Launch Deck Trellis M4

Created by John Park

https://learn.adafruit.com/launch-deck-trellis-m4

Last updated on 2024-06-03 02:34:30 PM EDT

Table of Contents

Overview 3

- Parts

App Launching with Automator 5

- Automator
- Action
- Launch Application
- Input & Dropdown List
- Test
- Save the Service
- Keyboard Shortcut Setup

Quicksilver 13

- Quicksilver
- Assign Shortcut

Windows App Launching 17

- Windows 10 Task Bar
- AutoHotkey
- Keyboard Launchpad

Code with CircuitPython 20

- CircuitPython Preparation
- HID Keyboard Basics
- Keyboard Press/Release
- Media Control
- Launch Deck Code
- Customizing
- Single, Unmodified Keystrokes
- Button Coordinates
- Add a Button
- Edit a Button
- Button Removal
- Layout Tips

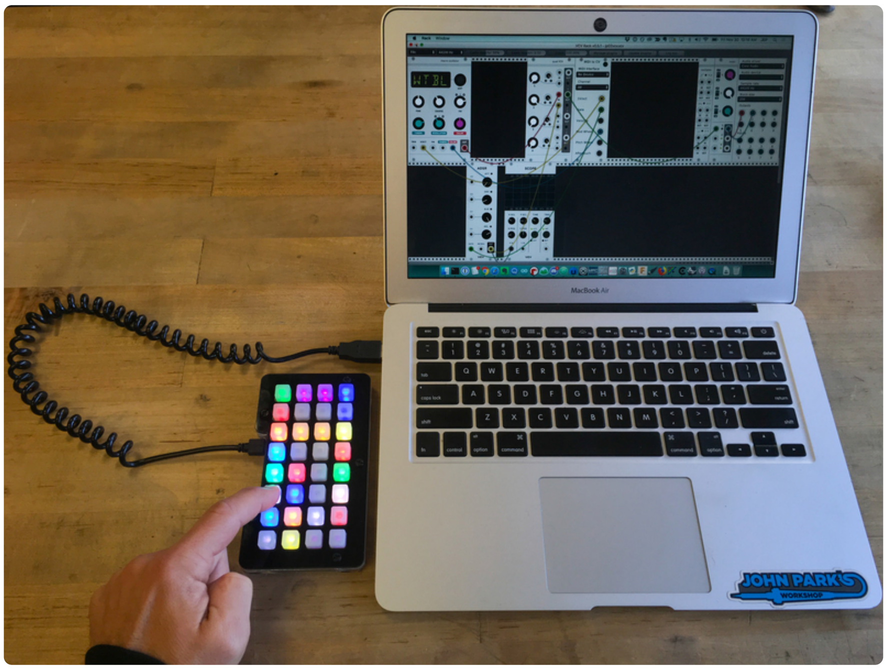
Overview

The general purpose computer coupled with the general purpose keyboard and mouse is a powerful combination, but wouldn't you like something a bit more... specific purpose for launch applications, playing media, and firing off keyboard-combos? Enter the Launch Deck Trellis M4!

The Trellis M4 can emulate a USB keyboard, so your computer has no idea that when you press a single button on the Trellis M4 it isn't actually you pressing a four button keyboard combo on your real keyboard!

The Launch Deck Trellis M4 uses a simple CircuitPython program to send totally customized HID USB keycode and media commands with any of its colorful 32 buttons. You can decide which buttons do what, and color code them as you like. A red button to launch Gmail, an orange button for Firefox, a green button for an old school homebrew themed terminal window, and so on.

This guide will show you how to set up the NeoTrellis M4 to send keyboard combos out to an app launching program, such as Automator and Quicksilver on macOS, and AutoHotkey in Windows.



Parts

Adafruit NeoTrellis M4 with Enclosure and Buttons Kit Pack

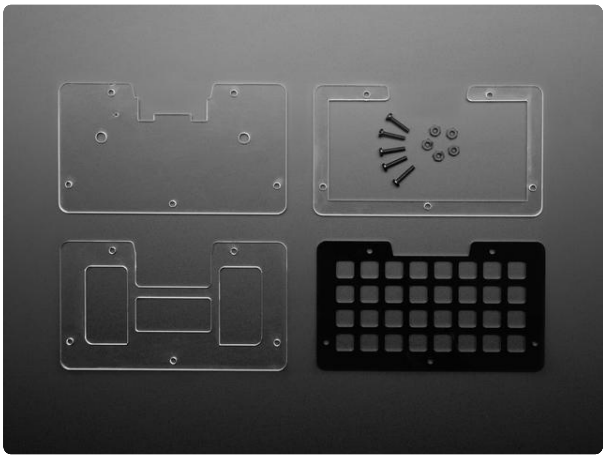
So you've got a cool/witty name for your band, a Soundcloud account, a 3D-printed Daft Punk...

https://www.adafruit.com/product/4020

USB cable - USB A to Micro-B

This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...

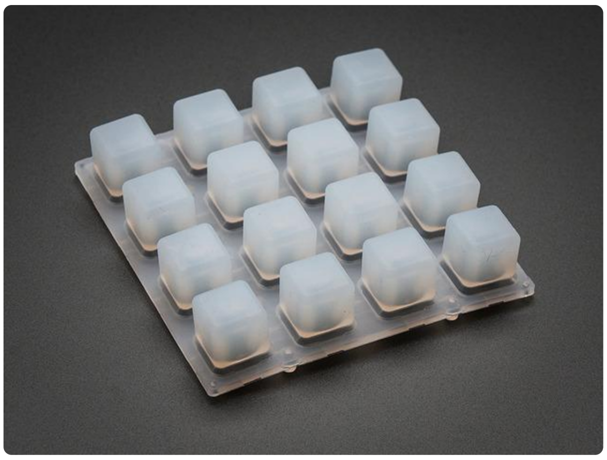
https://www.adafruit.com/product/592



NeoTrellis M4 Acrylic Enclosure Kit

So you've got your Adafruit NeoTrellis M4, a cool/witty name for your band, a Soundcloud account,

https://www.adafruit.com/product/3963



Silicone Elastomer 4x4 Button Keypad - for 3mm LEDs
So squishy! These silicone elastomer keypads are just waiting for your fingers to press them. Go ahead, squish all you like! (They're durable and easy to clean, just wipe with mild...
https://www.adafruit.com/product/1611

App Launching with Automator

Before we can set up the NeoTrellis M4 buttons to launch our apps, we need to lay the foundation with software that can assign keyboard shortcuts to launch apps.

We're all accustomed to using certain keyboard shortcuts (e.g., **control+c** or **command+c** for **copy**) and media keys built into our keyboards (**play/pause, mute, vol+/vol-**). Some are universal and work in most apps, some are system wide, and most operating systems even allow a small degree of freedom in creating custom key combos to launch applications. However, this is often restrictive.

There is, therefore, a whole world of customizable macro script applications and services that can run on your computer in order to allow maximum control. You can search online for many options, here we'll cover three in particular, **Automator**, **Quicksilver**, and **Windows 10 Taskbar**.

Automator

Automator is built into every macOS computer, so if you're on a mac, you've already got it!

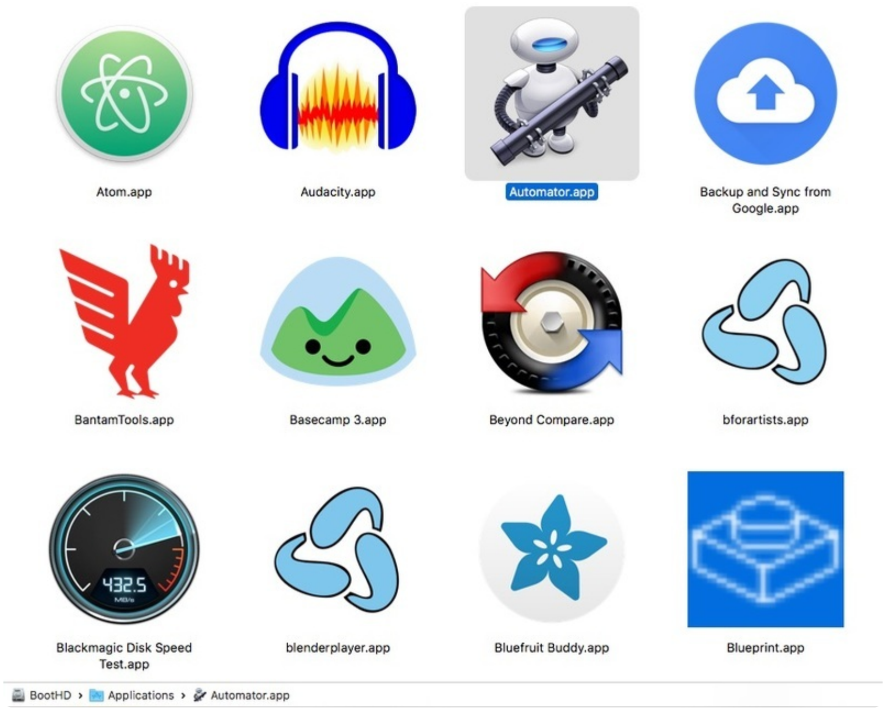
Automator is incredibly powerful, as it can be scripted to do nearly anything and is neatly integrated into the operating system. It is commonly used to automate repetitive tasks, such as resizing and renaming huge groups of image files at the push of a button.

