

User Manual

KMBOX

Keyboard & Mouse Box

20250707 V2.0

I. Introduction of KMBOX

KMBOX is a high-performance USB master-slave device controller designed for hardware development and applications. It adopts pure hardware architecture, which can accurately emulate and control USB devices such as keyboard, mouse, etc. at the hardware level, without relying on any additional drivers or DLL injections, which ensures the compatibility and stability of the devices.

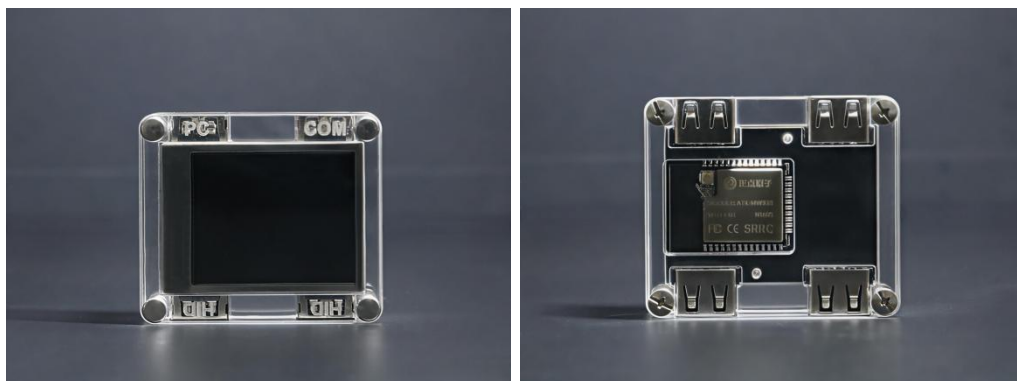
The core advantage of KMBOX is its powerful programmability. KMBOX is equipped with independent CPU, supports scripting, completely detached from the PC system, and can give advanced programmable functions to ordinary keyboards and mice (non-macro keyboards and mice), realizing complex macro operations. Through simple Python calls, users can easily achieve deep customization of the device, instantly upgrading ordinary peripherals into powerful intelligent devices.

KMBOX can be used in a wide range of applications, not only for rapid development of hardware-only physical plug-ins, but also to help users efficiently accomplish a variety of complex tasks. The hardware design is optimized to ensure safe and efficient operation, with excellent anti-detection capabilities to provide users with reliable protection.

KMBOX is also an ideal tool for programming and hardware learning. It uses the easy-to-use Python language as an entry point to help users quickly get started with programming, and gain a deeper understanding of the hardware logic and upper-layer application framework. Whether you are a beginner or an experienced enthusiast, you can use KMBOX to improve your skills through practice, and realize a comprehensive mastery of the underlying hardware to the upper layers of applications.

KMBOX is also the right hand of DIY enthusiasts, whether it is to explore the mystery of the underlying hardware, or to develop innovative applications, it can accompany you all the way to growth, to help you easily realize a variety of ideas and goals, to open the hardware development and application of a new journey.

kmbox physical picture





II. kmbox features

1.Core configuration

Project	Parameter
CPU	ESP32-S3, Xtensa® 32-bit LX6 dual-core processor, computing power up to 600 DMIPS 448 KByte
Memory	512KB SRAM + 8KB RTC Fast SRAM + 8KB RTC Slow SRAM
Memory	16MB (128Mbit) SPI Flash (file system support)
Interface	USB (PC) controlled device, USB (COM) serial port, USB (HID) × 2 device side
Communication	Bluetooth 5.0 (Bluetooth LE), WiFi 802.11 b/g/n
Power Consumption	3.3V/95mA-97mA (typical), 500mA (minimum power supply requirement)

2.USB(PC) Port Characteristics (Connection to Controlled PC)

After the USB(PC) port is connected to a PC, the kmbox will automatically enumerate into a standard keyboard and mouse. kmbox emulates a keyboard and mouse with a super high return rate. Below is a comparison of the key parameters of the kmbox emulated keyboard and a normal keyboard:

Comparison of the performance of KMBOX emulated keyboard and normal keyboard

Parameters	Kmbox Keyboard	Regular keyboard
Return rate	1ms (1000 times/sec)	10ms (100 cycles/sec)
Punchless Keys	10 keys	6 keys
Endpoint Length	64Byte	8Byte

1. The return rate is the time interval at which the keyboard is polled by the host.

The shorter the interval is, the more frequent the reports are, and the faster the keyboard's hardware is running. Conventional keyboards are 10ms, 100 reports per second. kmbox is 1ms, 1000 reports per second, 10 times faster than conventional keyboards. kmbox is 1ms, 1000 reports per second, which is ten times faster than a regular keyboard.

2. The report format is the number of punchless keys that can be sent in a packet of data.

Conventional keyboard adopts 1+6 mode, which supports 6 keys without punch. KMBOX supports 1+10 mode to realize 10 keys without punch. (PS: Theoretically, it can be 108 keys, but considering that human beings have only 10 fingers, 10 keys are already fully satisfied for daily use).

3.endpoint length is the size of the data hardware buffer.

It determines how many bytes the keyboard can transmit at one time. Normal keyboard is 8byte, Kmboxbuffer is 8 times of the regular keyboard. Kmbox buffer is 8 times bigger than the regular keyboard, and it can support more extensions.

As a simple example, suppose the keyboard inputs ten numbers "1234567890".

Conventional Keyboard :First packet 123456-->10ms after host reception-->Second packet 7890-->10ms after host reception-->End

Kmbox keyboard :First packet 1234567890-->1ms after host receive-->end

As you can see, for the same operation, normal keyboard needs 20ms, kmbox only needs 1ms.

Comparison of KMBOX Analog Mouse and Normal Mouse Performance

Parameter	Kmbox Mouse	Regular Mouse
Return Rate	1ms (1000 times/sec)	10ms (100 times/sec)
Number of buttons	8 keys (customizable)	3 keys
X coordinate range Y coordinate range	-32767 ~ 32767 (customizable)	-127 ~ 127
Roller Range	-127 ~ 127 (can be customized)	-127 ~ 127
Report Length	6 bytes (customizable)	4 bytes

1. Return rate:

The kmbox return speed is 10 times faster than normal mouse. So the response is more sensitive and rapid, lower delay.

2.Number of buttons:

In addition to the regular left, center and right buttons, kmbox also has 5 extra side buttons by default. Mouse without side buttons will have 5 side buttons after kmbox is connected. (PS: the number of buttons can be customized)

3. XY coordinate range:

The XY coordinate range of kmbox is 256 times of normal mouse.

Take an example:

The mouse moves from the lower right corner of the screen (1920x1080) to the upper left corner of the screen (0,0). The maximum value of X that a normal mouse can move at one time is 127, so it takes $1920/127=15.118$ to move 1920 units, rounded to the nearest 16 times. The time interval of one report is 10ms, so 16 times need $16*10\text{ms}=160\text{ms}$. kmbox XY coordinate range is plus or minus 32767, so only 1 packet of data is needed to move 1920 units. That is to say, 1ms is enough.

3. USB (COM) Function (Connection to Control PC)

- 1) As a communication interface, it connects to the controlling computer to control the controlled computer. The controlling computer sends commands to KMBOX through the serial port to control the keyboard and mouse operation of the controlled computer.
- 2) Device connection and data transmission, can be connected to other devices, such as sensors, controllers, etc., to realize data transmission and interaction.
- 3) Firmware upgrade, use the upgrade tool to update the firmware of KMBOX.
- 4) During development and debugging, send commands, configure parameters or receive debugging information to KMBOX through the serial port.

4. USB (HID) function (connecting keyboard and mouse)

Two **USB (HID)** ports are used for connecting the taken over USB devices. Currently, kmbox supports USB keyboard and mouse as default peripherals. That is to say, kmbox can get all the data of keyboard and mouse through USB(HID) port in real time, and in the middle of these underlying data processing, and then the processed data will be transmitted to the PC through USB(PC) port (or Bluetooth), so that the ordinary keyboard and mouse also have programmable macro functions, so as to realize any key change, any reorganization, any shielding of the keys, any logical judgment, keyboard point, mouse point and so on. The programmable macros can be used to change keys, reorganize keys, block keys, make logical judgments, connect keyboard, connect mouse, etc. See kmbox tutorial for details.

5. kmbox software features

- 1) kmbox built-in python interpreter. Support Python programming language.
- 2) Kmbox built-in keyboard and mouse control module (km) only need a simple API call to realize the powerful keyboard macro mouse macro and other functions.
- 3) Kmbox supports conditional judgment, variables, loops, multi-threaded python syntax.
- 4) Kmbox supports communication with the host computer. You can communicate with Kmbox through the serial port. For example, the host computer to find the map to find the color and so on.
- 5) Kmbox is currently compatible with all window and Linux hosts, no drive plugged in can be used.
- 6) Kmbox supports file system management, you can store multiple scripts on the board, call at will.

7) Kmbox supports firmware update. No need for other tools, directly through the serial port to upgrade the firmware.

6. Kmbox Typical Application

The core of Kmbox is to control the keyboard and mouse data at will. So as long as the keyboard and mouse can realize the kmbox can be realized. Please see the following examples

1. Intelligent auxiliary keyboard macros

- (1) Keystroke macro (click me to see the video effect)
- (2) [automatic shouting macro \(click me to see the video effect\)](#)
- (3) Keyboard change macro (click me to see the video effect)
- (4)

2. Intelligent auxiliary mouse macro

- (1) [Chicken pressure macro \(click me to see the video effect\)](#)
- (2) [Left click macro \(click me to see the video effect\)](#)
- (3) [Mouse drawing macro \(click me to see the video effect\)](#)
- (4) [Mouse precision control macro \(click me to see video effect\)](#)
- (5)

3. DIY extension application

- (1) [USB wired keyboard and mouse Bluetooth \(click me to see the video effect\)](#)
- (2) [change USB gamepad bluetooth to play King of Glory \(click me to see video effect\)](#)
- (3) [wireless mouse to control the cart](#)
- (4) WIFI Probe (link)
- (5) Weather Forecast (link)
- (6) IoT switch control (link)
- (7)

For more features, users are welcome to explore by themselves.

7. Kmbox Features

1. Security

For computer kmbox is a clean standard keyboard and mouse. Standard HID device, does not need any driver, the computer automatically recognizes, not software macros, driver macros can be compared, from the source to eliminate the risk of blocking and stealing.

2. Efficient

Independent dual-core CPU, 240Mhz ultra-high frequency. Strong data processing ability and efficient script execution speed. It does not take up the CPU time of the PC host. Its performance is

equivalent to 2.4G main frequency dual-core CPU to take 10% of the performance dedicated to the keyboard and mouse.

3. Simple

Kmbox uses flexible python3 scripts. Built-in powerful km module. Supports conditional judgment, variables, loops, multi-threading. Can easily write hardware plug-ins through simple API calls. python is recognized as a language that can be read.

4. Utility

KMBOX is a device that greatly expands the functionality of ordinary keyboards and mice. It gives the keyboard and mouse powerful customization capabilities, including key modification, auto click, macro programming and other functions. Whether it's mouse clicking, keyboard clicking, or complex mouse and keyboard macros, KMBOX can realize them all easily. KMBOX also supports advanced gaming operations, such as gun pressure and automatic monster swiping in "chicken" games, as well as triggering multiple skills with a single click. In addition, KMBOX can also transform ordinary keyboards and mice into Bluetooth devices, realizing wireless connection with computers, cell phones and other devices. With KMBOX, users can fully control all keyboard and mouse data, greatly enhancing work efficiency and gaming experience.

5. High speed

kmbox can directly 10 times improve the hardware performance of keyboard and mouse. Conventional keyboards and mice have a 10ms reporting interval, 100 times per second, while Kmbox has a 1ms reporting interval, 1000 times per second. After connecting kmbox, you can make your keyboard and mouse hardware overclocking, always keep 10 times speed lead.

6. Powerful

KMBOX comes with 16MB of storage space, this ample capacity allows it to easily store complex logic code and configuration files, while supporting efficient file system management. This makes KMBOX the perfect tool for learning USB technology, Python programming, and hardware and software development. Its powerful expansion capabilities allow you to externalize a variety of peripheral modules to achieve unlimited functionality expansion to meet your DIY needs.

III. Kmbox basic tutorial

This chapter introduces some basic usage of kmbox.

1. Unpacking

The list of accessories is as follows:

Material Name	Quantity	Remarks
Kmbox equipment	1	
USB Male-to-Male Cable	2	Length 1.5 meters



2. Connecting to the computer

First, connect one end of the USB extension cable to the USB1 port of the kmbox. Connect one end of the USB extension cable to the USB1 port of the kmbox and the other end to the USB port of your computer. Wait for a few moments. You will see these in the device manager:

Kmbox will appear 4 hardware devices by default after connecting to PC. As shown in the picture above.

1. Serial port (important)

The communication between Kmbox and the host computer is done through the serial port. If there is no serial port, please install the serial port driver. Serial driver is used for debugging code and downloading scripts. If you want to play programming and debugging, you must need to use it. After installing the serial driver, you can use any serial debugging software to log into the board.

Driver download: <https://github.com/SDRRADIO001/Keyboard-Mouse-Box>

2. A keyboard (auto-generated)

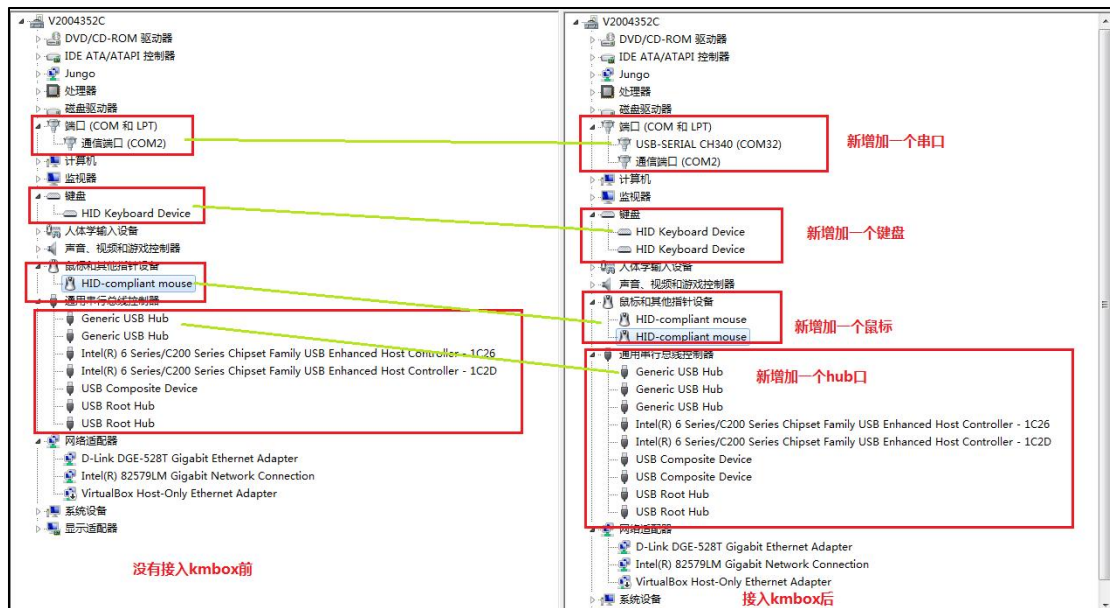
This keyboard is generated automatically by Kmbox simulation. It is a real physical device.

3. A mouse (automatically generated)

This mouse is also a mouse generated by kmbox auto enumeration. It is a real physical device.

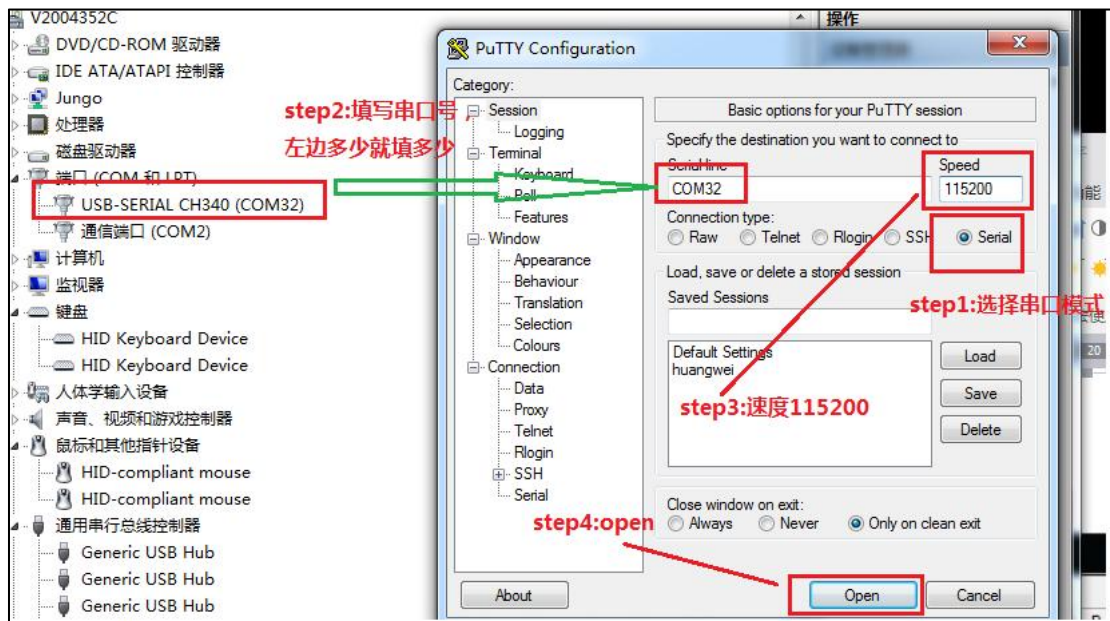
4. A HUB hub

Hub hub is used internally by kmbox for keyboard and mouse data channel and board debug transmission upgrade.



3. Login kmbox

Any serial tools can be used to login kmbox, common serial tools are SecureCRT, uploader, serial debugging assistant, etc. Here we take putty as an example. Here we take putty as an example to introduce a simple development process. Other tools are similar. Double click Putty, fill in the following steps, and then click open.



After clicking Open, you can type "Enter", and the symbol ">>>>" will appear, which means the communication with the board is successful. You can type "help()" and enter to see the help information inside the board. As shown in the following picture.

```

>>>
>>>
>>> km.help()
*****
欢迎使用km模块帮助系统。如果你对km模块的哪个函数不清楚，
请直接使用km.xxx('help')查看该函数的使用方法和返回值。
例如：isdown函数的使用方法和返回值查询：
输入km.isdown('help') 回车即可

想知道km包含哪些函数，请输入km.按键盘的Tab键。
*****
>>> 

```

Here is the python environment inside kmbox. You can write any algorithm and function you want according to python syntax.

Press tab to see what modules are currently available

```

>>>
__class__      __name__      Pin          SPI
bt             device       gc           km
math          os           sys          time
uos           bdev        TFT          sysfont
spi           tft         model        btl
>>> |

```

To see what functions are inside the module, for example the KM module, type "km." and press tab, the following are the functions contained in the KM module

```

>>>
>>> km.
__class__      MAC          Screen      baud
baud_index     catch_kb     catch_ml    catch_mm
catch_mr       catch_ms1    catch_ms2   click
debug          delay        down        freq
getint         getpos       help        init
isdown         keyboard     left        lock_ml
lock_mm        lock_mr      lock_ms1    lock_ms2
lock_mx        lock_my      mask        media
middle         mode         mouse       move
moveAuto       moveto       multidown   multipress
multiup        play         press       reboot
record         remap        rgb         rgb_free
right          rng          setpos      sidel
side2          side3        side4       side5
string         table        trace_debug trace_enable
trace_type     up           version     vertime
wheel          zero
>>> |

```

To find out what each function does, you can call the 'help' parameter function, which prints help information about the function. For example, for the press function, you can type

km.press('help') and enter, and get the following information

```
>>> km.press('help')
软件强制单击指定按键:
press() 包括按下和松开两个动作, 例如单击一次键盘a写法如下:
一: km.press('a') 输入参数是字符串a
二: km.press(4) 输入参数是a的键值
press支持模拟手动操作, 即控制一次按键按下的时间, press可以额外增加一个或者两个参数
press(4,50)-----两个参数, 表示单击a键, a键按下时间50ms
press(4,50,100)--三个参数, 表示单击a键, a键按下时间50-100ms的随机值
PS: 推荐使用键值类型, 效率更高, 速度更快, 手动按键正常人速度约为8次每秒。如需模拟请设置合适的按下参数
>>>
```

4. Keyboard Macro Programming

1) a little knowledge of the keyboard

How the computer recognizes the keys of the keyboard or mouse. In a USB system, all data communication is initiated by the host computer (PC). For example, a key is pressed on the keyboard. It is not the keyboard that actively sends the keystroke value to the host. Rather, the host initiates coming over and reading which keys were pressed by the keyboard. This is true for all USB devices. The communication between the host and the keyboard is similar to this pattern.

Host: keyboard do you have keys?

Keyboard: No!

After a while (Host waits for a polling time)

Host: keyboard do you have keys?

Keyboard: No!

After a while (host waits for a polling time)

Host: Do you have keys for the keyboard?

Keyboard: I pressed the keys 123456789

.....

As you can see from the above pattern, the USB device belongs to the slave device and the PC is the master device. The master and slave devices communicate with each other using a question and answer method. So when does the host read the keyboard data? How often is it read? This value is defined in the configuration descriptor of the USB device. The above "After a while" corresponds to this polling time. Usually it is 8ms-10ms for low-speed devices like keyboard and mouse (for more information, please check USB protocol documentation, HID documentation). That is 100 to 125 reads per second by the host. The smaller the value, the more diligent the host reads the data, the more responsive. Hi-Speed USB devices are read once in 125us. But not the smaller the better, the keyboard and mouse are physical devices, people can not do a second to press the keyboard 50 times or 50 times the mouse, the host is too fast reading is a waste of host resources. In fact, to measure the performance of a keyboard should mainly include the following two factors:

- **The rate of return**

Return rate is also the keyboard polling time interval. The shorter the interval, the faster the keyboard's hardware runs. Conventional keyboard is 10ms, kmbox is 1ms, the performance is 10 times faster than the conventional keyboard.

- **Report length**

Conventional keyboard report is 6 keys without punch, if you want to send 123456789 press, you need to send in two packets. The first package 123456, the second package 789. Kmbox currently uses 10 keys without punch, sending 123456789 can be sent in one packet, saving time.

2) How Kmbox handles keyboard data

The relationship between kmbox and the host computer and keyboard. kmbox is strung between the computer and the USB device.

The end that connects to the computer is a USB device and the Kmbox needs to emulate standard USB devices. By factory default, the kmbox emulates a keyboard and a mouse, the parameters of the keyboard emulated by the kmbox are listed in Table 1. The USB port on the right is connected to a USB peripheral. The right USB port is connected to a USB peripheral, so the right side of the kmbox plays the role of a USB host. As a host, kmbox recognizes HID type keyboard and mouse devices by default. kmbox will access and configure the peripherals according to their descriptors.

All raw data from the keyboard and mouse passes through the kmbox before being sent to the computer. Scripts are run inside the kmbox to process the raw keyboard and mouse data in real time and re-modify the packages according to the logic of the user scripts. Then it is sent to the host computer. For the host PC. He receives the macro processed data, and it is the real HID data. The logic running inside the script is equivalent to the logic of the external keyboard and mouse. That is, even if the external keyboard and mouse do not have macro functions, kmbox is equivalent to the keyboard and mouse have macro functions.

As an example, kmbox how to make ordinary mouse to realize the function of chicken pressure gun. The left mouse button is pressed to fire. If kmbox does not handle it. Then the PC host receives the left mouse button press to fire. This is a normal process. However, if you connect the kmbox, write a script inside the kmbox (if the left mouse button is pressed, then the mouse will move down). These two packets are sent to the host, the fire is realized by us and the mouse down is realized by the kmbox script. The whole process of kmbox help us automatically mouse down to press the gun, reducing the difficulty of operation. kmbox has encapsulated these functions, as long as you learn to call on it.

3) Kmbox function introduction

The following functions can also be used in the code by typing km.xxx('help'). xxx is the name of the function you want to query.

table function

call method: km.table()

Remarks: This function is used to display the corresponding table of common keyboard key names and key values.

Because the PC recognizes the key only recognizes the key value of the HID data, this function is used to query the physical keys on the keyboard corresponding to the decimal value, so if you need to query which key corresponds to which key value. You can call `km.table()` directly to view it. The return is as follows:

```
kmbox
>>> km.table()

按键名称      按键键值      按键名称      按键键值
space         44           w            26
a             4            s            22
d             7            1            30
2            31           3            32
4            33           5            34
6            35           7            36
8            37           9            38
0            39           b            5
c            6            .            53
e            8            f            9
g            10           h            11
i            12           j            13
k            14           l            15
m            16           n            17
o            18           p            19
q            20           r            21
t            23           u            24
v            25           x            27
y            28           z            29
tab           43           back         42
enter         40           del          76
esc           41           f1           58
f2            59           f3           60
f4            61           f5           62
f6            63           f7           64
f8            65           f9           66
f10           67           f11          68
f12           69           up           82
down          81           left         80
right         79           -            45
=            46           insert       73
home          74           [            47
]            48           \            49
end           77           caps         57
:            51           '            52
,            54           .            55
/            56           print        70
scroll        71           pause        72
pgup          75           f13          104
f14           105          f15          106
f16           107          f17          108
f18           109          f19          110
f20           111          pgdown       78
aplcat        101          ctr-l        224
shift-l       225          alt-l        226
gui-l         227          ctr-r        228
shift-r       229          alt-r        230
gui-r         231          help         254
>>>
```

As shown in the figure, the space key "space", the corresponding code value is 44, and the enter key "enter", the corresponding code value is 40. You don't need to memorize the key values, just look them up when you use them. If the key you want is not in this table, it is also supported. For example, the key value of the power key is 0x66, but we don't use it very often, so we don't write it in this table.

down function

call method: `km.down(value)`

The `down(value)` function is used to control the pressing of a keyboard key, which is equivalent to manually pressing a physical key. `value` can be either numeric (e.g. a key value of 4) or string (the 'a' string). It is recommended that you use a numeric type. It is more efficient. See the `km.table()` function for the value. `down()` is usually used in conjunction with `up()`.

