is lots of help available scattered through the application. These small graphics can be clicked on to get local help about the application's function.

In addition, you may notice that when the cursor lies over many of the labels, the labels turn a blue color. This is an indication that clicking on the label will also display some helpful information about its function. Much of the content of this manual is provided in bite-sized chunks within the application.

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.App-Palette

That folder (*com.bearboat.App-Palette*) will contain three things

1. A folder called: *ButtonPictures*
2. A folder called: *AppleScripts*
3. 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 **APP_Palette** work on a different machine, you would have to copy the *com.bearboat.App-Palette* folder with its sub-folders and place it into the *ApplicationSupport* folder of the different machine.

Upgrading From KM_GridPalettes

This topic is explained in greater detail in the Miscellaneous Odds and Ends section at the very end of the manual. It takes some familiarity with advanced features of the *Finder* to access the *~/Library/Application Support* folder. You can Google the topic to learn how. It varies slightly depending on the macOS version.

Run **ASK_Palette** for the first time and quit. This will assure the presence of a new folder in the *Application Support* folder called *com.bearboat.Ask-Palette.*

In the *Application Support* folder would also be living your old *com.bearboat.KM-GridPalettes* folder. It should contain a subfolder called *ButtonPictures* and a file called *PreserveState.txt* . Move copies of these two entities into your new folder, *com.bearboat.Ask-Palette*.

Now when you start your new program, **ASK_Palette**, it should now have ready to go the palettes you created in **KM_GridPalettes**.

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.

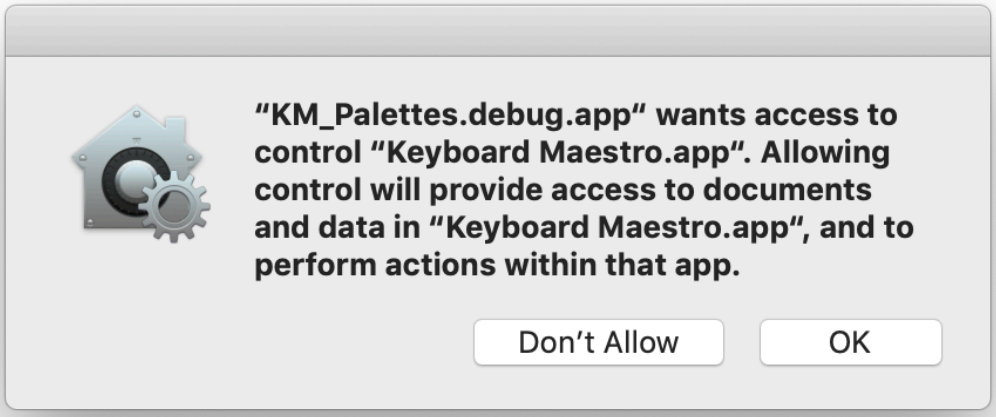
1. There is no **Keyboard Maestro** macro whose name corresponds to the name of the palette on which the button resides. Remember that every palette has a name ID like P00, P01, or P02. There has to be a **Keyboard Maestro** macro that you have created that has this identical name.
2. The **Keyboard Maestro** macro is in a Group that the **APP_Palette** program does not have access to. It is probably best to create a Group that **is** accessible by **APP_Palette** 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.
4. If the button is associated with an AppleScript, that named AppleScript has to exist in the expected folder
5. If the button is associated with a Shortcut, that Shortcut has to exist in the Shortcuts app of the hosting Mac.

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 **ASK_Palette** 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 of the example. It makes **ASK_Palette** the foremost application after the macro has run. It is often convenient to include this last step. If you want to use shortcuts inside of **ASK_Palette**, it really makes sense. For a keyboard shortcut to work, **ASK_Palette** has to be foremost. If you end up having to use the mouse to click on the **ASK_Palette** 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 in a palette. Even if **ASK_Palette** is not foremost, the **initial** click will invoke the buttons action. This is to say that you do not have to click first to "activate" **ASK_Palette** 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 ASK_Palette

- 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:

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%

- 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 ASK_Palette
  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.