It can also be used as an application launcher, based upon keyboard shortcut entry. Here's an example of how you can set up Automator to launch Firefox with the keyboard combination of **command+option+ctrl+1**

It's possible to create app launching shortcuts directly from the System Preferences without using Automator, but this is a good way to learn to use Automator, which is capable of much more sophisticated macro-like behaviors as well.

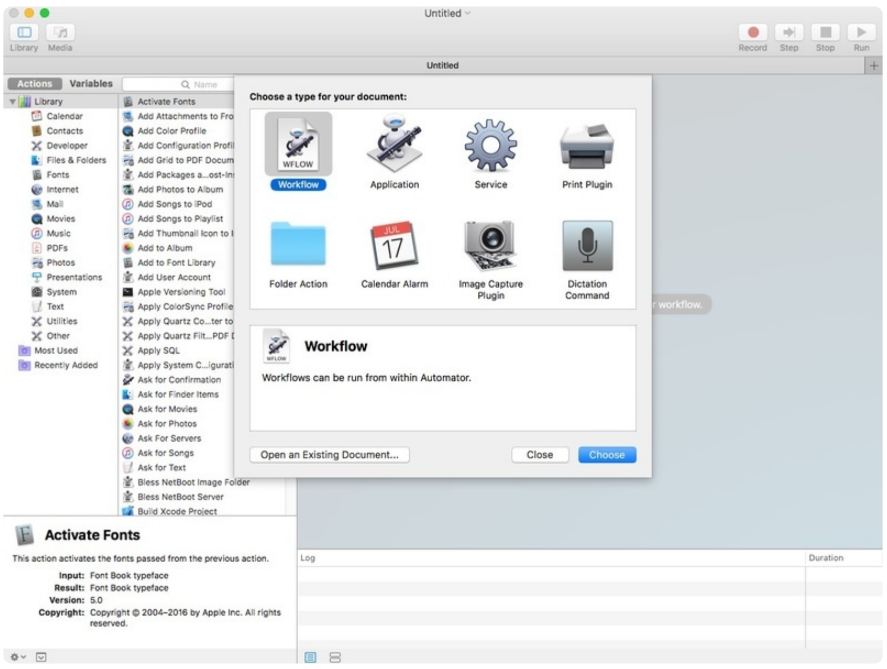
Launch Automator

First, open Automator by finding it in your /Applications folder and double-clicking it.



Automator will start up and you'll be presented with a document type chooser window.

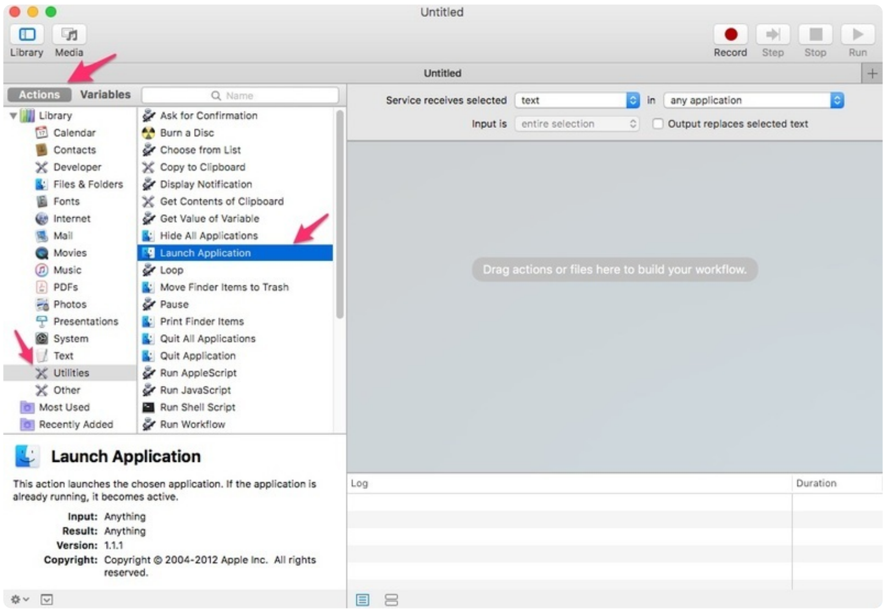
Click on the **Service** gear icon, then press the **Choose** button.



Action

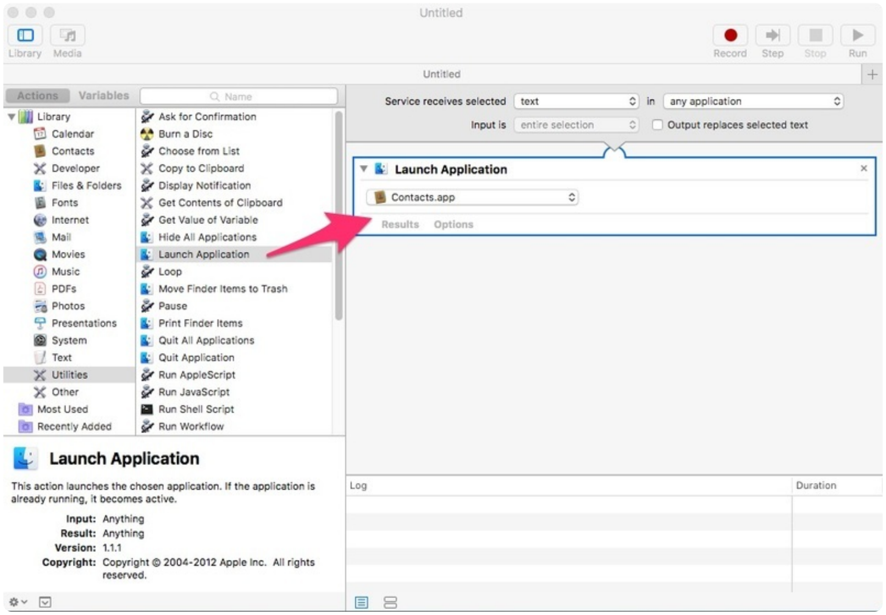
Now, we have an empty service window waiting to be given an action. Click on the **Actions** box in the upper left corner of the interface, then click the **Utilities** item in the

Library turndown to filter the actions to a smaller list. Then, click on **Launch Application** to highlight it.



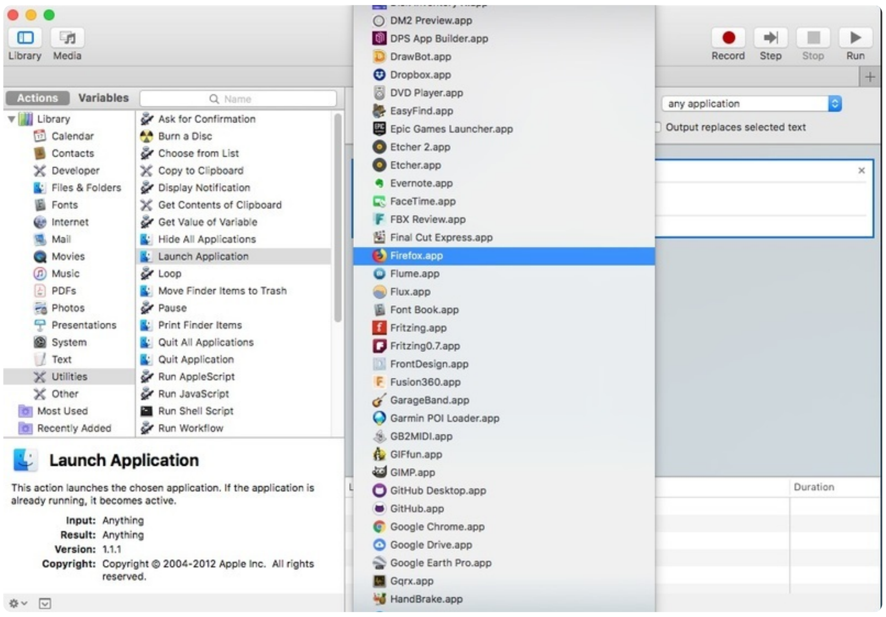
Launch Application

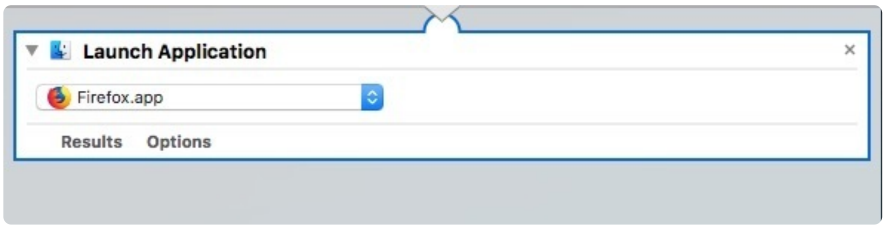
Now, you can drag the **Launch Application** action over to the empty workspace on the right, this will add it to the service.