For example, if you want the a key to remain pressed, you can write it in two ways:

1. `km.down('a')`---- directly enter the name of a, which is the key name in `table()`.

```
>>> km.down('a')
>>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Note that after entering this command above will always print a, to stop please press ctrl+C.

2. `km.down(4)`----直接输入a的键值(4), 这个键值就是`table()`中的键键键值.

```
>>> km.down(4)
>>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

`down` function is to control the key press with script, even if you physically did not press, as long as the call `km.down('a')`, then the PC will also recognize that a key press.

PS: If you are operating on the console, you can press ctrl+c to terminate the current script.

The Down function is described below:

```
>>> km.down('help')
软件强制指定按键一直按下:
例如想让a键按着不松可以有以下两种写法
一:km.down('a') 输入参数是字符串a
二:km.down(4)   输入参数是a的键值
PS:推荐使用键值类型,效率更高,速度更快
>>>
```

multidown function

The `multidown` function is the same as the `down` function. The only difference is that the `multidown` function supports multiple keystrokes at once. Up to 10 keys can be pressed at the same time (up to 10 parameters).

For example, if you want to call the task manager, the shortcut is `ctr-l` (left ctrl) + `alt-l` (left alt) + delete. You can write it like this:

Method 1: `km.multidown('ctr-l','alt-l','del')` ---- pass the characters directly

Method 2: `km.multidown(224,226,76)` ---- pass the key directly

Note that multi down is usually used in conjunction with multiup

up function

call method: `km.up(value)`

up(value) is used to set the keyboard to pop up the specified key, which is equivalent to manually releasing the physical key value can be either numeric (e.g. space 44) or string (the 'space' string). It is recommended to use the numeric type, as it is more efficient. See km.table() for the value, which is usually used in conjunction with the down() function.

For example, to release the space bar can be written in the following two ways:

1. km.up('space') ---- directly enter the name of the space bar, this name is the key name in table()

```
>>> km.up('space')
>>>
```

2. Km.up(44) ---- directly input the key value of the space bar, the key value is the key value in table().

```
>>> km.up(44)
>>>
```

Km.up(44) after the space bar pop up, the cursor does not continue to move.

PS: This function is equivalent to the release of the specified key, even if you physically hold down the space does not release, call the function after the space will also be released.

The Up function is described below:

```
>>> km.up('help')
软件强制松开指定按键:
  例如你按着键盘上的a不松, 通过软件让a松开。可以有以下两种写法
  一: km.up('a') 输入参数是字符串a
  二: km.up(4)   输入参数是a的键值
PS: 推荐使用键值类型, 效率更高, 速度更快
>>>
```

The multiup function

The multiup function is the same as the up function. The only difference is that the multiup function supports lifting multiple keys at once. Up to 10 keys can be lifted at the same time (up to 10 parameters).

For example, if you want to unlock the task manager, the shortcut is ctr-l(left ctrl)+alt-l(left alt)+delete to lift the keys. You can write it like this:

Method 1: km.multidup('ctr-l','alt-l','del') ---- pass the characters directly

Method 2: km.multiup(224,226,76) ---- pass the key directly.

Note that multi down is usually used in conjunction with multiup.

press function

press(value) function is used to press a key once, which is equivalent to manually clicking a physical key. value can be a number or a string (numeric type is more efficient), please refer to km.table() function for the value. This is equivalent to calling down() and then up().

For example, want to press the 'a' key on the keyboard. It can be written in the following two ways:

1. km.press('a') ---- directly enter the name of a, which is the key name in table()

```
>>> km.press('a')
>>> a
```

2. `Km.press(4)` ---- directly input the key value of a, this key value is the key value in `table()`.

```
>>> km.press(4)
>>> a
```

`press` function can also take two to three parameters, used to specify the time to press the specified key, you can simulate manual operation. For example, press a key 200ms, you can write:
`km.press(4,200)`

The 200 here means that the a key is pressed for 200ms, mainly used to simulate a long press of 200ms and then release. You can also take three parameters: `km.press(4,80,100)`

Here 80, 100 refers to the duration of a press between 80-100ms random value. It is mainly used for human simulation. If no parameter, `kmbox` will execute a key press as fast as possible. Usually 1ms. It is impossible for a normal person to do this, but with a parameter, it will make your script more human-like. It is possible to pass the behavior detection. Please pay attention to the reasonableness of the time parameter. For details on how to use `press`, please check `km.press('help'):: kmbox.press('help'):: kmbox.press('help')`:

```
>>> km.press('help')
软件强制单击指定按键:
press() 包括按下和松开两个动作, 例如单击一次键盘a写法如下:
一: km.press('a') 输入参数是字符串a
二: km.press(4) 输入参数是a的键值
press支持模拟手动操作, 即控制一次按键按下的时间, press可以额外增加一个或者两个参数
press(4, 50) ----- 两个参数, 表示单击a键, a键按下时间50ms
press(4, 50, 100) -- 三个参数, 表示单击a键, a键按下时间50-100ms的随机值
PS: 推荐使用键值类型, 效率更高, 速度更快,
>>>
```

The multipress Function

The `multipress` function is the same as the `press` function. The only difference is that the `multipress` function supports multiple keystrokes at once. It supports up to 10 keystrokes at the same time (up to 10 parameters).

For example, if you want to call the task manager, the shortcut is `ctr-l(left ctrl)+alt-l(left alt)+delete`. You can write it like this:

Method 1: `km.multipress('ctr-l','alt-l','del')` ---- pass the characters directly

Method 2: `km.multipress(224,226,76)` ---- passes the key directly.

string Function

The `string` function is used to simulate keyboard tapping to enter a string of characters, for example, we want to directly type "hellow word". The easiest way to do this is to call the `km.string('hellow word')` function, or if you don't mind the hassle, you can also call `h,e,l,l,o,w... km.press()` function one by one. This is actually what the `string` function does internally. Here's a screenshot of it in action:

```
>>>
>>> km.string('hellow word')
>>> hellow word
```


Note: string's strings support numbers from 0-9 and all upper and lower case letters and characters in the main keyboard area. Keyboards are not supported. This is because the keypad will conflict with the main keyboard. Be careful when using escape characters (\,').

The String function takes a string as its argument and supports escaping characters (\r(Enter) \n(Space) \t(Tab))for example, if you need to make an auto-login script. For example, if you need to make an automatic login script, you usually enter the account number, tab, password, and enter. You can use the string function to do this in one step, for example:

Account: KmBox

Password: www.clion.top

You can write it like this: km.string('KmBox\\twww.clion.top\\r')

Where \t and \r represent Tab and Return. (Note that the escape character needs an extra slash)

isdown function (need to connect the detected keyboard to kmbox)

isdown(value) function is used to check whether the specified value key is pressed. value can be a numeric type or a string type (numeric type is more efficient)please refer to km.table() function for the value. This function is mainly used in logical judgment to determine whether a specified key has been pressed or not.

For example, if you want to check if the 'a' key on the keyboard is pressed. There are two ways to write this function:

1. km.isdown('a') ---- directly enter the name of a, which is the key name in table().
2. km.isdown(4) ---- directly input the key value of a, this key value is table() in the key value

There are four return values:

- 0: not pressed -- physical software are not pressed
- 1: physically pressed - physically pressed but not pressed by the software
- 2: software pressed - software pressed but physical did not press
- 3: Physical and software both pressed - software and physical both pressed

Let's make a convention here: a real keyboard press is called a physical press, and a software script press is called a software press. So the return value of the isdown function has the above four is very good to understand.

The isdown function is described below:

```
>>> km.isdown('help')
用于查询键盘按键是否按下。例如要查询空格键是否按下可以这样写:
  isdown('space') 返回值.0:没有按下 1: 物理按下 2: 软件按下 3: 物理软件均按下
  isdown('44')    返回值.0:没有按下 1: 物理按下 2: 软件按下 3: 物理软件均按下
PS:isdown只管调用的瞬间按键是否按下。如果需要捕获处理所有时间的按键，请使用mask和catch处理
>>> █
```

Note: The isdown function detects what is in the HID report. If your keyboard is not plugged into the kmbox at all, the kmbox will not detect anything. Don't be so naive as to think that your computer will tell kmbox that it is working, write the driver yourself. Currently, kmbox detection is limited to keyboards and mice connected to kmbox.

getint function

This function is the same as the table function, it is used to query the value of the key corresponding to the keyboard string. For example, if we need to know the value of "esc" key on , we can use table() to find the value of "esc". Another way is to use the getint function. For example:

```
>>>
>>> km.getint('esc')
41
>>>
```

If you have a table, there is no need to use getint, right, there must be a reason. Let's take a look at the following code

```
Run=getint('f1')
if km.isdown(Run):
    km.string("welcom")
else:
    km.string("good bye")
```

As you can see above, we give Run the key value of the f1 key, and if Run is pressed, we print welcome. If Run is pressed, we print Welcome. If Run is released, we print Goodbye. If you want to change the shortcut to F10 instead of F1, the script changes only the string in getint and nothing else. If you don't have getint, then you need to change it once for each Run. So the advantage of this is that you can set the variables when you write the script, and it's easier to expose the interface. Convenient for the generality of the script.

The getint function is described below:

```
>>> km.getint('help')
用于查询常规按键对应的10进制键值:
例如要返回空格键对应的键值:km.getint('space')
返回值为44。也可以通过table()函数查看常用按键键值对照表
>>>
```

mask mask function (need to connect the masked keyboard to kmbox)

mask function is used to mask or query whether the physical keys of the keyboard are masked. For example, you want to mask the enter key on the keyboard. You can write it as follows: km.mask('enter',1). The first argument is the string enter. See the table() table for details. The second argument is 1 for mask.

```
>>> km.mask('enter',1)
>>>
```

After masking the enter key, the PC won't recognize the enter key on your keyboard, no matter if you press it or release it. This is equivalent to removing the Enter key from your keyboard. The other way to write it is the value of the enter key.

```
>>> km.mask(40,1)
```

Here, 40 is the key value for enter.

You can query the status of the blocked key (pressed or raised) with the isdown() function.

```
>>> km.isdown('enter')
0
>>> km.isdown('enter')
1
>>> 
```

As in the above figure, after blocking the carriage return, press the carriage return without letting go, and query by isdown. The return value is 1, which means it is physically pressed. Note that after masking, although kmbox will not send the key value to PC, it will still receive the key value. This is because when we do keyboard concatenation, we want the physical keystrokes not to affect our script. Let's say you want to do a space concatenation. You want to keep pressing down and up on without letting go of the space, because you're physically holding it down all the time. The moment the scripting software executes up. The moment the scripting software executes up, the keyboard is released. But then it will press down again. So in order to make sure that the script is not affected by the physical keystrokes, you can use the mask function. You can use the mask function to mask the physical keys. This way, the state of the spacebar is entirely up to the software. The physical keystrokes will not interfere with the logic of the script.

If you find that a key doesn't work. The first thing you can do is check to see if the key is masked. Just type: Km.mask

Km.mask (the name of the key or the value of the key).

```
>>> km.mask('enter')
1
>>> 
```

For example, if you have just masked and enter, and you see that the return value is 1, that means it is masked. To unmask a key, change the second parameter of mask to 0.

```
>>> km.mask('enter',0)
>>> km.mask('enter')
0
>>> 
```

After unmasking, query again whether enter is masked or not, the return value is 0, indicating that it is not masked. Press enter again and you can use it normally.

Mask function also has a more advanced use. It is used to set a certain key to capture mode. For example, if you want to know how many times the spacebar was pressed, if you use the normal isdown method, it may not poll fast enough. If you use the normal isdown method, it may not poll fast enough and you may miss detecting the specified key. You can use mask('space',2) to set the space bar to be captured in real time. Make sure you don't miss any keystrokes on . At this point you can use the catch_kb function to extract the key presses.

The Mask function is described below:

```
>>> km.mask('help')
设置和查询物理按键是否被软件屏蔽:
mask(44)      :查询空格键是否被软件屏蔽. 返回值: 0没有被屏蔽 1已被屏蔽 2软件捕获模式
mask('space') :查询空格键是否被软件屏蔽. 返回值: 0没有被屏蔽 1已被屏蔽 2软件捕获模式
mask(44, 0)   :解除空格键被软件屏蔽状态
mask('space', 1) 设置空格键被软件屏蔽
mask(44, 2)    :设置空格被屏蔽且捕获所有空格. 通过调用catch返回被捕获的按键和次数
PS:注意屏蔽了按键后, 物理按键值不会发送给主机. 可以通过isdown查询或者catch查询被屏蔽
按键是否被按下. 如果需要捕获记录所有时刻屏蔽的按键, 推荐使用catch函数
>>> █
```

init Initialization function

The init function is equivalent to restoring the initial state where nothing was done and nothing was pressed on the keyboard or mouse. This is because you may write scripts that mask many keystrokes. You can use this function when you want to release all the keyboard and mouse buttons of a script. Avoid restoring the original masks one by one.

media Multimedia Function Key Functions

media function is used to simulate the role of multi-function keys, your keyboard can not have multi-function keys, kmbox has the same as the external keyboard has. This function is called once to press the corresponding multifunction key. The multi-function keys include the following functions:

- 1) Volume down
- 2) Volume up
- 3) mute/unmute
- 4) pause / play (general player effective)
- 5) open browser
- 6) sleep Hibernates the computer
- 7) shutdown Shut down the computer

For example, if you want to reduce the volume of your computer, you can type `km.media(1)`. See `km.media('help')` for details.



remap function (you need to connect the keyboard to be remapped tkmbx)

The remap function is a fun function. It is commonly known as the remap function. The principle is to press the A key on the keyboard, kmbox will automatically remap the A key to B key. That is to say, it realizes the remap function of A key. With the remap function you can re-customize all the keys on the keyboard. Again, see the help file if you don't know how to do this.

One good thing about this function is that you can change the keys at any time. Often when playing a game, you'll have a key that just doesn't work. With this function you can change the keys to whatever you want.

catch_kb keystroke catching function (requires that the keyboard to be detected is connected to hkbmbx)

The catch_kb function is special in that it needs to be activated by calling [mask\(xxx,2\)](#). The catch_kb function is used to catch all the keys specified by the mask function. First of all, it should be noted. It takes time for the code to run. For example, if you want to check whether the a key is pressed or not. It's natural to think of using isdown('a') up front. Let's say you have a single-threaded script.

If isdown(a):

Print('a is down')

Sleep(10)

This script looks like it's detecting a. But it actually works badly. But it's actually very ineffective, because sleep(10) just stops the code for 10 seconds. No matter how you press a during those 10 seconds, the isdown function won't detect it. Because the code doesn't run into the isdown function. So the catch_kb function was born. As mentioned earlier, the catch_kb function requires a call to the mask function before it can be activated.

First call km.mask('a',2). a key press after the mask function returns will be captured in the catch_kb() function. Now, you count the number of times a is pressed. Then you call the km.catch_kb() function. You will find that each call to return a 4, returned 5 times to 0. (ps: the owner of two keyboards. So the call is blocked with another keyboard input 'a'.

```
>>> km.remap('help')
按键重映射(改键)函数:
重映射函数俗称改键函数。可以将任意按键映射为其他按键
例如你想把a键映射为b键。写法如下:
写法1:km.remap('a','b')即当按a时,等同于按b,a消息不会上传PC
写法2:km.remap(4,5) 其中4是a的键值,5是b的键值
如果要清除所有的重映射设定请调用:km.remap(0)
PS:重映射仅支持ctrl,shift,alt,gui以外的按键。被重映射的按键不可以调用mask屏蔽。
重映射之后的按键不支持获取按键状态。(因为实际的物理键没按下嘛,只是软件设置按下的)
>>>
```

The usage of catch_kb is as follows:

```
>>> km.catch_kb('help')
捕获键盘指定按键:
当调用mask(xxx,2)后,xxx按键将被实时监测,可以截获xxx按键所有按下消息
xxx按键值不会发送给pc,需要自己做处理。catch有两种模式,阻塞模式和非阻塞模式
km.catch_kb():是不带参数的非阻塞模式。没有发生捕捉事件返回0。捕捉到指定按键后返回按键值。
非阻塞模式常用在快速轮询,km.catch_kb(0)也是非阻塞模式
km.catch_kb(1):是带参数的阻塞模式。函数在没有捕捉到指定的按键前是不会返回的。
所以阻塞模式建议在多线程中使用。不然当前线程会被阻塞
PS:一定要调用mask(xxx,2)函数后才能使用catch_kb函数。
>>>
```

catch function also has a use is a key N skills. For example, you write a logic like this in your script. If a key is pressed, then 12345 is pressed in turn. There are many ways to implement this logic.

Method 1: Polling

If km.isdown('a):

Km.string('12345')

Do something ...

The advantage of this method is its simplicity, the disadvantage is that if Do something ... is longer than the time it takes you to press the key. Then the detection of a may not be timely.

Method 2: Multi-threaded catch

First we turn on catching for a, and then we enable a thread dedicated to the catch function. Inside the main thread, we do something ... in the main thread, which ensures that a's catch is not affected by do something. The method of multi-threading will be discussed later.

The above is the keyboard related API functions provided by kmbox. In fact, there are only two states of the keyboard, the key is pressed and the key is pressed up. If you have more needs, please contact me. Welcome to comment!

5. Mouse Macro Programming

Technical parameters of the mouse emulated by kmbox:

	Kmbox-Mouse	Other mice
Return rate	1ms(1000 times/S)	10ms(100 times/S)
Number of buttons	8 buttons (Left, center, right + 5 side buttons. Any key can be customized)	3 keys (left center right)
X coordinate range	-32767 ~32767 (can be customized arbitrarily)	-127~ 127
Y coordinate range	-32767 ~32767 (can be customized)	-127~ 127
Roller Range	-127 ~ 127 (can be customized)	-127~ 127
Report Length	6 bytes (customizable)	4 bytes

Return rate: normal mouse reports 10ms per command. kmbox is 1ms per command, 10 commands can be executed in 10ms.

Number of buttons: ordinary mouse has only three buttons: left, center and right. kmbox has 8 buttons (can be customized).

Coordinate range: ordinary mouse is one byte. kmbox is two bytes. The range is 256 times wider than normal mouse.

Take a simple example. On a 1920x1080 monitor. Ordinary mouse from the lower right corner of the screen (1920,1080) to move to the upper left corner (0,0), in accordance with the maximum step 127 to calculate (a byte of 8bit, 1bit symbol + 7bit (127) data). You need to move the x-axis direction $1920/127=15.118$ times, rounded to the nearest 16 times. A normal mouse needs to send 16 packets of data. Each packet of data interval 10ms, a total of 160ms. kmbox's X-step is 32767, the same movement, only need a packet of data, a transmission time 1ms.

Next start kmbox mouse macro function learning.

left Mouse left button control function

The left function is used for software control of the left mouse button. The left mouse button has two states, down and up. The definition of press is 1, pop up is 0. When no parameter is given, it is to query the state of the left mouse button, and the return value is as follows:

0: Pop up

1: Physical press

2: Software press

3: Physical and software pressed

When the parameter is given it is setting the state of the left mouse button:

0: set the left mouse button pop up

1: Set the left mouse button pressed

```
>>> km.left('help')
用于控制和查询鼠标左键状态:
没有参数时是查询鼠标左键状态, 返回值 0: 松开 1: 物理按下 2: 软件按下 3: 物理软件均按下
有参数时是设置鼠标左键状态, left(1)鼠标左键按下, left(0)鼠标左键松开
>>>
```

Note, if you want to check whether the left mouse button is pressed or not, please connect the mouse to be checked to the kmbox.

right mouse control function

The right function is used to control the right mouse button by the software. The right mouse button has two states, down and up. The definition of press is 1, and pop up is 0. When no parameter is given, it is to query the state of the right mouse button:

0: Bounced

1: physically pressed

2: Software pressed

3: Physical and software pressed

When the parameter is given, the state of the right mouse button is set:

0: Set right mouse button up

1: Set right mouse button down

The usage is shown in the following figure:

```
>>> km.right('help')
用于控制和查询鼠标右键状态:
没有参数时是查询鼠标右键状态, 返回值 0: 松开 1: 物理按下 2: 软件按下 3: 物理软件均按下
有参数时是设置鼠标右键状态, right(1)右键按下, right(0)右键松开
>>>
```

middle mouse center button control function

The middle function is used for software control of the middle mouse button. There are two states of the middle mouse button, press down and pop up. Definition is 1 for press, 0 for pop up, when no parameter is given, it is to query the state of the middle mouse button:

0: pop up

1: physically pressed

2: software pressed

3: both physical and software pressed

When the parameter is given, it sets the state of the middle mouse button:

0: middle mouse button up

1: middle mouse button pressed

The usage is shown as below:

```
>>> km.middle('help')
用于控制和查询鼠标中键状态:
  没有参数时是查询鼠标中键状态, 返回值 0: 松开 1: 物理按下 2: 软件按下 3: 物理软件均按下
  有参数时是设置鼠标中键状态, middle(1)中键按下, middle(0)中键松开
>>>
```

Ps: many mouse middle button is scroll wheel, scroll wheel can be pressed.

click mouse button click function

```
>>> km.click('help')
用于控制鼠标按键点击:
  输入参数有两个, 第一个参数是指定鼠标单击哪个按键
  第二个参数是要点击多少次, 没有第二个参数默认单击1次
km.click(0):单击鼠标左键(效果等于: left(1) left(0))
km.click(1):单击鼠标右键(效果等于: right(1) right(0))
km.click(2):单击鼠标中键(效果等于: middle(1) middle(0))
鼠标左键单击10次写法: km.click(0,10)
>>>
```

The click function is used for software control of mouse button click, click which button is decided by the click parameter. As shown above, the first parameter is to click which button. The second parameter is the number of clicks. Currently, the left click is 0, the center click is 1, and the right click is 2. The click function packs down and up into one. The second argument is the number of clicks. If you want to double-click, then changing the second argument to 2 is a double-click. And so on. Of course, you can also use down and up to simulate a click, with a delay in between, so you can control how long a click takes to press and how long it takes to lift. If you need to simulate human operation, try to use down and up to simulate it. click will be very fast, too fast for human operation.

wheel Mouse wheel control function

wheel function is used to control the mouse wheel. The wheel can be moved up and down, the input parameter is the value of the scrolling unit of the wheel, upward movement for positive, downward movement for negative. For example, the wheel up to move 10 units. And down to move 10 units screenshot as follows:

<pre>>>> km.wheel(10) 滚轮上移动 >>> >>> km.wheel(-10) 滚轮下移动 >>></pre>	<pre>>>> km.wheel('help') 鼠标滚轮设置: wheel(100):滚轮上移动100个单位 wheel(-100):滚轮下移动100个单位 滚轮移动范围为: -127~+127 >>></pre>
---	---

side1, side2, side3, side4, side5 Mouse side button control functions

These functions are used to control the mouse side button state, the input parameter can be with or without, no parameter is to query the side button state:

0: side button pop up

1: side button physically pressed

2: Side button software pressed

3: Side button physical software both pressed

With parameter is to set the side key status:

0: the side button is popped up

1: Side button pressed The screenshots are as follows:

```
>>>
>>> km.side1(1)
>>> km.side1(0)
>>> km.side1()
0
>>>
>>>
>>> km.side2()
0
>>> km.side2(1)
>>> km.side2(0)
>>>
```

Ps: side1 and side2 are generally page turn function, even if your mouse has no side button, it doesn't matter. kmbox can help to realize.

```
>>> km.side1('help')
用于控制和查询鼠标侧键1的状态:
没有参数时是查询侧键1状态, 返回值 0: 松开 1: 物理按下 2: 软件按下 3: 物理软件均按下
有参数时是设置侧键1状态, side1(1)侧键1按下, side1(0)侧键1松开
>>>
```

move mouse movement control function

move is used to control the relative movement of the mouse, the input parameter (x,y) is a numeric type. The range is -32767 to +32767, and x is positive to the right and negative to the left, and y is positive to the down and negative to the up. For example, to move the mouse 10 units to the right and 10 units down, you can write it like this:

```
>>> km.move(10,10)
>>>  
```

move Move is relative. The current coordinate point is used as a reference at all times.

