PikaPython

Release 0.1

Lyon

Jul 16, 2023

CONTENTS

: http://pikapython.com/doc

PikaPython is a completely rewritten ultra-lightweight python engine with zero dependencies, zero configuration, and can run under less than 4KB of RAM (such as stm32g030c8 and stm32f103c8), making it extremely easy to deploy and scale.

You can help us improve the document by Pull Request on the document source repo: https://github.com/pikasTech/pikadoc-en

CHAPTER

ONE

INTRODUCTION

1.1 Principle introduction

content:

- Introduction MCU and scripting language
- The principle analysis of PikaPython
- Light a lamp with Pikascript
- use PikaPython to implement an addition function

1.1.1 Introduction MCU and scripting languages

In embedded application scenarios such as IOT and smart terminals, script development is a convenient and fast solution.

When it comes to the development of embedded scripting languages, the first thing that comes to mind is micropython. Micropython allows engineers to use the scripting language python for MCU development, which greatly reduces the development threshold.

However, there are not many development boards that can be used directly in the development of micropython. It is obviously a huge project and a high threshold to transplant micropython for the MCU without ready-made micropython firmware.

Moreover, the operating efficiency of python is low, which is especially obvious in the MCU with limited resources. It is also difficult to make full use of the hardware features such as interrupt and dma of MCU for development with python.

In applications such as high real-time signal processing, data acquisition, and real-time control, it is difficult for python to be truly implemented in the production environment.

For now, in the development of mcu, about 80% of the development is still using the c language, and c++ only accounts for less than 20%.

But there is no doubt that the convenience of scripting languages is very obvious. Server-side developers are often familiar with object-oriented scripting languages such as python and JavaScript.

If the function of MCU can be called directly from the scripting language, the development difficulty will be significantly reduced.

Then, if you use the c language for MCU embedded development, and provide an object-oriented scripting language calling interface to the host computer or server, can you take into account the MCU operating efficiency and development efficiency?

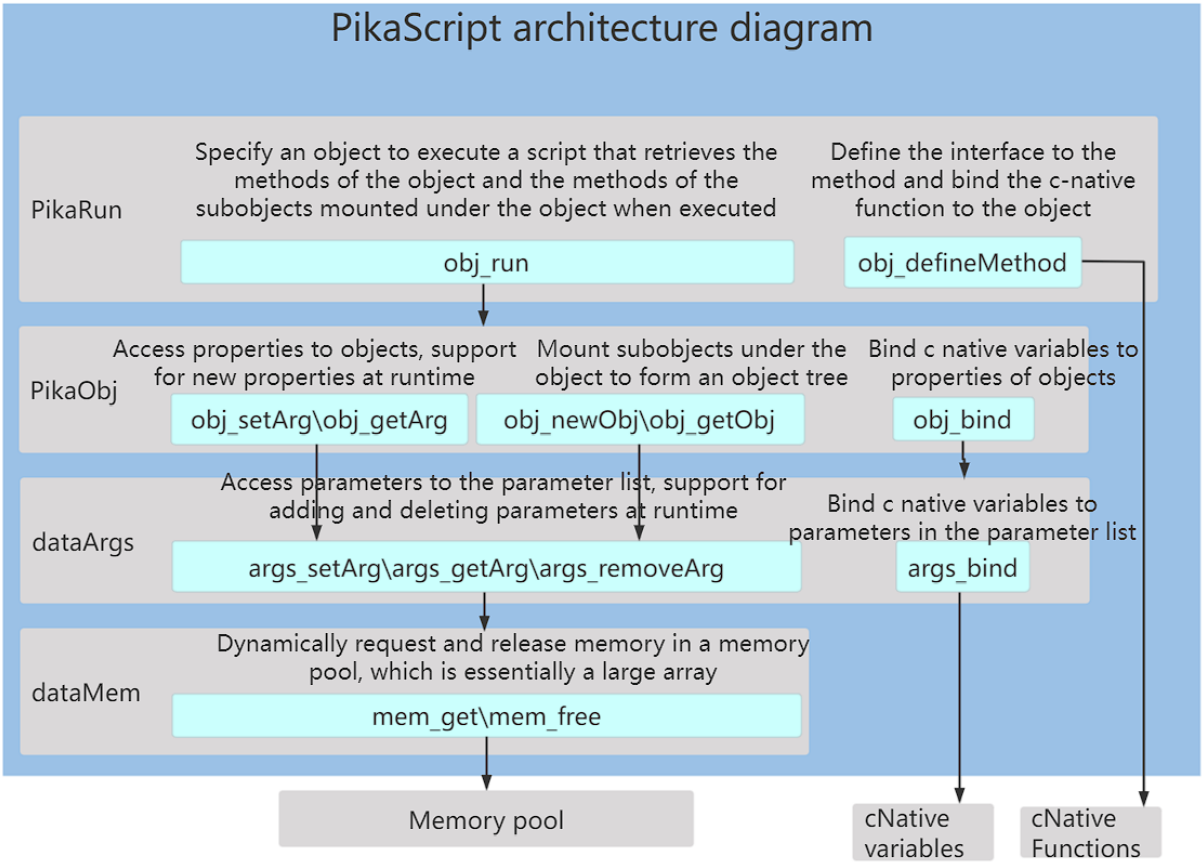
The Pikasciprt library introduced in this article does exactly that.

Pikascrpit library can provide object-oriented scripting language calling interface for mcu project developed in C language. PikaPython has the following features:

- Support bare metal operation, can run in mcu with more than 4Kb memory, such as stm32f103, esp32.
- Support cross-platform, can run in linux environment.
- Code is readable, uses only the C standard library, is structured as clearly as possible (as far as I can), and uses few macros.

1.1.2 The principle analysis of PikaPython

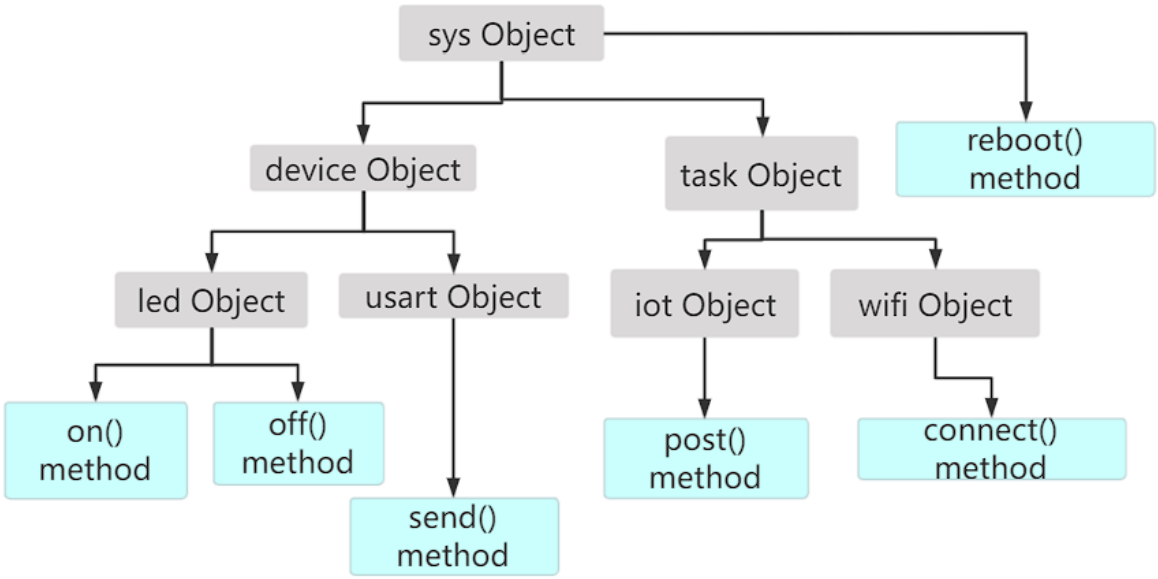
The schematic diagram of the architecture of PikaPython is shown in the following figure. We analyze it layer by layer from top to bottom.



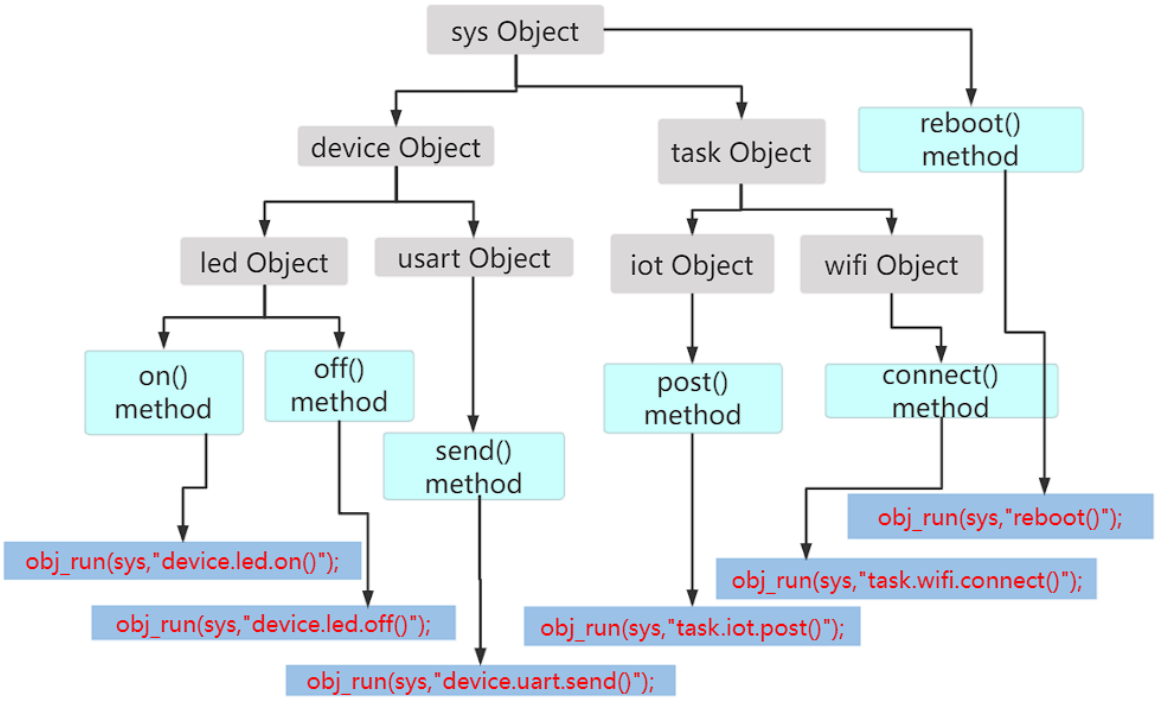
PikaRun script run layer

The PikaRun script running layer is the top-level calling interface of PikaPython, and script running can be realized only by calling obj_run. When calling obj_run, you need to specify an object. When the script runs, it will retrieve the methods of this object and the methods of the sub-objects of this object.

The following figure shows a common object structure in embedded development. sys is the top-level object. The sys object has a reboot() method. The device sub-object and the task sub-object are mounted under the sys object. These two objects The sub-objects are mounted below, and each sub-object has its own method.



At this time, we only need to pass in the pointer of the top `sys` object in `obj_run`, and you can call all methods of all objects with the method shown in the following figure. Among them, the `reboot()` method directly belongs to the `sys` object, so it can be called by directly running `obj_run(sys, "reboot()")`, and the led object is called through `obj_run(sys, "device.led. on()")` to call.



In actual development, we can let the mcu run the data received by the serial port directly as a script. E.g:

```
obj_run(sys, uartReceiveBuff);
```

Where uartReceiveBuff is the data received by the serial port.

At this time, send "`device.led.on()`" to the serial port of the mcu, and the led light can be turned on.

PikaObj Object Support Layer

As mentioned in the previous section, we already know how to use PikaPython to execute scripts within an existing object structure. So the next question is, how to construct objects, and how to define properties and methods for objects?

(1) Constructor function

PikaPython constructs objects through a constructor function. A constructor function corresponds to a class in PikaPython. The constructor function for an LED is shown below. In PikaPython, all constructor functions use the same entry and return parameters.

The entry parameter args is a parameter list. The args is internally based on a linked list, and any number and type of parameters can be passed in. Here args is the initialization parameter of the constructor, which will be used when constructing with parameters.

The return value of the constructor function is a PikaObj object.

```
PikaObj *New_LED(Args *args){
  // inherited from MimiObj base class
  PikaObj *self = New_PikaObj(args);
  // define properties for the object
  obj_setInt(self, "isOn", 0);
  // Bind the on() method to the LED1 object
```

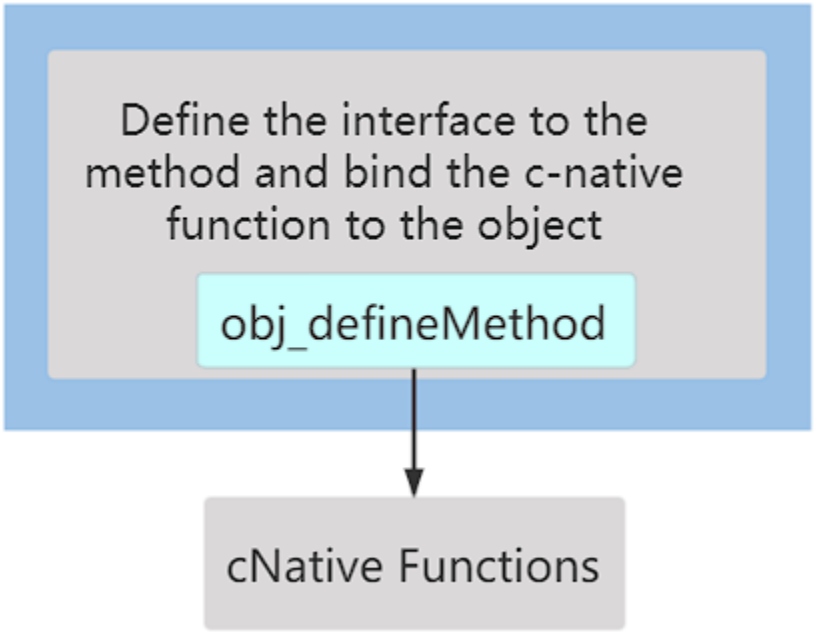
(continues on next page)

(continued from previous page)

```
  obj_defineMethod(self, "on()", onFun);
  return self;
}
```

The first line of the constructor is for class inheritance. The `LED` class inherits from the `Pikaobj` base class, which is the source of all classes.

`obj_setInt` defines a property for the `LED` class, the property name is `"isOn"`, and the initial value is `0`.

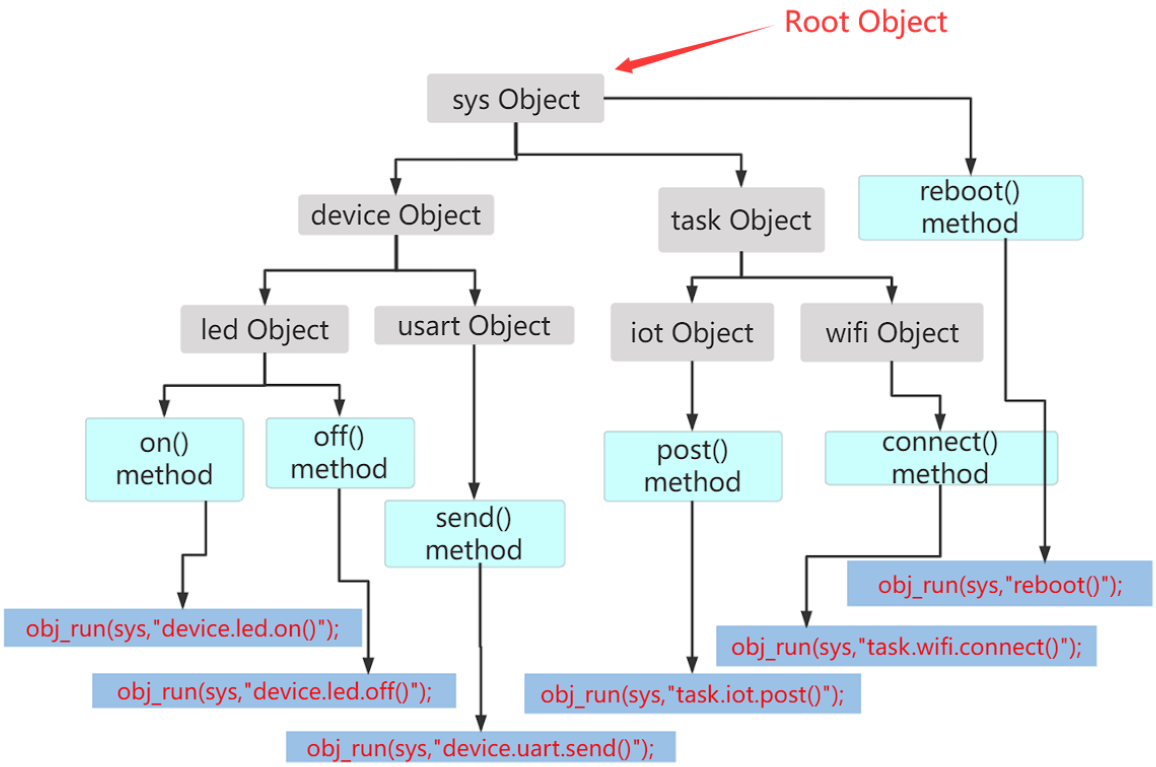


`obj_defineMethod` binds a method to the `LED` class, and the bound method is the `on()` method. `onFun` is a function pointer to the `c` native function to which the `on()` method is bound. The specific way of writing the `onFun` function is introduced in Chapters 3 and 4.

(2) Construct the object

There are two ways to construct objects. One is to construct the object passed in by `obj_run`, which is called the root object, such as the `sys` object in the following figure, and the other objects are general objects, which are mounted under the root object.

Generally, only one root object is constructed in a project.



The `newRootObj` function is used to construct the root object. To construct a root object, you need to pass in the object name `"led"` and the constructor function pointer. The return value of `newRootObj` is the pointer of the root object.

```
PikaObj *led = newRootObj("led", New_LED);
```

The construction of general objects is done in the constructor of the parent object. If you want to mount the `led` child object under the `sys` object, you can write the constructor function of the `SYS` class like this:

```
PikaObj * New_SYS(Args *args){
    // inherited from MimiObj base class
    PikaObj *self = New_PikaObj(args);
    // Import the LED class through the constructor of the LED class
    obj_import(self, "LED", New_LED);
    // Use the LED class to create a new led object, and the led object is used as a sub-
↪object of the sys object
    obj_newObj(self, "led", "LED");
    return self;
}
```

`obj_import` imports a class through the function pointer of the constructor. The imported class in the above code is named `LED`. `obj_newObj` creates a new object through the imported class, and the new object is mounted as a sub-object under the current class.