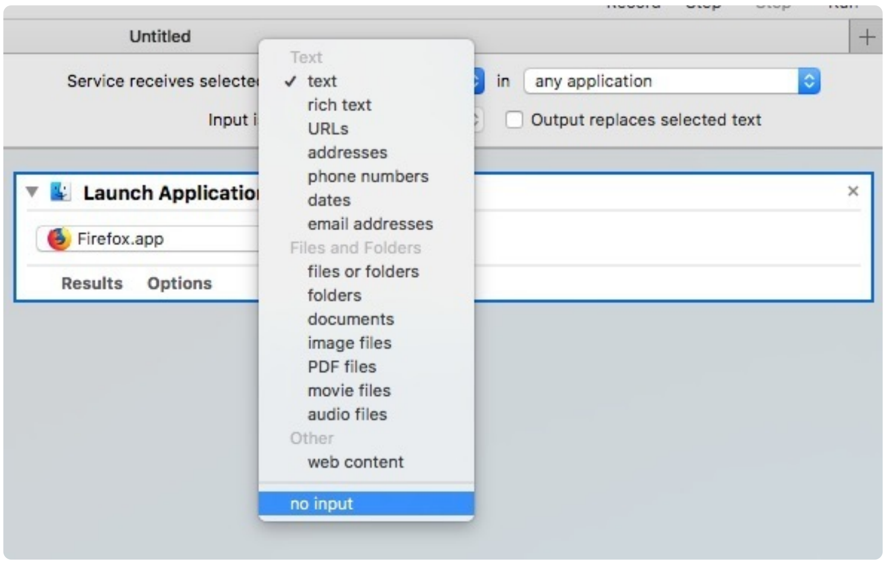
Input & Dropdown List

Next, we need to fine-tune some options. First of all, let's choose the application we want to launch from the dropdown list, in our case **Firefox.app**. Click where it currently says **Contacts.app** to show the full list popup, then find and click on **Firefox.app**.





Many services are set up to receive input of some kind, such as a file to manipulate, a bit of text to adjust, and so forth. Our service will not need any input, so we will change the **Service receives selected** field from the default of **text** to **no input**.



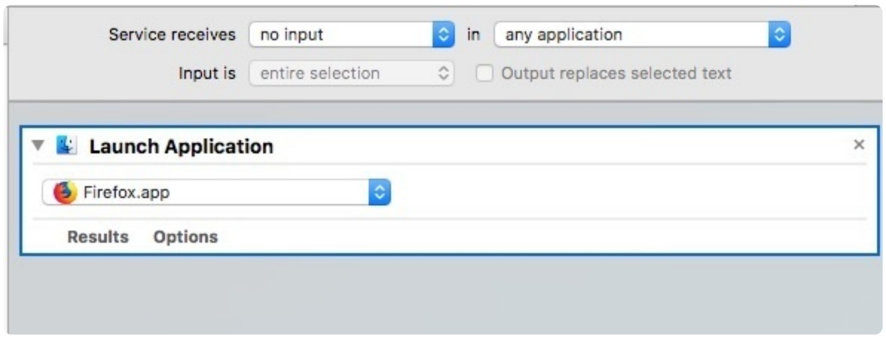
Test

It's a good idea to test out your action at this point -- simply click the **Run** button in the upper right corner and it will launch (or switch to) Firefox.

We want this service to run no matter where the system has focus, be it in the Finder, or a Terminal session, or inside of Photoshop, when we hit the shortcut to invoke the Firefox.app service, it should run. Therefore, we will leave the **in** field at the default of **any application**.

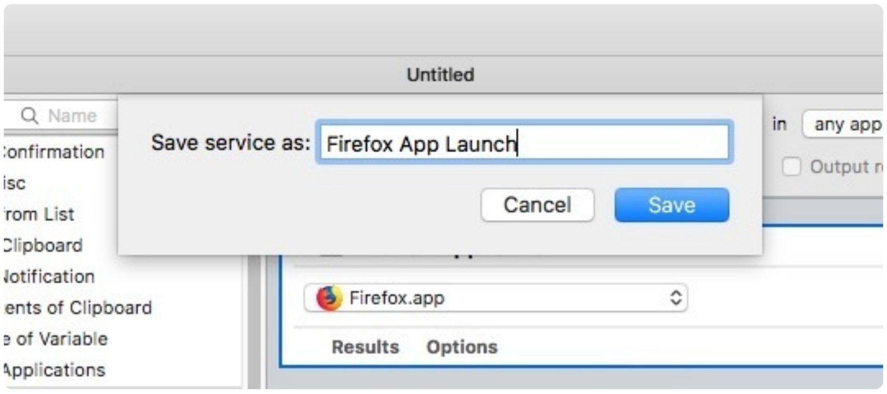
Save the Service

That's all there is to our service, so let's go ahead and save it. Click on **File** > **Save...** and then give it the name **Firefox App Launch** and click **Save**.



Service receives: no input in: any application
Input is: entire selection
Output replaces selected text

Launch Application
Firefox.app

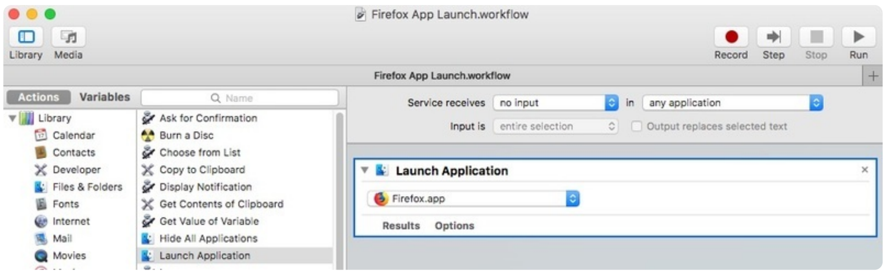


Untitled

Save service as: Firefox App Launch

That does it for the Automator section of the process. The service has now been saved and is accessible by the entire operating system. (Confusingly, it saves with the ".workflow" extension, even though we specified that it is a "service", you can ignore this.) You can quit Automator now.

If you're curious about it, the .workflow file is saved to the /Users/your_name/Library/Services/ directory, a.k.a., ~/Library/Services