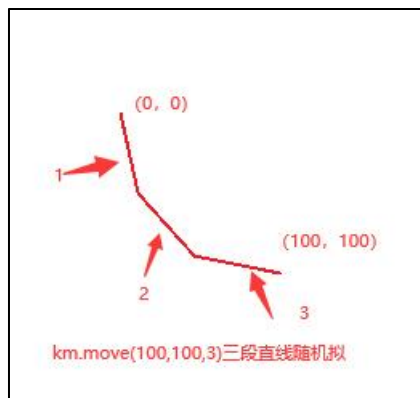
Relative moves are x and y units. If you want to precisely position the move, please use the [moveto](#) function later.

```
>>> km.move('help')
用于鼠标的相对移动, 设置鼠标相对于当前位置偏移(x,y)个单位:
向右和向下移动10个单位: move(10, 10)
向左和向上移动10个单位: move(-10, -10)
x,y允许的范围是-32767~+32767
注意: 如果想精确控制鼠标移动请关闭windows鼠标加速算法
>>>
>>>
```

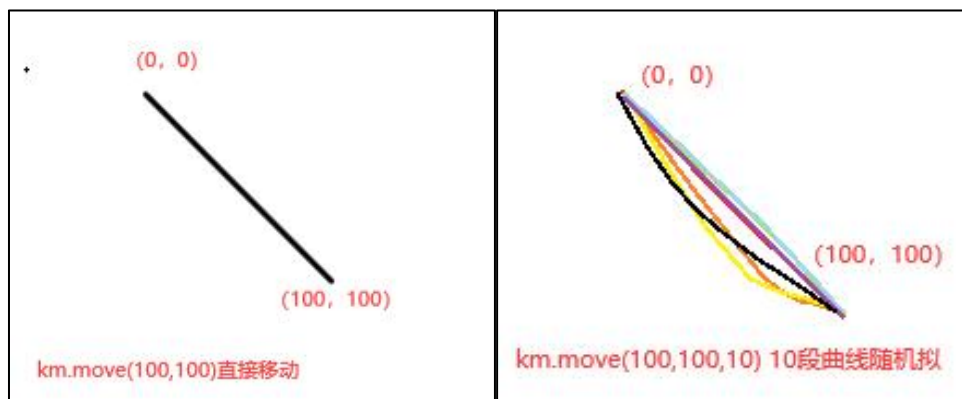
Above is the basic usage of move function, move function only give two parameters, then the mouse move will be completed within 1ms. There is no intermediate process, the actual effect is to move from the current point to the corresponding offset point in steps. This is the most efficient and fastest. If you need to simulate manual trajectory movement, you can add another parameter to the move function, which is used to constrain how many straight lines you expect to be fitted from

the start to the end. By common sense, the larger the third parameter number, the smoother the fitted curve will be.

3 straight lines fit a move from (0, 0) to (100, 100)



1 straight line and 10 straight lines fit a move from (0, 0) to (100, 100)

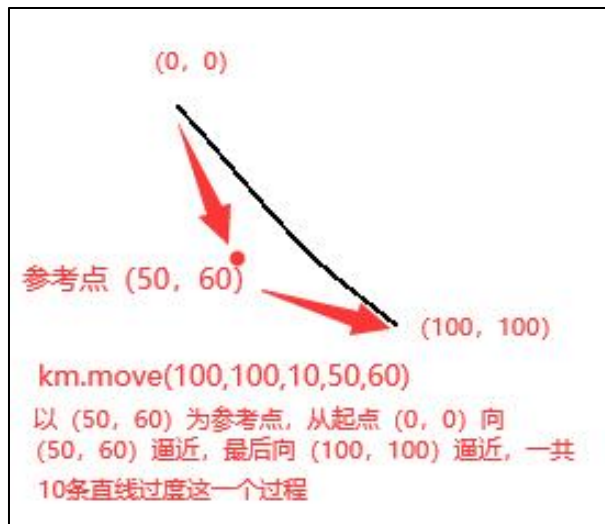


The image above shows the mouse trajectory. You can notice that the larger the number of points fitted, the smoother the excess curve from the start point to the end point. So, if you want to control the mouse to simulate human action, you can add a third parameter to move. You can change the step state into a continuous state.

In addition, you'll notice that each time you call `move(100, 100, 10)`, you get a different mouse path. Because there are countless curves that cross two points, `kmbox` will automatically randomize the points internally. It is guaranteed to move smoothly from the start point to the end point.

If you don't want `kmbox` to automatically randomize the points, but want to actually control the middle of the process curve and then add two parameters to the move function, used for reference approximation.

`km.move(100,100,10,50,60)`



where $(50, 60)$ are the coordinates of the reference point. kmbox will automatically fit the start point to the end point according to the reference point. Given the reference point, the trajectory of kmbox is uniquely determined. The direction of the curve and the degree of distortion are all determined by this reference point.

lock_mx(lock mouse x-direction mask function)

The `lock_mx(lock mouse X)` function is used to block the physical input of the x-axis direction of the mouse, and is mostly used to ensure that the x-direction movement of the software is not affected by the physical movement of the mouse.

The input parameter is 1 to lock the x-axis direction:

When the x-axis direction is locked, the left and right mouse movements will not work, only up and down movements.

An input parameter of 0 unlocks the x-axis direction.

If there is no input parameter, it will query the locked status of x-axis.

Return value 0: x-axis is not locked.

Return value 1: x-axis is locked.

The screenshot of the usage is shown below:

```

>>> km.lock_mx(1)  锁定X轴
>>> km.lock_mx()  查询X轴是否被锁定
1
>>> km.lock_mx(0)  解锁X轴
>>> km.lock_mx()  查询X轴是否被锁定
0
>>> 

```

After locking the X-axis, the physical movement of the mouse will not affect the movement in the script. This ensures that the script logic is not broken. Remember to use locking and unlocking together.

```
>>> km.lock_mx('help')
用于锁定鼠标X轴, 锁定后鼠标无法左右移动, 只能上下移动:
没有参数输入是查询X轴方向是否锁定. 0: 未锁定 1: 已锁定
有参数时是设置X轴是否锁定, lock_mx(1) 锁定, lock_mx(0) 解锁
>>>
```

lock_my Mouse y-direction blocking function

The lock_my(lock mouse Y) function is used to block the physical input of the mouse in the y-axis direction, and is mostly used to make sure that the y-direction movement of the software is not affected by the physical movement of the mouse.

The input parameter is 1 to lock the y-axis direction:

When the y-axis direction is locked, the mouse will not move up and down, only left and right.

When the input parameter is 0, the y-axis direction is unlocked.

If there is no input parameter, it queries the locked status of y-axis.

Return value 0: x-axis is not locked.

Return value 1: Mouse x-axis is locked.

```
>>> km.lock_my('help')
用于锁定鼠标Y轴, 锁定Y轴后鼠标无法上下移动, 只能左右移动:
没有参数输入是查询Y轴方向是否锁定. 0: 未锁定 1: 已锁定
有参数时是设置Y轴是否锁定, lock_my(1) 锁定, lock_my(0) 解锁
>>>
```

lock_ml left mouse button blocking function

Lock_ml(lock mouse left) function is used to block the physical input of the left mouse button, mostly used to ensure the software left button control is not affected by the physical left button of the mouse.

The input parameter is 1 to lock the left mouse button:

When the left mouse button is locked, it is useless to click the mouse physically.

A parameter of 0 unlocks the left mouse button.

No input parameter is used to query the locked status of the left mouse button.

Return value 0: Left mouse button is not locked.

Return value 1: the left mouse button is locked.

Please remember to use lock and unlock together.

Note that after lock_ml, you can call catch_ml() to catch the left mouse button pressed and lifted, see catch_ml for details.

```
>>> km.lock_ml('help')
用于软件锁定鼠标左键, 锁定鼠标后物理点击左键不会有反应:
没有参数输入是查询鼠标左键是否锁定. 0: 未锁定 1: 已锁定
有参数时是设置鼠标左键是否锁定, lock_ml(1) 锁定, lock_ml(0) 解锁
锁定后可以用left() 查询左键实际状态或者catch_ml捕获状态
>>>
```

lock_mm middle mouse button blocking function

Lock_mm(lock mouse middle) function is used to block the physical input of the mouse middle button, mostly used to ensure that the software middle button control is not affected by the physical mouse middle button.

The input parameter is 1 to lock the mouse middle button:

When the mouse middle button is locked, it is useless to click the mouse physically.

A parameter of 0 unlocks the middle button.

No input parameter is used to query the locked status of the middle button.

Return value 0: the center button is not locked.

Return value 1: Mouse center button is locked.

Please remember to use lock and unlock together.

Note that after lock_mm, you can call catch_mm() to catch the middle mouse button pressed and lifted, see the usage of catch_mm for details.

```
>>> km.lock_mm('help')
用于锁定鼠标中键, 锁定鼠标后物理点击中键不会有反应:
  没有参数输入是查询鼠标中键是否锁定. 0: 未锁定  1: 已锁定
  有参数时是设置鼠标中键是否锁定, lock_mm(1) 锁定, lock_mm(0) 解锁
  锁定后可以用middle() 查询中键实际状态或者catch_mr捕获状态
>>>
```

lock_mr right mouse button blocking function

The lock_mr(lock mouse right) function is used to block the physical input of the right mouse button, and is mostly used to ensure that the software right button control is not affected by the physical right button of the mouse.

The input parameter is 1 to lock the right mouse button:

When the right mouse button is locked, it is useless to click the mouse physically.

A parameter of 0 unlocks the right mouse button.

No input parameter is used to query the locked status of the right mouse button.

Return value 0: Right mouse button is not locked.

Return value 1: Right mouse button is locked.

Please remember to use lock and unlock together.

Note that after lock_mr, you can call catch_mr to catch the right mouse button pressed and lifted, see the usage of catch_mr for details.

```
>>> km.lock_mr('help')
用于锁定鼠标右键, 锁定鼠标后物理点击右键不会有反应:
  没有参数输入是查询鼠标右键是否锁定. 0: 未锁定  1: 已锁定
  有参数时是设置鼠标右键是否锁定, lock_mr(1) 锁定, lock_mr(0) 解锁
  锁定后可以用right() 查询右键状态或者catch_mr捕获状态
>>>
```

lock_ms1 Mouse Side Button 1 Masking Functions

Lock_ms1(lock mouse side 1) function is used to block the physical input of the mouse side button 1, mostly used to ensure that the software side button 1 control is not affected by the physical mouse side button 1.

The input parameter is 1 to lock the mouse side button:

When the mouse side button is locked, it is useless to click the mouse physically.

An input parameter of 0 unlocks the mouse sidebutton.

No input parameter is used to query the locked state of the sidebutton.

Return value 0: Mouse side button is not locked.

Return value 1: Mouse sidebutton is locked.

Please remember to use lock and unlock together.

Note that after lock_ms1, you can call catch_ms1 to catch the right mouse button pressed and lifted, see the usage of catch_ms1 for details.

```
>>> km.lock_ms1('help')
用于软件锁定鼠标侧键1, 锁定鼠标后物理点击侧键1不会有反应:
没有参数输入是查询鼠标侧键1是否锁定. 0: 未锁定 1: 已锁定
有参数时是设置鼠标侧键1是否锁定, lock_ms1(1) 锁定, lock_ms1(0) 解锁
锁定后可以用catch_ms1捕获侧键实际状态
>>>
```

lock_ms2 mouse side button 2 blocking function

Lock_ms2(lock mouse side 2) function is used to block the physical input of the mouse side button 2, mostly used to ensure that the software side button 2 control is not affected by the physical mouse side button 2.

The input parameter is 1 to lock the mouse side button:

When the mouse sidebutton is locked, it is useless to click the mouse physically.

A parameter of 0 unlocks the sidebuttons.

No input parameter is used to query the locked state of the sidebutton.

Return value 0: Mouse side button is not locked.

Return value 1: Mouse sidebutton is locked.

Please remember to use lock and unlock together.

Note that after lock_ms2, you can call catch_ms2 to catch the right mouse button pressed and lifted, see the usage of catch_ms2 for details.

```
>>> km.lock_ms2('help')
用于软件锁定鼠标侧键2, 锁定鼠标后物理点击侧键2不会有反应:
没有参数输入是查询鼠标侧键2是否锁定. 0: 未锁定 1: 已锁定
有参数时是设置鼠标侧键2是否锁定, lock_ms2(1) 锁定, lock_ms2(0) 解锁
锁定后可以用catch_ms2捕获侧键实际状态
>>>
```

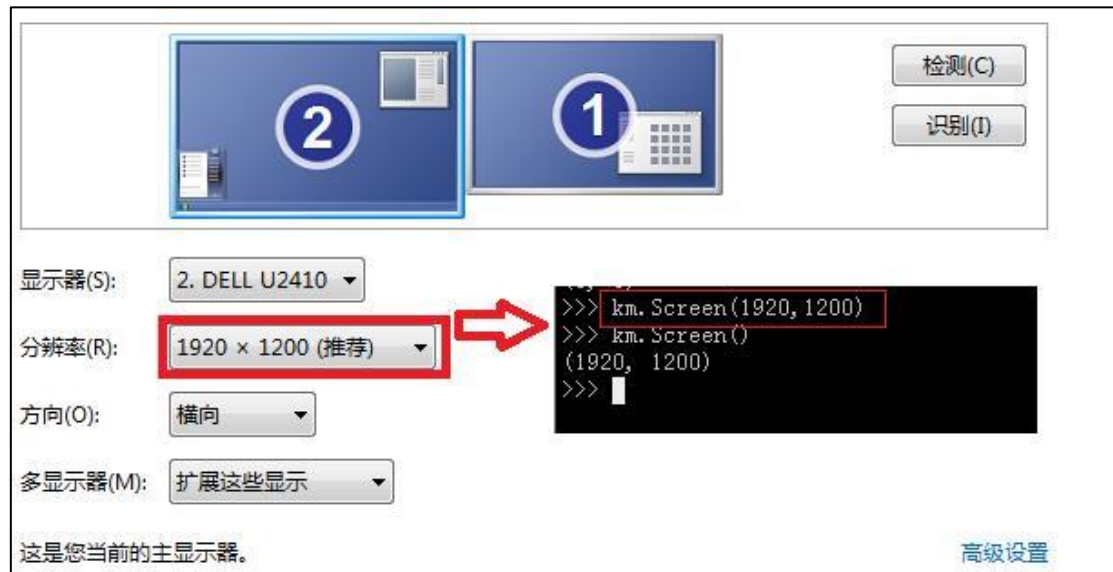
Screen mouse precise control range limit function

As we know before, all mouse data received by the computer are forwarded to the PC by the kmbox. In principle, Kmbox knows how much the mouse has moved in X,Y direction. If we can record the current XY, is it possible to know the current mouse in the screen at which point? This makes sense in principle, and Kmbox's internal coordinate statistics are also based on this idea. But there are also some cases will appear special cases. For example, when you move the mouse to the upper left corner of the screen, then you continue to move to the upper left corner. The mouse pointer will not move. Because the window that your mouse has been in his boundary, can not be smaller. But at this time the bottom of the mouse or the hair code has been there, to the left to move up, but the actual performance is the mouse will not move. If this time is still the statistics of the underlying data, then there must be a problem. In order to solve this problem, we introduce a mouse movement range limit function Screen. this function corresponds to the maximum range of

our mouse can move. For example, if our monitor resolution is 1920x1200, then we can limit the mouse to move only within this range. Beyond this range of movement will not be counted. This will realize the screen coordinates and kmbox coordinates correspond to each other. First of all do not give parameters to call the screen function, found that the return value is (0,0), that is, there is no limit on the screen size.

Now we limit the scope of the mouse according to the actual situation:

For example, the monitor is 1920x1200. Then directly set the allowable range of movement to Screen(1920,1200)



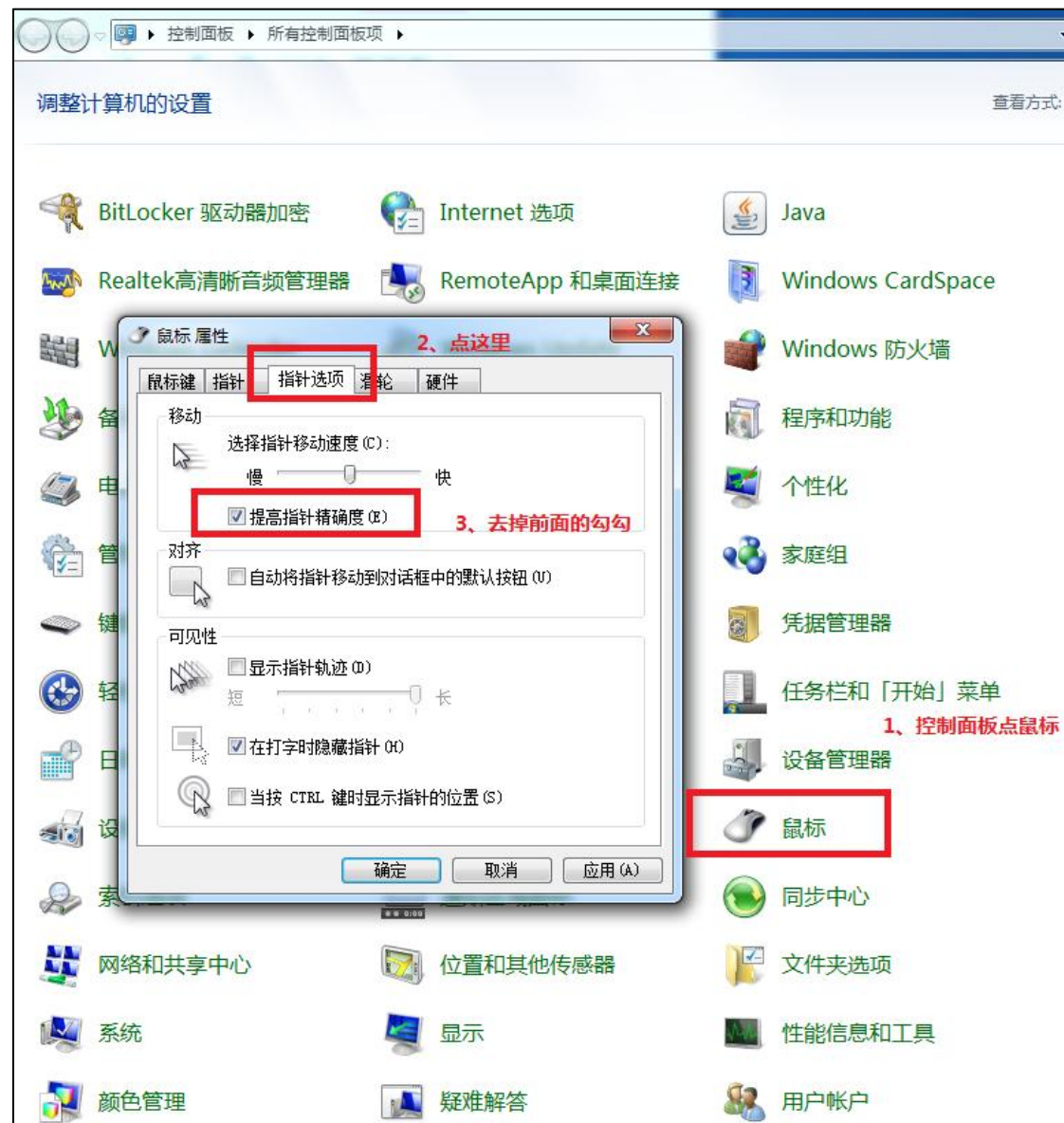
At this time, kmbox has limited the range of mouse movement internally. But this time it doesn't take effect. Because kmbox doesn't know which position of the screen the mouse is in and whether it is out of bounds or not. Next we need to call a calibration function zero.

```
>>> km.Screen('help')
用于设置鼠标允许移动的最大区域。避免鼠标边界错误
Screen():没有参数时是查询当前设置的限制尺寸
Screen(1920, 1080):有参数表示设置鼠标最大移动范围为:1920x1080
注意:当鼠标超过上面的范围时, 虽然底层在发移动数据, 但是鼠标已到达屏幕边缘
PC端不会有任何移动效果。在精确控制鼠标时会用到这个函数, 避免鼠标统计坐标出错
>>>
```

zero mouse precision control zero function

Zero function will make the mouse automatically calibrated. That is, the mouse moves to the upper left corner of the screen (0,0), and the back of the real-time record of the current mouse coordinates. Can be used to accurately control the mouse movement on the screen. And when the mouse exceeds the X,Y defined by screen function, the packet will be discarded automatically (because even if the underlying data is sent, the mouse will not move at the boundary). Note that, please remove windows mouse acceleration algorithm, otherwise the mouse will not move according to the actual data of the underlying mouse. windows has adopted a certain acceleration algorithm in order to increase the speed of the mouse, he will predict the target position according to the

acceleration of the mouse telegram. We need to disable this feature for precise control, otherwise it will also cause errors. The method to remove the mouse acceleration algorithm is as follows:



Zero function usage is as follows:

```
>>> km.zero('help')
鼠标坐标模式设置和查询
zero(0): 设置为相对模式坐标。坐标原点为当前位置。对应移动函数为move()
zero(1): 设置为绝对模式坐标。坐标原点在屏幕的左上角。对应移动函数为moveto()
zero() : 无参数是查询当前鼠标模式, 0: 相对模式 1: 绝对模式
>>>
```

getpos Mouse precise control to get the current position function

The getpos function is to get the current physical position of the mouse. That is, the coordinates of kmbox internal statistics. This function is only meaningful if it is called by zero. He can realize the one-to-one correspondence between kmbox coordinates and screen coordinates. Theoretically, if you choose a point on the screen, no matter how you move the mouse, the physical coordinate of this point is constant, the return value of Getpos function is the physical coordinate of

the current point of the mouse, the first element is the x-coordinate, the second element is the y-coordinate. The screenshot is shown below:

```
>>>
>>>
>>>
>>> km.getpos()
[109, 24]
>>>
```

The moveto mouse precision function

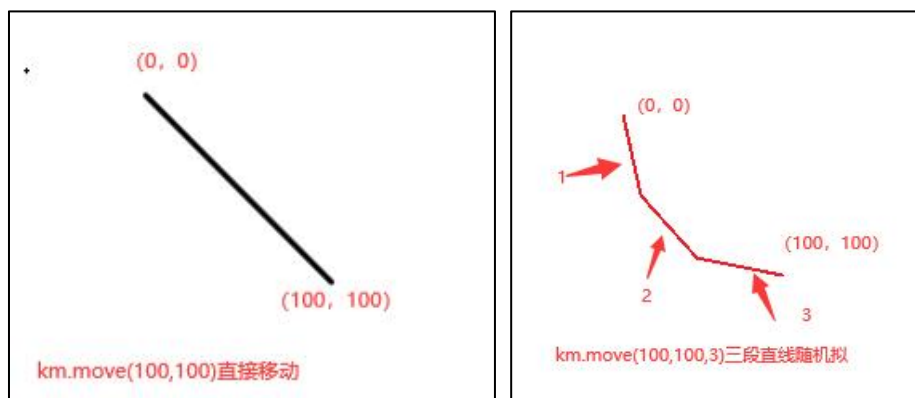
moveto moves the mouse to the absolute physical coordinates (x, y). So this function is also available after executing [the zero](#) function. The difference with the move function is that the origin of moveto is the upper left corner of the screen (0,0). The origin of move function is the current mouse position. moveto can quickly and precisely move the mouse to the specified position.

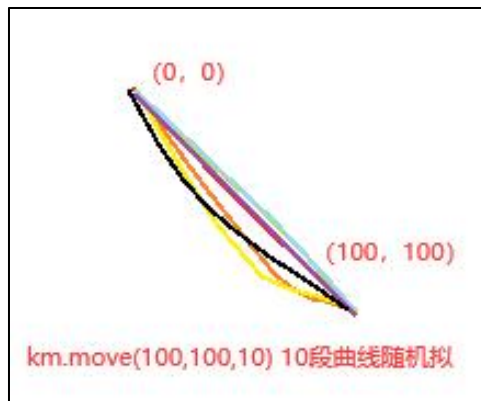
The usage of this function is as follows:

```
>>> km.moveto('help')
用于鼠标的绝对移动，将鼠标移动到绝对坐标(x,y)处：
此函数需要先调用zero(1)，且规定屏幕左上角为坐标原点(0,0)
推荐使用Screen(x,y)设定屏幕的分辨率大小，这样移动更准确
moveto(100,200)表示移动到绝对坐标(100,200)处
x,y的范围为0-32767，不存在负坐标
注意：请关闭windows鼠标加速算法
>>>
```

Note that generally this function is used in conjunction with getpos. For example, you want to move to a certain coordinate point. The coordinates of this point can be ~~data~~ by getpo. Then you can call moveto to move to that point exactly.

Above is the basic use of moveto function, moveto function if only give two parameters, then the mouse will move in 1ms to complete, no intermediate process. The actual effect is to move from the current point to the corresponding target point. This is the most efficient and fastest. If you need to simulate a manual trajectory, you can add another parameter to the moveto function, which is used to constrain the number of straight lines you expect to fit from the start point to the end point. By common sense, the larger the number of the third parameter, the smoother the fitted curve will be, as shown in the following figure: 1 straight line, 3 straight lines, and 10 straight lines to fit the move from (0,0) to (100,100), respectively.



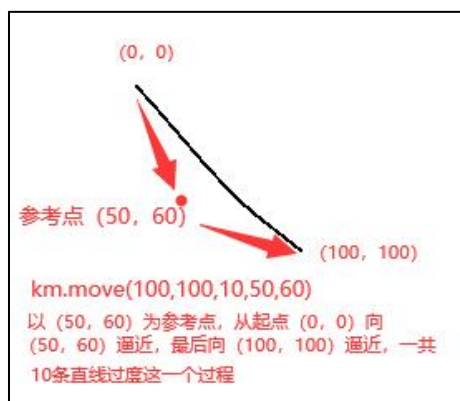


The above figure shows the mouse trajectory, and you can see that the larger the number of points fitted, the smoother the excessive curve from the starting point to the end point. So, if you want to control the mouse to simulate human action, you can add a third parameter to move. You can change the step state into a continuous state.