At this time, by calling the following function, you can get a `sys` root object that mounts the `led` object.

```
PikaObj *sys = newRootObj("sys", New_SYS);
```

dataArgs dynamic parameter list

dataArgs is a dynamic parameter list based on a linked list. Its structure is Args. dataArgs dynamically applies for and releases memory at runtime, so you can add, delete, modify, check parameters at runtime, and attribute and method information of Pikaobj The access is based on the dataArgs parameter list.

dataArgs supports integer, floating point, string, pointer type parameters, and also supports binding native C language variables as parameters in dataArgs.

The following example is the basic usage of Args. The implementation principle of dataArgs will be introduced in subsequent articles, and will not be emphasized in this article.

```
// create a new parameter list
Args *args = New_Args();
// Store an integer parameter a into the parameter list with a value of 1
args_setInt(args, "a", 1);
// Take the parameter a, the value is 1
int a = args_getInt(args, "a");
// modify the value of a to 2
args_setInt(args, "a", 2);
// Take out a again, the value is 2
a = args_getInt(args, "a");
// destroy the parameter list
args_deinit(args);
```

dataMemory

dataMemory provides dynamic memory allocation and release for dataArgs, which is not the focus of this article.

1.1.3 Light a light with PikaPython

Then let's light a light and see how PikaPython provides object-oriented scripting support for mcu in actual projects.

Let's take the HAL library of STM32 as an example. Suppose an LED light is connected to pin PA8, which we call led1. When PA8 is pulled high, the light is on, and when it is pulled low, the light is off.

Then to turn on the light led1, you need to use the following c language code:

```
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_8,SET)
```

We hope to use the following object-oriented script to turn on the lights more elegantly~

```
led1.on()
```

Let's see how to use PikaPython to achieve this requirement.

Write an onFun() function.

```
void onFun(MimiObj *self, Args *args){
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_8,SET);
}
```

This function will be registered in the script object as a method. After registration, it will no longer be called by the developer in C language development, but will only be called by the script interpreter when the script is running.

The entry parameters of the `onFun()` function are `self` and `args`, where `self` is the objectpointer, `args` is a list of incoming and outgoing arguments (used in Chapter 4).

In PikaPython, all functions bound as methods use these two entry parameters.

Write the constructor for the LED1 class.

```
PikaObj * New_LED1(Args *args){
    // Inherited from PikaObj base class
    MimiObj *self = New_PikaObj(args);
    // Bind the on() method to the LED1 object
    obj_defineMethod(self, "on()", onFun);
    return self;
}
```

`obj_defineMethod` is used to bind the written C language function as the method of the script object.

Here, the function pointer of the native function `onFun()` of the C language is registered into the object as a parameter, and the `"on()"` string declares the method name and parameters when the script is called, here `"on()"` Methods have no parameters, and method binding with parameters is introduced in Chapter 4.

Write the constructor for the root object.

```
PikaObj * New_MYROOT(Args *args){
    // inherited from MimiObj base class
    MimiObj *self = New_PikaObj(args);
    // Import the LED1 class
    obj_import(root, "LED1", New_LED1);
    // Construct sub-object "led1", the class of "led1" is "LED1"
    obj_newObj(root, "led1", "LED1");
    return self;
}
```

`obj_import` imports the `LED1` class through the function pointer of the constructor.

`obj_newObj` creates a new `led1` object through the imported `LED1` class, and the `led1` object is mounted as a sub-object under the `MYROOT` class.

Create a root object and listen for incoming data from the serial port. When the entire row of data is obtained, it is directly executed as a script.

```
int uartReceiveOk; //The flag bit that the serial port single-line reception is completed
char uartReceiveBuff[256];//Single-line data received by serial port
int main(){
    // Hardware initialization code is omitted

    // create root object
    PikaObj *myRoot = newRootObj("myRoot", New_MYROOT);
    while(1){
        // The serial port has received a single line of data
        if(uartReceiveOk){
            // Execute single-line data input from serial port
            obj_run(myRoot, uartReceiveBuff);
            // Clear the serial port receive flag
            uartReceiveOk = 0;
        }
    }
}
```

At this time, just send `led1.on()` to the serial port of mcu, the light will be on (magic no~)

1.1.4 Implement an addition function in PikaPython.

The method in the above example has no input and output. In the following example, we will define a `TEST` class and add an `add` method to the `TEST` class to implement the addition function. method of input and output.

Write an add() function.

Like the last `onFun` function, the function to be bound this time is the `addFun` function.

```
void addFun(PikaObj *self, Args *args) {
    //get the input parameters
    int val1 = args_getInt(args, "val1");
    int val2 = args_getInt(args, "val2");
    //implement method function
    int res = val1 + val2;
    // pass the return value back to the parameter list
    method_returnInt(args, res);
}
```

`args_getInt` is used to get integer parameters from the parameter list, here the input parameters `val1` and `val2` are taken from the parameter list. The parameter list also supports `float` type, string type and pointer type.

`method_returnInt` is used to pass the return value of the method, and it can also return `float` type, string type and pointer type.

Define the constructor of the test class

```
PikaObj *New_PikaObj_test(Args *args){
    //Inherit MimiObj base class
    MimiObj *self = New_PikaObj(args);
    // bind method
    obj_defineMethod(self, "add(val1:int, val2:int)->int", addFun);
    return self;
}
```

This time use `obj_defineMethod` to bind a method with input and output parameters.

`"add(val1:int,val2:int)->int"` is python's typed function declaration syntax, indicating that the `add` method has two input parameters, `val1` and `val2` of type `int`, and the output The parameter is also of type `int`. Likewise, pass a function pointer to the `addFun` function.

Write the constructor for the root object.

```
PikaObj * New_MYROOT(Args *args){
    // Inherited from PikaObj base class
    PikaObj *self = New_PikaObj(args);
    // import the TEST class
    obj_import(self, "TEST", New_PikaObj_test);
    // Construct sub-object "test", the class of "test" is "TEST"
    obj_newObj(self, "test", "TEST");
    return self;
}
```

Mount the `test` child object in the root object.

Create object and test run script

```
void main(){
    // create a new root object
    PikaObj *root = newRootObj("root", New_MYROOT);
    //Run the script (also supports the calling method of "res = test.add(val1 = 1,␣
↪val2= 2)")
    obj_run(root , "res = test.add(1, 2)");
    // Get the attribute value res from the root object
    int res = obj_getInt(root, "res");
    //destroy the root object
    obj_deinit(root);
    /* print return value res = 3*/
    printf("%d\r\n", res);
}
```

After `obj_run` runs the script, it will dynamically create a `res` property, which belongs to the `root` object.

`obj_deinit` is used to destroy the object, all child objects mounted under the `root` object will be automatically destroyed.

In this example, the `root` object mounts the `test` object, so the `test` object will be automatically destroyed before the `root` object is destroyed.

1.1.5 Constructing classes and objects more easily

Implementing a `class` by writing a constructor function is still a bit cumbersome. In practice, PikaPython provides a tool to automatically generate constructor functions: **PikScript precompiler**.

Just declare a `class` in Python syntax and it will automatically link to C functions, see C Module -> Making C Libraries into Python Libraries

1.2 Demo show

I want to run a Python with a microcontroller, I have to use linux virtual machine + cross-compilation tool chain + command line compile micropython firmware, but also have to use the DfuSe tool to burn the firmware, burned also can not use the C debugger to debug.

I want to expand a C module of my own, but I have to learn to use some completely unintelligible macro functions, and I have to register them manually, and I have to write makeFile, and I can't debug C after compilation.

I am poor, can not afford to buy STM32F4, want to buy a STM32F103C8T6 micropython development board, Taobao a search, seems not.

Now the C8T6 is also expensive, I still want to use F0, use G0, with domestic chips, can it work?

It seems that it is not very easy to port micropython to G0.

So? Is there another way to play?

In other words, I want to develop with Keil, debug with Keil, and I want to use the cheapest microcontroller, and it's very easy to develop C modules.

How about trying PikaPython?

What is PikaPython?

PikaPython provides extremely easy to deploy and extend Python scripting support for resource-constrained mcu. It doesn't require an OS, it runs bare metal, and it doesn't require a filesystem.

PikaPython supports bare-metal operation, at least for mcu with RAM 4kB and FLASH 32kB, the recommended configuration is RAM 10kB and FLASH 64kB, such as stm32f103c8t6 and stm32g070RBT6, which have no pressure at all and even meet the recommended configuration.

And support Keil, IAR, RT-Thread studio, segger embedded studio and other IDE development, zero dependencies, zero configuration, out-of-the-box, extremely easy to integrate into the existing C project.

These are all demos of STM32G070RBT6.

1.2.1 Demo 01 Light up

```
import PikaStdLib
import machine

mem = PikaStdLib.MemChecker()
io1 = machine.GPIO()
time = machine.Time()

io1.setPin('PA8')
io1.setMode('out')
io1.enable()
```

(continues on next page)

(continued from previous page)

```
io1.low()

print('hello pikascript')
print('mem.max:')
mem.max()
print('mem.now:')
mem.now()

while True:
    io1.low()
    time.sleep_ms(500)
    io1.high()
    time.sleep_ms(500)
```

Look at the script, it's all Python3 standard syntax.

The light is flashing.

1.2.2 Demo 02 Serial port test

```
import PikaStdLib
import machine

time = machine.Time()
uart = machine.UART()
uart.setId(1)
uart.setBaudRate(115200)
uart.enable()

while True:
    time.sleep_ms(500)
    readBuff = uart.read(2)
    print('read 2 char:')
    print(readBuff)
```

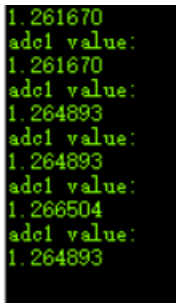
Open a serial port and try to read two characters

very smooth

1.2.3 Demo 03 Try reading an ADC

```
import PikaStdLib
import machine

time = machine.Time()
adc1 = machine.ADC()

adc1.setPin('PA1')
adc1.enable()

while True:
    val = adc1.read()
    print('adc1 value:')
    print(val)
    time.sleep_ms(500)
```

Again a few lines of script fixes it.

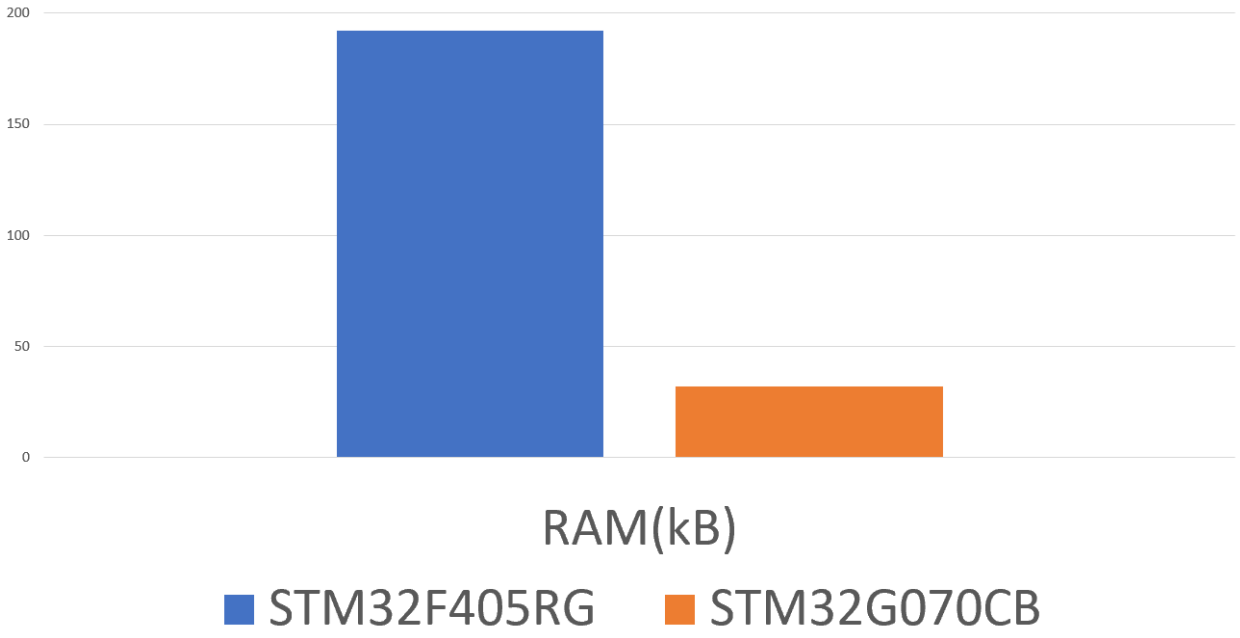


This is the output result.

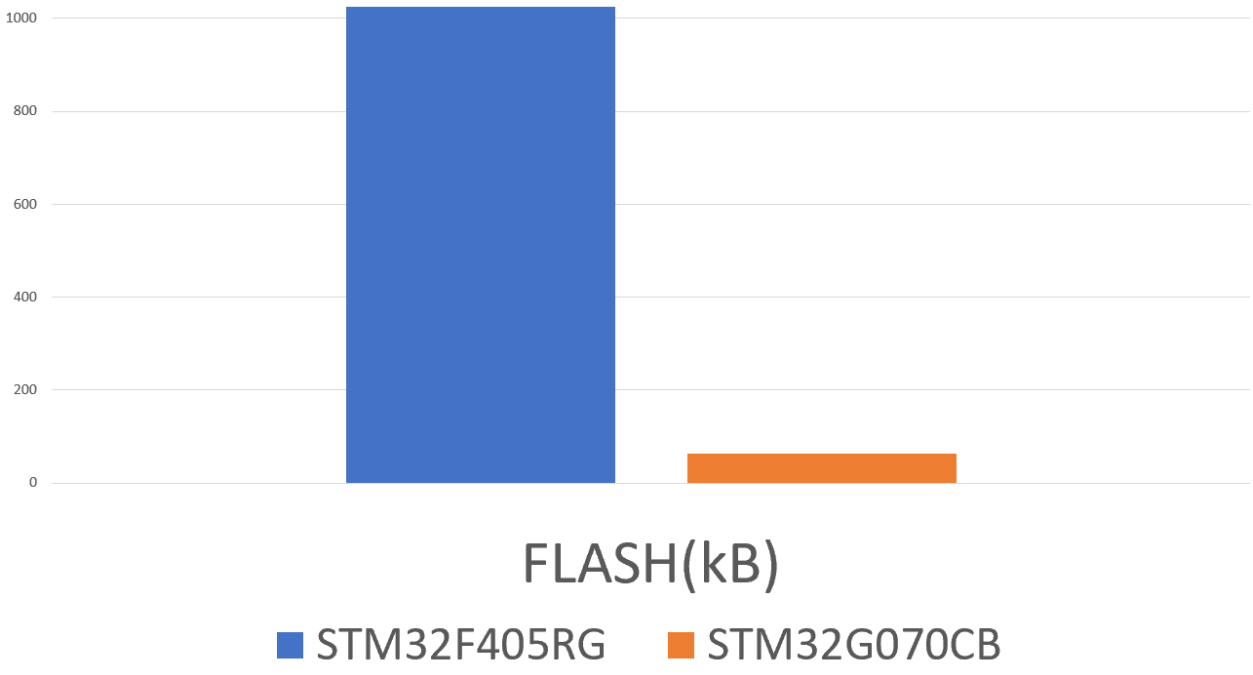
The maximum value of RAM occupied by these demos is only 3.56K, including the 1K stack is also 4.56K, the maximum Flash occupation is 30.4K, using the STM32F103C8T6's 20K RAM and 64K Flash as the standard, RAM is only used up less than 25%, Flash is only used up less than 50%, simply more resources do not know how to spend. This is a lot of resources.

Also running Python, we can briefly compare the common chip STM32F405RG for micropython and the chip STM32G070CB for PikaPython.

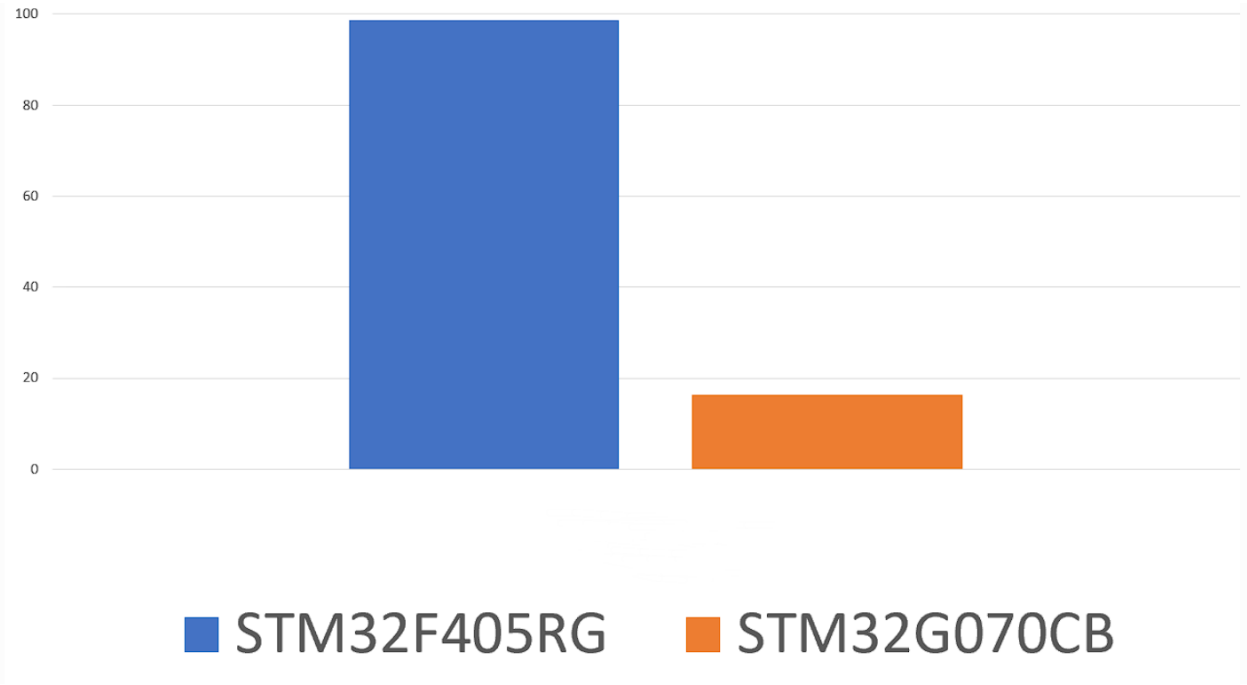
1.2.4 RAM resource comparison



1.2.5 Flash resource comparison



1.2.6 Reference price comparison (take the price of 10 pieces of Lichuang Mall on September 11, 2021 as a reference)



image

1.2.7 How about the expansion ability?

In addition to device drivers, developing custom python script bindings for mcu is very easy with the pikascript development framework. The following two demos are custom C module extensions that develop some python script interfaces based on the ARM-2D image driver library.