Service receives no input in any application

Input is entire selection

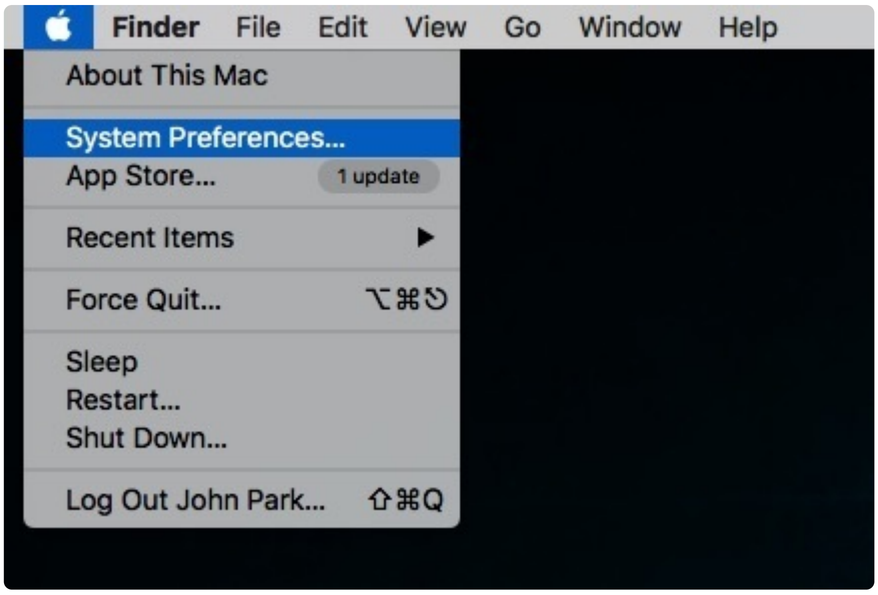
Output replaces selected text

Launch Application

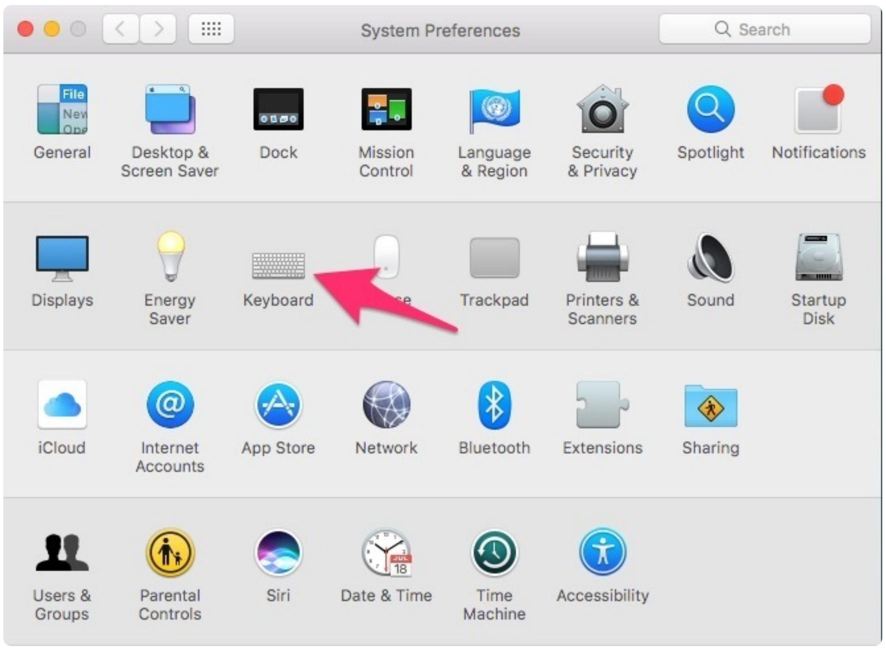
Firefox.app

Keyboard Shortcut Setup

Now that we have created the service we need a way to invoke it with a keyboard shortcut. We'll do so from the Keyboard system preferences. Click on the **Apple Menu > System Preferences...**

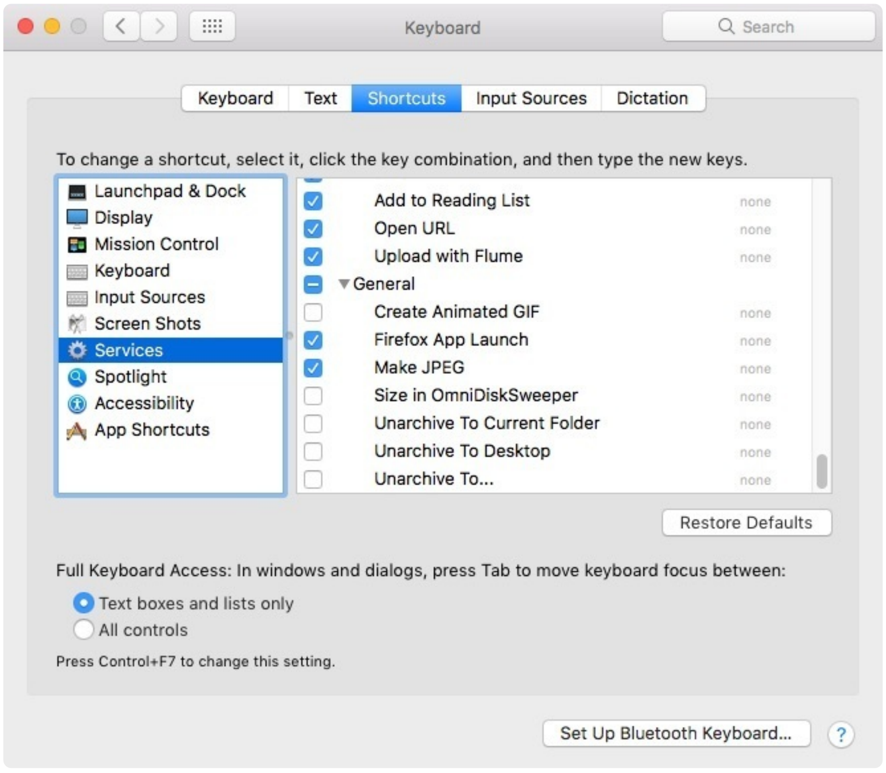


Once the **System Preferences** window opens, click on the **Keyboard** entry.



Click on the **Shortcuts** button at the top, and then the **Services** entry on the left. This window allows you to set and change keyboard shortcuts. Scroll to the bottom of the list on the right and you'll see our **Firefox App Launch** service.

If you really wanted to skip using Automator, jump to the App Shortcuts menu item on the left instead.



To change a shortcut, select it, click the key combination, and then type the new keys.

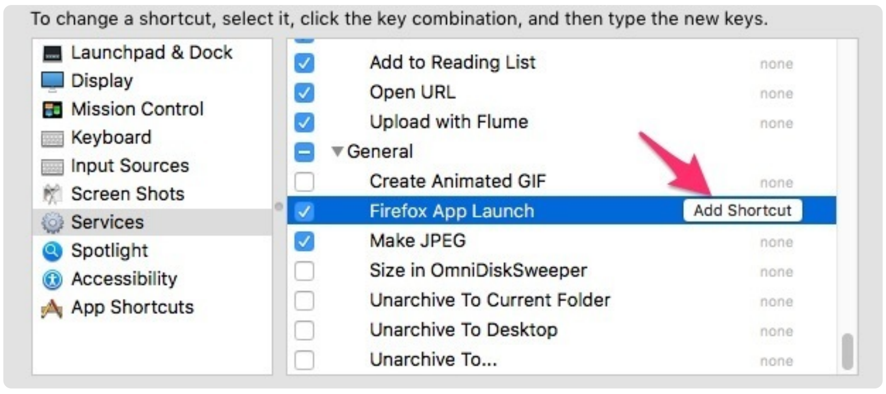
✓	Add to Reading List	none
✓	Open URL	none
✓	Upload with Flume	none
−	▼General	
☐	Create Animated GIF	none
✓	Firefox App Launch	none
✓	Make JPEG	none
☐	Size in OmniDiskSweeper	none
☐	Unarchive To Current Folder	none
☐	Unarchive To Desktop	none
☐	Unarchive To...	none

Full Keyboard Access: In windows and dialogs, press Tab to move keyboard focus between:

[✓ Text boxes and lists only]
All controls

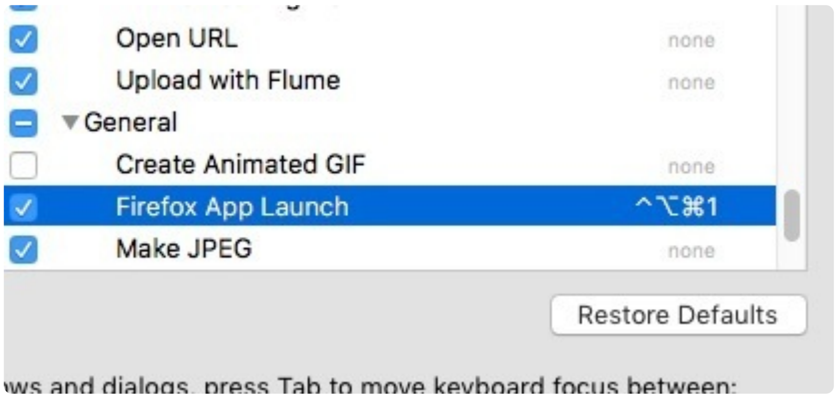
Press Control+F7 to change this setting.

Click on the **Firefox App Launch** service and then click the **Add Shortcut** button.


To change a shortcut, select it, click the key combination, and then type the new keys.

Launchpad & Dock	✓ Add to Reading List	none
Display	✓ Open URL	none
Mission Control	✓ Upload with Flume	none
Keyboard	▾ General	
Input Sources	Create Animated GIF	none
Screen Shots	✓ Firefox App Launch	Add Shortcut
Services	✓ Make JPEG	none
Spotlight	Size in OmniDiskSweeper	none
Accessibility	Unarchive To Current Folder	none
App Shortcuts	Unarchive To Desktop	none
	Unarchive To...	none

The field wants you to type your shortcut, so go ahead and press **command+option+ctrl+1**


✓ Open URL	none
✓ Upload with Flume	none
▾ General	
Create Animated GIF	none
✓ Firefox App Launch	^⌥⌘1
✓ Make JPEG	none

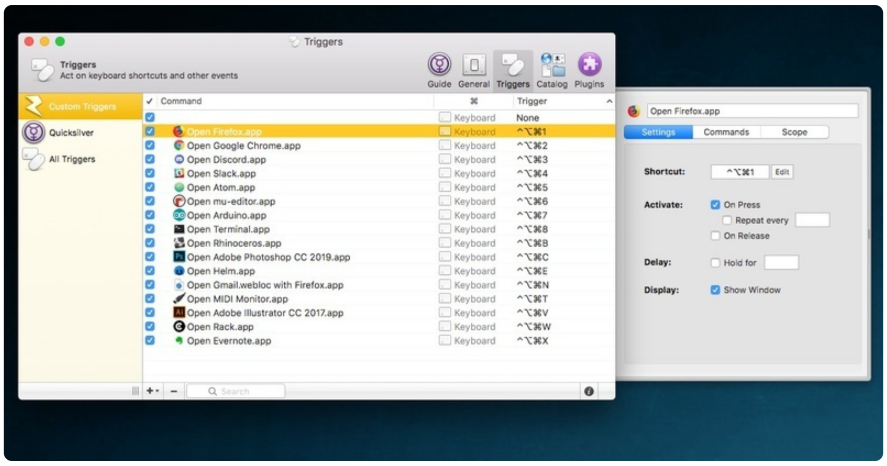
Now, you can close the Keyboard Preference window and test it out! Press **command+option+ctrl+1** and you'll launch or switch to Firefox!