In addition, you will find that the mouse trajectory is different every time you call move(100, 100, 10). Because there are countless curves over two points, kmbox will automatically randomize the points internally. It is guaranteed to move smoothly from the start point to the end point.

If you don't want kmbox to automatically randomize the points, but want to actually control the middle of the process curve and then add two parameters to the moveto function to do the reference approximation.

km.moveto(100,100,10,50,60)



where (50, 60) are the coordinates of the reference point. kmbox will automatically fit the start point to the end point according to the reference point. Given the reference point, the trajectory of kmbox is uniquely determined. The direction of the curve and the degree of distortion are all determined by this reference point.

catch_ml left mouse button catch function

catch_ml(catch mouse left) function is used to catch the physical input of the left mouse button, mostly used in cases where special handling of the left button is required. Its usage is the same as the keyboard catch_kb. The following figure shows how catch_ml is used

```
>>> km.catch_ml('help')
捕获鼠标左键：
    当调用lock_ml(1)后，鼠标左键被实时监测，且鼠标左键按下和弹起的消息
    不会发送给pc，需要自己做处理。catch_ml有两种模式，阻塞模式和非阻塞模式
    km.catch_ml():是不带参数的非阻塞模式。没有发生捕捉事件返回0.按下返回1，松开返回2
    非阻塞模式常用在快速轮询，km.catch_ml(0)也是非阻塞模式
    km.catch_ml(1):是带参数的阻塞模式。函数在没有捕捉到指定的按键前是不会返回的。
    所以阻塞模式建议在多线程中使用。不然当前线程会被阻塞
    PS:一定要调用lock_ml(1)函数后才能激活使用catch_ml函数。
>>>
```

catch_mm Middle mouse button catch function

catch_mm(catch mouse middle) function is used to catch the physical input of the middle mouse button, mostly used in cases where special handling of the middle button is required. Its usage is the same as the keyboard catch_kb. The following figure shows how catch_mm is used

```
>>> km.catch_mm('help')
捕获鼠标中键：
    当调用lock_mm(1)后，鼠标中键被实时监测，且鼠标中键按下和弹起的消息
    不会发送给pc，需要自己做处理。catch_mm有两种模式，阻塞模式和非阻塞模式
    km.catch_mm():是不带参数的非阻塞模式。没有发生捕捉事件返回0.按下返回1，松开返回2
    非阻塞模式常用在快速轮询，km.catch_mm(0)也是非阻塞模式
    km.catch_mm(1):是带参数的阻塞模式。函数在没有捕捉到指定的按键前是不会返回的。
    所以阻塞模式建议在多线程中使用。不然当前线程会被阻塞
    PS:一定要调用lock_mm(1)函数后才能激活使用catch_mm函数。
>>> █
```

catch_mr Right mouse button catch function

The catch_mr(catch mouse right) function is used to catch the physical input of the right mouse button, and is often used when special handling of the right button is required. Its use is the same as the keyboard catch_kb. The following diagram shows how catch_mr is used.

```
>>> km.catch_mr('help')
捕获鼠标右键：
    当调用lock_mr(1)后，鼠标右键被实时监测，且鼠标右键按下和弹起的消息
    不会发送给pc，需要自己做处理。catch_mr有两种模式，阻塞模式和非阻塞模式
    km.catch_mr():是不带参数的非阻塞模式。没有发生捕捉事件返回0.按下返回1，松开返回2
    非阻塞模式常用在快速轮询，km.catch_mr(0)也是非阻塞模式
    km.catch_mr(1):是带参数的阻塞模式。函数在没有捕捉到指定的按键前是不会返回的。
    所以阻塞模式建议在多线程中使用。不然当前线程会被阻塞
    PS:一定要调用lock_mr(1)函数后才能激活使用catch_mr函数。
>>> █
```

catch_ms1 Mouse side button 1 catch function

The catch_ms1(catch mouse side 1) function is used to catch the physical input of mouse side 1, and is mostly used in cases where special handling of side 1 is required. Its usage is the same as the keyboard catch_kb. The following figure shows how catch_ms1 is used.

```
>>> km.catch_ms1('help')
捕获鼠标侧键1：
    当调用lock_ms1(1)后，鼠标侧键被实时监测，且鼠标侧键按下和弹起的消息
    不会发送给pc，需要自己做处理。catch_ms1有两种模式，阻塞模式和非阻塞模式
    km.catch_ms1():是不带参数的非阻塞模式。没有发生捕捉事件返回0.按下返回1，松开返回2
    非阻塞模式常用在快速轮询，km.catch_ms1(0)也是非阻塞模式
    km.catch_ms1(1):是带参数的阻塞模式。函数在没有捕捉到指定的按键前是不会返回的。
    所以阻塞模式建议在多线程中使用。不然当前线程会被阻塞
    PS:一定要调用lock_ms1(1)函数后才能激活使用catch_ms1函数。
>>> █
```

catch_ms2 Mouse side button 2 catch function

The `catch_ms1`(catch mouse side 2) function is used to catch the physical input of mouse side 2, which is mostly used in cases where special processing is needed for side 2. Its usage is the same as the keyboard `catch_kb`. The following figure shows how `catch_ms2` is used.

```
>>> km.catch_ms2('help')
捕获鼠标侧键2:
    当调用lock_ms2(1)后, 鼠标侧键被实时监测,且鼠标侧键按下和弹起的消息
    不会发送给pc,需要自己做处理。catch_ms2有两种模式, 阻塞模式和非阻塞模式
    km.catch_ms2():是不带参数的非阻塞模式。没有发生捕捉事件返回0. 按下返回1, 松开返回2
    非阻塞模式常用在快速轮询,km.catch_ms2(0)也是非阻塞模式
    km.catch_ms2(1):是带参数的阻塞模式。函数在没有捕捉到指定的按键前是不会返回的。
    所以阻塞模式建议在多线程中使用。不然当前线程会被阻塞
    PS:一定要调用lock_ms2(1)函数后才能激活使用catch_ms2函数。
>>>
```

These are the functions supported by mouse macros. If you have more new requirements or comments, please contact me.

6. Other Functions and Modules

In the previous sections. You should have grasped what functions are included in `kmbox`'s keyboard macros and mouse macros. This section talks about other functions. You may use them.

debug function

```
>>> km.debug('help')
打印调试数据:(开发人员专用)
    当需要查看底层传输数据时可以打开这个调试。
    输入参数, 0: 关闭所有调试打印
    输入参数, 1: 打开鼠标底层报文数据
    输入参数, 2: 打开键盘底层报文数据
>>>
```

delay function

```
>>> km.delay('help')
delay(...):延迟函数
    1个参数是精确延时, 例如延迟10ms,km.delay(10)
    2个参数是随机延时, 例如随机延迟100ms-200ms,km.delay(100,200)
>>>
```

reboot function

```
>>> km.reboot('help')
软复位重启板卡
    km.reboot()会在3秒后软件复位重启板卡
>>>
```

version version query function

```
>>> km.version('help')
返回当前kmbox的版本号:
    最新固件请去官网:www.kmbox.top
    或者论坛:www.clion.top
    也欢迎提交bug或者意见!
>>> km.version()
kmbox: 2.0.0 Aug 31 2020 21:49:51
>>>
```

rgb_free Free the GP IO of the tri-color LEDs.

```
>>> km.rgb_free('help')
释放三色LED占用的3个GPIO:
    当你需要复用这个3个IO时, 你需要调用这个函数将3个GPIO从kmbox释放出来
    不然你在操作IO, kmbox也在操作IO. 那么最后就乱套了
    释放IO调用:km.rgb_free()
>>>
```

mode mode switch function

Kmbox has a total of 7 transmission modes, namely USB mode, bluetooth mode, kvm mode.

You can use this function to switch between different modes in multiple hosts.

```
>>> km.mode('help')
kmbox传输模式设置:
    kmbox一共有7种传输协议, 分别对应如下:
    0:usb+bt+kvm模式 (等同于模式1+模式2+模式3)
    1:USB模式 (模式1, 有且仅有USB传输--kmbox的PC端口传输)
    2:蓝牙模式 (模式2, 有且仅有蓝牙传输)
    3:kvm模式 (模式3, 有且仅有kvm传输, 需要kvm扩展卡)
    4:usb+kvm模式 (等同于模式1+模式3)
    5:bt+kvm模式 (等同于模式2+模式3)
    6:usb+bt模式(等同于模式1+模式2)
使用方法: km.mode(1)->切换到USB模式
           km.mode() ->查询当前模式
ps:此函数的用途一般是多系统切换, 例如你可以USB连接A电脑, 蓝牙连接B电脑
    kvm接C电脑, 调用此函数可在多个系统中切换键鼠连接的设备
```

rng random number generator function

```
>>> km.rng('help')
产生随机函数:
    无参数是产生一个随机数,km.rng()
    2个参数是产生一个指定范围内的随机数, 例如需要一个100-200之间的随机数,km.rng(100,200)
>>> █
```

freq keyboard and mouse report frequency overclocking setting function

km.freq() function is used to force the keyboard and mouse to set the reporting frequency. No setting is required by default. The default value is 200, i.e. the keyboard and mouse upload 200 this report every second. If you do not give the parameter is to query the current reporting frequency, to give the parameter is to force the setting of the reporting frequency, the value range is [100, 1000]. Reporting frequency is an inherent property of USB devices. Forced overclocking may cause instability of the device. The reporting frequency of regular keyboard and mouse is 125Hz. If you think the mouse lags slowly, you can force the reporting frequency to be higher. freq's usage is described in the km.freq('help') function.

MAC Get Machine Code Function

Machine code is the identity card of the board, the machine code of each board has uniqueness. Machine code is mostly used for script binding machine. Implement the registration authorization function to prevent scripts from being used on unauthorized devices. What encryption algorithm is defined by the script author.

The return value of the MAC function is the group element, six elements in total. The return value is different for each board.


```

>>> km.MAC()
(124, 158, 189, 193, 30, 80)
>>>

```

These are the API functions provided by the km module. Keyboard and mouse operations with km module should be basically enough. But kmbox is not just a keyboard and mouse operation. Keyboard and mouse are just some of the functions. kmbox has many more modules! You can type `help('modules')` to see what modules are included by default:

```

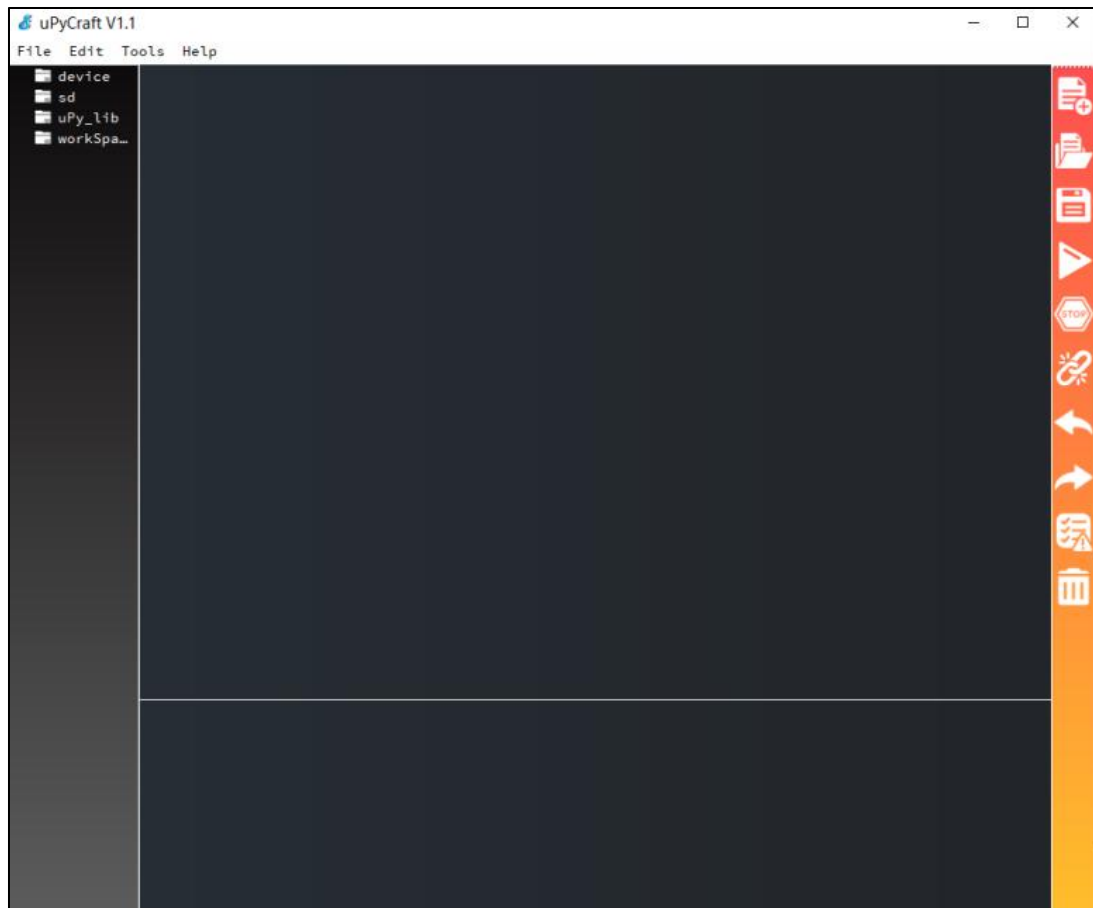
>>> help('modules')
__main__      esp32          os              ujson
_boot         flashbdev      random          uos
_ewire        framebuf      re              upip
_thread       gc           select          upip_utarfile
_webrepl      hashlib      socket          urandom
apa106        heapq        ssl             ure
array         host         struct          uselect
binascii      inisetup     sys             usocket
bt            io           time            ussl
btree         json         ubinascii       ustruct
builtins      km           ucollections    utime
cmath         machine      ucryptolib      utimeq
collections   math         uctypes         uwebsocket
device        micropython uerrno          uzlib
dht           neopixel     uhashlib        webrepl
ds18x20       network      uhashlib        webrepl_setup
errno         ntptime      uheapq          websocket_helper
esp           onewire      uio             zlib
Plus any modules on the filesystem
>>>

```

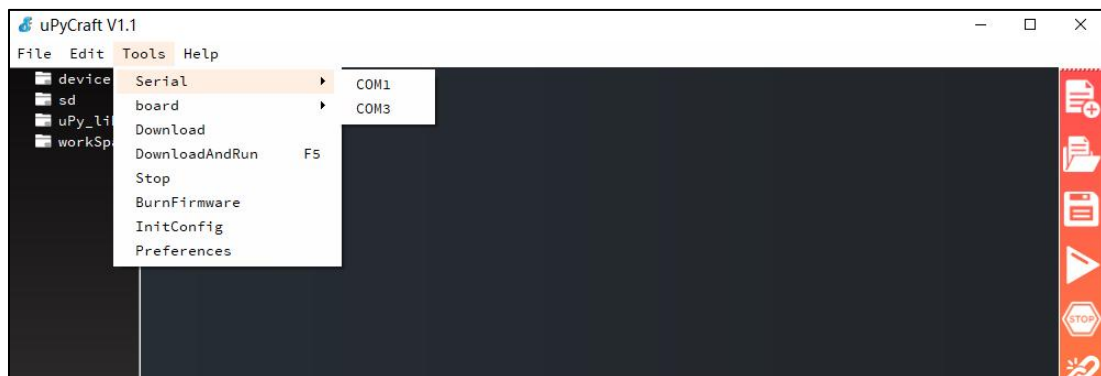
The km is only one of the many modules. We won't go through them one by one here. We will not talk about each one of them here, but we will learn about the km module from the previous section. You can use kmbox to drive the hardware. You can use kmbox to drive hardware, you can also use it to write software algorithms, DIY all kinds of electronic products are very suitable.

IV. kmbox practical chapter

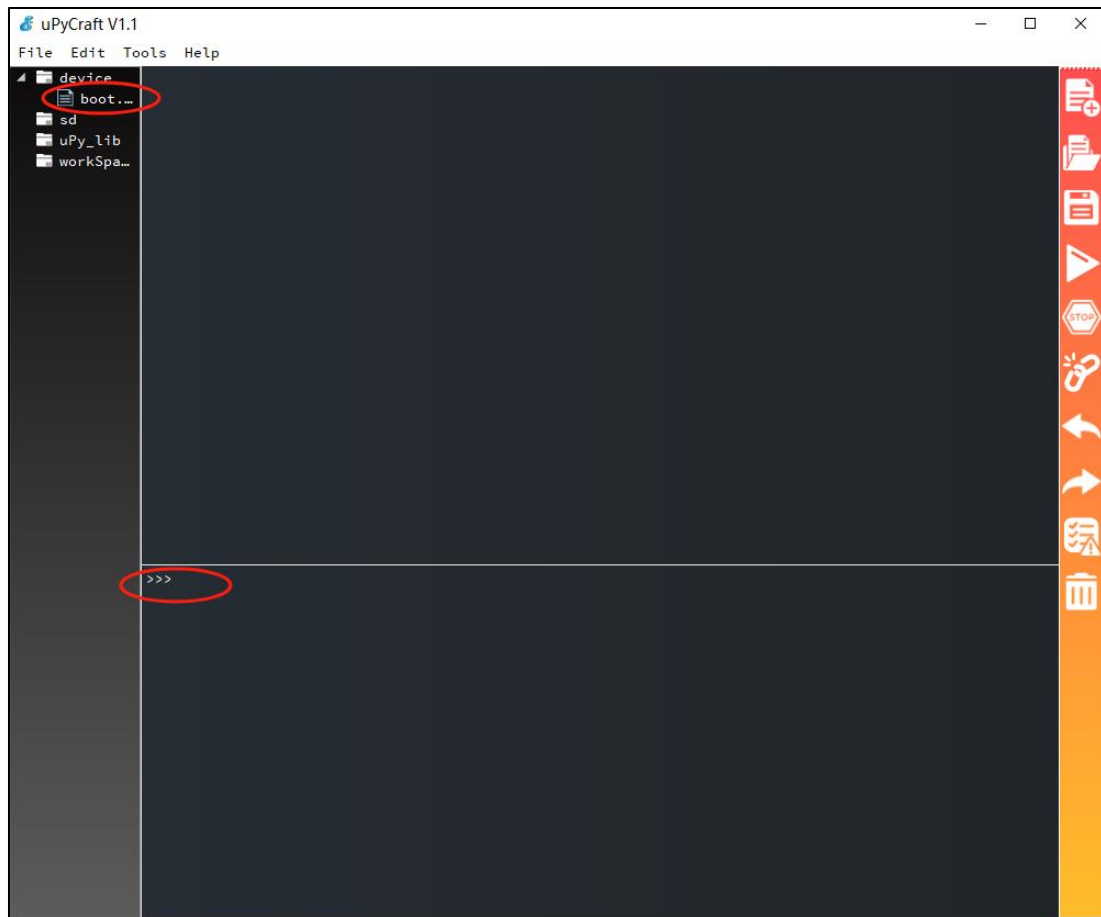
In the previous chapters, through putty can easily try to use each function, but this is still a little distance from the actual application of the script. This chapter will learn how to write scripts to link the previous API. From theory to practice over. Writing scripts is very simple, that is, the previous API calls and combinations, plus conditional judgment, loops and so on. All of this is based on the function you want to achieve. As you can see from the previous section, python is an online interpreter. It executes the code as soon as you knock it out of the box. If you have hundreds or thousands of lines of code, writing it in putty can drive you crazy. This section introduces a new tool, uPyCraft.exe, which is a specialized code editor and downloader. It can also be used for debugging. Double clicking uPyCraft.exe will bring up the following application screen:



After opening, you need to connect to the development board: in order tools --> serial-->COMxx connect to the development board.

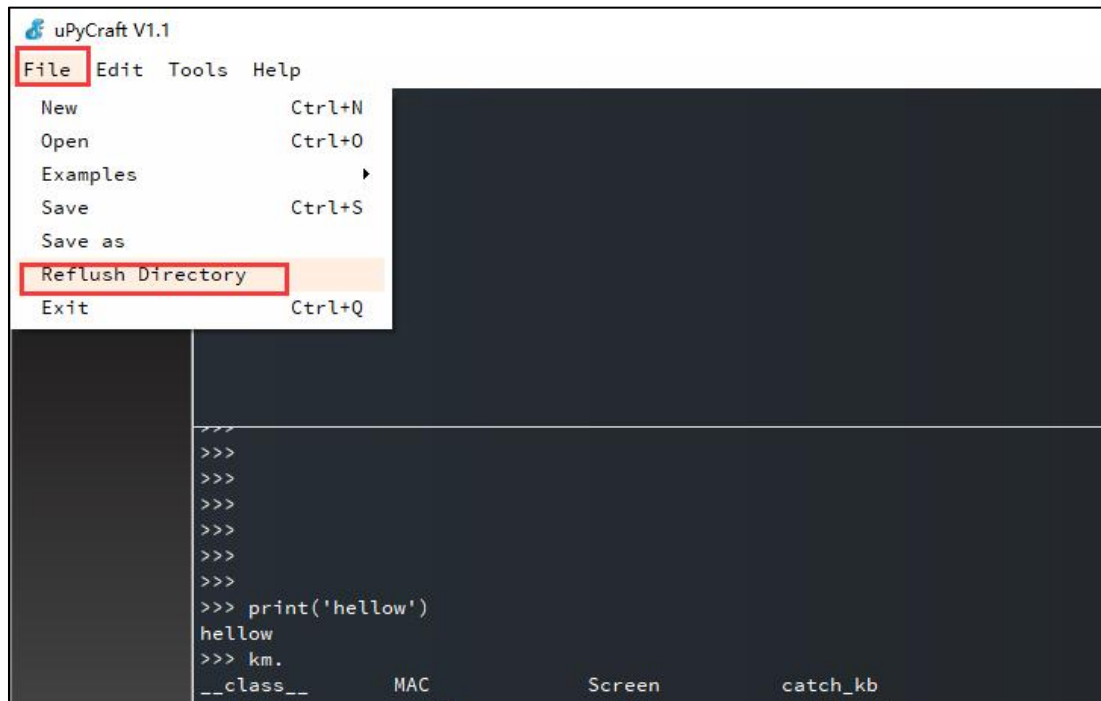


If the connection fails, please test the serial driver. After connecting, it looks like this. Here is actually a putty console.

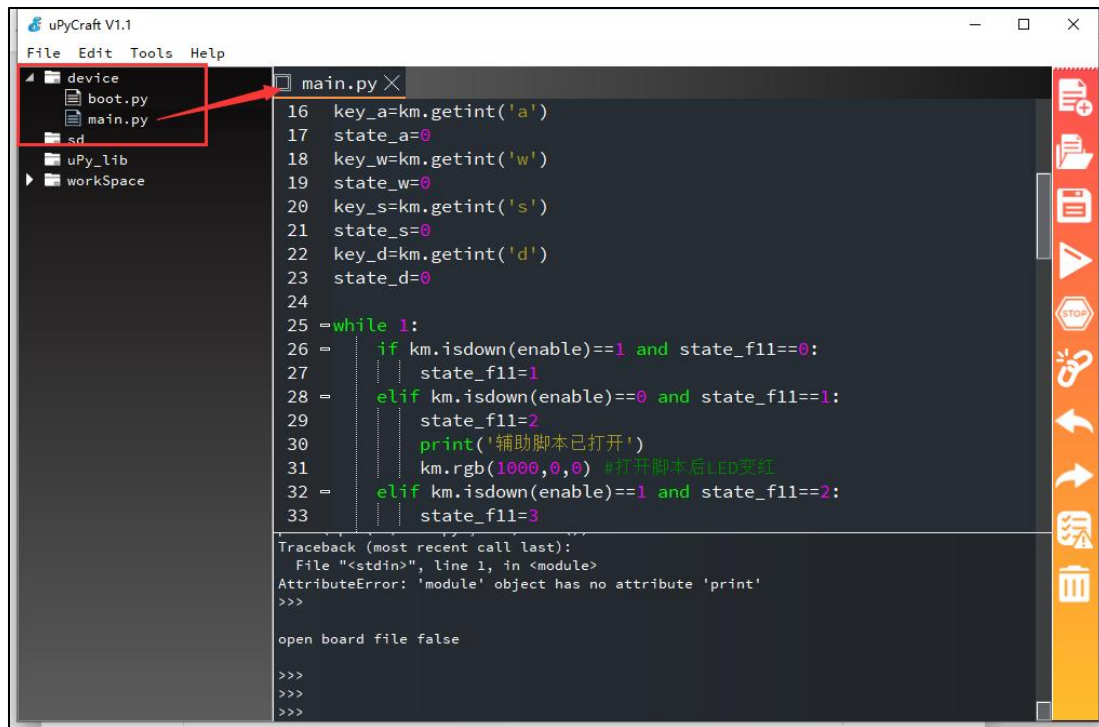


1. Read kmbox internal scripts

Click File -->Reflush Directory to read the scripts inside the board (**mpy scripts cannot be read**).