AKS_Palette is strictly a one way street. **AKS_Palette** is almost entirely a GUI. The user interacts with the program only to specify actions that will be performed by **Keyboard Maestro** or AppleScript or Shortcuts and the apps that they are, in turn, 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

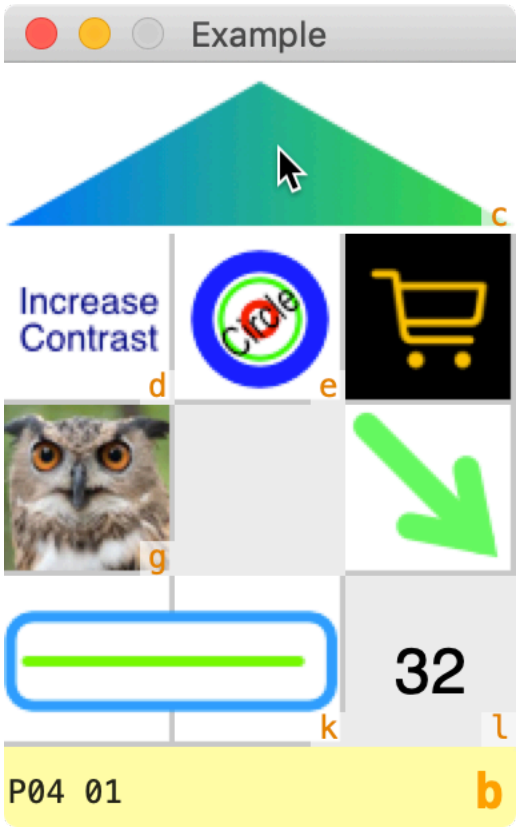
Twitter: @_rlivingston or @bearboat

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

Miscellaneous Odds and Ends



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.ASK-Palette*.

While the Mac user familiar with accessing the *~Library* world could directly access this folder, Apple discourages it.

ASK_Palette 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.

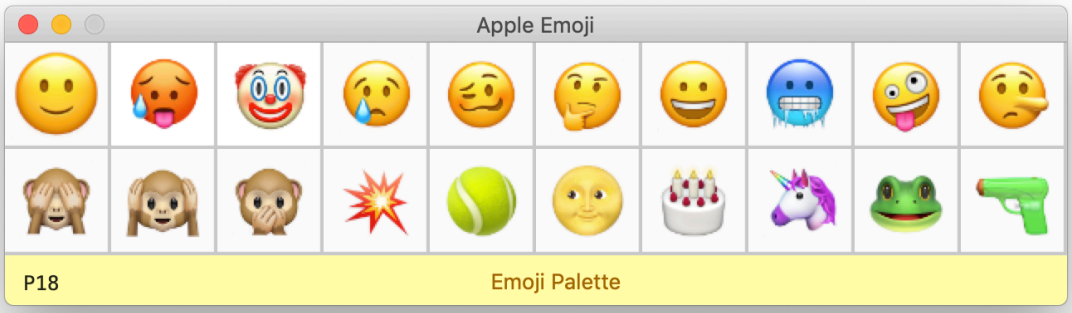
The *Select Picture To Add* button will open a Finder choose file dialog. You can select any picture that you want that is appropriately named and sized, and it will end up being copied and moved to the correct location so that subsequently the palette will show this image over its button.

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

An Example Palette

The data for an example palette has been provided. As befits an example, it is simple. A palette of emoji pictures has been created. Clicking on any of these buttons does essentially the same thing. If you are in a program capable of accepting an emoji (*Mail*, *TextEdit*, *Pages*), clicking on one of these buttons merely inserts that emoji into the text.



This palette works through a **Keyboard Maestro** macro. The macro basically consists of a large Switch statement.

P18, name name of the macro, is activated by the P18 palette. **ASK_Palette** sends which button the user clicked to the macro. You have to make sure that the name of the palette corresponds to the name of the macro that it controls.

P18

No triggers specified.
Will execute the following actions:

- Comment “P18”

  P18 contains all the actions relating to the P18 palette. When the 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 another Macro associated with the button.

- Comment “Type Emoji”
  Presumably, you are originally in a program that can accept Emoji. So at the start of the macro, we return to that program (Activate Last Application).

- Activate Last Application

- Comment “Variable whichButton gets assigned”
  When you click on the ASK_Palette %TriggerValue% indicates which button was clicked by the user.

- Set Variable “whichButton” to Text %TriggerValue%

- Switch of Variable “whichButton”
  If it is “00”, Execute the Following Actions:
  - Insert Text by Typing
    🙂

- If it is “01”, Execute the Following Actions:

- Insert Text by Typing
  🥵

- If it is “02”, Execute the Following Actions:
  - Insert Text by Typing
    🤡

- If it is “03”, Execute the Following Actions:
  - Insert Text by Typing
    😢

- If it is “04”, Execute the Following Actions:
  - Insert Text by Typing
    😢

Etc.