Later, we'll set up the Launch Pad Trellis M4 to send the same four-button shortcut at the press of a single, color-coded Firefox-orange button!

Quicksilver

Quicksilver

Quicksilver is an alternate way to create app launching shortcuts on macs. It is a free, open source app that can be used for all kinds of searching, automation, triggered actions, and more.



✓	Command	⌘	Trigger
✓		Keyboard	None
✓	Open Firefox.app	Keyboard	^⌥⌘1
✓	Open Google Chrome.app	Keyboard	^⌥⌘2
✓	Open Discord.app	Keyboard	^⌥⌘3
✓	Open Slack.app	Keyboard	^⌥⌘4
✓	Open Atom.app	Keyboard	^⌥⌘5
✓	Open mu-editor.app	Keyboard	^⌥⌘6
✓	Open Arduino.app	Keyboard	^⌥⌘7
✓	Open Terminal.app	Keyboard	^⌥⌘8
✓	Open Rhinoceros.app	Keyboard	^⌥⌘B
✓	Open Adobe Photoshop CC 2019.app	Keyboard	^⌥⌘C
✓	Open Helm.app	Keyboard	^⌥⌘E
✓	Open Gmail.webloc with Firefox.app	Keyboard	^⌥⌘N
✓	Open MIDI Monitor.app	Keyboard	^⌥⌘T
✓	Open Adobe Illustrator CC 2017.app	Keyboard	^⌥⌘V
✓	Open Rack.app	Keyboard	^⌥⌘W
✓	Open Evernote.app	Keyboard	^⌥⌘X

Open Firefox.app

Settings | Commands | Scope

Shortcut: ^⌥⌘1 Edit

Activate: [✓] On Press
[ ] Repeat every
[ ] On Release

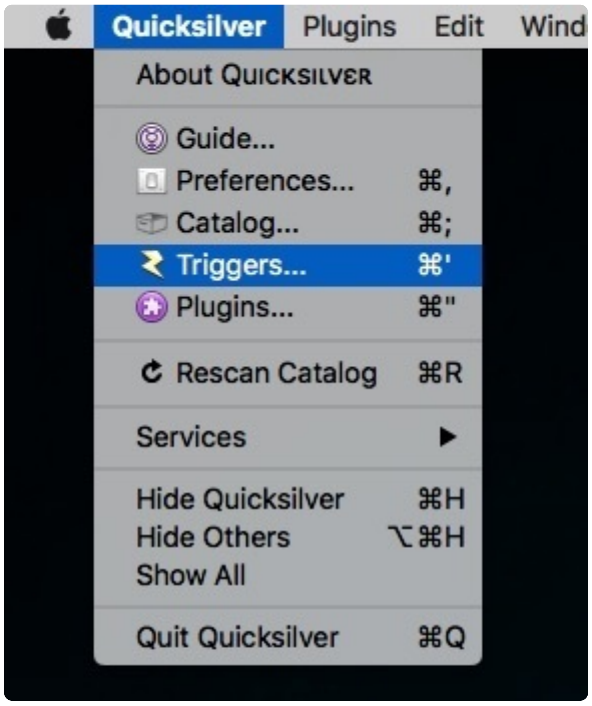
Delay: [ ] Hold for

Display: [✓] Show Window

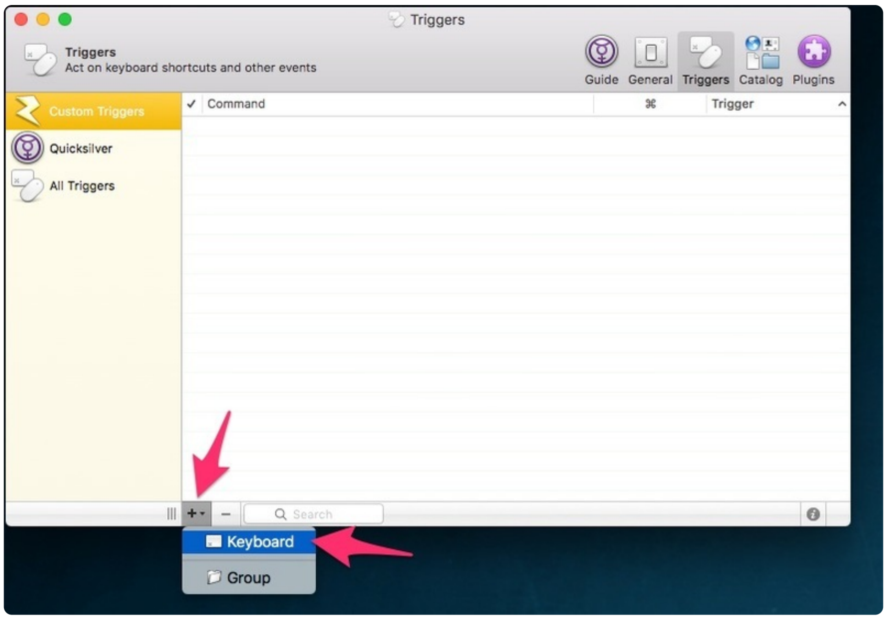
Don't create duplicate sets of keyboard shortcuts to launch apps in both Automator and Quicksilver or you'll create a rip in the space/time continuum.

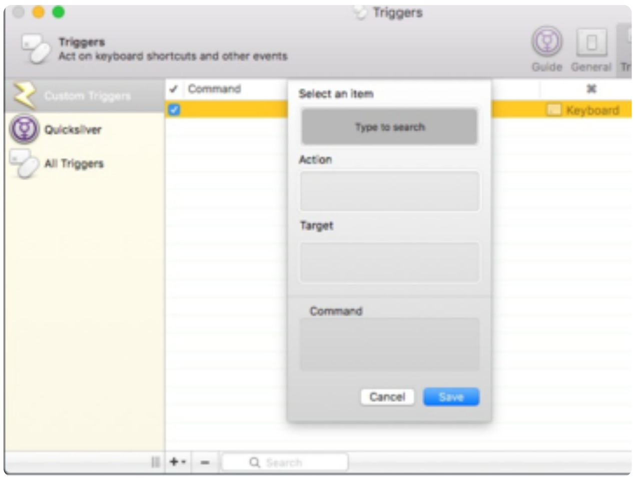
First, download the latest Quicksilver (https://adafru.it/D9w)and install it on your computer. Double-click the Quicksilver icon in your Applications folder to run it.

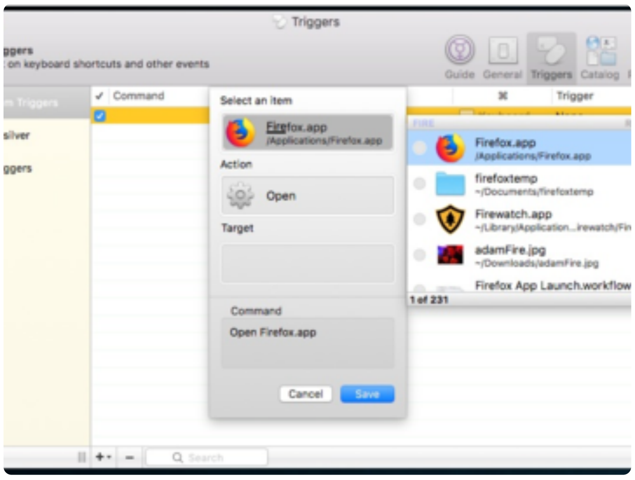
To add app launching actions, click on the **Quicksilver** > **Triggers...** menu item.



In the **Triggers** window, click on the + sign and **Keyboard** item to add a trigger.