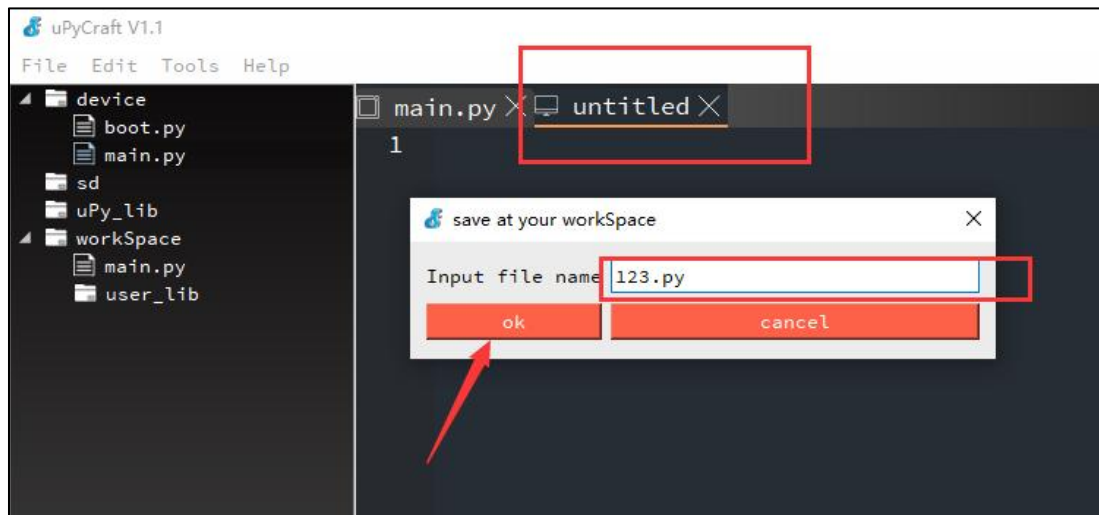


You can see the script files inside the board in the Device folder, double click the corresponding file to view the contents. You can modify, save and rename it.



2. New Script

File --> New can create a new file (shortcut key is Ctrl+N). The new file does not have a name, press ctrl+s to save the file to give him a name, click OK at this time you can write the script functions you need to this file.



3. Debugging scripts

For example, in 123.py, we wrote a simple script. It prints a hellow every 1 second.

Debugging this script is very simple. There are several ways to do it:

Way one:

Running in memory (recommended), press ctrl+e in the console below. kmbox will go into copy mode. Copy the contents of the 123.py file directly (ctrl+c) and right-click on the console to

paste it (**do not use the shortcut ctrl+v**). Press ctrl+d on the console after the paste is done, and the copied code will be run inside kmbox.

After Ctrl+d the script is already running, as shown in the following picture.

```
=== print('hellow ')
=== km.delay(1000)
===
===
hellow
hellow
hellow
hellow
hellow
```

If you want to end the run, press ctrl+c on the console.

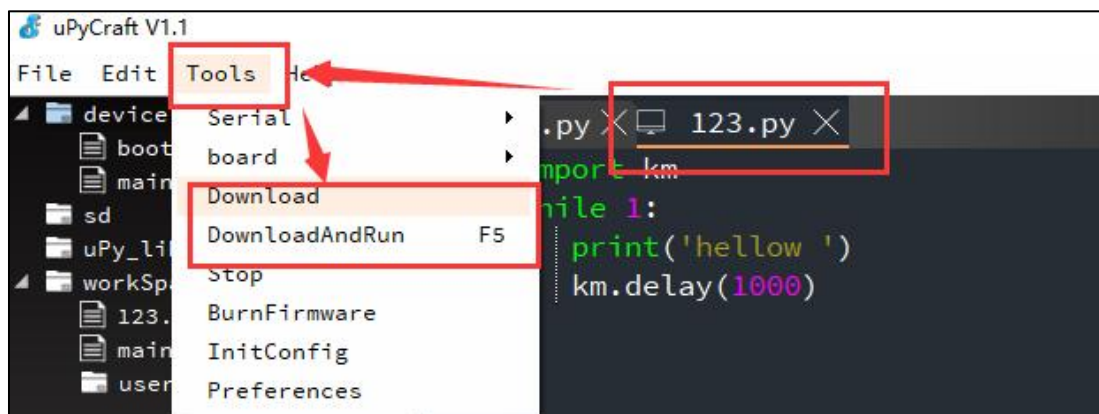
Way two:

Directly click RUN on the right, this method is not recommended. This method is not recommended because it will be downloaded to the board, Flash has a write life, usually 100,000 times will not work. It is recommended to use way 1, the script OK to confirm that there is no error and then burned to the internal board.

4. Burning Scripts

If there is no problem with the debugging script, we want to save the script permanently, we need to burn the script to the development board. For example, the 123.py in the previous section needs to be burned to the development board after confirming that the function is correct.

Click 123.py--->Tools---> Download to burn to the board.



Where downloadAndRun is to run the script immediately after downloading. If you don't need to run it, just download it. Burning scripts do not support mpy type files.

5. Simple application of the mouse macro (chicken pressure gun macro production)

This chapter will use the previous knowledge to make a simple and practical value of the chicken pressure gun macro. In fact, in the previous chapters have been mentioned in the gun

pressure is how to realize. The basic idea is to press the left button and move the mouse down automatically.

But you want to ask me exactly how much to move down? I can only tell you that I don't know. The only people who know this parameter are the programmers who write the chicken program. But we can test it. Today, we will take the wilderness action to do a simple ox knife small test it. Teach you how to use kmbox to write a chicken pressure gun plug-in.

See this video for detailed steps:

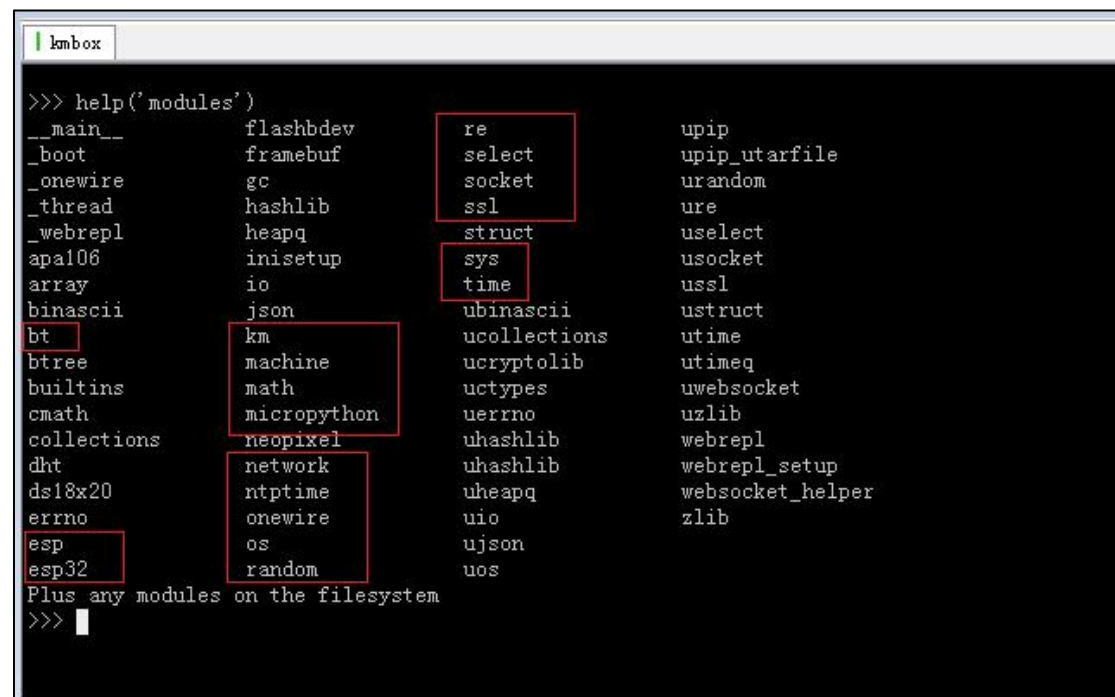
<https://www.bilibili.com/video/BV1Ya4y1Y7ku/>

6. How to use third-party libraries

In the basic tutorial above you were able to test and verify simple logic in a Repl environment. But a real project may require a lot of py files that depend on each other. So how to call other libraries or modules inside kmbox?

7. Default Modules

To import other modules, you need to see which modules are already included. You can type `help('modules')` in the repl interface to see which modules are included in kmbox by default:



```
>>> help('modules')
__main__      flashbdev      re             upip
__boot        framebuf       select          upip_utarfile
__onewire     gc             socket          urandom
_thread       hashlib        ssl             ure
_webrepl      heapq          struct          uselect
apa106        inisetup       sys             usocket
array         io             time            ussl
binascii      json           ubinascii       ustruct
bt            km             ucollections    utime
btree         machine        ucryptolib      utimeq
builtins      math           uctypes         uwebsocket
cmath         micropython    uerrno          uzlib
collections   neopixel       uhashlib        webrepl
dht           network        uhashlib        webrepl_setup
ds18x20       ntptime        uheapq          websocket_helper
errno         onewire        uio             zlib
esp           os             ujson
esp32         random         uos
Plus any modules on the filesystem
>>>
```

From the above, we can see the common km module, sys module, time module, machine module, os module and so on, that is to say, these module names can be directly included in our scripts by using **import xxxx**.

Press tab to see the default modules loaded at boot, as shown below:

```
>>>
Pin          SPI          __dict__      gc
os           spi          sys           vfs
KMMMain      TFT          bdev          sysfont
km           uart_speed  km_trace_type km_trace_enable
bt_enable    km_mode      km_freq       km_vertime
tft
>>> |
```

If you want to use the machine module. If you want to use the **machine** module, you can use **"import machine"** and press tab to see that the machine module has been packaged into our environment.

```
>>> import machine
>>>
Pin          SPI          __dict__      gc
machine     os           spi          sys
vfs          KMMMain      TFT          bdev
sysfont      km           uart_speed  km_trace_type
km_trace_enable km_mode      bt_enable    km_mode
km_freq      km_vertime  tft          uos
>>>
```

The machine module contains some hardware operations such as ADC, DAC, PWM, I2C, GPIO and so on.

```
>>> machine.
__class__    __name__      ADC          DAC
DEEPSLEEP    DEEPSLEEP_RESET EXT0_WAKE
EXT1_WAKE    HARD_RESET    I2C          PIN_WAKE
PWM          PWRON_RESET   Pin          RTC
SDCard       SLEEP         SOFT_RESET    SPI
Signal       TIMER_WAKE    TOUCHPAD_WAKE Timer
TouchPad     UART          ULP_WAKE      WDT
WDT_RESET    deepsleep     disable_irq   enable_irq
freq         idle          lightsleep    mem16
mem32        mem8          reset         reset_cause
sleep        time_pulse_us unique_id      wake_reason
>>> machine. |
```

If you do want to introduce something that is not in the default library. There are several ways to introduce other modules. kmbox draws a pentagram automatically. Before writing the script, you first have to figure out how to draw a pentagram with the mouse. The idea is as follows:

- Step 1: Draw a line of length L at an angle of 72° to the x-axis.
- Step 2: Deflect 144° to the right, and draw a line segment of length L.
- Step 3: Deflect 144° to the right and draw a line segment of length L.
- Step 4: Deflect 144° to the right, and draw a line of length L.
- Step 5: Deflect 144° to the right and draw a line segment of length L.

After the above 5 steps, you will get a five-pointed star. Then how to use the mouse macro to realize, please see the following code:

```

1  """
2  鼠标左键单击绘制五角星
3  mp_turtle是海龟绘图模块。移植标准的turtle模块
4  """
5  from mp_turtle import *
6
7  while 1:
8      if m.left()==1: #如果鼠标左键按下
9          home()      #将当前鼠标点设置为坐标原点
10         left(72)     #龟头72度 准备画图
11         for i in range(5): #循环5次画边
12             forward(100)  #绘制长度为100的边
13             right(144)    #右转144度
14             m.delay(100)  #延时100ms, 避免绘制过快您看不清过程
15
16         while m.left()==1: #此时五角星已经绘制完成, 等待鼠标左键松开避免不停绘制
17             m.delay(100)   #空等延时
18
19
20
21

```

In the above figure, the left mouse button is pressed, the current coordinate is set as the origin, and then it is shifted to the left by 72 degrees. In the for loop 5 times to draw a line length of 100 and deflection, that is, to get a five-pointed star.

The video of the actual running of the script is as follows: [Click me to see the video!](#)

The home, left, forward, and right functions used here are defined in the mp_turtle module. mp_turtle in turn calls the km module. This is equivalent to repackaging the km module again.

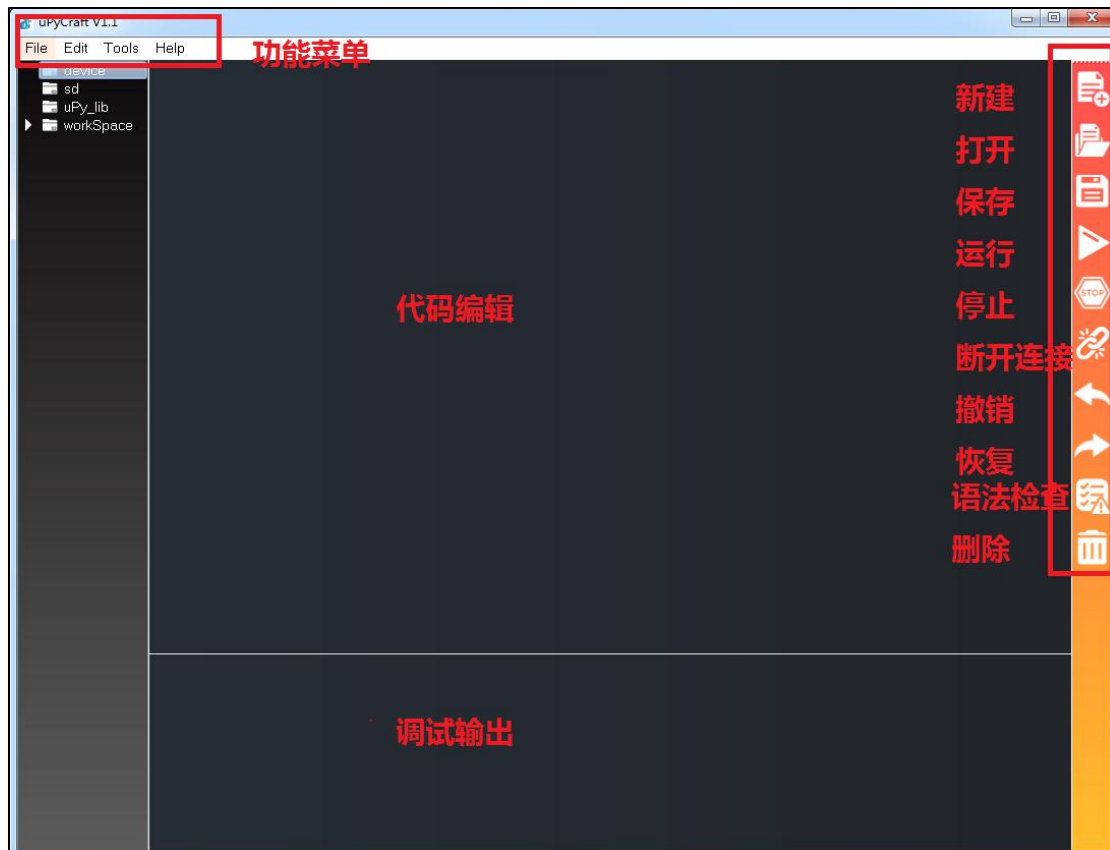
8. File Storage (Entry Level)

File storage is to save the source code of the module you want to import into kmbox. Then you can import the module directly. You can use the following tools to save py files inside kmbox.

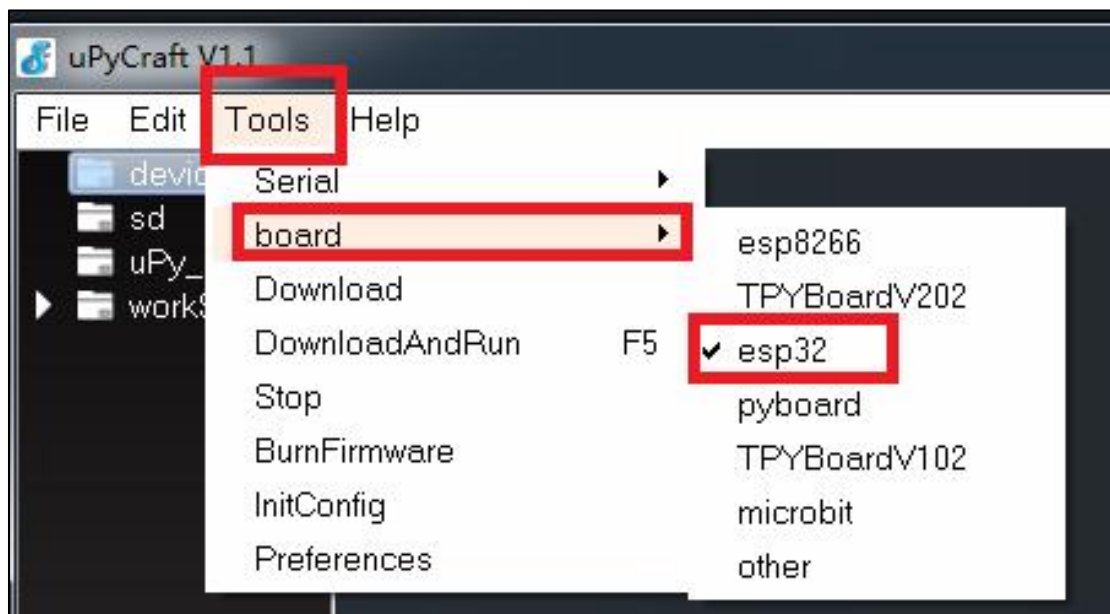
1. Upycraft

This is a small code editor and debugger specially designed for micropython.

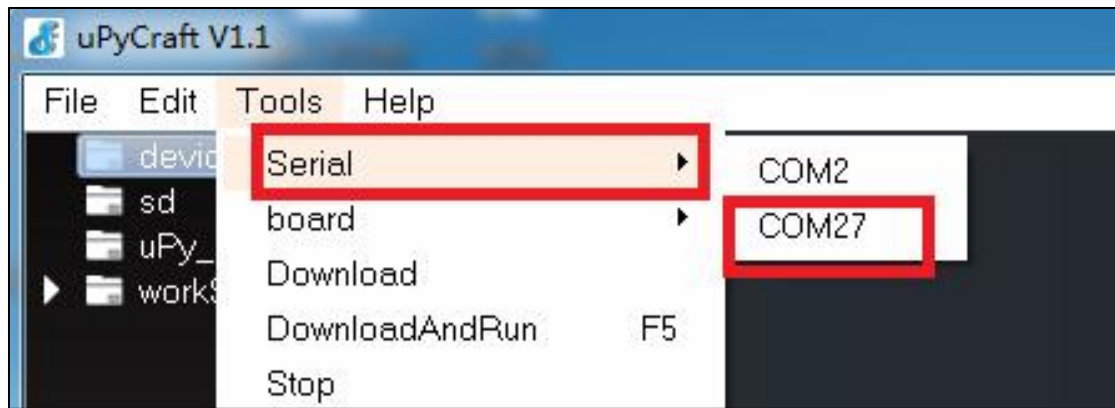
To download it, please visit: <https://github.com/SDRRADIO001/Keyboard-Mouse-Box>



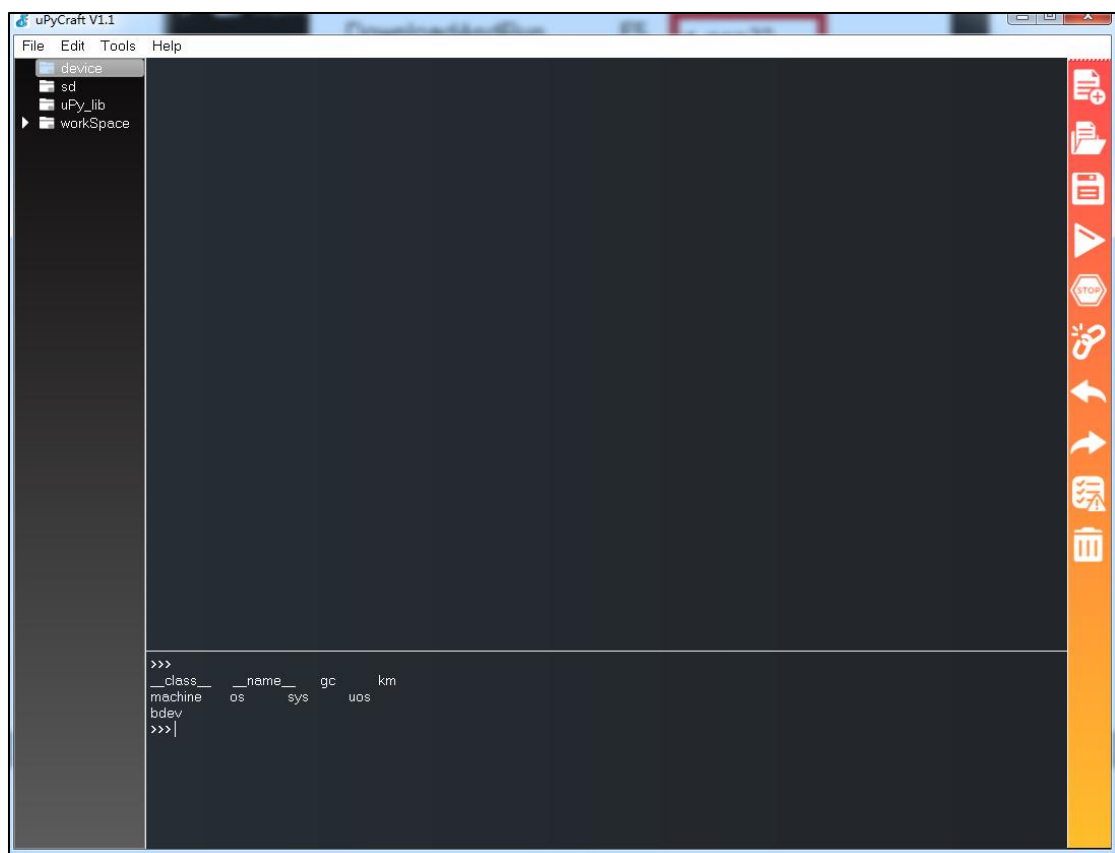
Let's take kmbox as an example to introduce how to use it and how to transfer files. After opening the software, under the tool menu, select the board chip model as esp32 in the board tab as shown in the following figure:



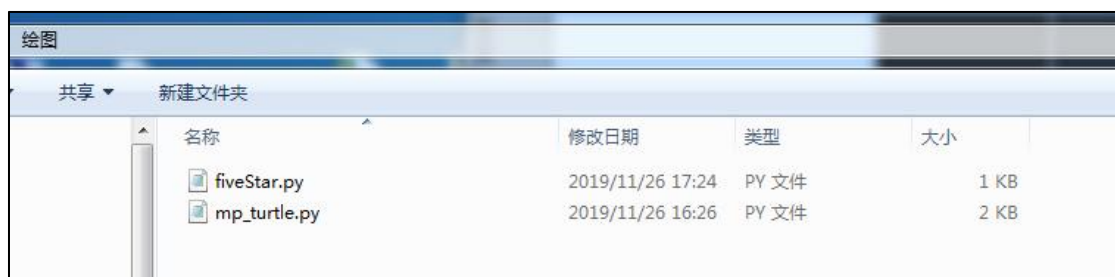
Then choose the serial port number of kmbox as the actual serial port number.



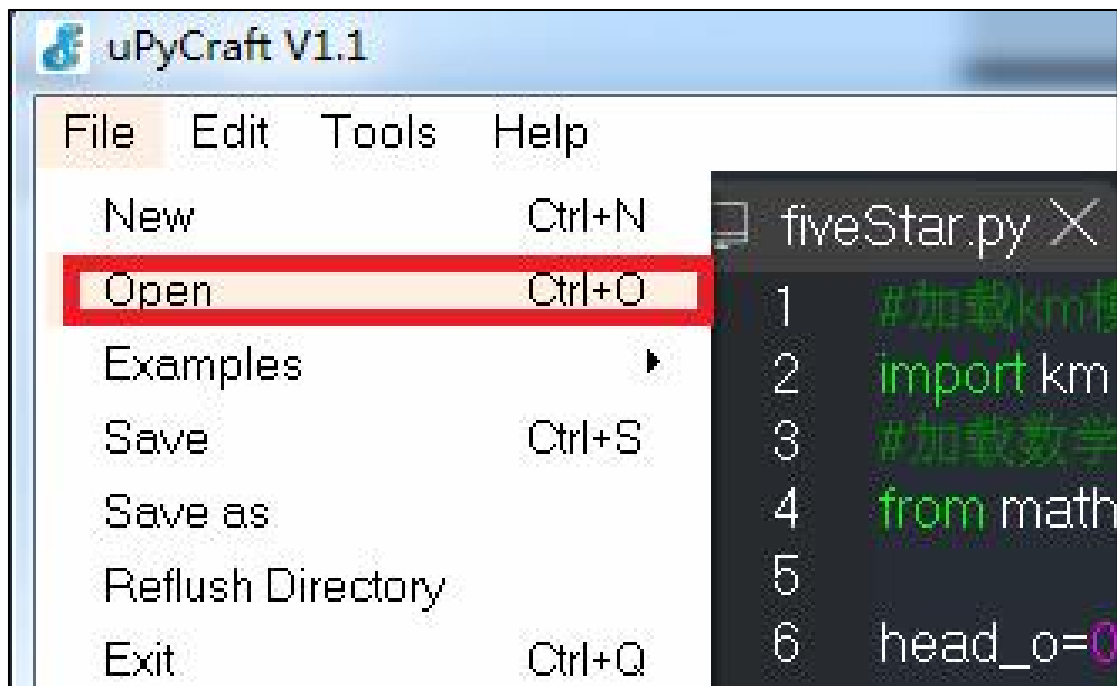
After the setting is done, you can carry out the repl operation in the debug window (same as putty in the previous section).