1.2.8 A few small squares~

1.2.9 Several rotating suns~

1.2.10 So, is PikaPython open source?

Of course, this is the github home page of PikaPython: https://github.com/pikasTech/pikascript

1.2.11 Is it difficult to develop?

PikaPython has prepared rich demos and development guides from shallow to deep for developers, and the guides will continue to be improved and maintained.

1.2.12 Can it be commercialized?

Of course! PikaPython uses the MIT protocol and allows modifications and commercialization, but be careful to keep the original author's byline.

1.3 Syntax support

Support for a subset of python3 standard syntax.

1.3.1 object support

1.3.2 Operator

1.3.3 Control flow

1.3.4 Module

1.3.5 List/Dict

1.3.6 Exception

1.3.7 Slice

1.3.8 Other keywords/Syntax

CHAPTER

TWO

GET START

2.1 How to Get Started with PikaPython using KEIL Simulator

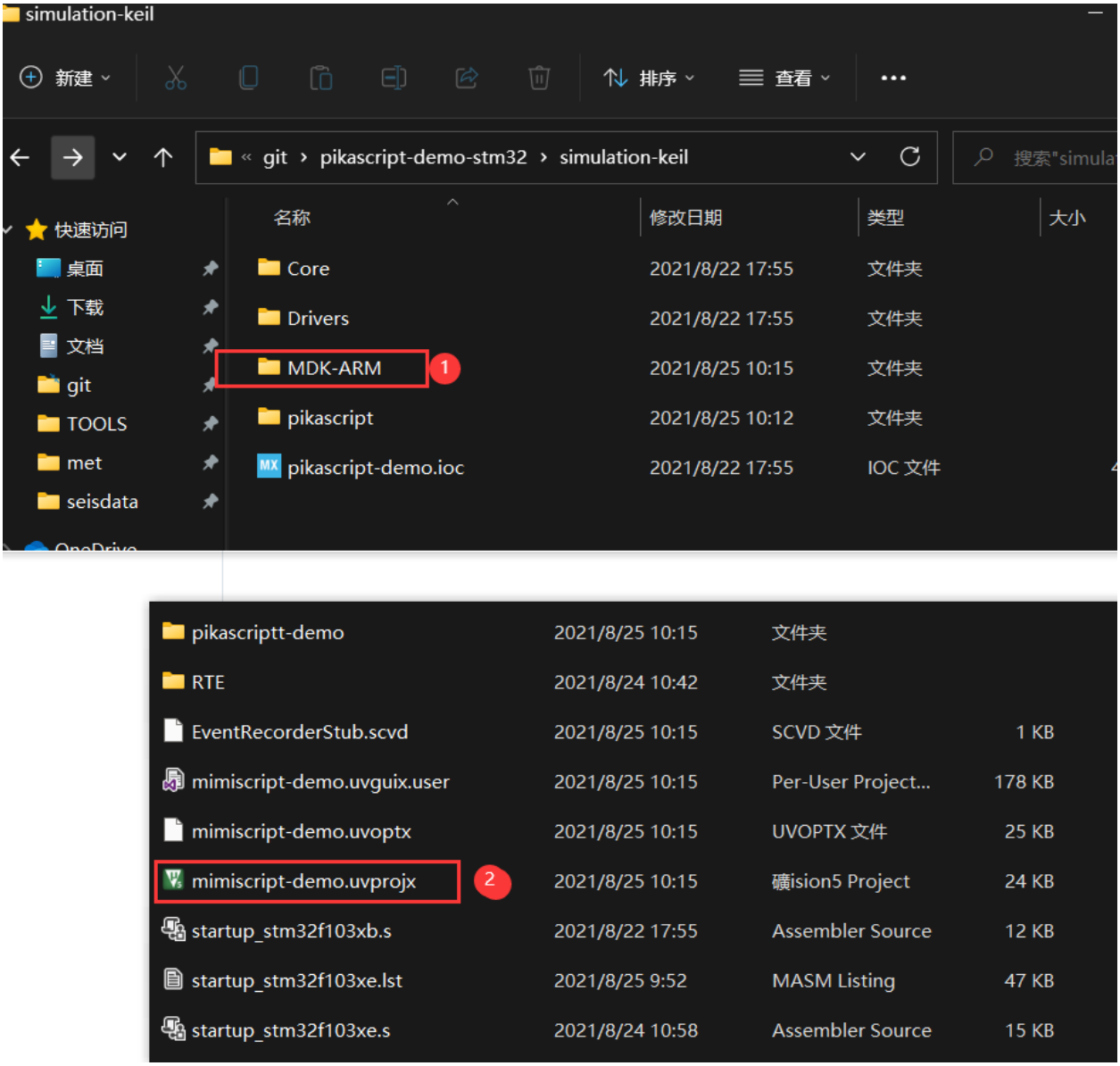
In this article, we introduce a new way of playing PikaPython without hardware, i.e. using simulation in MDK. The target board of simulation is stm32f103, and you can experience the fun of pikascript immediately after downloading the project.

2.1.1 Create project

Open the pikascript official website http://pikascript.com

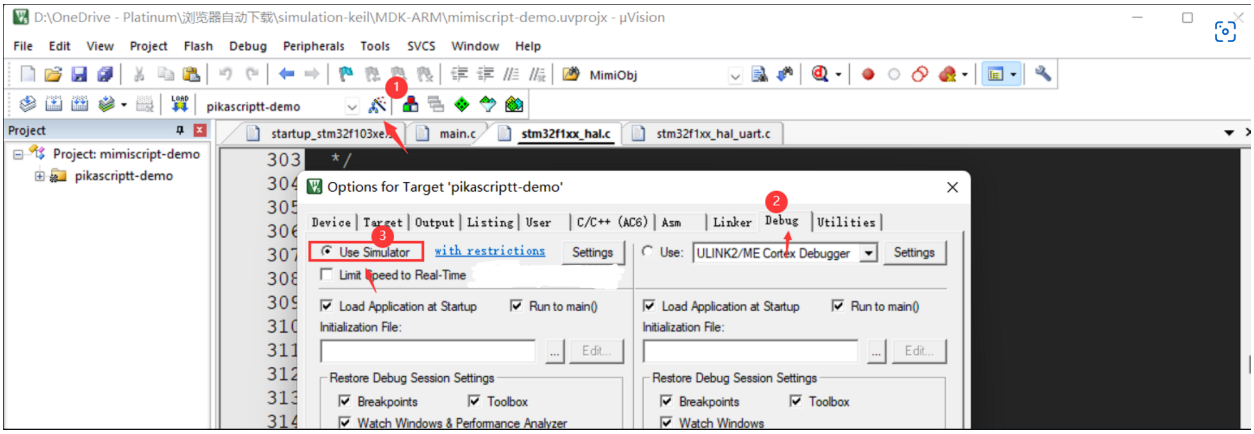
Select simulation-keil and click “Start Generation”

Unzip the downloaded zip archive and open the project

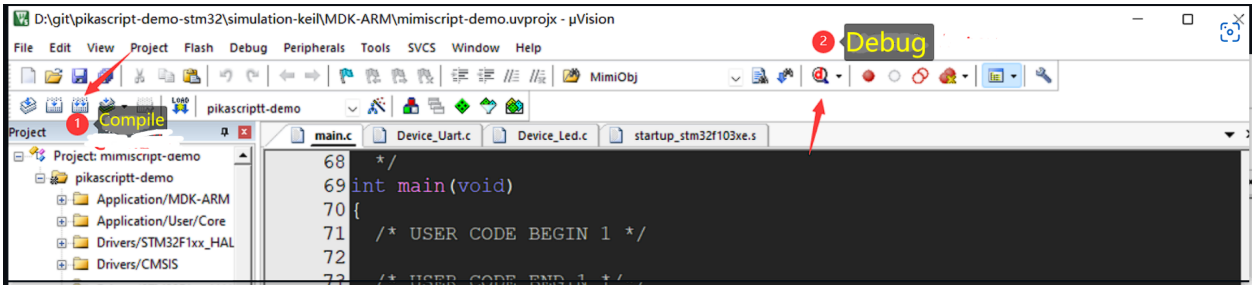


Run the simulation project

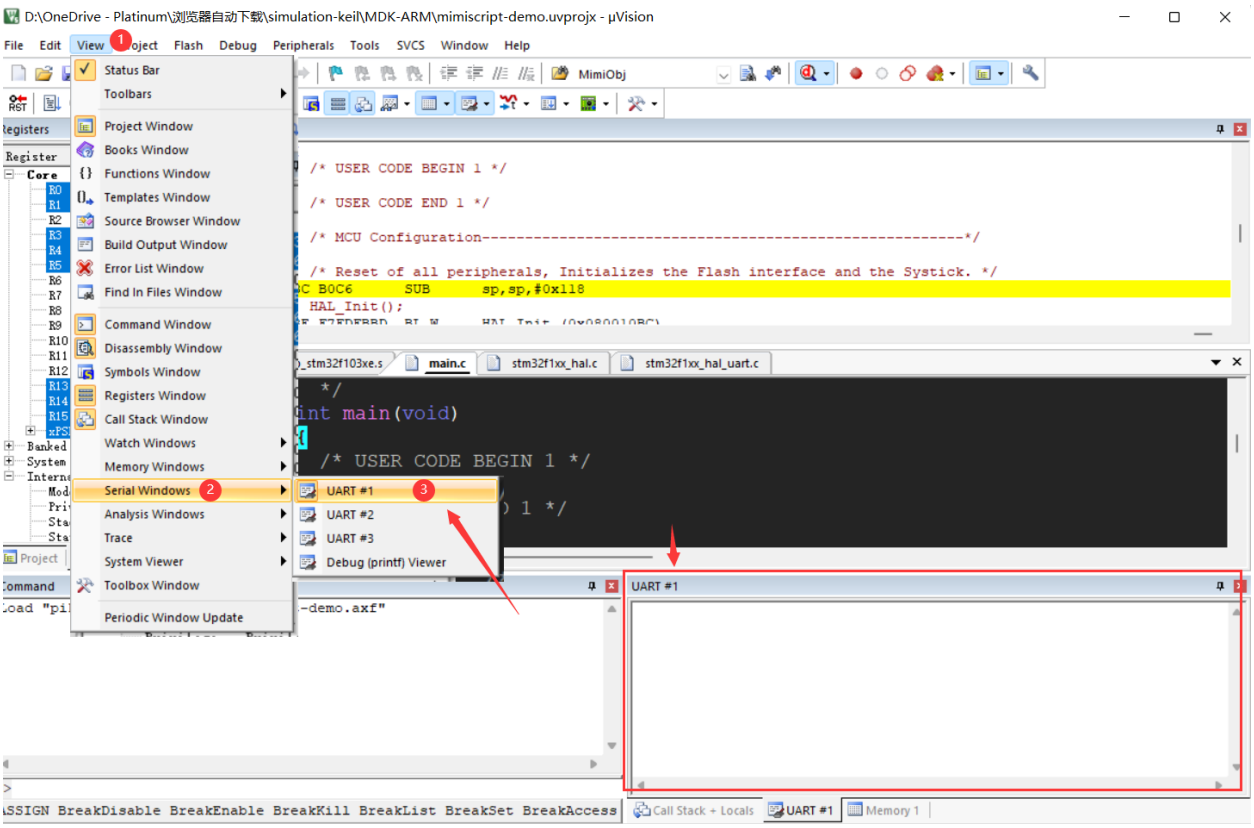
Make sure you have select the simulator as the debugging target



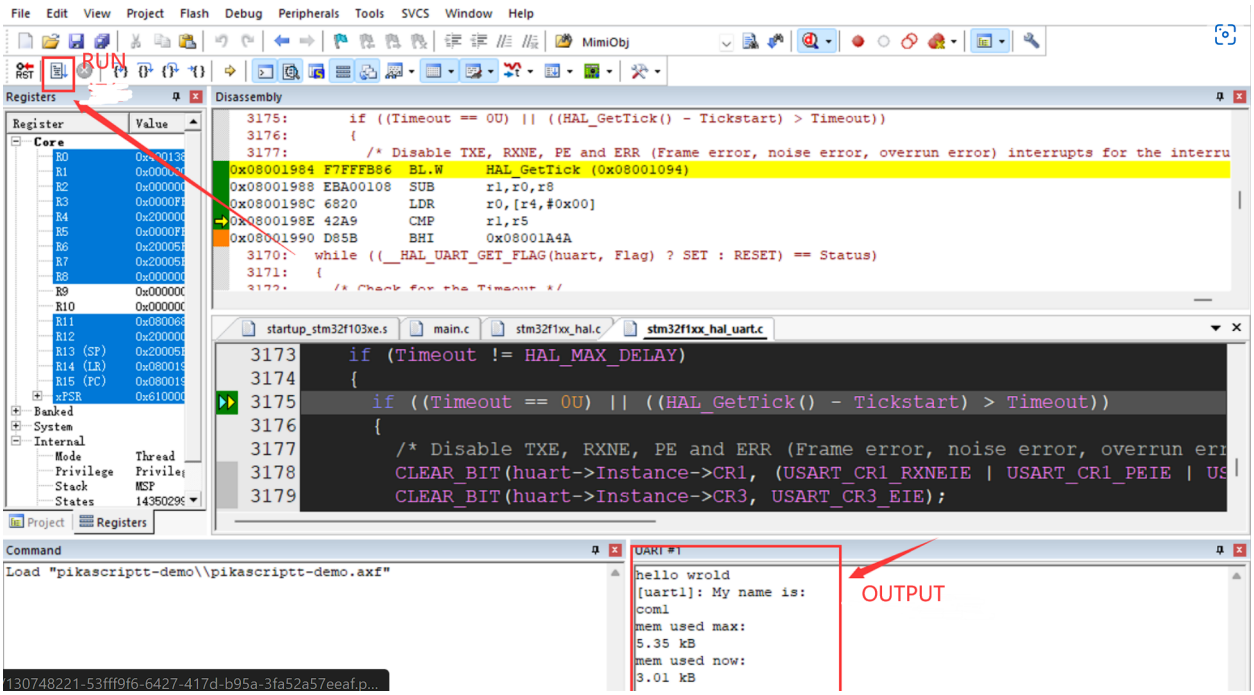
Compile and debug:



Once entering the debug session, make sure you have opened the serial windows as shown below:



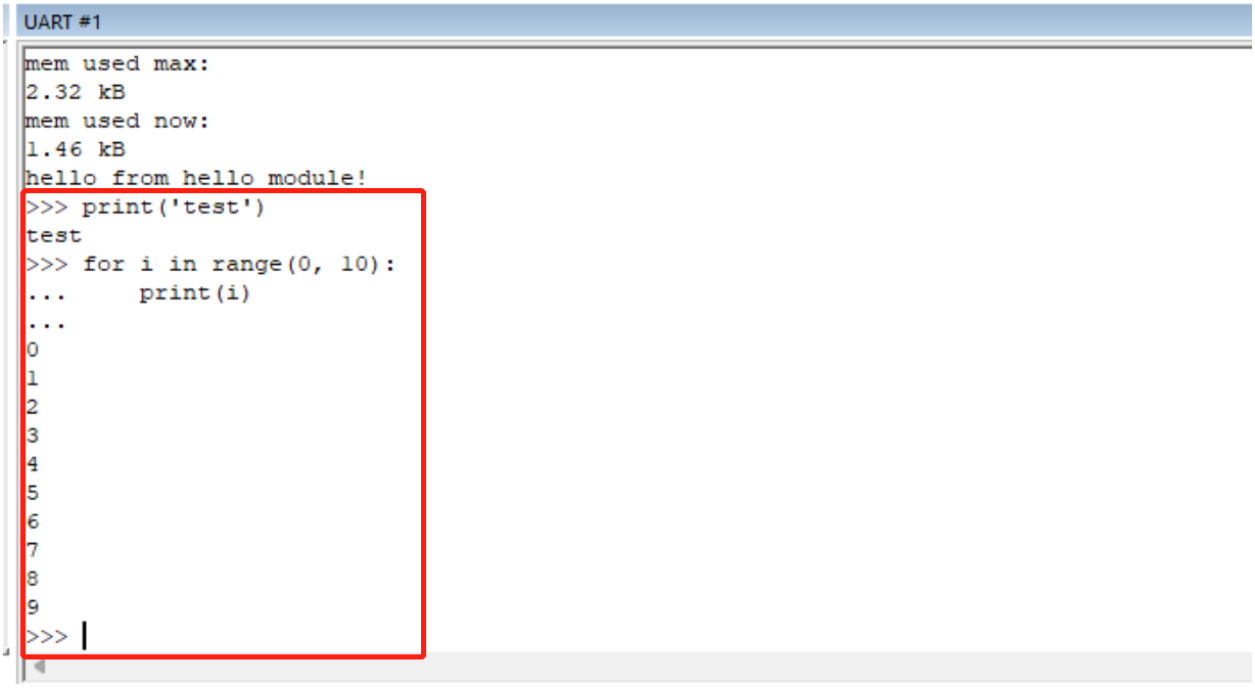
run and check the output:



2.1.2 REPL

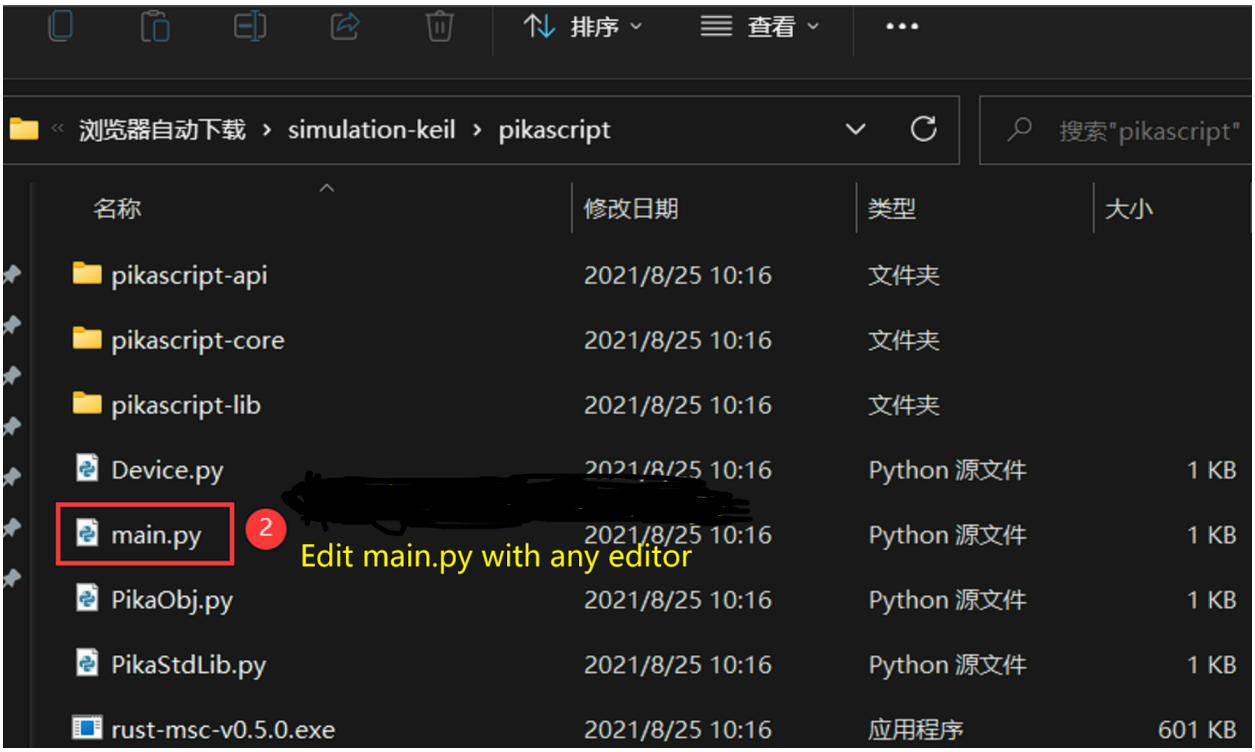
Python scripts can be run interactively by typing them directly in the UART window.