Tips for the Button on this Palette

00 Slight Smile

01 Hot

02 Clown

03 Sad But Relieved

04 Woozy

05 Thinking

06 Grinning

07 Cold

08 Zany

09 Lying

10 See No Evil

11 Hear No Evil

12 Speak No Evil

13 Explosion

14 Tennis

15 Full Moon

16 Birthday Cake

17 Unicorn

18 Frog

19 Water Pistol

Practice by Making this Palette

To make this palette, open the **ASK_Palette** application and go to the **Configure** menu and select *New Palette…*. Give it a *Name/Alias* - something like Type Emoji. Make note of the true name of the palette you have created. It will be of the form Pdigitdigit. Let's say that it is P07. This is important, because the name of the **Keyboard Maestro** macro that it controls has to be also P07. The name of the macro provided for this example, P18, has to be changed to whatever name was assigned to the palette that you are creating.

Now go back to the **Configure** menu and select *Grid Construct*. A window will open up with the tools to define your palette. The first thing to do is to specify that you want 2 rows and 10 columns. The active buttons will show up on the grid.

Now click on the individual grid elements. The first one is (00). Activate it and you will be able to assign it a *Tip*. The appropriate text for the tips has

been provided above. The *Tip* for the first button is **Slight Smile**. Accept your edit with the OK button and then activate the second grid element (01). Fill in **Hot** for its *Tip*. The activate the third grid element (02). Supply **Clown** for its *Tip*.

Continue this process until all 20 buttons have been assigned the appropriate *Tip* text. At this point, click on the Done button in the lower right corner and you will see your palette. It will not have any pictures, but it will function if there is an active **Keyboard Maestro** macro that has the name of the palette (Pdigitdigit) and the content of the macro is as has been provided with this example.

Now let's get pictures on the buttons. A folder of 64 pixel images has been provided with this example called *Sample ButtonPictures*. Go to the **Configure** menu and select *Button Pictures*. Click on the button, Select Picture To Add, and navigate to the folder of *Sample ButtonPictures* that has been provided with this example. One by one select these pictures and add them to the palette.

Once this process is complete, you will have a working palette with decorative pictures on the palette buttons. Make sure that the **Keyboard Maestro** engine is active and that there is a macro called P07. Fire up an application (like *TextEdit*) that accepts emoji and you can use your palette and see it working.

Upgrading from KM_GridPalettes

A few of you may be migrating from **KM_GridPalettes** which is an earlier version of the **ASK_Palette** application. You can replace your use of **KMGrid_Palettes** with **ASK_Palette** if you proceed with the following steps. This requires enough comfort in your use of the *Finder* to open up the *Library* folder and retrieve and place files there.

The *Library* folder on your Mac contains a folder called *Application Support*. This folder contains files that are used by the various applications on your computer and you should be careful not to disturb these files which in general are not intended to be directly accessed by users.

The *Application Support* folder can contain two folders relevant to this discussion.

The first is *com.bearboat.KM-GridPalettes* and the second is *com.bearboat.ASK-Palette*. These folders contain all the relevant data for **KM_GridPalettes** and **ASK_Palette**. To upgrade to using **ASK_Palette**, without losing all the work that you previously did in **KM_GridPalettes**, you copy files that exist in the *KM_GridPalettes* folder to *ASK-Palette* folder.

Instructions

1. Run the **ASK_Palette** program for the first time and immediately quit the program. This will take care of creating the *com.bearboat.ASK-Palette* folder in the *Application Support* folder. Within this folder, an empty folder called *ButtonPictures* and a text file called *PreserveState.txt* will have been created.
2. Go to the *Application Support* folder. To discourage inadvertent mucking around in this area, Apple makes accessing the *Library* folder slightly tricky. When in the Finder, go to the **Go** menu. You will see a list of destination folders that you might be interested in accessing. The *Library* folder will not be one of them. However, if you hold down the

Option Key, then *Library* will appear. So hold down the Option Key and select *Library*.

3. That will take you to the *Library* folder and you can see all its subfolders. The one that you are interested in is *Application Support*. Open that folder and look for *com.bearboat.KM-GridPalettes* and *com.bearboat.ASK_Palette*. Copy the *ButtonPictures* folder in the *com.bearboat.KM-GridPalettes* folder to the *com.bearboat.ASK_Palette* folder. This will replace the empty *ButtonPictures* folder that is already there. Then copy the file *PreserveState.txt* from the *com.bearboat.KM-GridPalettes* folder and place it in the *com.bearboat.ASK_Palette* folder. Replace the *PreserveState.txt* that is has already been created in the *com.bearboat.ASK_Palette* folder when you ran the program for the first time.

4. Now close all these Finder windows making sure that you do not disturb the other files and folders that live in the *Library* folder.

5. Open the **ASK_Palette** application and it should now contain all the palettes that you made while running its predecessor program **KM_GridPalettes**.