The following section describes how to import third-party files or modules, for example, you find a drawing module on the Internet.



For example, if you find a drawing module on the Internet, this drawing module consists of two files. Let's open these two files in turn.

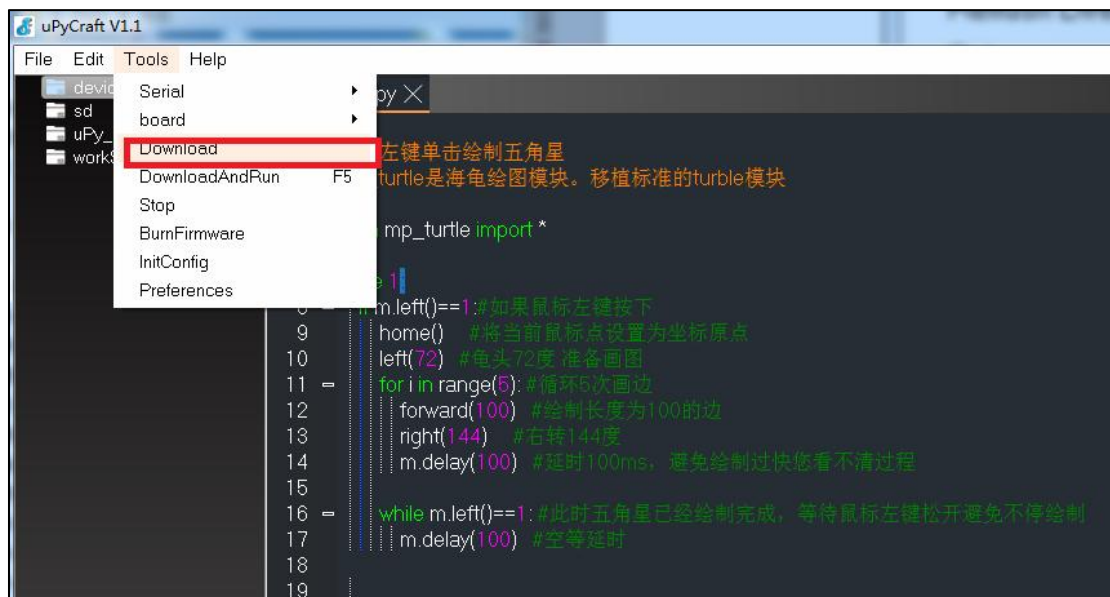


Open them as shown in the figure below:

Then download to the development board Tools -> Download . Note that download is to download, only save the file, will not run line file, the following downloadAndRun is to save and then run the file. From the source code, we can see that the dependency of fiveStar.py file is mp_turtle. Since there is no mp_turtle.py file on the board now, it will definitely run wrong.



```
__class__  __name__  gc      km
os         sys      uos      bdev
myfile
>>>
>>>
>>>
>>> import fiveStar
|
```



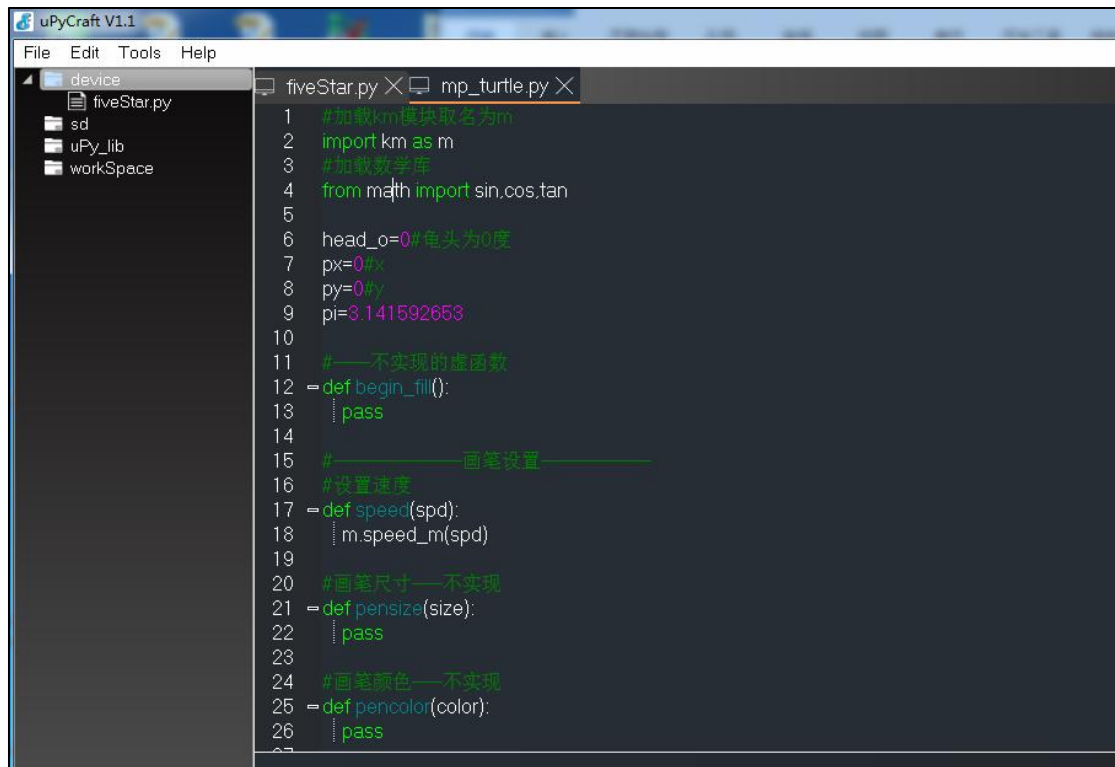
After downloading the file, the following message will appear on the console.

```
>>>
>>>

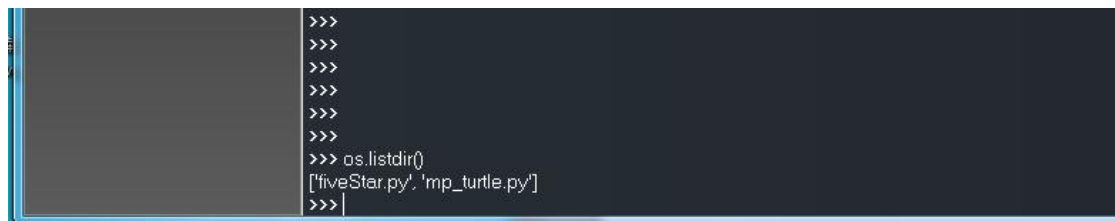
Ready to download this file, please wait!
.....
download ok
```

Similarly, open the mp_turtle.py file and burn it to the board.

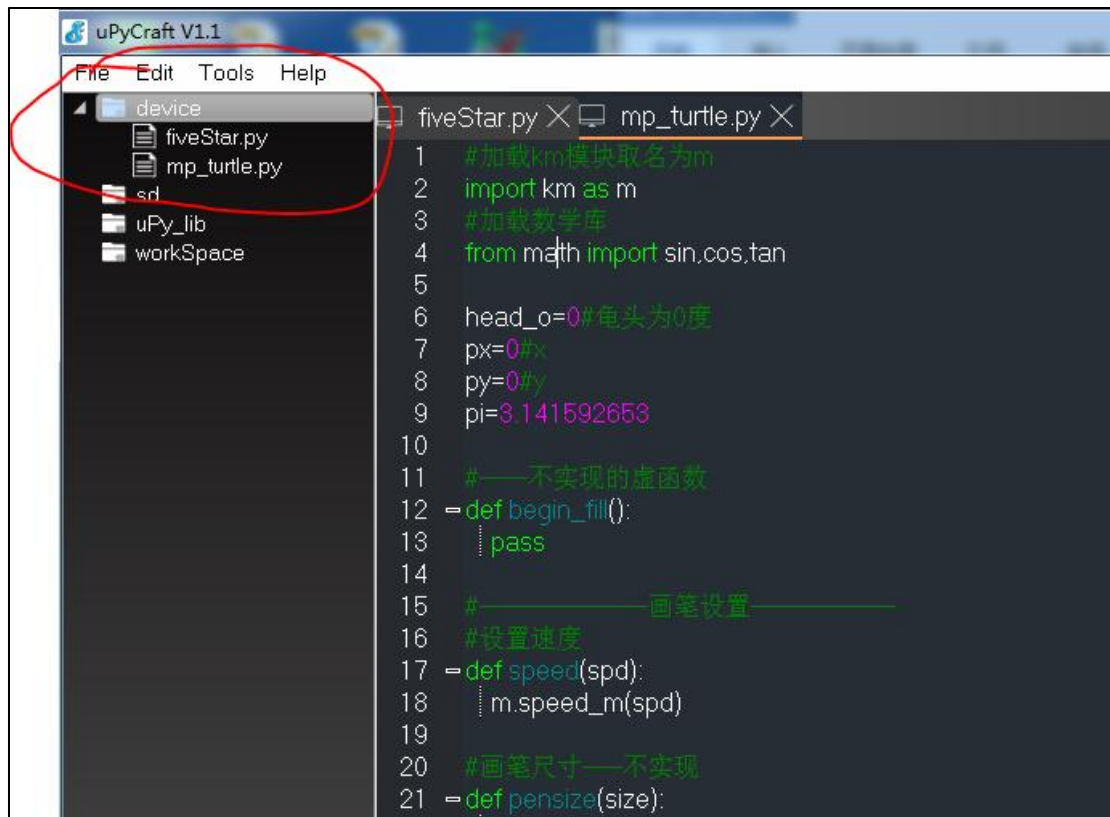
Ps: As you can see from the source file of mp_turtle.py, its underlying dependencies are the km module and the math module. These two modules are supported by the board. So all the dependencies of this script are satisfied. Then he can definitely run it.



You can see these two files on the left after the burn is done. You can also type `os.listdir()` in the console to see what files are currently mounted under the file system. This is shown in the following figure:



At this point these two files are already inside kmbox. You can call `fiveStar.py` directly to realize the function.



If you call `help('modules')` at this time, the two modules you just passed in will not appear.

`help('modules')` shows the modules compiled directly from the source code. The above file transfer is for user-defined modules.

To use an imported third-party module.

You can use this module by typing `import fiveStar` in the console.

Notice how it looks like it's dead when you run it, but it's not. Look at the source code of `fiveStar.py`.

```

1
2
3 鼠标左键单击绘制五角星
4 mp_turtle是海龟绘图模块。移植标准的turtle模块
5
6 from mp_turtle import *
7
8 =while 1:
9 = if m.left()==1: #如果鼠标左键按下
10     home() | #将当前鼠标点设置为坐标原点
11     left(72) #龟头72度,准备画图
12 = for i in range(5): #循环5次画边
13     forward(100) #绘制长度为100的边
14     right(144) #右转144度
15     m.delay(100) #延时100ms,避免绘制过快您看不清过程
16
17 = while m.left()==1: #此时五角星已经绘制完成,等待鼠标左键松开避免不停绘制
18     m.delay(100) #空等延时
19
20
21

```

In while 1 wait for your left mouse button to be pressed and then draw the five-pointed star.

The mouse needs to be plugged into the Kmbox to know whether the mouse is pressed or not. If

the mouse is connected to the computer, the computer will not tell kmbox that the mouse is pressed.

If you want to stop the script, just press ctrl+c on the console.

After the interruption, you can continue to use import fiveStar and check the member functions inside. Here's the picture

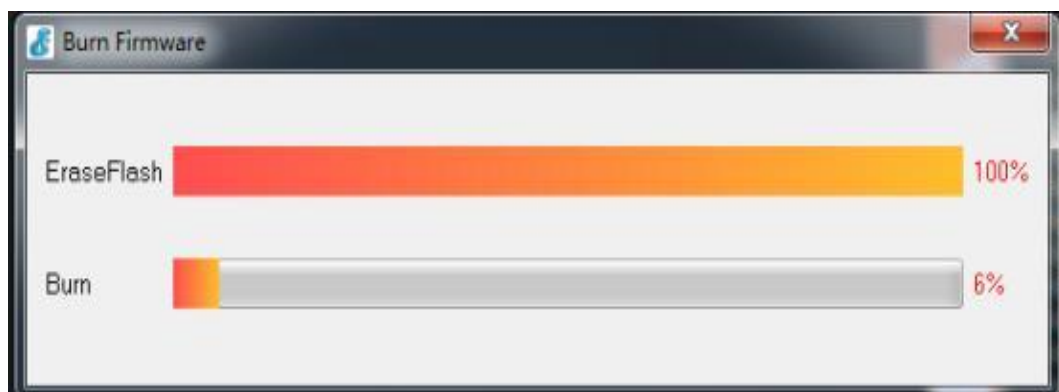
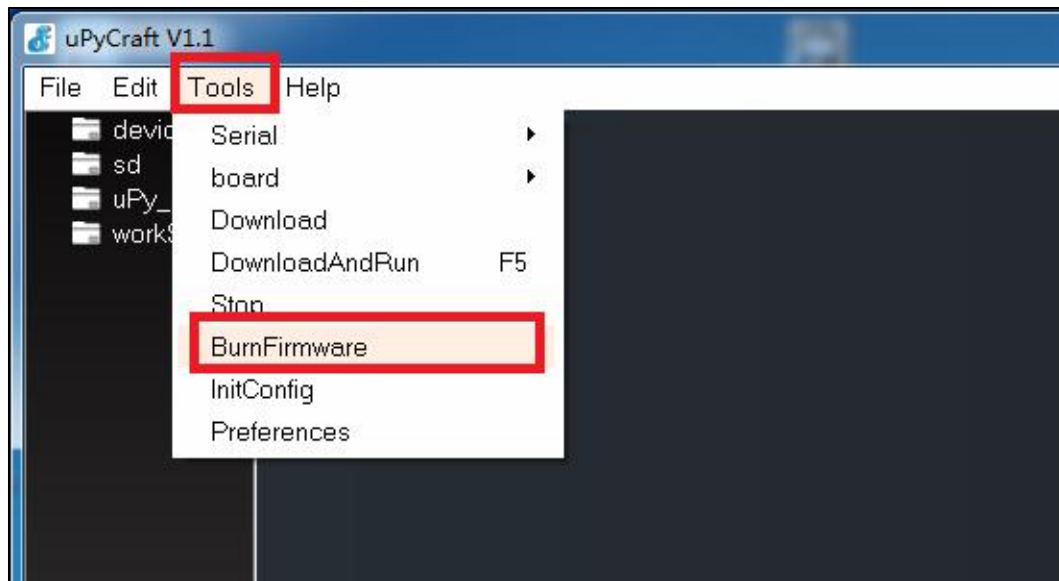
```
>>>
__class__  __name__  gc      km
os         sys      uos      bdev
myfile     fiveStar
>>> fiveStar.
__class__  __name__  __file__  cos
home       left      pi          right
sin         tan       m          forward
head_o      px        py          begin_fill
speed       pensize   pencolor   pendown
penup       backward  goto       setx
sety        setheading heading     position
>>> fiveStar|
```

As long as the source code is available and the dependencies are resolved, the script will eventually run inside the kmbox. Usually the py files burned into the board are already debugged. If it is the development stage, it is recommended to run it in memory, and burn it into the board only when it needs to be released.

V. Firmware upgrade (board repair)

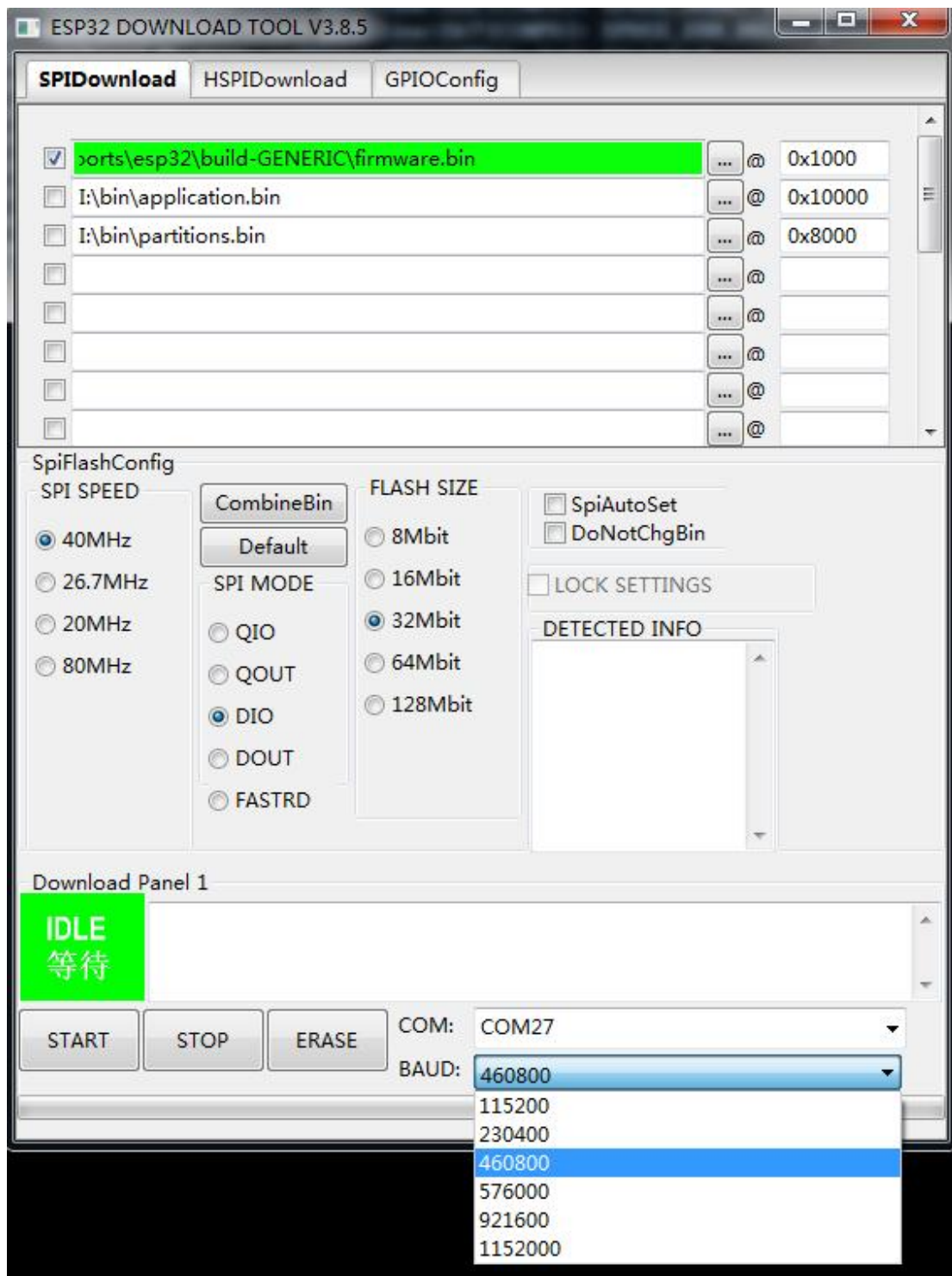
In the previous tutorials, it was described how to use the scripts and how to extend them yourself. If in case there is a problem with the burned script, it does not run properly, you can use the brush to restore. kmbox's brush there are a total of two ways to brush, be sure to power off and restart the development board after brushing (plugging and unplugging the USB).

The first way requires tools: uPyCraft, open tools->BurnFirmware. The following interface will appear, choose the kmbox firmware there, click OK.



Wait for two progress bars to finish. Power off and reboot, the KMBOX card will be restored to its original state.

The second way is `flash_download_tool_v3.8.5.exe`, which is a download tool from Loxin website. He can be used for batch burning.



VI. Kmbox Advanced Tutorial

1. How to make the script run automatically after powering up

Very often after writing a script, we want to save the script to kmbox, so that kmbox can run automatically as soon as it is powered on. Instead, we need to input commands into the console to invoke the script. In fact, this has been introduced in the previous section " [How to use third-party](#)

libraries". If you want the script to run as soon as you start the kmbox, first download the script to kmbox. First, download the script to kmbox, second, set the script to start at boot. In "**How to use third-party libraries**", it describes how to download scripts. The boot process of Kmbox is fixed. He will run the boot.py script first, and then run the main.py script. Usually boot.py contains some initialization configuration parameters of the system. So, if you want your script to run on boot, just change the name of the script to main.py or boot.py. It is recommended to use the name main.py, boot.py may be used later for other front configuration.

2. Single-threaded code optimization

In the previous tutorials, we mostly used `while True` loop and wrote detection logic in it. When the amount of logic is very small and the execution takes very little time. Basically, you don't have to think about efficiency. But sometimes you need to consider efficiency. Here is an example. Learn how to optimize the script to improve execution efficiency.

For example: there is a need for an automatic monster script, which needs to have two functions.

- 1) If you press the A key, it will automatically release ~~15~~ skills 1, 2, 3, 4 and 5.
- 2) Automatically shout every 10 seconds.

According to the previous learning, it is easy to think of judging the A key pressed with `isdown`, throw skills with `press`, every 10 seconds with `delay`. it is easy to write the code as this.

```
while True:
    if km.isdown('a'):#If a is down
        km.press('1')# throw 1 skill
        km.press('2')#Throw 2 skill
        km.press('3')#throw 3 skill
        km.press('4')#throw 4 skill
        km.press('5')#throw 5 skill
    print('I'm going to yell every 10 seconds')
    km.delay(10000)#delay 10 seconds
```

Running the code above will bring up a print every 10 seconds. But pressing a is hardly likely to bring up 12345. as shown below:

```
Hi3559AV100 | kmbox
paste mode; Ctrl-C to cancel, Ctrl-D to finish
===
while True:
===
    if km.isdown('a'):#如果a按下
===
        km.press('1')#丢1技能
===
        km.press('2')#丢2技能
===
        km.press('3')#丢3技能
===
        km.press('4')#丢4技能
===
        km.press('5')#丢5技能
===

    print('我要每隔10秒喊话一次')
===
    km.delay(10000)#延迟10秒
===

我要每隔10秒喊话一次
我要每隔10秒喊话一次
我要每隔10秒喊话一次
```

Actually, it is because of the instruction `km.delay(1000)`, because the code is running line by line. `km.delay(10000)` takes 10s from start to return. Because the code runs line by line, the `km.delay(10000)` instruction takes 10s to run from start to return. During these 10 seconds, the CPU does nothing but wait for these 10 seconds to finish before running the `km.isdown('a')` instruction. So if you happen to press the a key when `kmbox` executes `isdown('a')`, `kmbox` will only execute the 12345 instructions.

Let's talk about how to optimize the code. If you remove the delay, you will see that the main loop can reach hundreds of thousands of times a second. So it is equivalent to detecting `isdown` hundreds of thousands of times per second. According to Nyquist Sampling Theorem, if the sampling frequency is more than 2 times, the sampling information will not be lost. If the main loop runs 10,000 times a second. This means that the data will not be lost even if you press the button 5000 times per second. In reality, you can't press a key more than 50 times a second. So in theory, as long as the main loop can run 100 times per second, basically no keystrokes will be lost. So to make sure that a key press is not lost. We should try to increase the number of calls to `isdown` as much as possible, so that it is more than 100. So here's the first way, short polling.

```

import time # for time counting
last_time=0 # Record the time of the call.
while True:
    if km.isdown('a'):# if a down
        km.press('1')# throw 1 skill
        km.press('2')#Throw 2 skills
        km.press('3')#Throw 3 skills
        km.press('4')#throw 4 skill
        km.press('5')#throw 5 Skill
    if time.time()-last_time>=10:
        print('I want to shout every 10 seconds ')
        last_time= time.time()#refresh the time of the last call
    #last_time= time.time()#refresh the time of the last yell.

```

Comparing the previous code, the above code is tens of thousands of times more efficient. This code wastes no CPU time. It uses an if statement to make a quick judgment on the 10-second delay. When the time is up, it prints. The above code runs at least 10,000 times per second. There is no chance that a press is not detected. So, at this point you see that not only does it immediately execute 12345 when you press a, but it also prints the data every 10 seconds. But there is still a problem with this code. The problem is that it runs too fast. It will do things you don't expect. You can see the result by running it yourselves on the board. When you find the problem, you can figure out how to fix it. If it runs too fast, just find a way to make it not run so fast. Different people have different ways to realize the same function. There is no best, as long as you can achieve the needs on the line.

3. Multi-threaded use

In the previous section of the automatic monster script as an example. How to use multi-threaded method to realize. Need an automatic monster fighting script, the script needs to have two functions:

- 1) If you press the A key, it will release the 5 skills 1,2,3,4,5 automatically.
- 2) Automatically shout every 10 seconds.

Logically speaking, shouting every 10 seconds has nothing to do with releasing 12345 if A is pressed. They can be seen as two separate things. You can write a function A that detects if A is pressed. A function B to let him automatically shout, and then these two functions "run at the same time". This is where we come to multithreading.