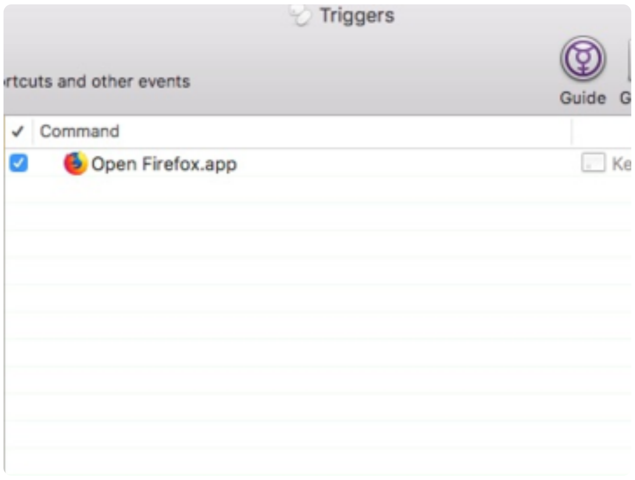

Triggers
Triggers
Act on keyboard shortcuts and other events
Guide
General
Triggers
Catalog
Plugins
Custom Triggers
Quicksilver
All Triggers
Command
⌘
Trigger
Search
Keyboard
Group





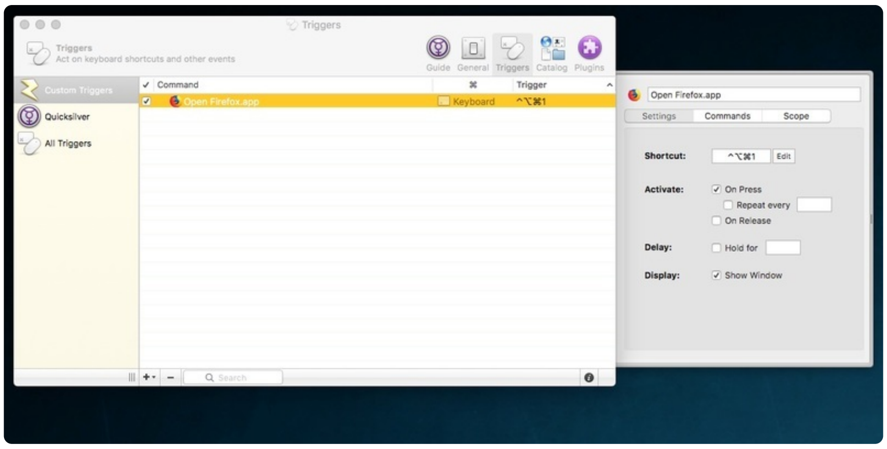
This will pop up the trigger creation window. Click in the top field and start typing "firefox" to have it autofill **Firefox.app**

Click **Save**



Assign Shortcut

Now, we'll add a keyboard shortcut. Double-click the word **None** under the **Trigger** column for the **Open Firefox.app** trigger entry. Then, in the shortcut field, create your shortcut, **command+option+ctrl+1**



✓	Command	⌘	Trigger
✓	Open Firefox.app	Keyboard	^⌥⌘1

Open Firefox.app

Settings | Commands | Scope

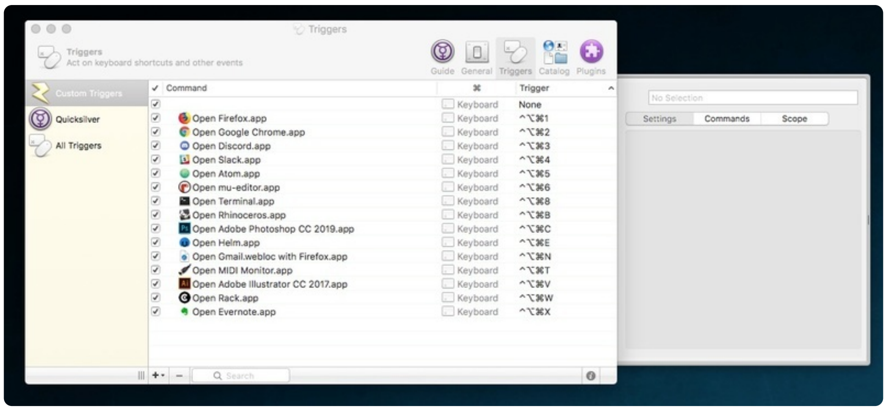
Shortcut: ^⌥⌘1 Edit

Activate: [✓ On Press] Repeat every, On Release

Delay: Hold for

Display: [✓ Show Window]

As you add more of them, you will build up a nice list of apps to launch with shortcuts!



✓	Command	⌘	Trigger
✓		Keyboard	None
✓	Open Firefox.app	Keyboard	^⌥⌘1
✓	Open Google Chrome.app	Keyboard	^⌥⌘2
✓	Open Discord.app	Keyboard	^⌥⌘3
✓	Open Slack.app	Keyboard	^⌥⌘4
✓	Open Atom.app	Keyboard	^⌥⌘5
✓	Open mu-editor.app	Keyboard	^⌥⌘6
✓	Open Terminal.app	Keyboard	^⌥⌘8
✓	Open Rhinoceros.app	Keyboard	^⌥⌘B
✓	Open Adobe Photoshop CC 2019.app	Keyboard	^⌥⌘C
✓	Open Helm.app	Keyboard	^⌥⌘E
✓	Open Gmail.webloc with Firefox.app	Keyboard	^⌥⌘N
✓	Open MIDI Monitor.app	Keyboard	^⌥⌘T
✓	Open Adobe Illustrator CC 2017.app	Keyboard	^⌥⌘V
✓	Open Rack.app	Keyboard	^⌥⌘W
✓	Open Evernote.app	Keyboard	^⌥⌘X

When we set up the CircuitPython code we'll use this list of shortcuts as well as some new ones that don't require an app launching program, such as the media keys and screenshot hotkeys.

Windows App Launching

There are many different ways to launch applications from keyboard shortcuts in Windows. Here are a few to try.

Windows 10 Task Bar

A very quick and easy way to launch apps in Windows 10 is with keyboard shortcuts which correspond to the first ten apps pinned to your taskbar. By holding the Windows key plus any number from 1 - 0 (pretend the '0' is a '10') you can automatically launch that item.

So, place Chrome as the second item on your taskbar and then press **windows+2** to launch. Easy!

This method works in Ubuntu Linux as well! The 'Super' key stands in for the 'Windows' key.

AutoHotkey

AutoHotkey (https://adafru.it/Dao) is a free, Open Source scripting language for Windows that allows you to automate all kinds of tasks on your machine, including setting up app launching keyboard shortcuts.

You can download it here (https://adafru.it/Dap) and then check out this tutorial to get started. (https://adafru.it/Daq)

Once you've installed AutoHotkey, it's a good idea to get familiar with it by doing the basic tutorial. Then, you can create shortcut launch scripts for any application you like. Here's an example script that uses windows+ctrl+alt+w to launch Chrome:

```
#^!w::
Run, Chrome.exe
return
```

Save and launch that script, and now when you set up your Launch Deck Trellis M4 code in CircuitPython you'll specify the **windows+ctrl+alt+w** keycode combo for your Chrome button!

Keyboard Launchpad

From Stardock, the $10 Keyboard Launchpad app (https://adafru.it/Dar) makes it very easy to set up app launching hotkeys.

Code with CircuitPython

If you are new to CircuitPython, we recommend you start with the Welcome to CircuitPython Guide (https://adafru.it/Bid) then come back here.

Adafruit recommends installing and using the Mu editor on your computer to edit CircuitPython files. Mu is available for Windows, macOS, and Linux. You can learn about Mu in this guide (https://adafru.it/ANO).

CircuitPython Preparation

To prepare the NeoTrellis M4 to run the code, follow these steps:

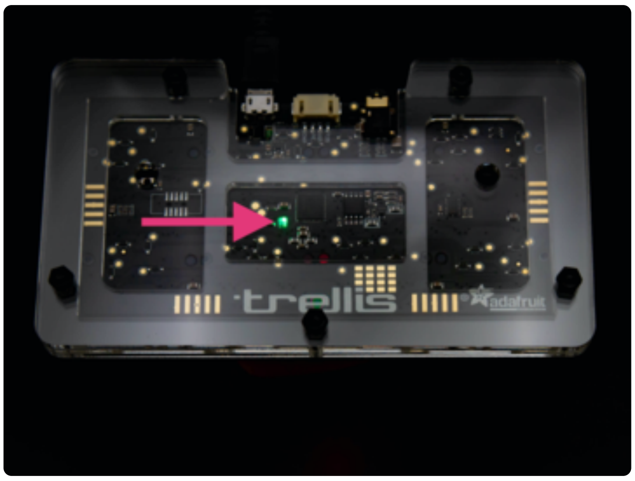
- Update the bootloader for NeoTrellis (https://adafru.it/RAw)from the Trellis M4 guide
- Install the latest version of CircuitPython (at least 4.0.0 Alpha 3) for NeoTrellis M4.

Download the latest version of CircuitPython for this board via CircuitPython.org

https://adafru.it/Em6

Click the link above to download the latest UF2 file.

Download and save it to your desktop (or wherever is handy).



Plug your NeoTrellis M4 Express into your computer using a known-good USB cable.

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the **Reset** button next to the USB connector on your board, and you will see the status DotStar RGB LED turn green. If it turns red, check the USB cable, try another USB port, etc.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

You will see a new disk drive appear called **TRELM4BOOT**.

Drag the **adafruit_circuitpython_etc.uf2** file to **TRELM4BOOT.**

The LED will flash. Then, the **TRELM4BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

- Get the latest CircuitPython library pack (https://adafru.it/zB-) matching your version of CircuitPython and save it onto your hard drive. You will need three **.mpy** files and one folder within the library pack for this guide. Drag the files listed below over into the **/lib** folder on **CIRCUITPY:**
  - **adafruit_trellism4.mpy**
  - **adafruit_hid** folder
  - **neopixel.mpy**
  - **adafruit_matrixkeypad.mpy**

USB keyboards and mice show up on your computer as 'HID' devices, which stands for 'Human Interface Device'

HID Keyboard Basics

This guide page (https://adafru.it/DaD) has a great intro to CircuitPython HID Keyboard.