NOTE: Please use With **4 white-spaces** for indentation.



2.1.3 How to Run a different python script

Open the `main.py` in any editor, e.g. vscode:



In `main.py`, you might see something similar to:

```
# main.py
import Device
import PikaStdLib

led = Device.LED()
uart = Device.Uart()
mem = PikaStdLib.MemChecker()

print('hello wrold')
uart.setName('com1')
uart.send('My name is:')
uart.printName()
print('mem used max:')
mem.max()
print('mem used now:')
mem.now()
```

This script uses standard python3 syntax. Suppose we have already modified this script, so how to run it on the device?

The interesting part is, pikascript uses a method similar to java, i.e. it is semi-compiled and semi-interpreted. For example, the pikascript compiler compiles classes and methods, while PikaVM interprets method-calls and object-creation/destruction at runtime.

The pikascript compilation is a two-step process:

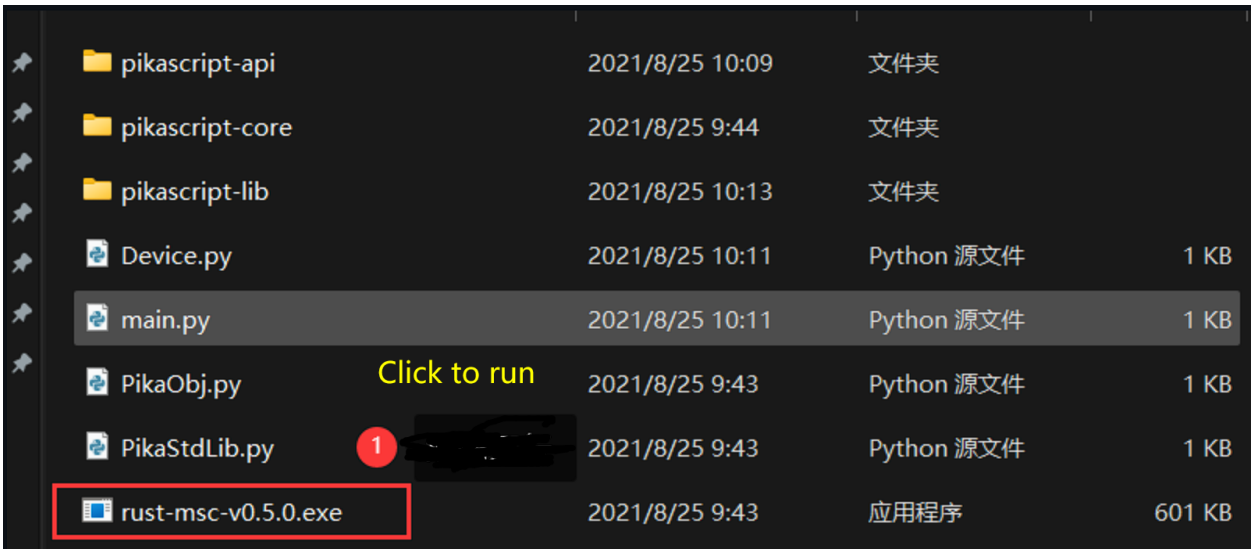
1. Using pikascript compiler to compile the `.py` files into `.c` and `.h` files and store them in the `pikascript-api`

folder.

2. Using the ordinary c compiler to compile all the c source files and generate an executable image for the target device.

Double-click `rust-msc-vxx.yy.zz.exe` to run the pika precompiler which is written in Rust.

NOTE: Here `xx.yy.zz` is the version number.

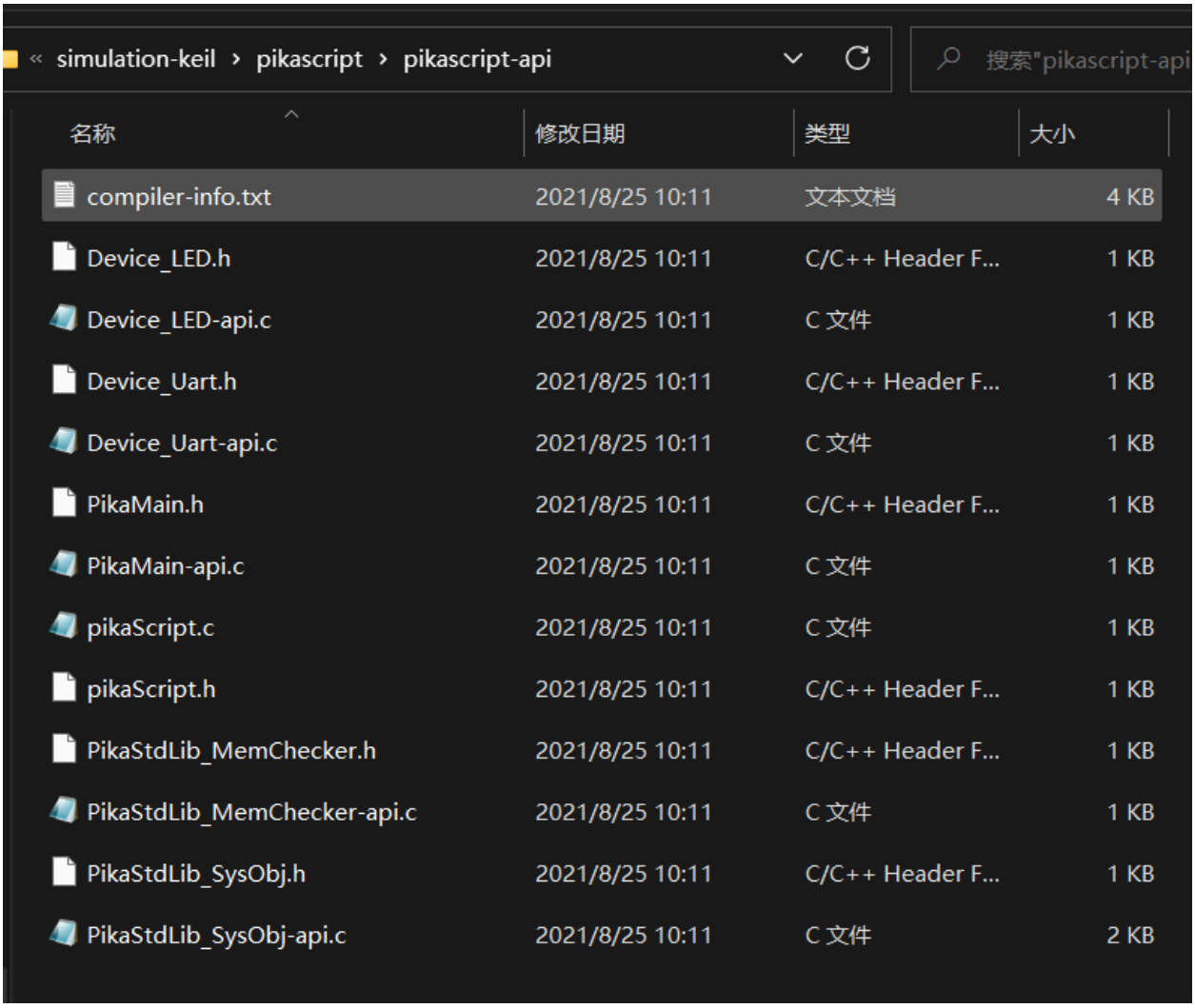


If you want to ensure that the updated script is compiled as expected, please

1. delete all files in the `pikascript-api` folder,
2. run the precompiler and
3. check whether the new `.c` and `.h` files have been generated or not.

IMPORTANT: Please do **NOT** remove the `pikascript-api` folder but only the files inside.

Here is an example that shows the `*.c` and `*h` files generated in the `pikascript-api` folder



Now, let's modify main.py as a practice:

```
import Device
import PikaStdLib

led = Device.LED()
uart = Device.Uart()
mem = PikaStdLib.MemChecker()

print('hello wrold')
uart.setName('com1')
uart.send('My name is:')
uart.printName()
print('mem used max:')
mem.max()
print('mem used now:')
mem.now()

# new code start
print('add new code start')
```

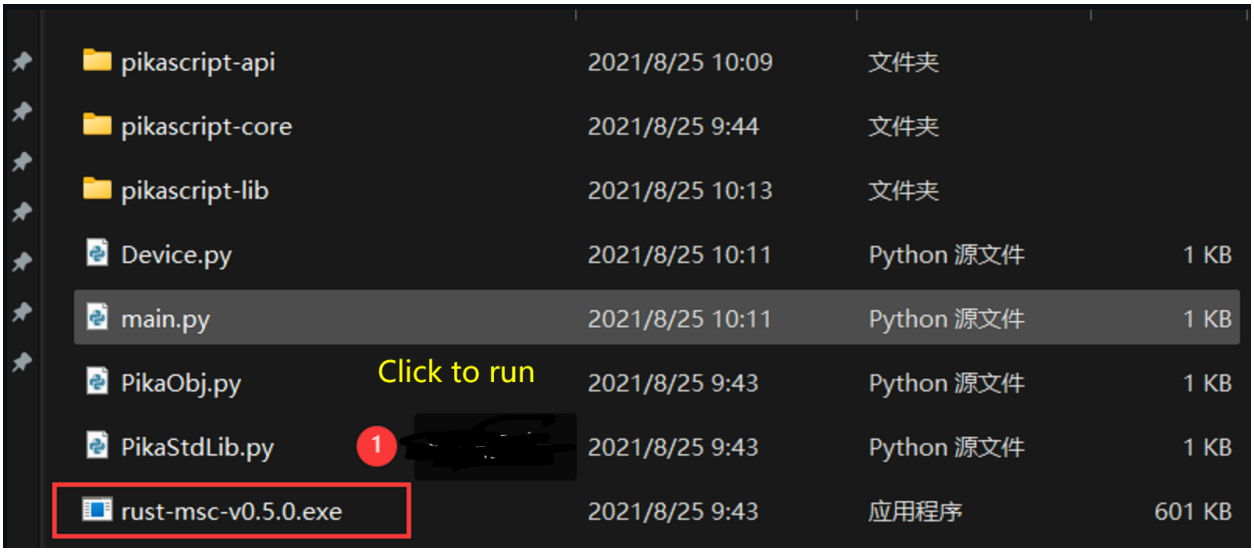
(continues on next page)

(continued from previous page)

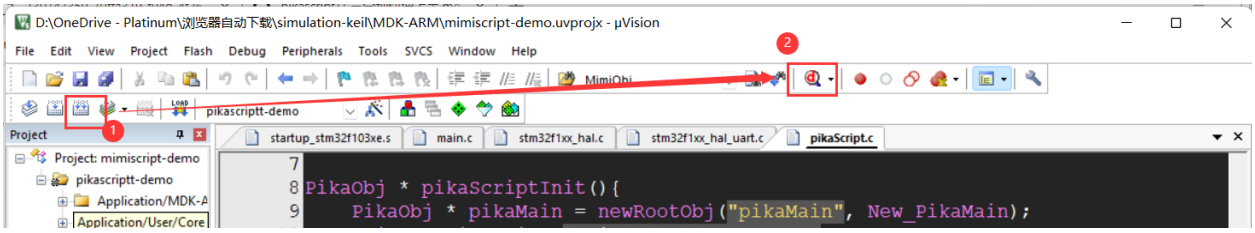
```
uart.setName('com2')
uart.printName()
print('add new code end')
# new code end
```

As you can see, we have added 4 new lines to the `main.py`. Let's compile and run:

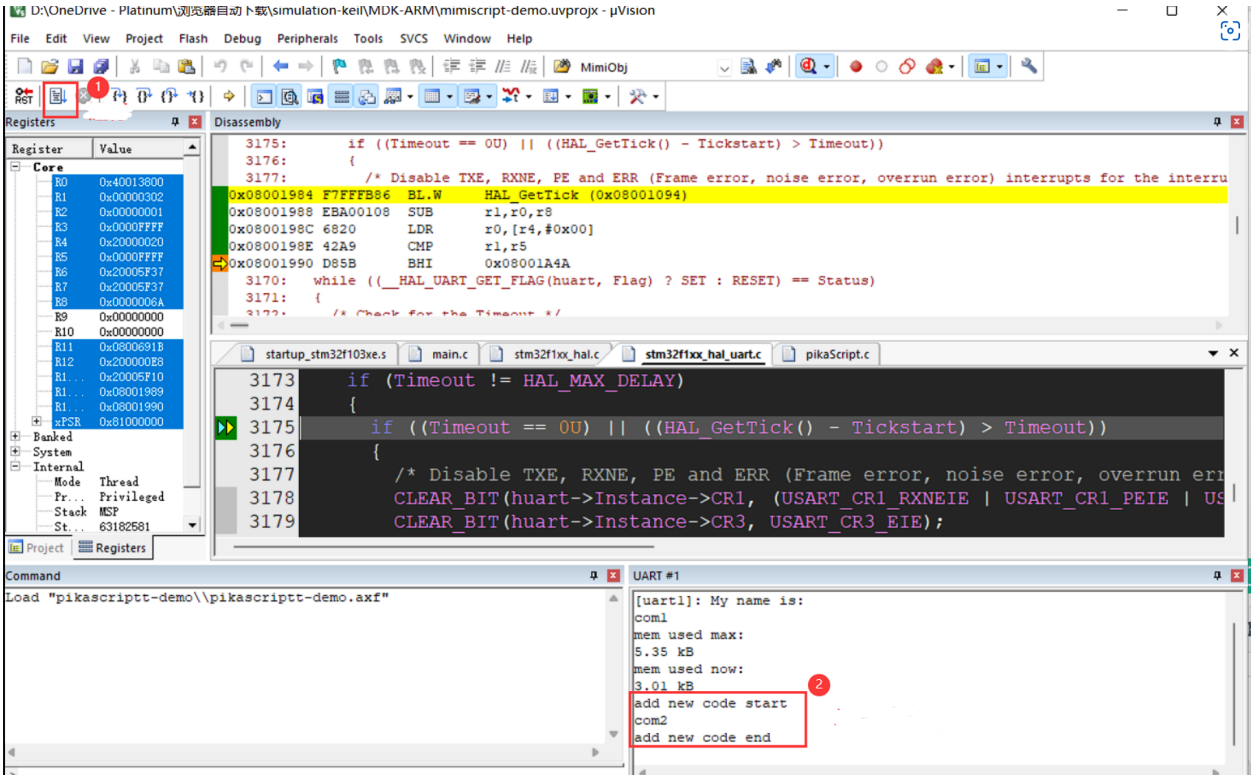
Compile `pikascript-api`



Compile the keil project and enter the debugging session:



run and observe the output



As shown above, there are 3 new lines in the output, indicating that our modification works as expected.

That's all, enjoy!!

2.2 Use BSP project

2.2.1 create project

Enter pikascript official website http://pikascript.com Select the platform, module, and click "Start Build". (The default module will be automatically selected after selecting the platform)

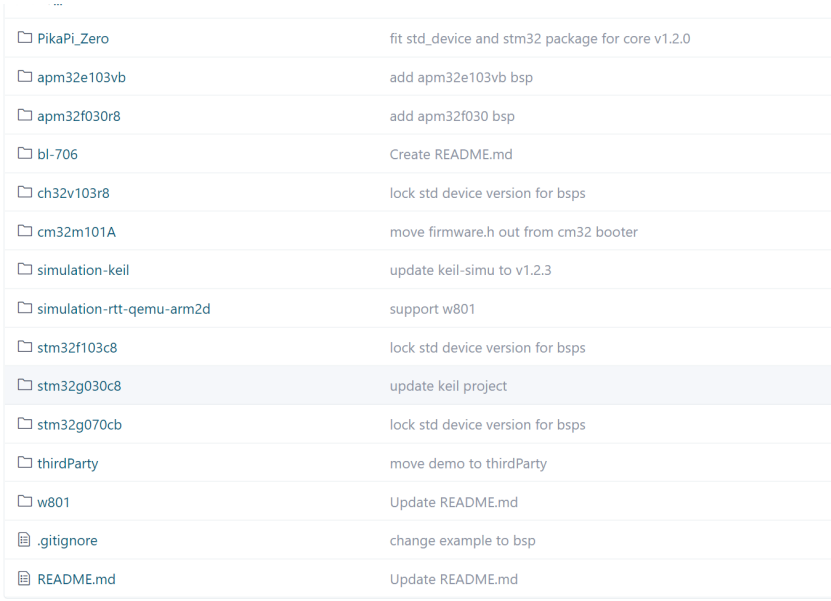

Create PikaScript Project
平台选择 Platform Selection
stm32g030c8
模块安装 Package install
(提示: 如果你是新手，为避免兼容性问题，请使用默认模块)
(Tips: if you're new to it, use the default packages to avoid compatibility issues)
pikascript-core v1.8.0
PikaStdLib v1.8.0
PikaStdDevice v1.8.0
STM32G0 v1.3.0
STM32F1 v1.1.0
STM32F4 v0.0.2
PikaPiZero v1.2.0
Arm2D v0.4.0
CH32V103 v1.0.0
pikaRTThread v1.3.0
pikaRTDevice v1.1.0
SmartLoong v0.0.1
PikaVSF v0.0.1
W801Device v1.0.0
CH582 v1.0.0
ctypes v0.0.2
生成工程 Generate Project
开始生成 Start

2.2.2 The source of the project

The transplanted bare metal MCU project is in the pikascript/bsp directory, and each folder in it is a transplanted bare metal project.