Although logically A, B two functions are running at the same time. But there is only one CPU. Any time A is running, B is not running (except for multi-core CPUs). What appears to be simultaneous running is actually serial AB alternating. It's just that they're alternating so fast that you can't tell. Here's an example of multithreading:

```

"""
kmbox supports multi-threaded operation, the example is as follows:
Those that need to be processed quickly are placed in thread A.
Time-consuming operations are in thread B.
A and B are logically independent. They run concurrently.
import thread #Introduce the multithreading module.
import time #Introduce the time module

```

```

keyval a=km.getint('a') #a
km.mask(keyval a,2) #mask monitor a and add to catch queue via catch kb

#A thread - fast response to a task
def thread_A():
    while 1:
        retVal=km.catch kb() #Non-blocking capture of keystroke data
        if retVal==keyval a: #capture a press 1 time
            print('a pressed... Execute logic after a press, e.g. one-key N-skill.')
#B thread - time-consuming task
def thread_B(): #B threading function
    while 1: #B thread dead loop
        print('B thread running.... .cputick=',time.ticks ms())
        time.sleep(1.5) #1.5 seconds to sleep
thread.start_new_thread(thread_A, ()) #start thread A
thread.start_new_thread(thread_B, ()) #start thread B

```

You can see that no matter how much delay there is in B, it doesn't affect the detection of the A function, which runs very smoothly. There's a lot more to multithreading than that. There are also inter-thread synchronization, mutexes, and so on. I won't go into all of them here.

PS: The above code does not write thread exit, please improve yourselves.

4. Script Encryption and Authorization

In the previous sections, our scripts are in plaintext. In other words, as long as you have the py file, you can modify and use it without restriction. You can modify and use it without restriction. However, in many cases, you don't want to make your scripts available to the public so easily. So you can encrypt the script and give the encrypted file to the user. This way you don't have to worry about the source code leaking. And you can also authorize the user to charge by the time or the number of charges can be.

Let's take a simple example: suppose this is a one-shot deal. You implement a certain feature and sell it to the user. You use password authentication. The user buys a key from you. If the key is correct, you can use it, if not, you can't. For example, the plaintext source code looks like this (test.py)

```

import km
IsAuthor=0 # Is authorization initialized to --- Authorization conditions can be defined by
yourself, e.g., key, machine code, time, etc. 0 time, etc. Here is a simple demo
def author( key):
    global IsAuthor
    if key=='123456': #Set the script key to 123456.
        IsAuthor=1 #Allow the run function if the key is correct.
        print('Authorization code is correct')
    else:
        print('The authorization code is incorrect! Please contact the author of the script:
xxx@xx.com')
def run():

```

```

if IsAuthor==0: #The key is incorrect
    print("Sorry you are not authorized, please call author to enter the registration code ")
    return
while 1:
    print("Script is running with authorization...")
    km.delay(1000)

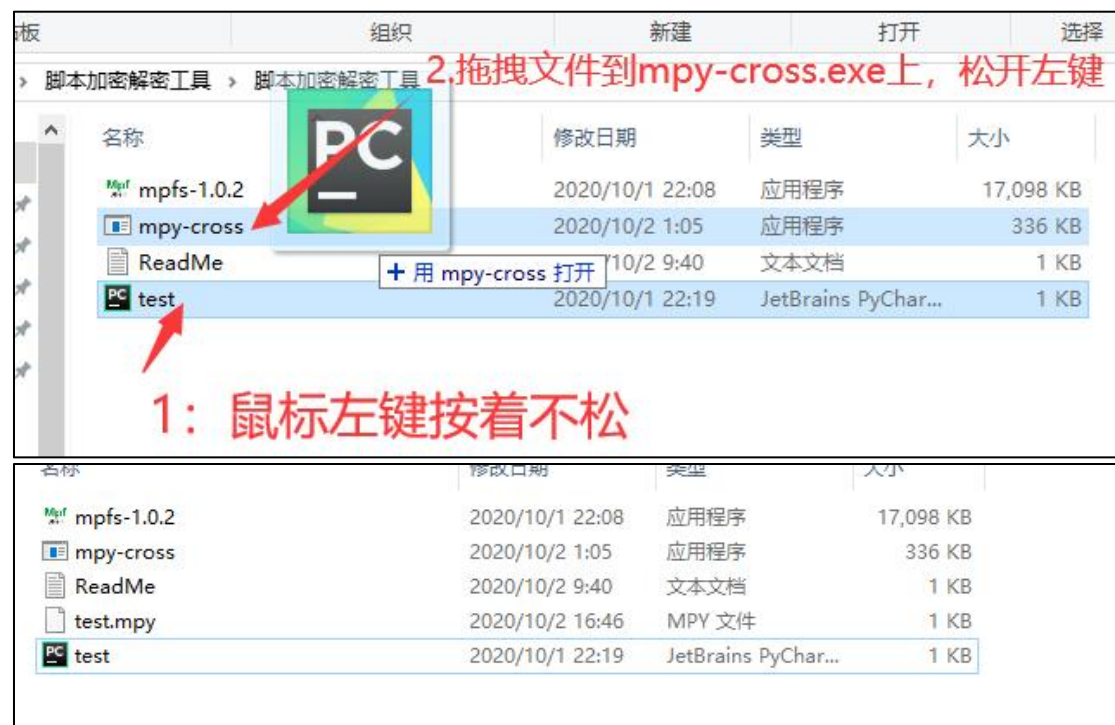
```

Suppose the key string for the script is 123456. The first function, author, is used for authorization. If authorization is successful, it sets the IsAuthor flag. When the run function is executed, it checks whether authorization has been granted. If not authorized, the function will not run. To protect the author's intellectual property, we need to encrypt the script using the mpy-cross tool, in a Windows environment. The encryption method can be chosen freely, but after encryption, only the author should know how to decrypt it. Now, let's explain how to encrypt and use the script from both the author's and the user's perspectives.

5. Script author encryption script

Method 1: Directly drag and drop the file to be encrypted (recommended)

Compile mpy_cross, drag and drop the encrypted script directly onto mpy-cross.exe and release the left mouse button. The mpy file will be automatically generated in the folder where the exe is located:



Method 2: Compile mpy-cross and generate it under python using the command

Step 1: The tool you need for encryption is **mpycross** (build your own python environment) - note that the version number is 1.11. Other versions may not work.

```

C:\Users\hw\MPY>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

```

To install it, run this command: `pip install mpy-cross==1.11` The installation process is shown below:

```
C:\Users\hw\MPY>pip install mpy-cross==1.11
Defaulting to user installation because normal site-packages is not writeable
Collecting mpy-cross==1.11
  Downloading mpy_cross-1.11-py2.py3-none-win_amd64.whl (151 kB)
    | 151 kB 5.8 kB/s
Installing collected packages: mpy-cross
Successfully installed mpy-cross-1.11
```

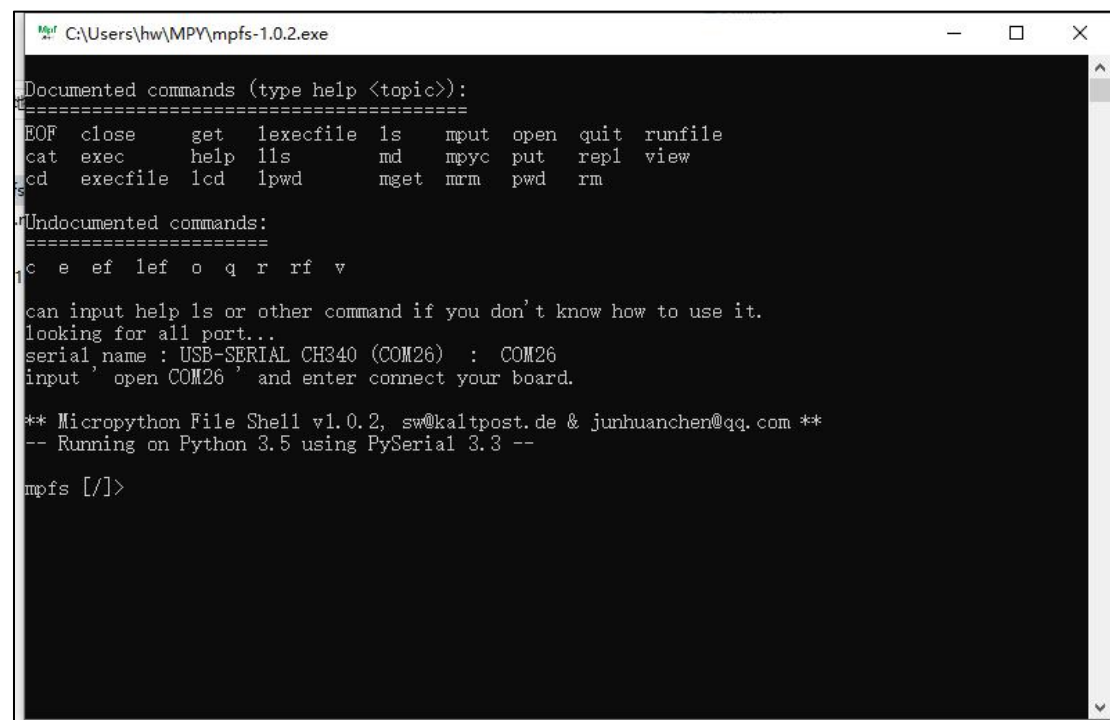
Once installed, you can convert the plaintext py script to a ciphertext mpy file.

Execute the following command: `python -m mpy_cross test.py` ----- The yellow part of test.py is the plaintext file to be converted. After executing this command, the test.py file will automatically generate a test.mpy file. This test.mpy is the ciphertext file. This is the ciphertext file that you can distribute to your clients. You can compare the test.py and test.mpy files. test.mpy is a smaller file and is more efficient than test.py. And test.mpy is invisible. You don't have to worry about source code leakage.

PS: Script encryption actually converts the source code to machine code, so the mpy file is smaller than the original file and more efficient than the py file.

6. Users using encrypted files

The user can use an encrypted file (mpy) in the same way as a normal plaintext file (py). You can think of mpy as the equivalent of py, except that the user does not know what is in the mpy file. To use a mpy file, you need to burn it into kmbox, and then Import it to call the functions in it. Upcraft is not able to transfer mpy files (because the author didn't consider mpy files in the first place). Here we introduce a new tool, Mpfs, whose function is to download any type of file into kmbox. You can think of it as a downloader, double-click it and run it as shown in the picture below:



```
C:\Users\hw\MPY\mpfs-1.0.2.exe
Documented commands (type help <topic>):
=====
EOF  close  get  lexecfile  ls  mput  open  quit  runfile
cat  exec   help  lls       md  mpyc  put  repl  view
cd   execfile  lcd  lpwd     mget mrm  pwd  rm

Undocumented commands:
=====
c e ef lef o q r rf v

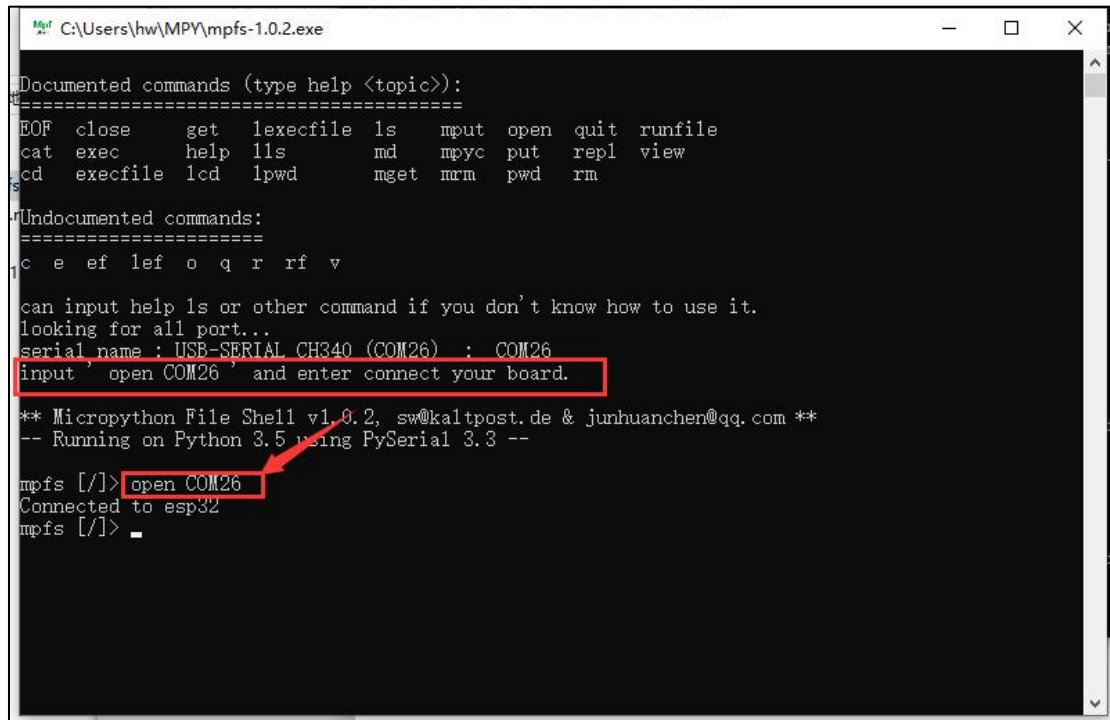
can input help ls or other command if you don't know how to use it.
looking for all port...
serial name : USB-SERIAL CH340 (COM26) : COM26
input 'open COM26' and enter connect your board.

** Micropython File Shell v1.0.2, sw@kaltpost.de & junhuanchen@qq.com **
-- Running on Python 3.5 using PySerial 3.3 --

mpfs [/]>
```

To use this software, you need the board to be in repl mode. If you don't know how to enter the repl mode, you can just refresh the kmbox directly. The default is repl mode.

Since mpy needs to be placed in the internal file system of the board, you need to connect to kmbox. So you need to connect to kmbox, type: open COMxx --- connect to the board (COMxx is the serial number).



```
C:\Users\hw\MPY\mpfs-1.0.2.exe

Documented commands (type help <topic>):
=====
EOF  close      get      lexecfile  ls      mput     open     quit      runfile
cat  exec        help     lls        md      mpyc     put      repl      view
cd   execfile    lcd      lpwd       mget    mrm      pwd      rm

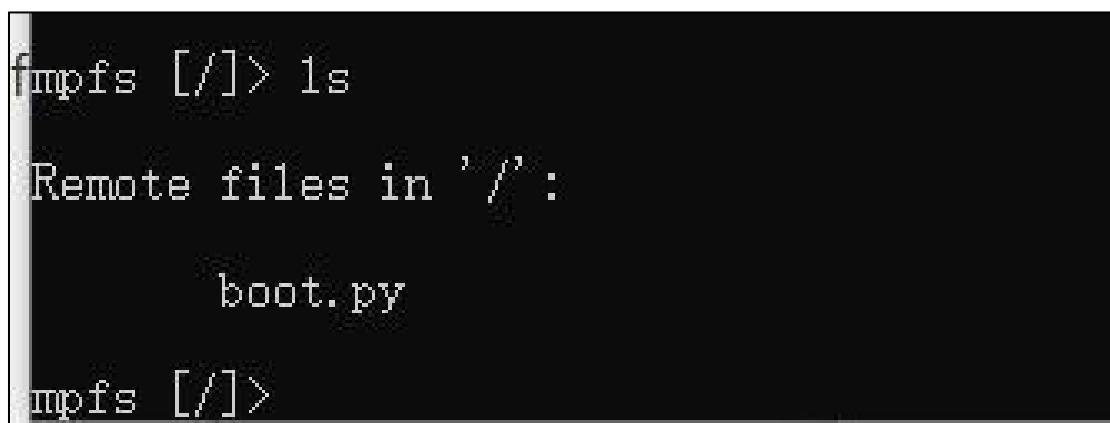
Undocumented commands:
=====
c e ef lef o q r rf v

can input help ls or other command if you don't know how to use it.
looking for all port...
serial name : USB-SERIAL_CH340 (COM26) : COM26
input 'open COM26' and enter connect your board.

** Micropython File Shell v1.0.2, sw@kaltpost.de & junhuanchen@qq.com **
-- Running on Python 3.5 using PySerial 3.3 --

mpfs [/]> open COM26
Connected to esp32
mpfs [/]> _
```

After the connection is done, you can see that the connection is successful. Here you can enter commands like ls mv, etc. as you would do in linux.



```
mpfs [/]> ls

Remote files in '/':

    boot.py

mpfs [/]>
```

For example, after ls, you can see that there is a boot.py script inside the current file system. Kmbox has an internal file system that can store various types of files. You can use the os library to create files, change file names, and so on. Now you need to write the mpy file to the kmbox file system.

You only need to execute one command: put test.mpy


```
mpfs [/]> put test.mpy
test.mpy
transfer 378 of 378
mpfs [/]>
```

When executing this command, note that test.py and mpfs.exe need to be in the same directory:

名称	修改日期	类型	大小
mpfs-1.0.2	2020/10/1 22:08	应用程序	17,098 KB
test.mpy	2020/10/2 1:05	MPY 文件	1 KB
test	2020/10/1 22:19	JetBrains PyChar...	1 KB

At this point, the mpy file has been burned into the kmbox. Now mpfs can be closed. The next step is to run the encryption script. How to use it is the same as a normal py file, just import test.

```
>>>
>>> import test
>>> test.
__class__      __name__      __file__      km
run            IsAuthor     author
>>> test.█
```

As you can see, all the functions written by the author are here. You can call them. From the source code of the author's side, we know that you have to call author first and pass the key 123456 in order to use the run function. Let's try calling run as follows:

```
>>>
>>> import test
>>> test.
__class__      __name__      __file__      km
run            IsAuthor     author
>>> test.run()
对不起您还未授权，请调用author输入注册码
>>>
```

Let's say I paid a dollar from the author to know that the key is 123456, and I call author('123456').

```
>>>
>>> import test
>>> test.
__class__      __name__      __file__      km
run            IsAuthor     author
>>> test.run()
对不起您还未授权，请调用author输入注册码
>>> test.author('123456')
授权码正确
>>>
```

Once the key is correct, I can use the run function.

```

>>>
>>>
>>> import test
>>> test.
__class__      __name__      __file__      km
run            IsAuthor      author
>>> test.run()
对不起您还未授权，请调用author输入注册码
>>> test.author('123456')
授权码正确
>>>
>>>
>>> test.run()
脚本已授权正在运行中...
脚本已授权正在运行中...
脚本已授权正在运行中...
脚本已授权正在运行中...
脚本已授权正在运行中...
脚本已授权正在运行中...
脚本已授权正在运行中...
脚本已授权正在运行中...
脚本已授权正在运行中...
脚本已授权正在运行中...

```

Of course, the above code is just a demo. isAuthor is given directly to the variable, which should be encapsulated as a class and set as a private variable. The encryption is also not a simple string. You can use machine code to bind it. The key is different for each device. The rules are up to you.

VII. Prevention of blocking

1. Hardware Anti-Blocking

This chapter mainly explains how to avoid hardware sealing. Although kmbox belongs to the pure hardware physical plugin, you can't block kmbox unless you are not allowed to use keyboard and mouse, as I said before. You should know that USB devices have VIDs and PIDs, and if some games have already torn their faces off by not allowing the specified VIDs and PIDs to play the game, then there is nothing you can do about it. There's nothing you can do about it. After all, the game company has the right to do so. Back then, 360 and QQ disliked each other! But the general game companies do not dare to do so. Because devices with VIDs and PIDs are legal devices. If you disable a VID, millions of devices all over the world may not be able to use it, but the VID and PID of kmbox are regular and legal. The VID and PID of the kmbox are valid and legal. The VID and PID may also be used for other standard keyboards and mice. If a gaming company limits the VID and PID, it will block all other types of keyboards and mice that use the IC. It's not worth the effort to hurt a lot of people just to mess with kmbox. So very few games do this. But if they do, how do you fix it.

Then change the VID and PID. Note: VID and PID need to be paid to the USB Association, please do not change them, and using other companies' VID and PID without authorization is a violation of the law. You will be responsible for the consequences.

Check the VIDs and PIDs of major vendors: <http://www.linux-usb.org/usb.ids>

For example, the following Logitech VID=046d, Toshiba VID=046C, kmbox VID=4348 046b American Megatrends, Inc.

0001 Keyboard

0101 PS/2 Keyboard, Mouse & Joystick Ports
0301 USB 1.0 Hub
0500 Serial & Parallel Ports
ff10 Virtual Keyboard and Mouse
046c Toshiba Corp., Digital Media
Equipment 046d Logitech, Inc.
0082 Acer Aspire 5672 Webcam
0200 WingMan Extreme Joystick
0203 M2452 Keyboard
0242 Chillstream for Xbox 360
0301 M4848 Mouse
0401 HP PageScan
0402 NEC PageScan
4348 WinChipHead
5523 USB->RS 232 adapter with Prolific PL 2303 chipset
5537 13.56Mhz RFID Card Reader and Writer
5584 CH34x printer adapter cable

If the kmbox (4348) is blacklisted, we can manually change the VID to 046D and the computer will think that the device connected is a Logitech product and not a kmbox. Changing the VID and PID is a violation of the law, unless the VID and PID belong to you. The following is for reference only. The VID is the vendor number and the PID is the product number.

How to change VID

Vid belongs to the manufacturer number of USB device, which is assigned by USB association. Kmbox default VID is 0x4348. To read or write VID, you need to use device module. First import device You can input the following command to check the VID.



Type `device.VID('help')` to see how to use the VID.

```
>>> device.VID()
VID=4348
>>> device.VID('help')
自定义Kmbox的VID:
  kmbox的默认VID是4348.VID相当于设备的身份证,不可以随意修改.主机会根据VID匹配不同的驱动
  有的游戏检测到特定VID后会禁用设备数据,此时可以通过修改VID绕开这个屏蔽,修改VID是侵权行为。
  请不要乱用别人的VID。本函数仅供学习使用。修改VID方法如下:
  device.VID()      : 无参数是查询当前VID
  device.VID('046D'): 有参数是设置VID=046D(戴尔设备)
  设置完VID后需要调用enable(1)函数重新枚举
>>> █
```

For example, now change VID=046D (to fake a Logitech product).

```
>>> device.VID('046D')
自定义VID=046d,启用新参数请调用enable(1)函数
>>> █
```

Changing the VID does not take effect immediately, you need to call `device.enable(1)`. To allow the computer to re-recognize the USB device, you need to call `device.enable(1)`. After calling Your computer will re-enumerate it.



This way you can perfectly disguise yourself as a Logitech keyboard or mouse. The original VID blocking for the 4348 will of course be gone. Again, VID infringement is illegal, please do not modify it at your own discretion, otherwise you will be responsible for the consequences.

How to change PID

PID is the product ID, which is used to distinguish different products of the same company. For example, Logitech has various models of keyboards and mice. For example, Logitech has various models of keyboards and mice, and their VIDs are the same, so they use PID to differentiate them, and the modification method of PID is the same as that of VID. Kmbox default PID=0x1234.

```
>>> device.PID('help')
自定义Kmbox的PID:
  PID是产品ID,不同的PID可以区分同一厂商的不同设备,便于主机加载不同驱动
  device.PID()      : 无参数是查询当前PID
  device.PID('046D'): 有参数是设置PID=046D
  设置完PID后需要调用enable(1)函数重新枚举
>>> █
```

Kmbox default PID=0x1234. Please try by yourself.

How to Change USB Device Name

When Kmbox is plugged into the computer, the default name is kmbox, some people said the anti-hacker mechanism may restrict the device according to the device name. So how to change the name? Actually, you should know that most of the keyboards and mice don't have the character descriptor of the device in order to save the cost. In fact, you should know that most of the keyboards and mice do not have the character descriptor of the device (they need ROM to store it) in order to save the cost. By default, there is no device name. What can you do if you really want to disable a device according to its name? As long as you enable any one of the two functions VID and PID, the character description of Kmbox will be cleared. The host PC will not read the device name. If you can't read the name, you can't block it. And I think it's unlikely to recognize the device by its name. If I were an anti-plugin programmer, I would start with VID and PID. Behavioral detection is the main focus, so I don't provide a function to modify the name.

2. Software Anti-Blocking

In the previous hardware anti-sealing can be solved in the hardware causes sealing, but there is a saying in the field of hardware: anti-dull not anti-stupid. Many seals are not caused by hardware, but by software. As an example, you made a 500 times per second mouse pointing wildly continuous pointing device, lasted 1 day. It's impossible for a person to think about it, first, 500 times per second, ask which god can do it. Can point 10 times per second has been inhuman. 500 times do not even have to think about it, is absolutely plug-in. Secondly, it lasts for a day, which is also unlikely. It is also unlikely that the high intensity of the clicks will last for a day. So write hardware hang to try to simulate human operation. If you want to use hardware to hang, you should act like a bit, don't be so fake, and restrain your own behavior. When writing scripts, it is best to follow the actual situation, more random numbers, more simulation process, everything is controlled within a reasonable range. Here can only slowly write their own script to experience.

To give a few examples of often easy to make mistakes:

1. the delay is too accurate (each keyboard and mouse events delay pattern is too accurate, unlike human operation.)
2. key time is too fast (kmbox in order to improve the efficiency of the code, a key command will be the fastest speed of execution about 1ms, however, the actual key, from press to lift, the time is at least 80ms above. Please try to use down and up to simulate the key, less press)
3. The span of mouse movement is too big. (Try to split the large span of mouse movement into small span of movement, less use move(800,600) such code, but use move(1,1), x direction cycle 800 times, y direction cycle 600 times.)