For even more details, check out the documentation at https://circuitpython.readthedocs.io/projects/hid/en/latest/ (https://adafru.it/B-7) which includes all of the keycodes and media codes you can use.

First, we'll import the **adafruit_hid** library into our program. This will allow us to make calls to send keyboard keys and media keys.

```
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode

kbd = Keyboard(usb_hid.devices)
cc = ConsumerControl(usb_hid.devices)
```

Keyboard Press/Release

Now we can send this command to "type" the letter 'a':

`kbd.press(Keycode.A)`

`kbd.release(Keycode.A)`

This would send a lowercase 'a' to the computer just as if you had typed it yourself. To send a capital 'A', we'd add the shift key to the command like this:

`kbd.press(Keycode.SHIFT, Keycode.A)`

`kbd.release(Keycode.SHIFT, Keycode.A)`

This is pretty cool, since it means we can layer on lots of keys all at the same time, just like you do on your physical keyboard when using keyboard shortcuts!

So, if there's some keyboard shortcut you want to use (or create for yourself in something like Quicksilver or AutoKeys) that is **command+option+ctrl+a** the CircuitPython code would look like this:

`kbd.press(Keycode.GUI, Keycode.ALT, Keycode.CONTROL, Keycode.A)` `kbd.release(Keycode.GUI, Keycode.ALT, Keycode.CONTROL, Keycode.A)`

The adafruit_hid library allows for operating system specific names such as 'Keycode.COMMAND' on macOS which is 'Keycode.WINDOWS' on Windows. Or, you can use the generic 'Keycode.GUI' on any operating system. Same goes for 'ALT/OPTION'

Media Control

There is a second command we'll use when we want to adjust volume, play/pause, skip tracks, and so on with media such as songs and videos. These are often represented on a physical keyboard as icons silkscreened onto the rightmost function keys.

In USB HID speak, these are known as "Consumer Control codes". To play or pause a track we'll use this command:

`cc.send(ConsumerControlCode.PLAY_PAUSE)`

Launch Deck Code

Here's the full code for the Launch Deck. Copy it and then paste it into Mu and save it to your Trellis M4 as **code.py**

Now, you'll need to edit the keycodes sent in the code to the ones you want to use! Just remember to match them up with your app launcher, such as Quicksilver, AutoKeys, or whatever you choose!

```
# SPDX-FileCopyrightText: 2018 John Edgar Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

#  Launch Deck Trellis M4
#  USB HID button box for launching applications, media control, camera switching
and more
#  Use it with your favorite keyboard controlled launcher, such as Quicksilver and
AutoHotkey

import time
import random
import adafruit_trellism4
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode

# Rotation of the trellis. 0 is when the USB is upself.
# The grid coordinates used below require portrait mode of 90 or 270
ROTATION = 270

# the two command types -- MEDIA for ConsumerControlCodes, KEY for Keycodes
# this allows button press to send the correct HID command for the type specified
MEDIA = 1
KEY = 2
# button mappings
# customize these for your desired postitions, colors, and keyboard combos
# specify (button coordinate): (color hex value, command type, command/keycodes)
keymap = {
    (0,0): (0x001100, MEDIA, ConsumerControlCode.PLAY_PAUSE),
    (1,0): (0x110011, MEDIA, ConsumerControlCode.SCAN_PREVIOUS_TRACK),
    (2,0): (0x110011, MEDIA, ConsumerControlCode.SCAN_NEXT_TRACK),
    (3,0): (0x000033, MEDIA, ConsumerControlCode.VOLUME_INCREMENT),

    (0,1): (0x110000, MEDIA, ConsumerControlCode.MUTE),
    # intentional blank button
    # intentional blank button
    (3,1): ((0,0,10), MEDIA, ConsumerControlCode.VOLUME_DECREMENT),

    (0,2): (0x551100, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL,
Keycode.ONE)),
    (1,2): (0x221100, KEY, (Keycode.CONTROL, Keycode.SHIFT, Keycode.TAB)),  # back
cycle tabs
    (2,2): (0x221100, KEY, (Keycode.CONTROL, Keycode.TAB)),  # cycle tabs
    (3,2): (0x333300, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL,
Keycode.TWO)),

    (0,3): (0x001155, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL,
Keycode.THREE)),
    # intentional blank button
    # intentional blank button
    (3,3): (0x330000, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL,
Keycode.FOUR)),

    (0,4): (0x005511, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL,
Keycode.FIVE)),
    (1,4): (0x440000, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL,
Keycode.SIX)),
    # intentional blank button
    (3,4): (0x003300, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL,
Keycode.EIGHT)),

    (0,5): (0x222222, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL, Keycode.W)),
    (1,5): (0x000044, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL, Keycode.E)),
    # intentional blank button
    (3,5): (0x332211, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL, Keycode.T)),

    (0,6): (0x001133, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL, Keycode.C)),
    (1,6): (0x331100, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL, Keycode.V)),
    (2,6): (0x111111, KEY, (Keycode.GUI, Keycode.SHIFT, Keycode.FOUR)),  # screen
shot
    (3,6): (0x110000, KEY, (Keycode.GUI, Keycode.ALT, Keycode.CONTROL, Keycode.N)),

    (0,7): (0x060606, KEY, (Keycode.GUI, Keycode.H)),  # hide front app, all windows
    (1,7): (0x222200, KEY, (Keycode.GUI, Keycode.GRAVE_ACCENT)),  # cycle windows
of app
    (2,7): (0x010001, KEY, (Keycode.GUI, Keycode.SHIFT, Keycode.TAB)),  # cycle
apps backards
    (3,7): (0x010001, KEY, (Keycode.GUI, Keycode.TAB))}  # cycle apps forwards


# Time in seconds to stay lit before sleeping.
TIMEOUT = 90

# Time to take fading out all of the keys.
FADE_TIME = 1
```

```
# Once asleep, how much time to wait between "snores" which fade up and down one
button.
SNORE_PAUSE = 0.5

# Time in seconds to take fading up the snoring LED.
SNORE_UP = 2

# Time in seconds to take fading down the snoring LED.
SNORE_DOWN = 1

TOTAL_SNORE = SNORE_PAUSE + SNORE_UP + SNORE_DOWN

kbd = Keyboard(usb_hid.devices)
cc = ConsumerControl(usb_hid.devices)

trellis = adafruit_trellism4.TrellisM4Express(rotation=ROTATION)
for button in keymap:
    trellis.pixels[button] = keymap[button][0]

current_press = set()
last_press = time.monotonic()
snore_count = -1
while True:
    pressed = set(trellis.pressed_keys)
    now = time.monotonic()
    sleep_time = now - last_press
    sleeping = sleep_time > TIMEOUT
    for down in pressed - current_press:
        if down in keymap and not sleeping:
            print("down", down)
            # Lower the brightness so that we don't draw too much current when we
turn all of
            # the LEDs on.
            trellis.pixels.brightness = 0.2
            trellis.pixels.fill(keymap[down][0])
            if keymap[down][1] == KEY:
                kbd.press(*keymap[down][2])
            else:
                cc.send(keymap[down][2])
            # else if the entry starts with 'l' for layout.write
        last_press = now
    for up in current_press - pressed:
        if up in keymap:
            print("up", up)
            if keymap[up][1] == KEY:
                kbd.release(*keymap[up][2])

    # Reset the LEDs when there was something previously pressed (current_press)
but nothing now
    # (pressed).
    if not pressed and current_press:
        trellis.pixels.brightness = 1
        trellis.pixels.fill((0, 0, 0))
        for button in keymap:
            trellis.pixels[button] = keymap[button][0]

    if not sleeping:
        snore_count = -1
    else:
        sleep_time -= TIMEOUT
        # Fade all out
        if sleep_time < FADE_TIME:
            brightness = (1 - sleep_time / FADE_TIME)
        # Snore by pausing and then fading a random button up and back down.
        else:
            sleep_time -= FADE_TIME
            current_snore = int(sleep_time / TOTAL_SNORE)
            # Detect a new snore and pick a new button
            if current_snore > snore_count:
                button = random.choice(list(keymap.keys()))
                trellis.pixels.fill((0, 0, 0))
                trellis.pixels[button] = keymap[button][0]
                snore_count = current_snore

            sleep_time = sleep_time % TOTAL_SNORE
            if sleep_time < SNORE_PAUSE:
                brightness = 0
            else:
                sleep_time -= SNORE_PAUSE
                if sleep_time < SNORE_UP:
                    brightness = sleep_time / SNORE_UP
                else:
                    sleep_time -= SNORE_UP
                    brightness = 1 - sleep_time / SNORE_DOWN
        trellis.pixels.brightness = brightness
    current_press = pressed
```

Customizing

In order to customize the Launch Deck, you'll want to know a few things about how the Trellis M4 code in CircuitPython works. This guide (https://adafru.it/D2W) is a great place to read up on all the details, but we'll go over a few specifics here.