Each project is independent and can be copied out of the pikascript repository for separate use.

(simulation-keil-dev and pico-dev are not listed. These two bsp can only be used in the warehouse and are used to develop the kernel.)



https://github.com/pikastech/pikascript/tree/master/bsp

2.2.3 Support list

In the README.md in the bsp folder, the current platform support and the usage of bsp are marked.

(The table below is not up-to-date)

Click here for the latest form

MCU support

MCU	bsp	gpio	uart	pwm	adc	i2c
stm32g030c8	√	√	√	√	√	√
stm32g070cB	√	√	√	√	√	√
stm32f103c8	√	√	√	√	√	√
stm32f103rb	√	√	√	√	√	√
stm32f103rc	√	√	√	√	√	√
ch32v103r8t6	√	√				
cm32m101a	√					
w801	√					
apm32f030r8	√					
apm32e103vb	√					
bl-706	√					

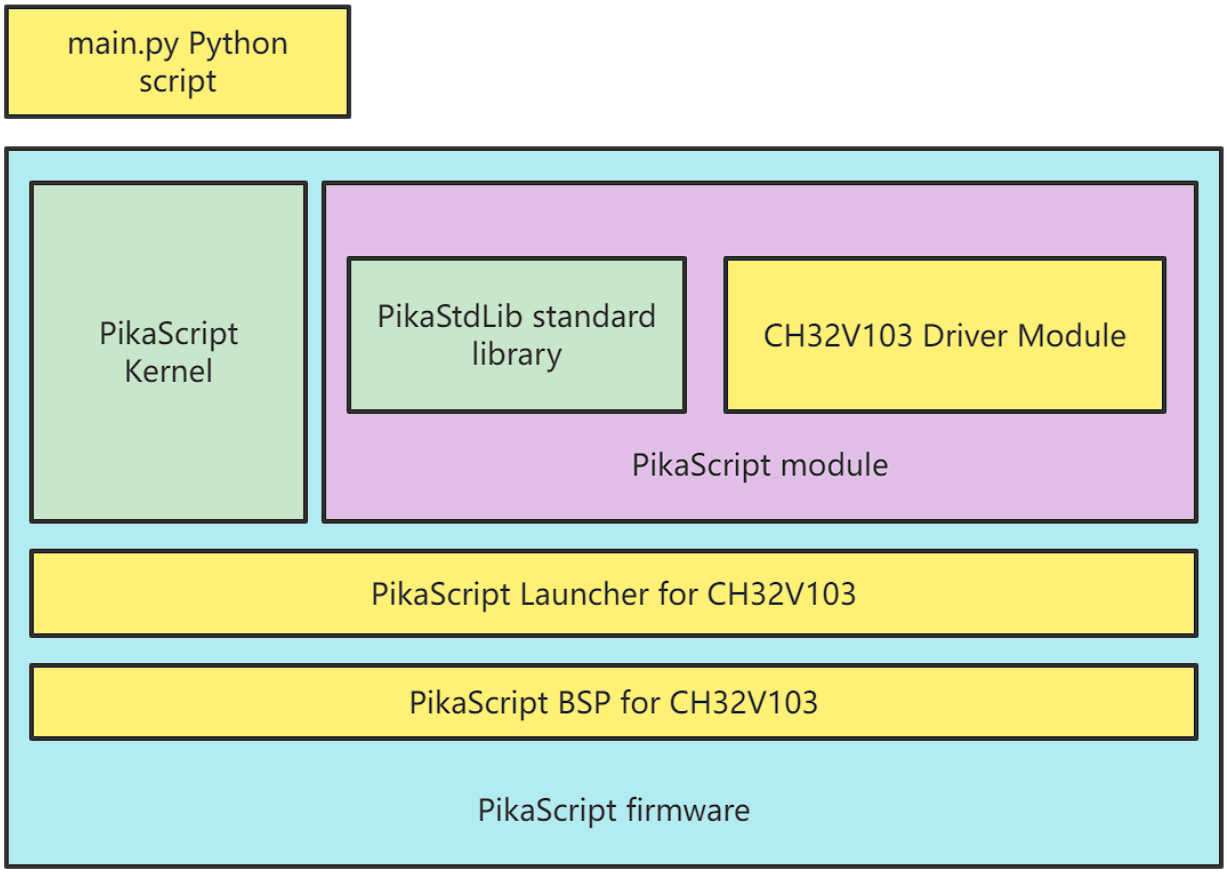
Board support

Board	bsp	gpio	uart	pwm	adc	i2c	rgb	lcd	arm-2d
Pika-Pi-Zero	√	√	√	√	√	√	√	√	√
QEMU-arm2d	√								√
Raspberry Pico	√								

You can help PikaPython extend this table by contributing **driver modules** or **bsp**, please refer to the **New Platform Porting Guide**, **Module Development** and **Package Management** in the documentation for details.

2.2.4 Projcet structure

Taking CH32V103 as an example, a PikaPython project includes the following parts.



1. The first is the part of the BSP folder except the PikaPython folder. This part is the real BSP, including the basic peripheral library provided by the manufacturer, CMSIS and other common libraries on some platforms. You can get it sorted.
2. The above part is the launcher of PikaPython, including the main.c entry file, the pika_config.c configuration file, and the *.s assembly startup file. The launcher is responsible for supporting printf, stack settings, the startup of PikaPython, as well as some functions such as interactive operation, serial port download of Python, etc.

pika_config.c is used to support some advanced functions such as downloading Python through serial port. PikaPython can still run without this file.

1. The above is the main part of PikaPython, which is divided into two parts: the kernel and the module. The kernel is the file in pikascript/src. You can choose a version and add it to compile. **No modification is required.**
2. Module part can be developed by yourself, or pulled from the warehouse, PikaStdLib standard library module is required. Other modules are optional.

For how to use modules and how to make modules, please refer to the **Module Development** section, and for how to contribute modules to the PikaPython reference, please refer to the How to contribute PikaPython modules section.

1. The top layer is the Python script that the PikaPython project can support. The Python script can be directly interpreted and run. There are various ways to load the script, including **pre-compiled into firmware, interactive operation, serial port download of Python scripts**, etc., pre-compiled For firmware, please refer to the **Module Development** section, and for interactive operation and serial port download, please refer to the **New Platform Porting** section.

Only modules imported in main.py will be compiled into the firmware, so main.py can also play the role of **trimming modules**.

2.2.5 module management

Launchers, kernels and modules can all be managed using the package manager.

Therefore, the PikaPython folder in the BSP only contains the package manager **pikaPackage.exe** itself, the **requestment.txt** module description file and the **main.py** sample script three files.

requestment.txt uses the same module description syntax as general python. Running pikaPackage.exe directly can identify requestment.txt in the current folder and pull the corresponding module.

Taking requestment.txt in the bsp of stm32g030 as an example, the pulled modules are:

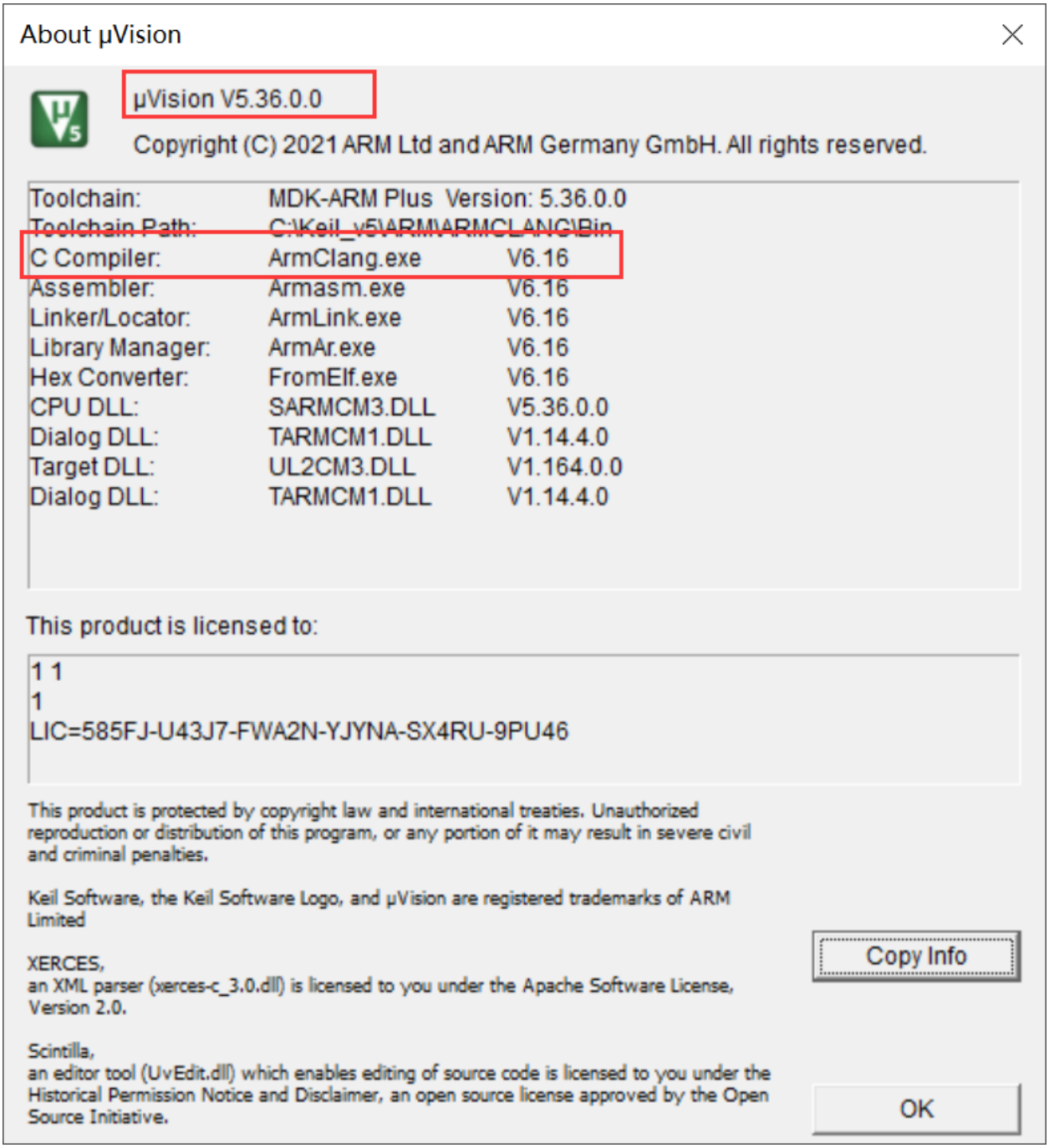
- Kernel: pikascript-core
- Standard library: PikaStdLib
- Peripheral module: STM32G0 PikaPiZero PikaStdDevice

```
pikascript-core==v1.10.0
PikaStdLib
PikaStdDevice==v1.6.0
STM32G0==v1.2.0
PikaPiZero==v1.1.3
```

The pulled module needs to be precompiled, just run rust-msc-latest-win10.exe directly.

2.2.6 Precautions

1. Keil version **strongly recommended** not lower than **5.36**



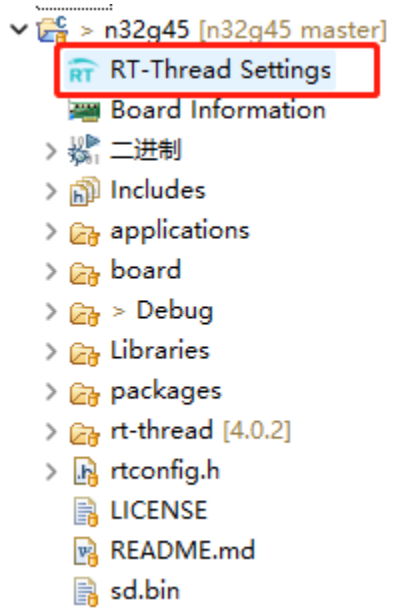
2.3 Start with RT-Thread package

PikaPython has been added to the RT-Thread package. Under the programming language category, you can quickly use PikaPython by directly adding packages.

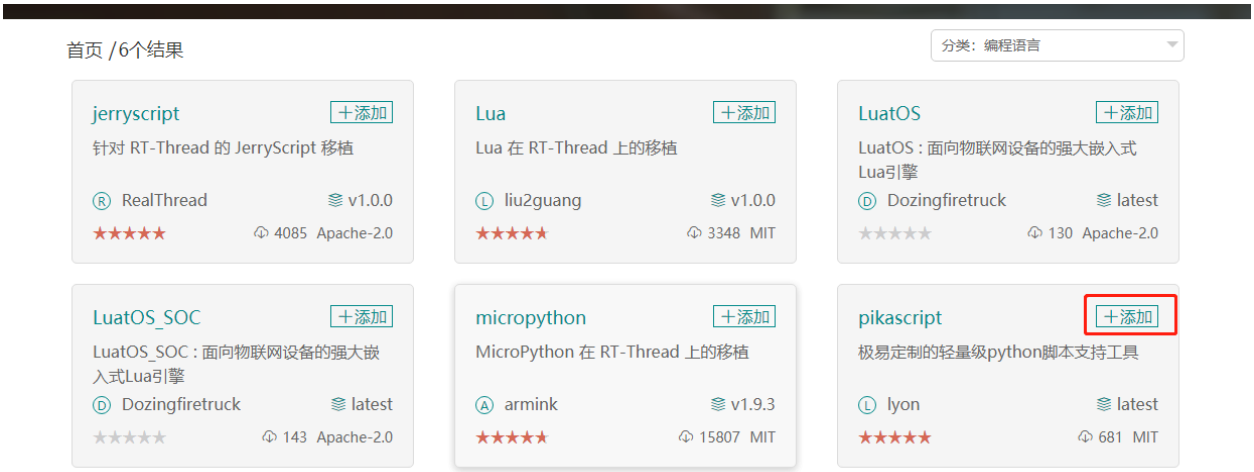
The PikaPython package supports **full RT-Thread BSP**.
If you encounter compatibility problems during use, you can file an issue at github or Forum to ask questions.

2.3.1 Install

Import the pikascript package


n32g45 [n32g45 master]
RT-Thread Settings
Board Information
二进制
Includes
applications
board
Debug
Libraries
packages
rt-thread [4.0.2]
rtconfig.h
LICENSE
README.md
sd.bin


RT-Thread Settings
软件包(2)
添加软件包
IOT
与物联网相关的软件包，包括网络相关软件包，云接入软件包等
外设
与底层外设硬件相关的软件包，sensor软件包
系统
系统级软件包，监控系统行为、其他文件系统等
编程语言
工具
辅助使用的一些工具软件包
杂类
一些未归类的软件包，demo，示例等



Add RT_WEAK before rt_vsnprintf in rt-thread/src/kservice.c **(only for rt_thread version 4.1.0 and below)**

```
777 RT_WEAK rt_int32_t rt_vsnprintf(char       *buf,
778                                 rt_size_t   size,
779                                 const char *fmt,
780                                 va_list     args)
781 {
```

Delete the static **(only for rt_thread version 4.1.0 and below)** of finsh_getchar in rt-thread/components/finsh/shell.c

```
167 static int finsh_getchar(void)
168 {       删除
169 #ifdef RT_USING_DEVICE
170 #ifdef RT_USING_POSIX
171     return getchar();
172 #else
```

2.3.2 start pikascript

Option 1: Start with msh (default mode)

Use the pikaRTThread module in packages/pikascript-latest/requestment.txt (included by default).

The latest default requestment.txt can be viewed here.

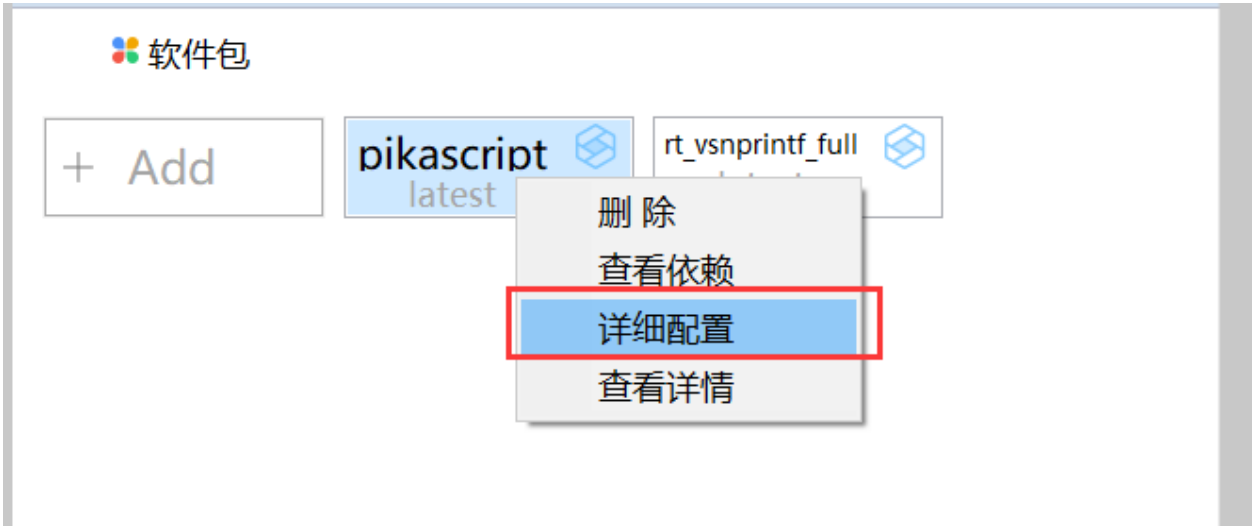
Type “pika” in msh to **start** PikaPython in a thread.

The initial startup will execute the /pikascript-latest/main.py initialization script. After execution, enter pika **interactive running** mode, Enter “exit()” to return to msh, enter “pika” again to enter pikascript, and enter directly into interactive mode.