All in all, try to act like a human being is doing it.

3. Behavioral mimicry in kmbox

In the previous section on behavior detection, it was emphasized again and again that you should be able to pretend to use the plugin, and try to act like a human being. Kmbox provides some simple and useful APIs to prevent your operation from being too mechanized.

1. Delay is too accurate.

For this, kmbox provides a random delay function. See delay function usage

2. Key press time is too fast.

For this problem, kmbox provides a press function, you can press with three parameters, used to specify the time of key press. See press function usage.

3. the span of mouse movement is too big.

Kmbox has built-in move and moveto functions to support process simulation. You don't need to write complex move logic, just give a few parameters, kmbox can automatically make the mouse move into a smooth curve. You no longer need to worry about the mechanical operation of straight to straight, horizontal and vertical, instead of the same starting point to the end of the movement, kmbox can be smooth and continuous over. Kmbox's smoothing principle is based on Bezier curves (if you need to know the principle, please go to Baidu).

Let's talk more about the move and moveto functions. move and moveto, if given only two arguments, are used to quickly move to a given coordinate at once. If given only two arguments, move and moveto are one-time quick moves to the given coordinates. If you give them three arguments, you can use several polylines to move from the starting point to the end point. If you give three arguments, it means that you have to fit several polylines from the start to the end. Common sense dictates that the larger the value of the third parameter, the smoother the fitted curve will be. And there are countless curves over two points. So when no other auxiliary qualifiers are given, every call to the function will result in a different curve. This is the random curve of kmbox. If you want to control the degree of distortion of the curve, you can give a reference point coordinate. This reference point coordinate can modify the direction of the curve's distortion, so that the curve is distorted towards the reference point. So the move and moveto functions take either three or five arguments, three to fit a random Bézier curve, and five to fit a Bézier curve closer to the reference point. You can run the script below to see what each parameter does.

To re-emphasize:

One, using multiple lines to fit a curve way is definitely slower than a single horizontal and vertical. Currently, kmbox takes 1ms to fit each additional line. The larger the length of the fit, the smoother the curve, but the higher the time consumed. The smaller the length of the fit, the more obvious the inflection steps are. Therefore, it is better to fit the curve according to the actual situation. It is recommended to use 127 as the maximum scale unit to calculate how many straight lines are needed. For example, the maximum span from (0, 0) to (1920, 1080) is 1920 units. Then divide 1920 by 127 is equal to 15.118. Therefore, it is recommended to fit a straight line with a minimum value of 16.

Second, at present, kmbox is not subject to any restrictions, unless there is a need to simulate the process can be used to simulate the process, otherwise the normal situation of how to write the script on how to write.

Third, if you do not know how to use the reference point to constrain the curve. It is recommended to use a function with three parameters directly. Kmbox will automatically randomly select a reference point within the range of the starting point and the end point, and automatically fit the curve according to the fitting parameters.

VIII. Appendix

Appendix 1 Keyboard Key Value Correspondence Table

This table shows the keyboard HID data and the corresponding values of the keys, which is the content of the table function. The left side is the key name and the right side is the HEX value. If you use a key that is not in the table function, remember to convert the HEX to decimal.

#define KEY_NONE	0x00
#define KEY_ERRORROLLOVER	0x01
#define KEY_POSTFAIL	0x02
#define KEY_ERRORUNDEFINED	0x03
#define KEY_A	0x04
#define KEY_B	0x05
#define KEY_C	0x06
#define KEY_D	0x07
#define KEY_E	0x08
#define KEY_F	0x09
#define KEY_G	0x0A
#define KEY_H	0x0B
#define KEY_I	0x0C
#define KEY_J	0x0D
#define KEY_K	0x0E
#define KEY_L	0x0F
#define KEY_M	0x10
#define KEY_N	0x11
#define KEY_O	0x12
#define KEY_P	0x13
#define KEY_Q	0x14
#define KEY_R	0x15
#define KEY_S	0x16
#define KEY_T	0x17
#define KEY_U	0x18
#define KEY_V	0x19
#define KEY_W	0x1A
#define KEY_X	0x1B
#define KEY_Y	0x1C
#define KEY_Z	0x1D
#define KEY_1_EXCLAMATION_MARK	0x1E
#define KEY_2_AT	0x1F
#define KEY_3_NUMBER_SIGN	0x20
#define KEY_4_DOLLAR	0x21
#define KEY_5_PERCENT	0x22

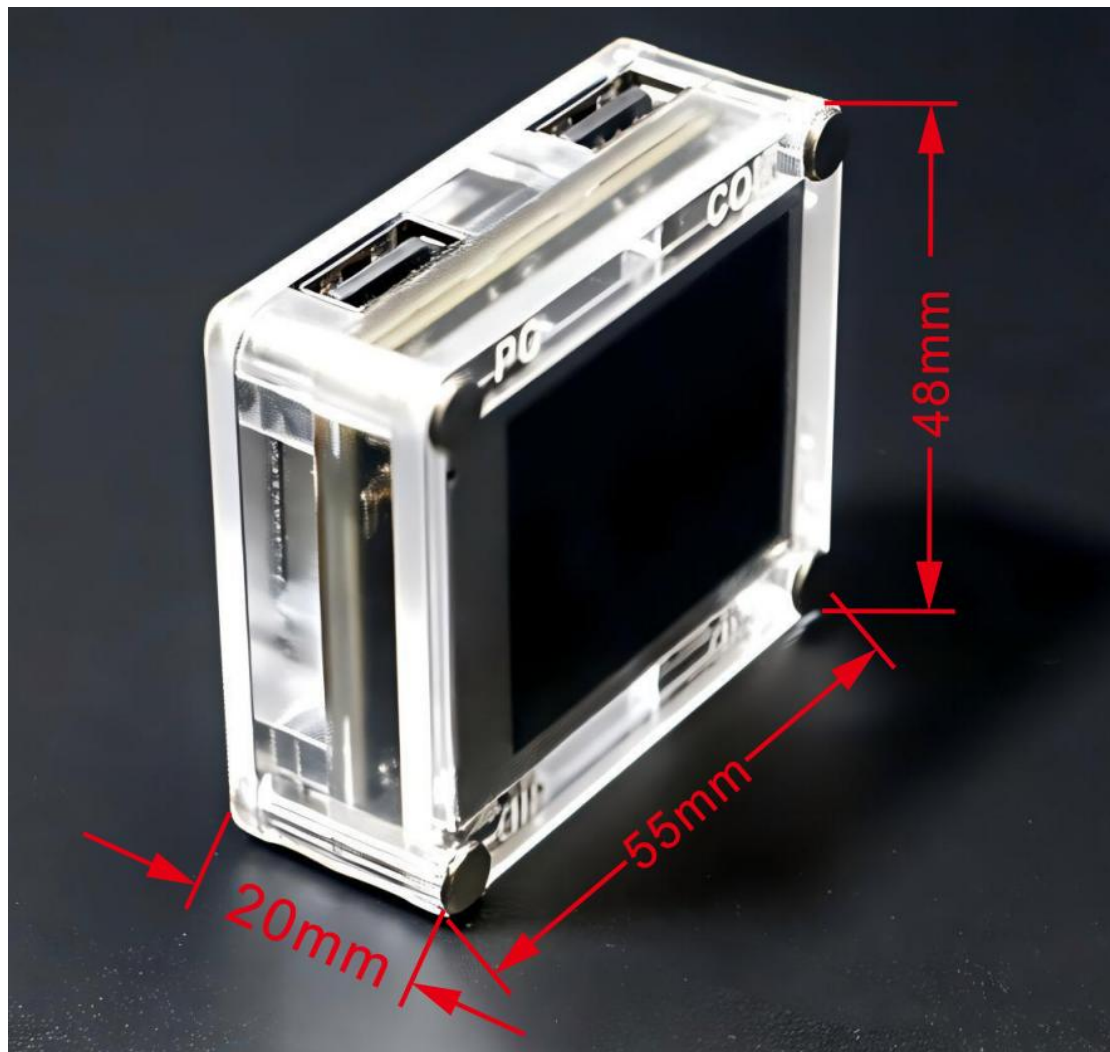
#define KEY_6_CARET	0x23
#define KEY_7_AMPERSAND	0x24
#define KEY_8_ASTERISK	0x25
#define KEY_9_OPARENTHESIS	0x26
#define KEY_0_CPARENTHESIS	0x27
#define KEY_ENTER	0x28
#define KEY_ESCAPE	0x29
#define KEY_BACKSPACE	0x2A
#define KEY_TAB	0x2B
#define KEY_SPACEBAR	0x2C
#define KEY_MINUS_UNDERSCORE	0x2D
#define KEY_EQUAL_PLUS	0x2E
#define KEY_OBRACKET_AND_OBRACE	0x2F
#define KEY_CBRACKET_AND_CBRACE	0x30
#define KEY_BACKSLASH_VERTICAL_BAR	0x31
#define KEY_NONUS_NUMBER_SIGN_TILDE	0x32
#define KEY_SEMICOLON_COLON	0x33
#define KEY_SINGLE_AND_DOUBLE_QUOTE	0x34
#define KEY_GRAVE_ACCENT AND TILDE	0x35
#define KEY_COMMA AND LESS	0x36
#define KEY_DOT_GREATER	0x37
#define KEY_SLASH_QUESTION	0x38
#define KEY_CAPS_LOCK	0x39
#define KEY_F1	0x3A
#define KEY_F2	0x3B
#define KEY_F3	0x3C
#define KEY_F4	0x3D
#define KEY_F5	0x3E
#define KEY_F6	0x3F
#define KEY_F7	0x40
#define KEY_F8	0x41
#define KEY_F9	0x42
#define KEY_F10	0x43
#define KEY_F11	0x44
#define KEY_F12	0x45
#define KEY_PRINTSCREEN	0x46
#define KEY_SCROLL_LOCK	0x47
#define KEY_PAUSE	0x48
#define KEY_INSERT	0x49
#define KEY_HOME	0x4A
#define KEY_PAGEUP	0x4B
#define KEY_DELETE	0x4C
#define KEY_END1	0x4D
#define KEY_PAGEDOWN	0x4E
#define KEY_RIGHTARROW	0x4F
#define KEY_LEFTARROW	0x50
#define KEY_DOWNARROW	0x51
#define KEY_UPARROW	0x52

#define KEY_KEYPAD_NUM_LOCK_AND_CLEAR	0x53
#define KEY_KEYPAD_SLASH	0x54
#define KEY_KEYPAD_asteriks	0x55
#define KEY_KEYPAD_MINUS	0x56
#define KEY_KEYPAD_PLUS	0x57
#define KEY_KEYPAD_ENTER	0x58
#define KEY_KEYPAD_1_END	0x59
#define KEY_KEYPAD_2_DOWN_ARROW	0x5A
#define KEY_KEYPAD_3_PAGEDN	0x5B
#define KEY_KEYPAD_4_LEFT_ARROW	0x5C
#define KEY_KEYPAD_5	0x5D
#define KEY_KEYPAD_6_RIGHT_ARROW	0x5E
#define KEY_KEYPAD_7_HOME	0x5F
#define KEY_KEYPAD_8_UP_ARROW	0x60
#define KEY_KEYPAD_9_PAGEUP	0x61
#define KEY_KEYPAD_0_INSERT	0x62
#define KEY_KEYPAD_DECIMAL_SEPARATOR_DELETE	0x63
#define KEY_NONUS_BACK_SLASH_VERTICAL_BAR	0x64
#define KEY_APPLICATION	0x65
#define KEY_POWER	0x66
#define KEY_KEYPAD_EQUAL	0x67
#define KEY_F13	0x68
#define KEY_F14	0x69
#define KEY_F15	0x6A
#define KEY_F16	0x6B
#define KEY_F17	0x6C
#define KEY_F18	0x6D
#define KEY_F19	0x6E
#define KEY_F20	0x6F
#define KEY_F21	0x70
#define KEY_F22	0x71
#define KEY_F23	0x72
#define KEY_F24	0x73
#define KEY_EXECUTE	0x74
#define KEY_HELP	0x75
#define KEY_MENU	0x76
#define KEY_SELECT	0x77
#define KEY_STOP	0x78
#define KEY_AGAIN	0x79
#define KEY_UNDO	0x7A
#define KEY_CUT	0x7B
#define KEY_COPY	0x7C
#define KEY_PASTE	0x7D
#define KEY_FIND	0x7E
#define KEY_MUTE	0x7F
#define KEY_VOLUME_UP	0x80
#define KEY_VOLUME_DOWN	0x81
#define KEY_LOCKING_CAPS_LOCK	0x82

#define KEY_LOCKING_NUM_LOCK	0x83
#define KEY_LOCKING_SCROLL_LOCK	0x84
#define KEY_KEYPAD_COMMA	0x85
#define KEY_KEYPAD_EQUAL_SIGN	0x86
#define KEY_INTERNATIONAL1	0x87
#define KEY_INTERNATIONAL2	0x88
#define KEY_INTERNATIONAL3	0x89
#define KEY_INTERNATIONAL4	0x8A
#define KEY_INTERNATIONAL5	0x8B
#define KEY_INTERNATIONAL6	0x8C
#define KEY_INTERNATIONAL7	0x8D
#define KEY_INTERNATIONAL8	0x8E
#define KEY_INTERNATIONAL9	0x8F
#define KEY_LANG1	0x90
#define KEY_LANG2	0x91
#define KEY_LANG3	0x92
#define KEY_LANG4	0x93
#define KEY_LANG5	0x94
#define KEY_LANG6	0x95
#define KEY_LANG7	0x96
#define KEY_LANG8	0x97
#define KEY_LANG9	0x98
#define KEY_ALTERNATE_ERASE	0x99
#define KEY_SYSREQ	0x9A
#define KEY_CANCEL	0x9B
#define KEY_CLEAR	0x9C
#define KEY_PRIOR	0x9D
#define KEY_RETURN	0x9E
#define KEY_SEPARATOR	0x9F
#define KEY_OUT	0xA0
#define KEY_OPER	0xA1
#define KEY_CLEAR_AGAIN	0xA2
#define KEY_CRSEL	0xA3
#define KEY_EXSEL	0xA4
#define KEY_KEYPAD_00	0xB0
#define KEY_KEYPAD_000	0xB1
#define KEY_THOUSANDS_SEPARATOR	0xB2
#define KEY_DECIMAL_SEPARATOR	0xB3
#define KEY_CURRENCY_UNIT	0xB4
#define KEY_CURRENCY_SUB_UNIT	0xB5
#define KEY_KEYPAD_OPARENTHESIS	0xB6
#define KEY_KEYPAD_CPARENTHESIS	0xB7
#define KEY_KEYPAD_OBRACE	0xB8
#define KEY_KEYPAD_CBACE	0xB9
#define KEY_KEYPAD_TAB	0xBA
#define KEY_KEYPAD_BACKSPACE	0xBB
#define KEY_KEYPAD_A	0xBC
#define KEY_KEYPAD_B	0xBD

```
#define KEY_KEYPAD_C 0xBE
#define KEY_KEYPAD_D 0xBF
#define KEY_KEYPAD_E 0xC0
#define KEY_KEYPAD_F 0xC1
#define KEY_KEYPAD_XOR 0xC2
#define KEY_KEYPAD_CARET 0xC3
#define KEY_KEYPAD_PERCENT 0xC4
#define KEY_KEYPAD_LESS 0xC5
#define KEY_KEYPAD_GREATER 0xC6
#define KEY_KEYPAD_AMPERSAND 0xC7
#define KEY_KEYPAD_LOGICAL_AND 0xC8
#define KEY_KEYPAD_VERTICAL_BAR 0xC9
#define KEY_KEYPAD_LOGIACL_OR 0xCA
#define KEY_KEYPAD_COLON 0xCB
#define KEY_KEYPAD_NUMBER_SIGN 0xCC
#define KEY_KEYPAD_SPACE 0xCD
#define KEY_KEYPAD_AT 0xCE
#define KEY_KEYPAD_EXCLAMATION_MARK 0xCF
#define KEY_KEYPAD_MEMORY_STORE 0xD0
#define KEY_KEYPAD_MEMORY_RECALL 0xD1
#define KEY_KEYPAD_MEMORY_CLEAR 0xD2
#define KEY_KEYPAD_MEMORY_ADD 0xD3
#define KEY_KEYPAD_MEMORY_SUBTRACT 0xD4
#define KEY_KEYPAD_MEMORY_MULTIPLY 0xD5
#define KEY_KEYPAD_MEMORY_DIVIDE 0xD6
#define KEY_KEYPAD_PLUSMINUS 0xD7
#define KEY_KEYPAD_CLEAR 0xD8
#define KEY_KEYPAD_CLEAR_ENTRY 0xD9
#define KEY_KEYPAD_BINARY 0xDA
#define KEY_KEYPAD_OCTAL 0xDB
#define KEY_KEYPAD_DECIMAL 0xDC
#define KEY_KEYPAD_HEXADECIMAL 0xDD
#define KEY_LEFTCONTROL 0xE0
#define KEY_LEFTSHIFT 0xE1
#define KEY_LEFTALT 0xE2
#define KEY_LEFT_GUI 0xE3
#define KEY_RIGHTCONTROL 0xE4
#define KEY_RIGHTSHIFT 0xE5
```

Appendix 2 kmbox Structure Dimension Drawing



IX. Common Problems

1. Keyboard and mouse do not work properly

When the keyboard or mouse received kmbox can not work properly, kmbox as a host, need to be compatible with all the keyboard and mouse, I have limited equipment at hand, the keyboard and mouse on the market thousands and thousands, can not know its characteristics. So remember to open the serial port and insert the USB device. You may see this line printed below.

```
I (37833) : VID==0x04CA&&PID==0x0061&&DID==0x0100&&HID==0x002E Version 3.0.0 @Feb 10 2021 17:42:16
E (37843) : host.configX(1226.97.256.46.mType=?&kType=?&)
此设备可能无法正常使用。如果不能正常工作，您可以使用host.config函数来强制匹配。
如果您尝试过所有的匹配还是无法正常使用，请复制以上内容，保存到一个txt中，发送到366021972@qq.com
如果操作正常可以不用理会，kmbox感谢您的添砖加瓦！
I (37878) kmbox: 设置配置值为: 01
I (37883) kmbox: 设置配置OK
I (37887) kmbox: 总电流需求: 100mA 其中Hub1:100mA,Hub2:0mA
```

Here are two ways to make your keyboard and mouse work:

Method 1 : Call the host.configX function. To debug which parameter is suitable for your keyboard and mouse, type import host in the console. There are two config functions in the host module, config0 and config1. Because kmbox has two USB ports. These two USB ports can be used to plug

in a keyboard or a mouse. configX (X=0 or 1) function is used to force the USB configuration attributes of the two ports. the first four parameters of configX you can get from the log.

E (37843) : `host.configX(1226,97,256,46,mType=?,kType=?)` where

mType: indicates the mouse report parsing method, and its value ranges from 0 to 12 .

kType: the keyboard report parsing method, the value range is 0 to 6.

If the mouse doesn't work, you need to change the value of mType, which ranges from 0 to 12. You can set the value of mType to one of the values from 0 to 12 in turn. After the call, see if the mouse works. If it works, then the mType of your mouse has been determined. Similarly, if the keyboard doesn't work properly, change the value of kType in turn. Until the keyboard can work normally.

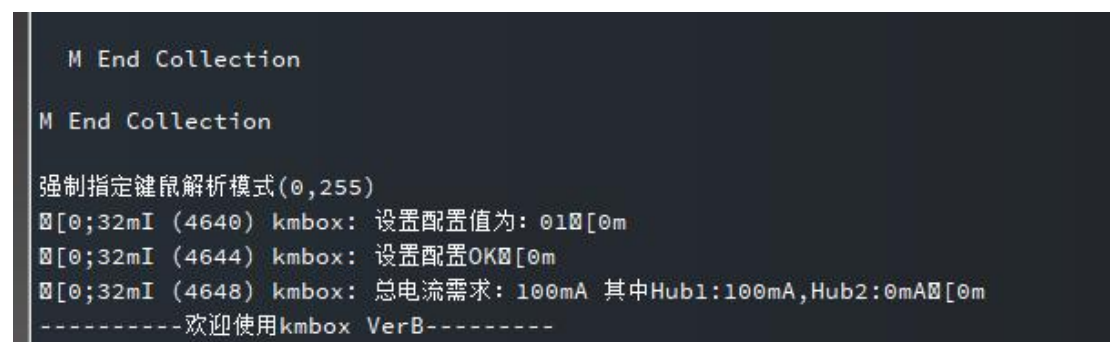
Note:

When the wireless keyboard and mouse receiver is connected to kmbox, the receiver will receive keyboard and mouse data. You need to set the value of mType and kType at the same time. If your device is pure keyboard, then please set the value of mType to 255, if your device is pure mouse, then set the value of kType to 255.

Second, config0 and config1 do not correspond to USB ports, they correspond to devices. Devices can be plugged into any USB port. If you have a device that doesn't work and you use config0 to force a match. The second unusable device you need to use the config1 function. If you continue to use the config0 function, it will flush out the config0 configuration. kmbox controls up to two devices. config0 and config1 are used to correspond to these two devices.

After you have successfully adapted the keyboard and mouse using the above method, you can create a config.py file. Write the configuration values into the config.py file. Then download it to the development board. Then it will be permanently auto-recognized and matched in the future. As shown in the picture below:

Reboot the development board after downloading and it will be recognized automatically:



```
M End Collection
M End Collection
强制指定键鼠解析模式(0,255)
[0;32mI (4640) kmbox: 设置配置值为: 010[0m
[0;32mI (4644) kmbox: 设置配置OK0[0m
[0;32mI (4648) kmbox: 总电流需求: 100mA 其中Hub1:100mA,Hub2:0mA0[0m
-----欢迎使用kmbox VerB-----
```

If you have tried to match all the parameters of the keyboard and mouse and still can't use it normally. Then please remember to copy all the print content to a txt file. And named after the device name. For example, Logitech G102 mouse is not recognized. Please copy all printouts and name them "Logitech G102 Mouse.txt" (below). And send the file [to the author](#), after adding the adaptation can be supported.

2. The serial port of the mouse disconnects as soon as it is plugged in.

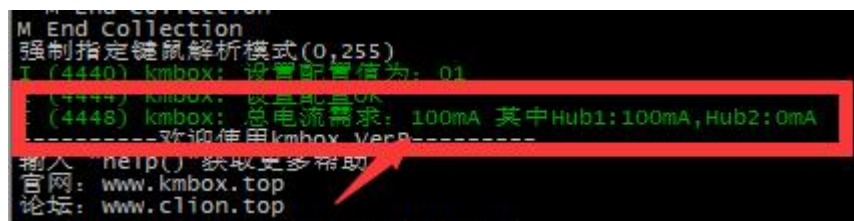
This is a typical problem of insufficient power supply. Do not connect the USB1 port of kmbox to any HUB extender. Please make sure it is directly connected to the USB port of your computer. And make sure the USB port has enough power supply.

Common sense: the power supply voltage of a USB port is 5V, the maximum current is 500mA, when the USB of your computer is connected to the kmbox, and you connect a keyboard or a mouse to the kmbox, the power supply is still the USB of your computer. kmbox needs to consume 100mA of current to work (300mA for Bluetooth mode). If your keyboard or mouse consumes more current, its total current demand plus the kmbox's is more than 500mA. then kmbox may not work properly. Because your computer's USB port can't provide enough current. So please make sure the USB port is sufficiently powered.

Keyboard and mouse current demand can be found in the boot log, kmbox doesn't do power management. By default, your computer has enough power supply:

```
I (161224) kmbox: ===== configuration descriptor start =====  
I (161230) kmbox: bLength:      09  
I (161234) kmbox: bDescriptorType: 02 (fixed to 02)  
I (161240) kmbox: wTotalLengthH/L: 0054 (collection length)  
I (161246) kmbox: bNumInterfaces: 03 (number of interfaces)  
I (161251) kmbox: bConfigValue: 01 (configuration value)  
I (161257) kmbox: iConfiguration: 04 (configuration character index)  
I (161263) kmbox: bmAttributes: A0 (bus powered remote wakeup support)  
I (61270) kmbox: MaxPower:      31 (Maximum current 98mA)
```

Please make sure that the sum of all device currents is less than the USB specification of 500mA and that your computer is able to provide sufficient current. Currently the boards have integrated a total current requirement hint.



```
M End Collection  
强制指定键鼠解析模式(0,255)  
I (4440) kmbox: 设置配置值为: 01  
(4444) kmbox: 设置配置OK  
(4448) kmbox: 总电流需求: 100mA 其中Hub1:100mA, Hub2:0mA  
-----欢迎使用kmbox VerB-----  
输入 "help()" 获取更多帮助  
官网: www.kmbox.top  
论坛: www.clion.top
```

If the total current is greater than 500mA, the kmbox may not work properly. It is also unlikely that the keyboard and mouse behind it will work properly. After all, a USB current is only 500mA. The original keyboard and mouse were powered by separate ports. Now one USB port powers the kmbox, the external keyboard and power supply, and the external mouse power supply. If this situation occurs it is recommended to use a dual port USB cable to increase the USB power supply. The dual port USB cable is connected as follows:

3. After plugging in the board, it keeps rebooting

This may be the serial port driver installation is not correct resulting in frequent reboot of the board, please download and install the driver.