Single, Unmodified Keystrokes

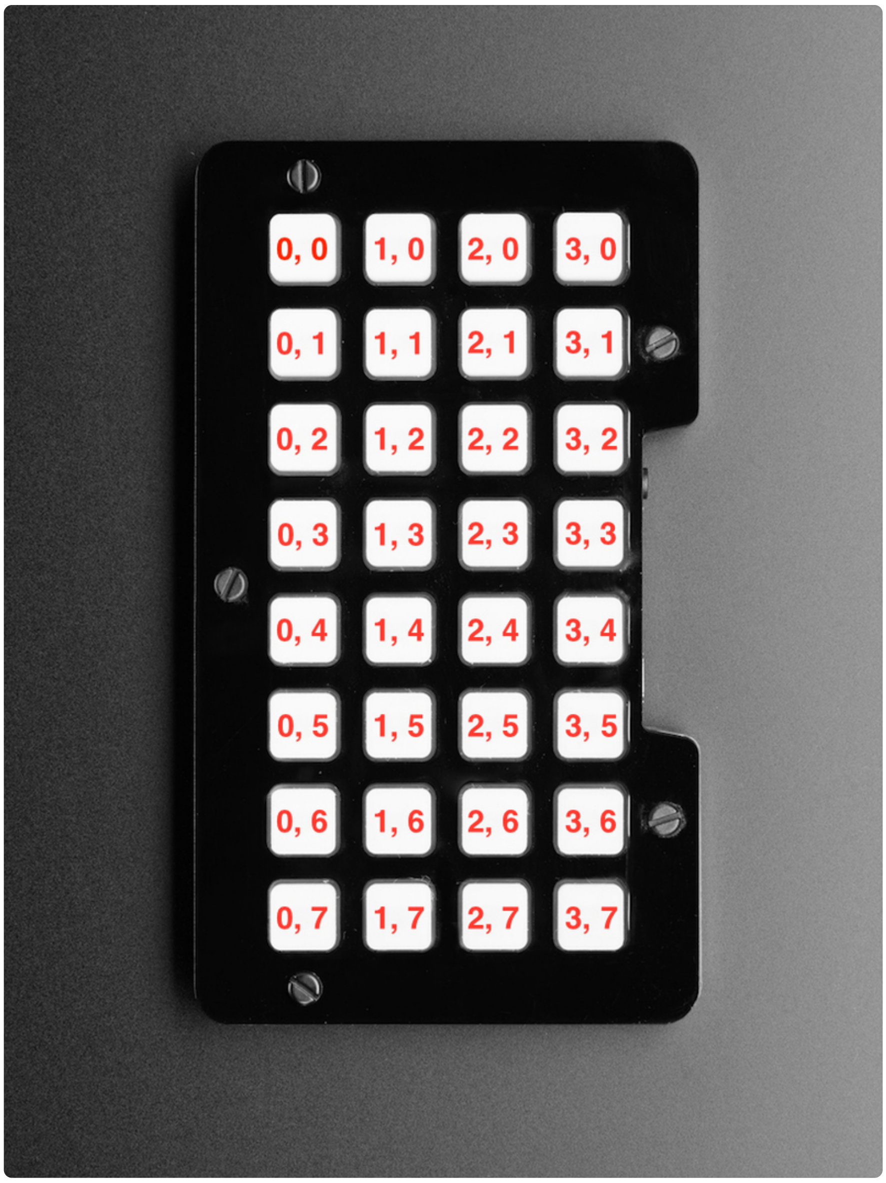
If you want to use the Launch Deck to send a single, unmodified keystroke, we need to enclose it in brackets. This is because of the code's expectation of a tuple.

So, to send a letter **g** with the bottom, left button in portrait orientation, this would be the command (replacing line 61 in the code above):

```
(0, 7): (0x052405, KEY, ([Keycode.G])),
```

Button Coordinates

The Trellis M4 can be oriented in landscape (wide) or portrait (tall) modes. We've set the rotation to portrait mode, so the button grid coordinate system consists of eight rows of four buttons. Here's an example with the rotation set at 90.



Add a Button

Let's say you wanted to add a button to the second row. Currently, there are two unused ones in that row in our example code.

This is the code currently:

```
(0,1): (0x110000, MEDIA, ConsumerControlCode.MUTE),
# intentional blank button
# intentional blank button
(3,1): ((0,0,10), MEDIA, ConsumerControlCode.VOLUME_DECREMENT),
```

Here you can see we've made it easy to drop in two more buttons by commenting out the second and third lines.

So, let's add a shortcut to the second button that is green and can be used to force a reload on a web page in a web browser inside of Windows (you can also add this on a macOS machine by following these instructions (https://adafru.it/Dbe)). That's the **ctrl+F5** key combo.

First, we need to specify the button coordinate: `(1,0):`

Then, we'll specify the green color. You can use an RGB color value such as `(0,50,0)` or a hexidecimal color value. `(0x004400)` You don't want them too bright, so these are dimmer green values.

Next, we specify that this is a keycode, rather than a media key with the `KEY` flag.

Lastly, we list out any keys that will be held down simultaneously, in this case `(Keycode.CONTROL, Keycode.F5)`

This is the full line of code: `(1,1): (0x004400, KEY, (Keycode.CONTROL, Keycode.F5))`

And, here it is inserted among the rest of that row:

```
(0,1): (0x110000, MEDIA, ConsumerControlCode.MUTE),
(1,1): (0x004400, KEY, (Keycode.CONTROL, Keycode.F5)),
# intentional blank button
(3,1): ((0,0,10), MEDIA, ConsumerControlCode.VOLUME_DECREMENT),
```

Edit a Button

If you want to edit the functionality or color of an existing button, simply edit that section. For example, to set the **mute** button to blue:

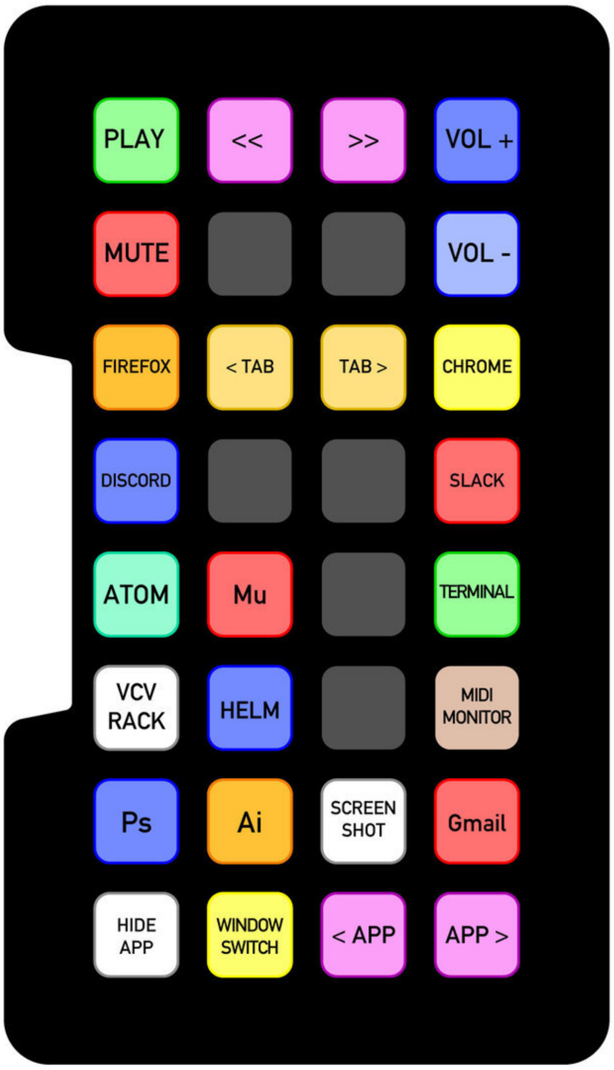
```
(0,1): (0x000011, MEDIA, ConsumerControlCode.MUTE),
```

Button Removal

To remove a button you can simply delete the line of code, or just comment it out with a '#' symbol like this:

```
# (0,1): (0x110000, MEDIA, ConsumerControlCode.MUTE),
(1,1): (0x004400, KEY, (Keycode.CONTROL, Keycode.F5)),
# intentional blank button
(3,1): ((0,0,10), MEDIA, ConsumerControlCode.VOLUME_DECREMENT),
```

That will effectively remove the first button of the second row.



Layout Tips

Here's how I have mine laid out. Here are a few tips for an easy-to-use Launch Deck:

- Color code buttons to their icons
- Group like things together in rows or columns. The entire upper two row set is used for media control here
- Notice the brighter colored **Vol** + above the dimmer **Vol -** which mimics a remote control layout

- Leave some space. Blank buttons help serve as visual landmarks for the important functional buttons nearby