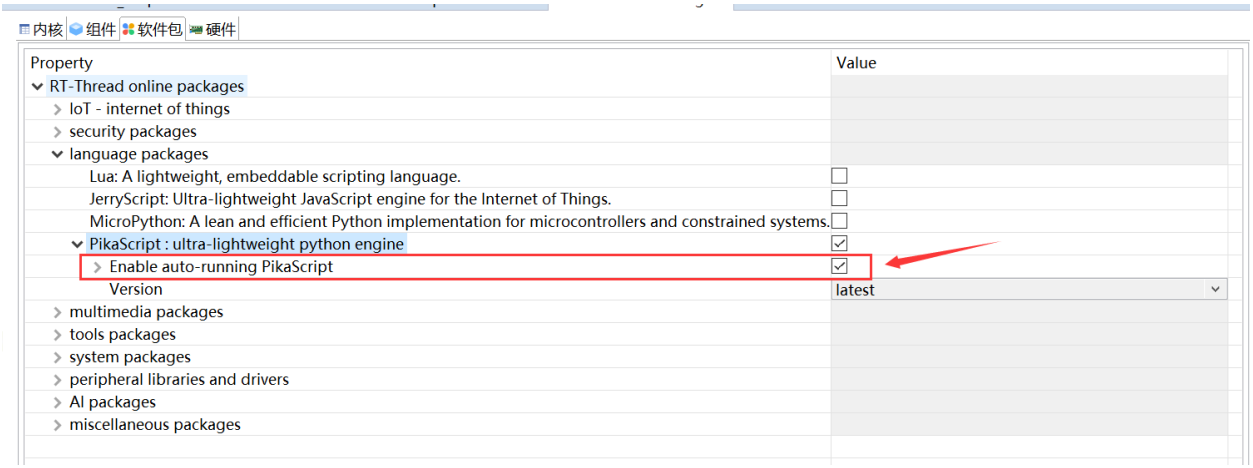

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Dec  9 2021
 2006 - 2020 Copyright by rt-thread team
msh >pika
----------------------------------------------------------------------
|                                                                    |
|     ____  _  __                 _____           _       _   __    |
|    / __ \ (_) / /__ ____ _      / ___/ _____ ____ (_) ____   / /_   |
|   / /_/ // / / //_// __ `/      \__ \ / ___// ___// / / __ \ / __/   |
|  / ____// / / ,<  / /_/ /      ___/ // /__ / /   / / / /_/ // /_     |
| /_/    /_/ /_/|_| \__,_/      /____/ \___//_/   /_/ / .___/ \__/     |
|                                                     /_/            |
|          PikaScript - An Ultra Lightweight Python Engine           |
|                                                                    |
|           [ https://github.com/pikastech/pikascript ]             |
|           [  https://gitee.com/lyon1998/pikascript  ]              |
|                                                                    |
----------------------------------------------------------------------
hello PikaScript!
hello PikaScript!
hello PikaScript!
>>> print('test')
test
>>> print(a)
[error] print: can not print val
[info] input commond: 0 RUN print
>>> a = 2
```

Option 2: Automatically start at boot

Enter the package detailed configuration



Check Enable auto-running PikaPython



3 After setting, it will automatically start PikaPython, run the main.py script, and then go back to msh

Enter **pika** in msh to run interactively.

Option 3: Manual start

If you need **custom start**, you can use the following methods to start manually.

Import header file:

```
#include "pikaScript.h"
```

Start PikaPython:

```
PikaObj * pikaMain = pikaScriptInit();
```

run interactively

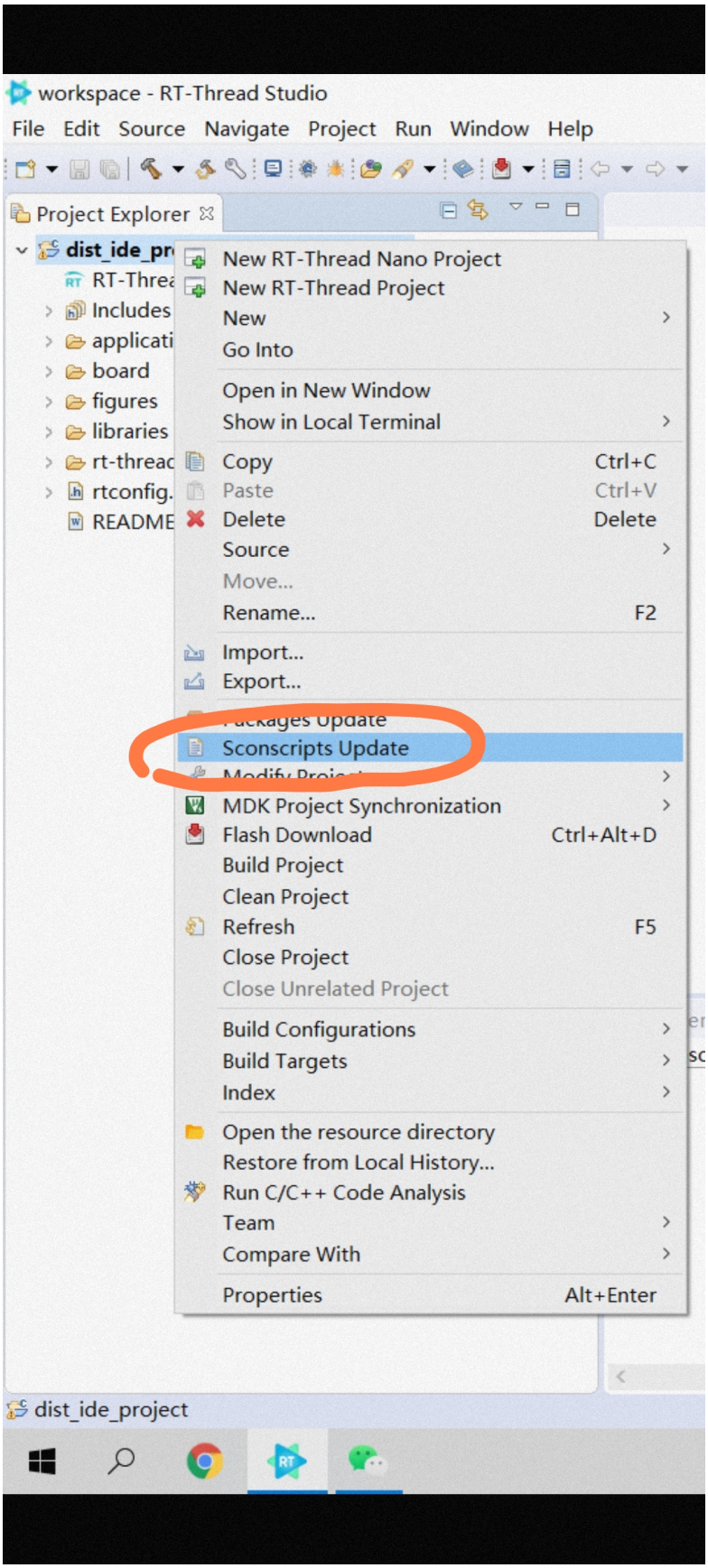
Refer to the **Support Interactive Run** section of the documentation.

Serial download Python script

Refer to the **Support Serial Port Download Python** part of the document.

Using the PikaPython module and package manager

Modify pikascript-latest/requestment.txt, then right-click the project, Sconscripts Update, you can install the module/modify the module version, and precompile.


workspace - RT-Thread Studio
File Edit Source Navigate Project Run Window Help
Project Explorer
dist_ide_pr
RT-Threa
Includes
applicati
board
figures
libraries
rt-thread
rtconfig.
README
New RT-Thread Nano Project
New RT-Thread Project
New
Go Into
Open in New Window
Show in Local Terminal
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Sconscripts Update
MDK Project Synchronization
Flash Download Ctrl+Alt+D
Build Project
Clean Project
Refresh F5
Close Project
Close Unrelated Project
Build Configurations
Build Targets
Index
Open the resource directory
Restore from Local History...
Run C/C++ Code Analysis
Team
Compare With
Properties Alt+Enter
dist_ide_project

For more usage, please refer to the **package manager**, **module usage, module development** part of the documentation.

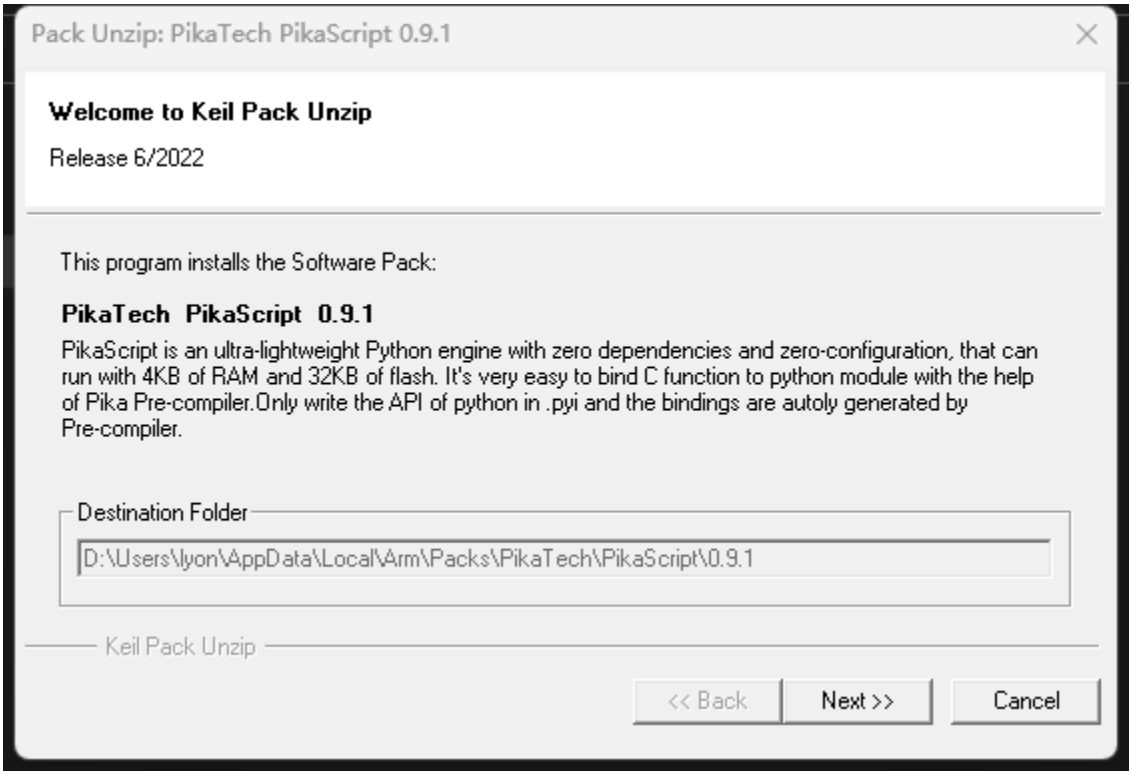
2.4 Start with CMSIS-PACK

Users developing with Keil can use CMSIS-PACK to install PikaPython with one click.

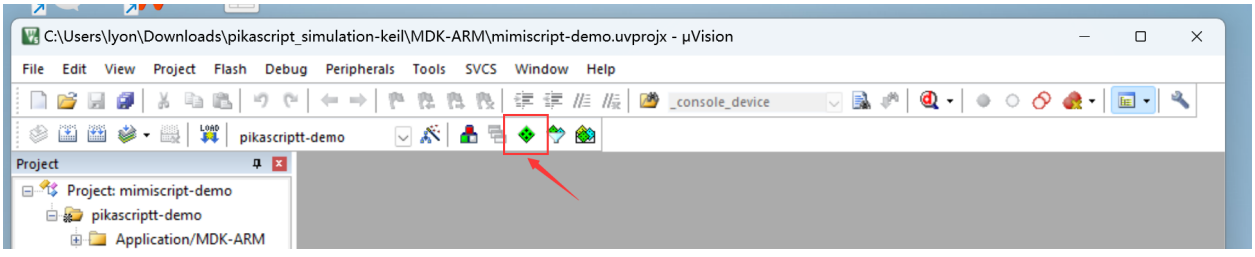
2.4.1 Install PikaTech.PikaPython.x.x.x.pack

Click to download

Just go all the way to Next and install



2.4.2 Set in the project

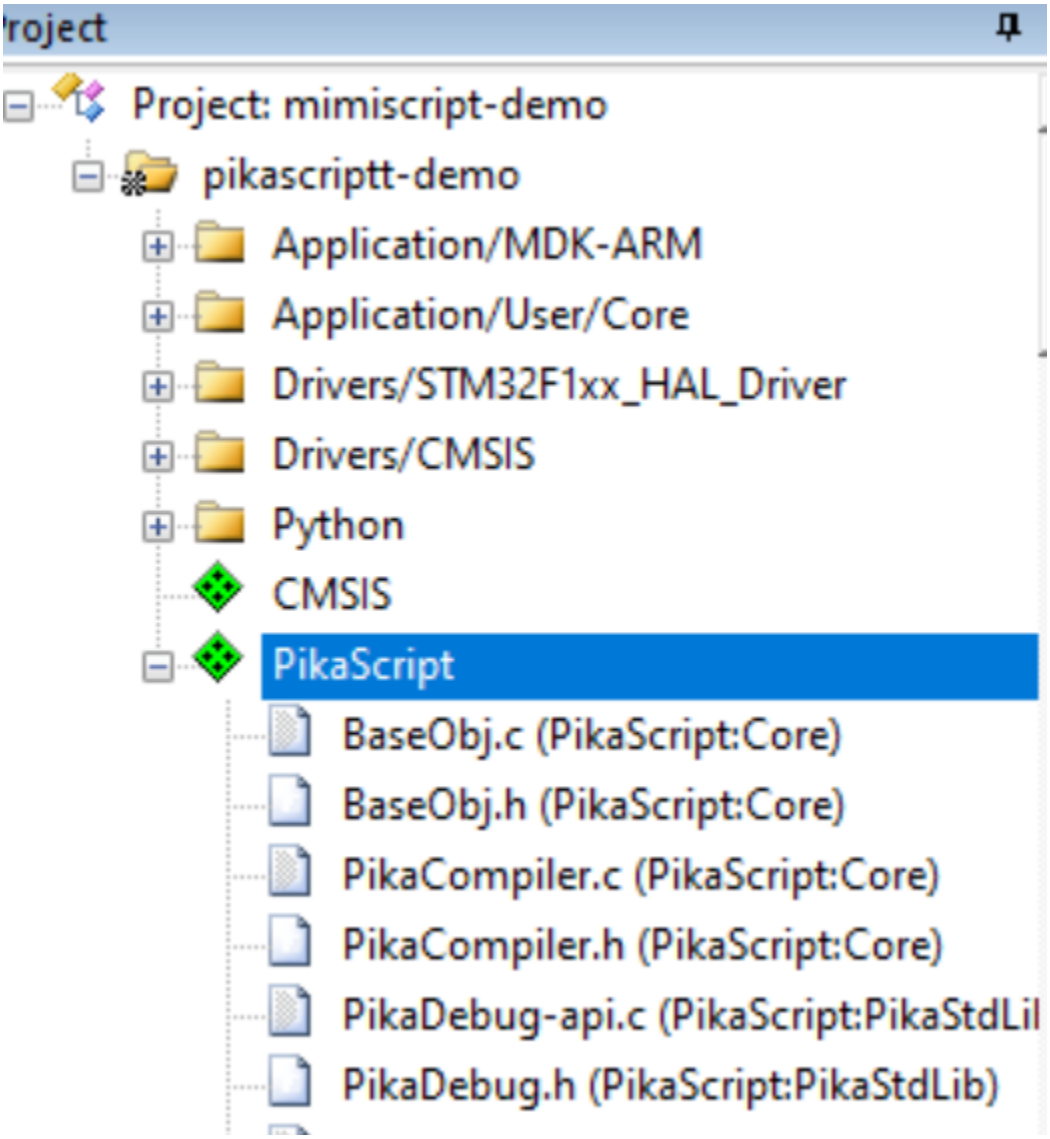


Check PikaPython, including Core and PikaStdLib

Manage Run-Time Environment

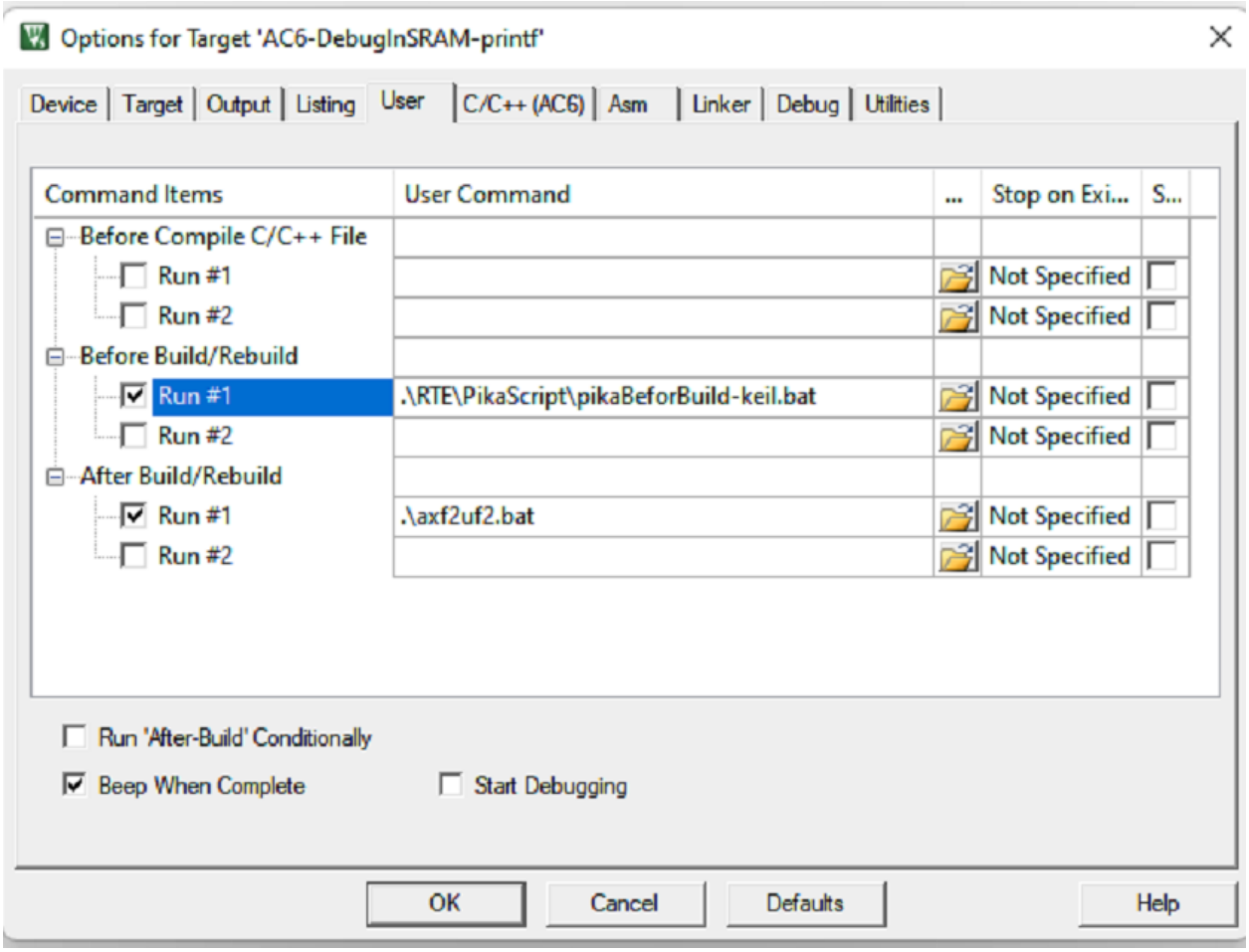
Software Component	Sel.	Variant	Version	Description
Board Support		MCBSTM32C	2.0.0	Keil Development Board MCBSTM32C
CMSIS				Cortex Microcontroller Software Interface Components
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler		ARM Compiler	1.7.2	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device				Startup, System Setup
File System		MDK-Plus	6.15.0	File Access on various storage devices
Graphics		MDK-Plus	6.24.0	User Interface on graphical LCD displays
Network		MDK-Plus	7.17.0	IPv4 Networking using Ethernet or Serial protocols
PikaScript		PikaScript	1.8.6	an ultra-lightweight Python engine
PikaScript				
Core	☑		1.8.6	PikaScript Kernel
PikaStdLib	☑		1.8.6	The standard library for PikaScript
USB		MDK-Plus	6.16.0	USB Communication with various device classes
Utilities		Performance Counter	1.9.4	A dedicated performance counter for Cortex-M systick.

Here you can see that PikaPython has been added



In Before Build add

```
.\RTE\PikaPython\pikaBeforBuild-keil.bat
```



Then introduce in main.c

```
#include "pikaScript.h"
```

Start PikaPython after initializing the system and printf

```
PikaObj *pikaMain = pikaScriptInit();
```

Compile successfully.

Build Output

```
compiling PikaStdLib_PikaObj-api.c...
compiling PikaStdTask_Task-api.c...
compiling PikaStdTask-api.c...
compiling __asset_pikaModules_py_a.c...
compiling pikaScript.c...
compiling PikaStdLib_SysObj.c...
linking...
Program Size: Code=88468 RO-data=3932 RW-data=20 ZI-data=24668
FromELF: creating hex file...
"pikascriptt-demo\pikascriptt-demo.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:06
```

Run successfully !

UART #1

```
======[pikascript packages installed]======
PikaStdLib==v1.8.6
pikascript-core==v1.8.6
===========================================
hello PikaScript!
mem used max:
1.78 kB
>>>
```

For more usage, please refer to porting guide

2.5 Start with the Docker Development Environment

2.5.1 Why use docker development environment

PikaPython's kernel and standard libraries are developed in a docker environment, which can be prone to some hard-to-debug problems when developing features that involve the kernel internals, such as

- memory leaks
- memory overruns
- broken kernel functionality

This problem can be avoided by using PikaPython's docker development environment, which has been installed with **unit testing framework** and **memory checking tool**, so that if there is a memory security problem, it can be quickly found and solved to avoid memory hazards.

PikaPython's linux development platform also needs to install go, rust, GoogleTest, GoogleBenchmark, valgrind and other tools, which is rather cumbersome, Docker-based development environment can install these tools in one click, and ensure that all developers' development environment is consistent.

2.5.2 Build Docker container

Please make sure you have installed Docker on the host:

- Install Docker directly on Linux platform
- Install Docker-Desktop on Windows platform
  - Docker-Desktop requires the installation of wsl2

(For windows platform, you can use the following command in wsl, not PowerShell)

step1: Clone the repository

```
git clone https://github.com/pikastech/pikascript
cd pikascript/docker
```

step2: Build the Docker image, then start the container

```
bash build.sh
sh run.sh
```

step3: Initialize the port/linux

```
cd port/linux
sh pull-core.sh
sh init.sh
```

step4: Run GoogleTest, BenchMark, and valgrind

```
sh gtest.sh
sh ci_benchmark.sh
sh valgrind.sh
```

step5: Run REPL

```
sh run.sh
```

For more development guidelines under Docker, please refer to Development Process for Standard Libraries .

2.6 Start with the LVGL GUI Simulation Project

The LVGL GUI Simulation Project provides an experimental environment for co-simulation of PikaPython and LVGL.

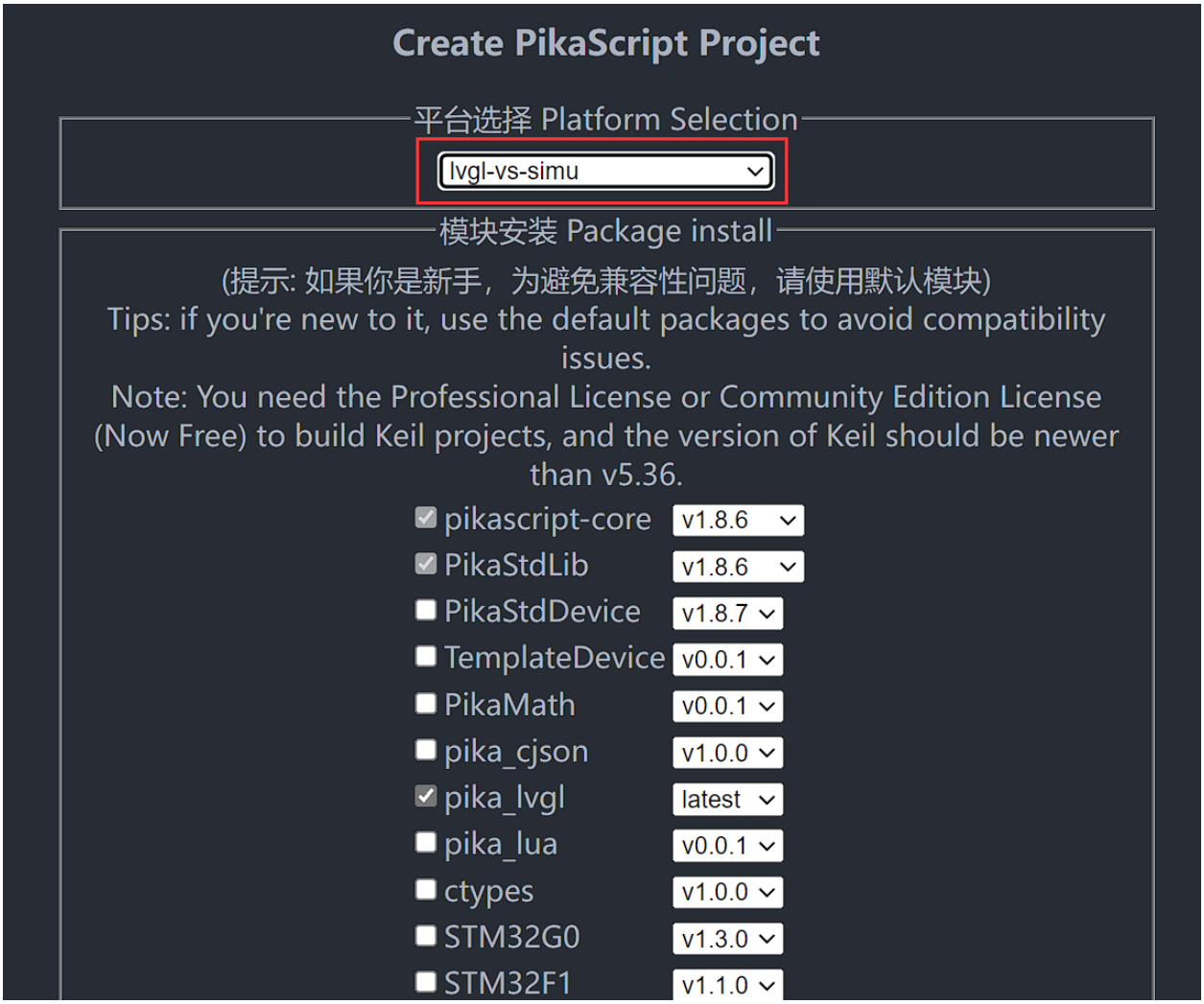
The GUI simulation can be performed on a PC using Visual Studio.

2.6.1 Get the project

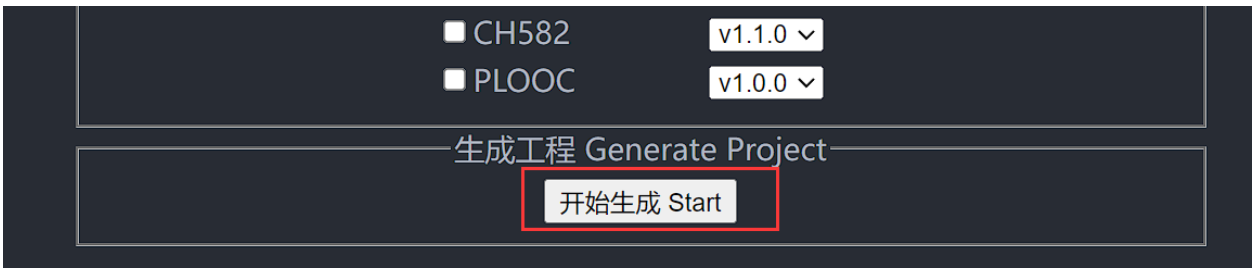
http://pikascript.com/

Select lvgl-vs-simu, a Visual Studio simulation project, from the Project Builder on the official PikaPython website.

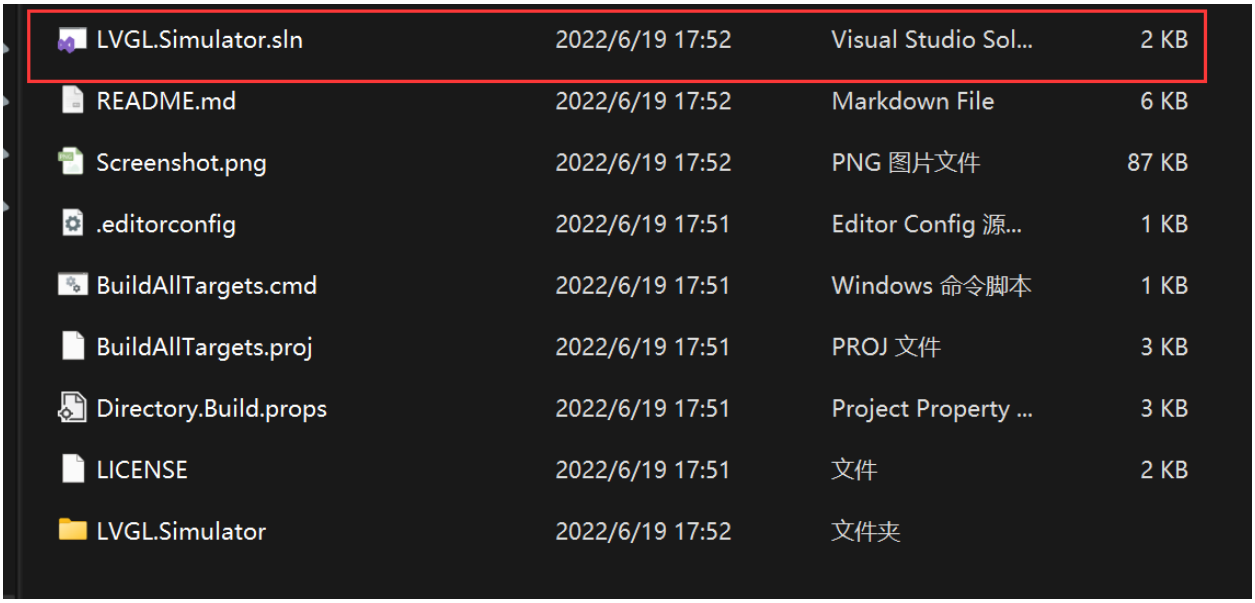
This project is branched from the official LVGL Visual Studio simulation project.



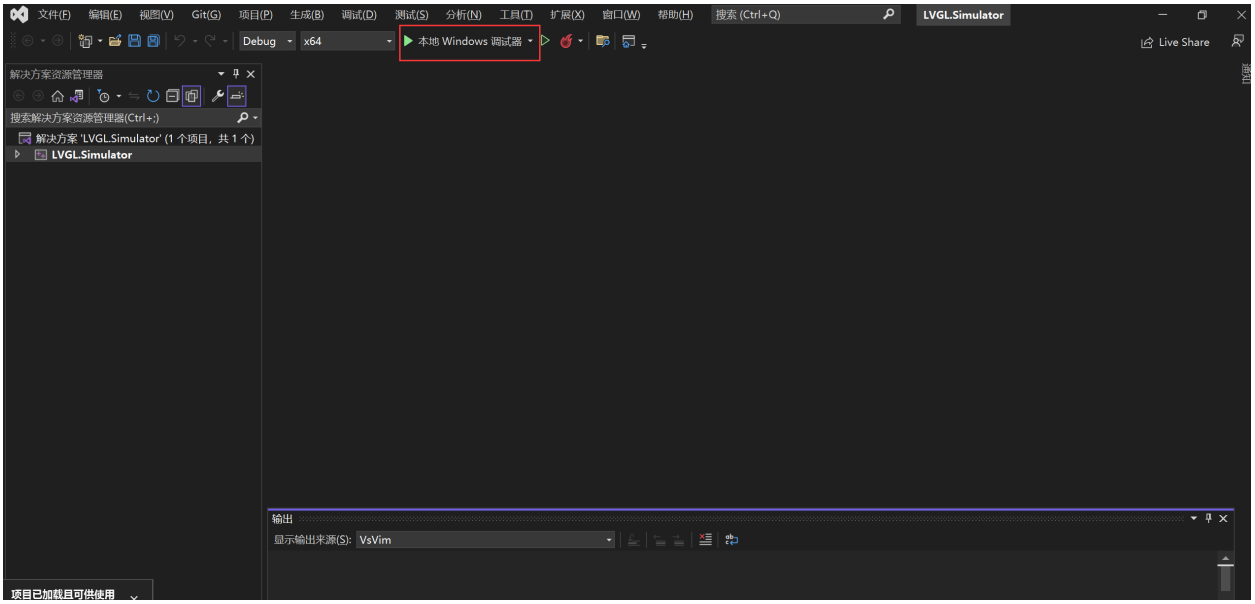
Click Generate Project and wait about 1 minute.



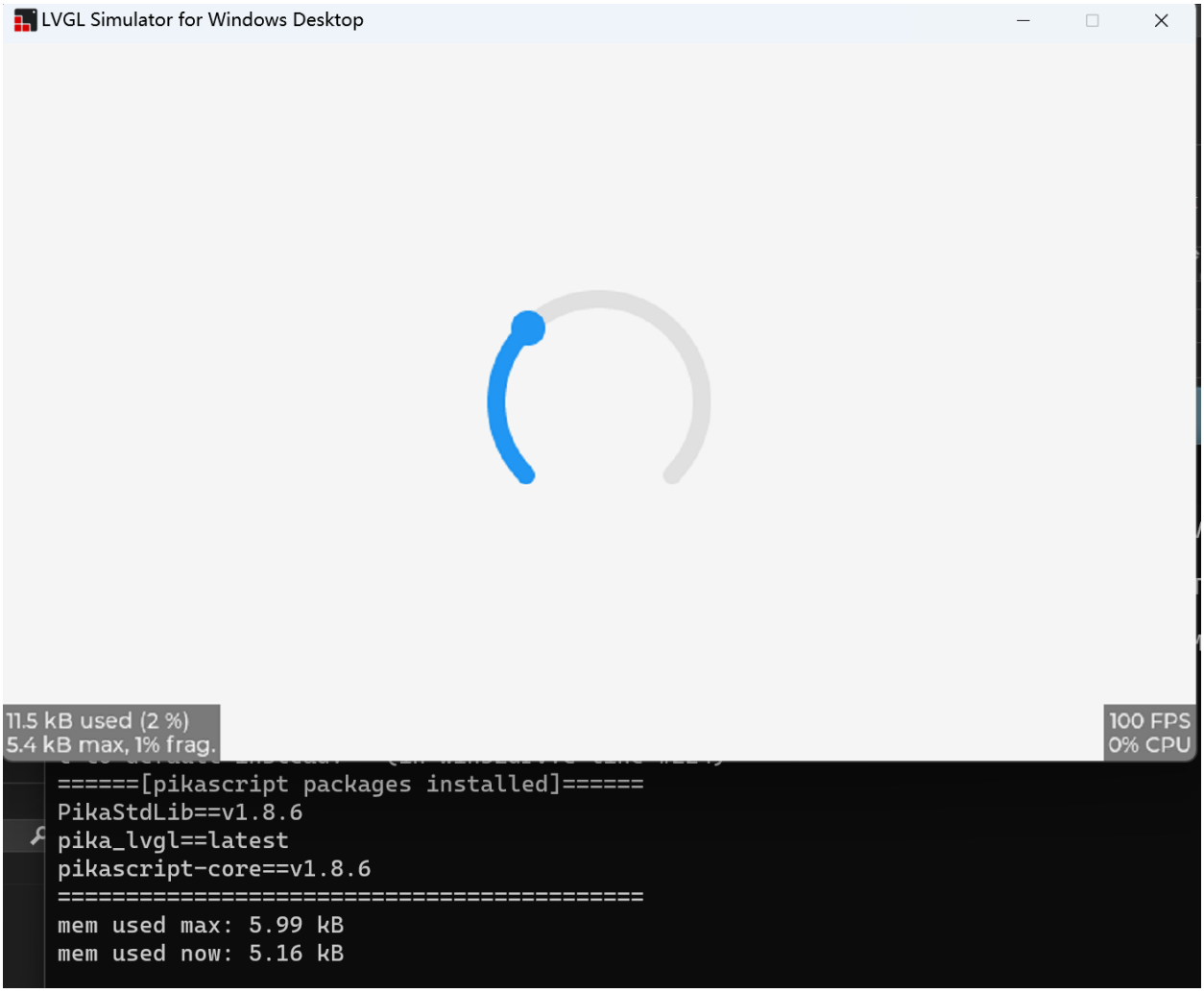
Unzip the project and open LVGL.Simulator.sln



Start compiling and running directly

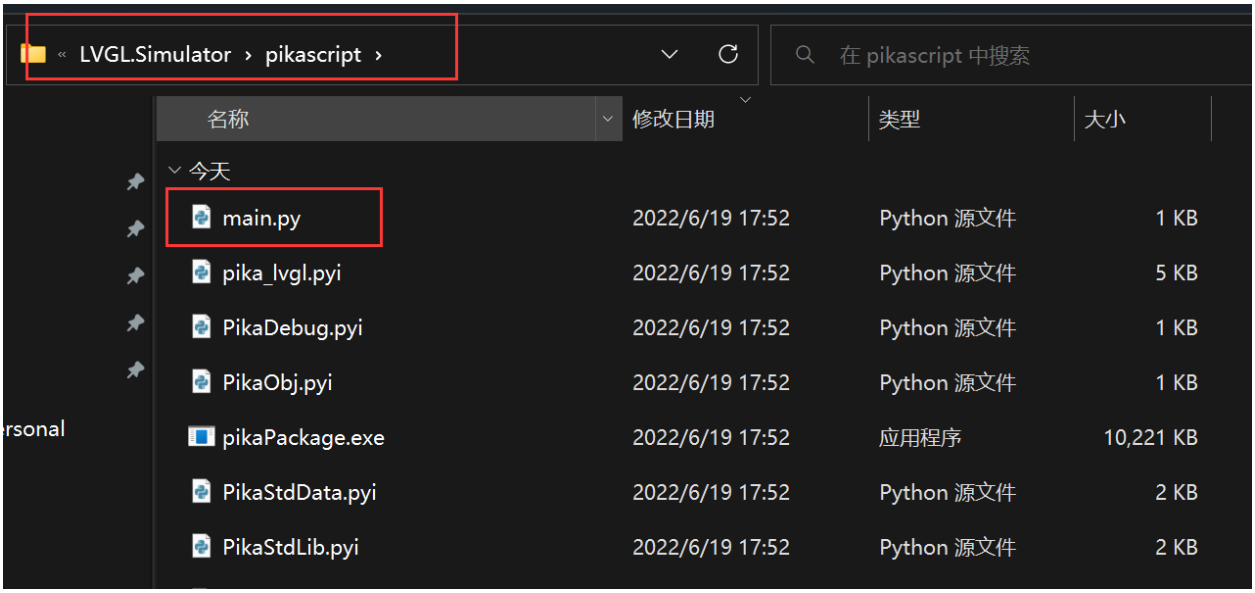


You can see that the lvgl emulator has been successfully started



2.6.2 Programming with Python

The Python file for running the project is in LVGL.Simulator/pikascript/main.py, and it is recommended to edit the Python file with VSCode.



The code in main.py is shown below, and the project will run this main.py when it starts

```
# main.py
import pika_lvgl as lv
import PikaStdLib
mem = PikaStdLib.MemChecker()

# Create an Arc
arc = lv.arc(lv.scr_act())
arc.set_end_angle(200)
arc.set_size(150, 150)
arc.center()

print('mem used max: %0.2f kB' % (mem.getMax()))
print('mem used now: %0.2f kB' % (mem.getNow()))
```

More sample code

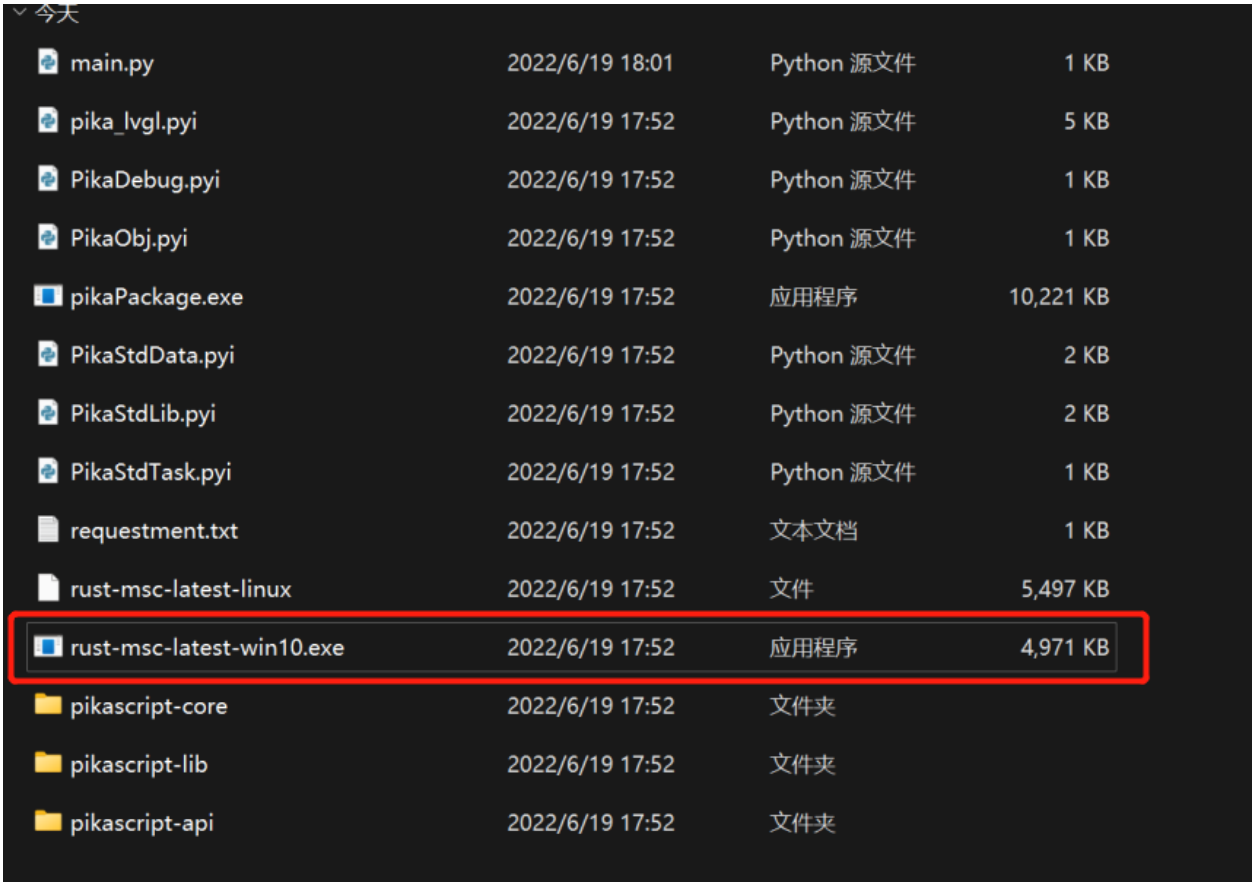
You can see more sample code in the /pikascript/examples/lvgl folder.

lv_arc1.py	!27 add lvgl package and examples
lv_arc2.py	!27 add lvgl package and examples
lv_bar1.py	!27 add lvgl package and examples
lv_btn1.py	!27 add lvgl package and examples
lv_callback1.py	!27 add lvgl package and examples
lv_checkbox1.py	!27 add lvgl package and examples
lv_label1.py	!27 add lvgl package and examples
lv_list1.py	!27 add lvgl package and examples
lv_obj1.py	!27 add lvgl package and examples
lv_obj2.py	!27 add lvgl package and examples
lv_roller1.py	!27 add lvgl package and examples
lv_slider1.py	!27 add lvgl package and examples
lv_style1.py	support style for lvgl
lv_switch1.py	!27 add lvgl package and examples
lv_table1.py	!27 add lvgl package and examples
lv_textarea1.py	!27 add lvgl package and examples

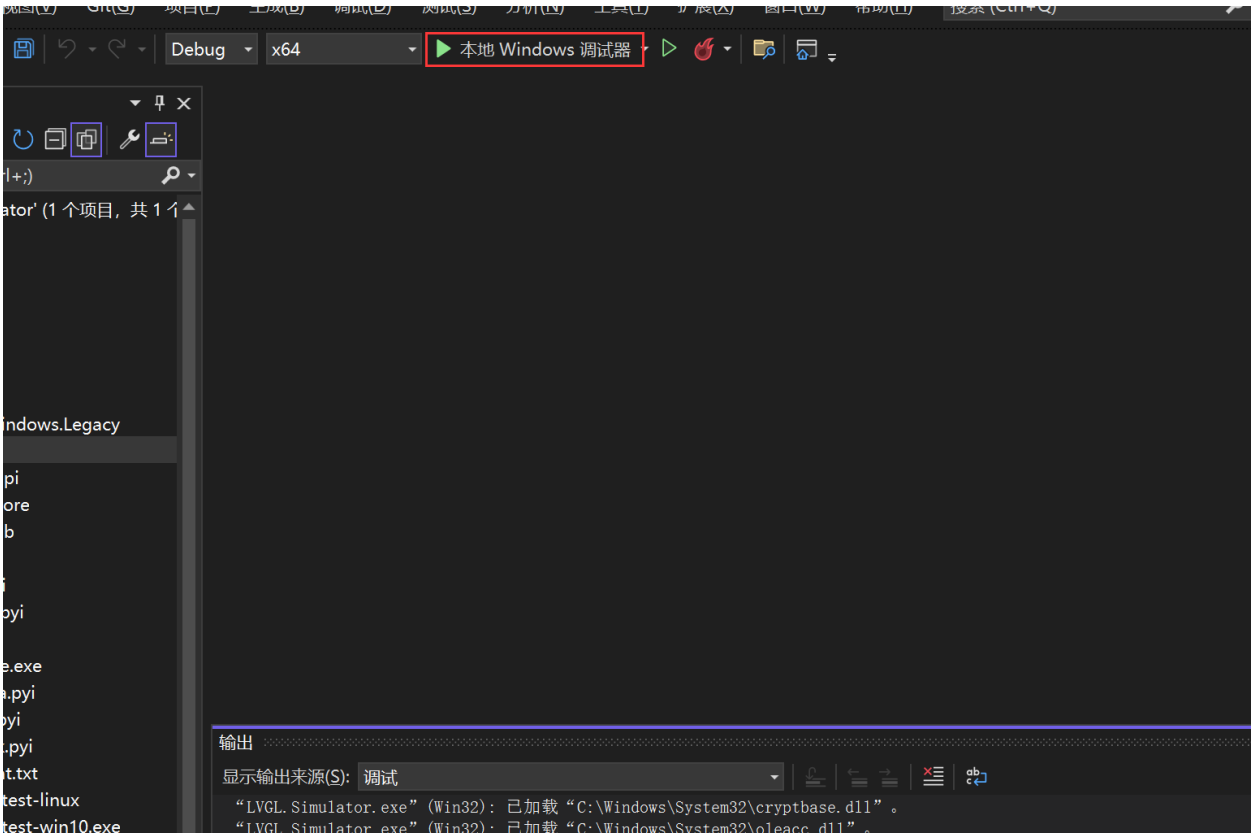
For example, you can copy lv_callback1.py into main.py.

```
# lv_callback1.py
import pika_lvgl as lv
import PikaStdLib
mem = PikaStdLib.MemChecker()


def event_cb_1(evt):
    print('in evt1')
    print('mem used now: %0.2f kB' % (mem.getNow()))


def event_cb_2(evt):
    print('in evt2')
    print('mem used now: %0.2f kB' % (mem.getNow()))


btn1 = lv.btn(lv.scr_act())
btn1.align(lv.ALIGN.TOP_MID, 0, 10)
btn2 = lv.btn(lv.scr_act())
btn2.align(lv.ALIGN.TOP_MID, 0, 50)
btn1.add_event_cb(event_cb_1, lv.EVENT.CLICKED, 0)
btn2.add_event_cb(event_cb_2, lv.EVENT.CLICKED, 0)

print('mem used max: %0.2f kB' % (mem.getMax()))
print('mem used now: %0.2f kB' % (mem.getNow()))
```

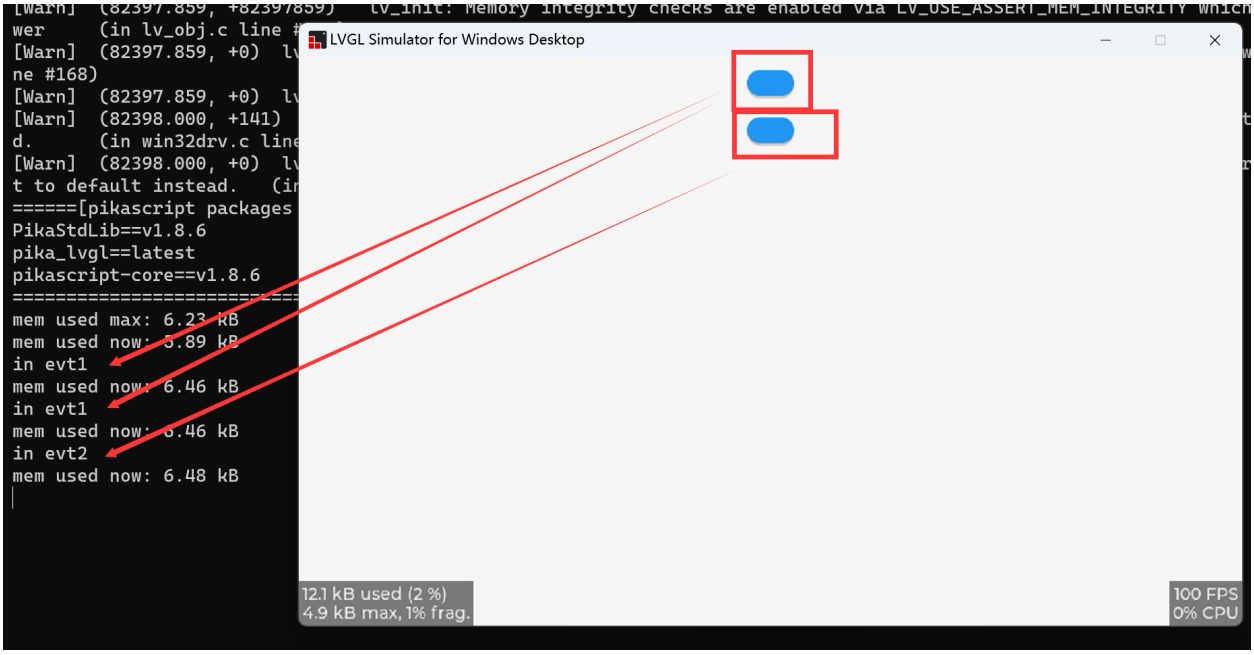
After replacing main.py, run PikaPython's pre-compiler



and then start running



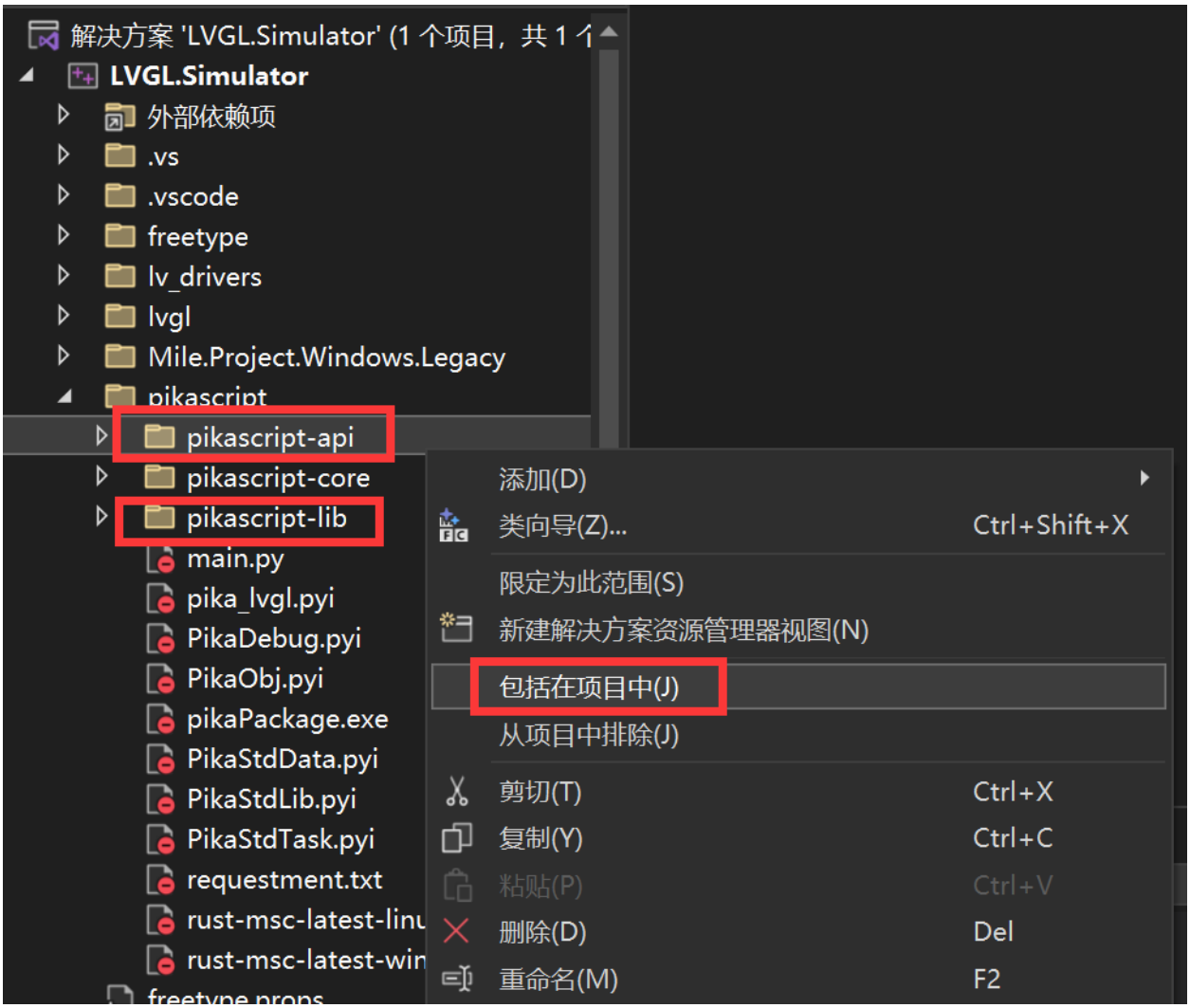
In this example you can click the button and then view the output.



2.6.3 Frequently Asked Questions

If you are prompted for missing functions, you need to manually add the files to be compiled

Right-click on pikascript/pikascript-api and pikascript/pikascript-lib and click “Include in project”, then recompile.



2.7 Play Python on Raspberry Pi Pico in MDK

It is well known that MicroPython supports the Raspberry Pi Pico, and we see there some room to improve, not only about the memory footprint, but also about the way to bind your own c modules. It’s not rare to see people in the community complain about the complexity and debugging experience.

The resources and price of the Raspberry Pi Pico are really good, it is fun to play with, not to mention there is a big community behind it. One question for most of the MCU developer is that can we use MDK to develop Raspberry Pi Pico and play with PikaPython? Why not? Thanks to a open-source project called Pico_Template, dream becomes reality. Please note that Pico_Template allows you to compile the latest pico-sdk using the Arm Compiler 6, debug without an extra pico and retarget printf to MDK without using any Serial2USB adapter.

For details, see:

I'm going to use MDK to develop Raspberry Pi Pico, how come!

As we mentioned before, binding C modules in MicroPython is very complicated and difficult to debug. Is there a more convenient way to do it for python running on MCUs?

YES! Our answer to this question is PikaPython. PikaPython is a completely rewritten ultra-lightweight python vitual machine, with zero dependency on toolchain, simple configuration, ultra-low memory footprint (i.e. you can use it with less than 4KB of SRAM). Using framework based C module development tools, your API calling written in Python can be automatically connect to the your C modules. Cannot be more simple or convenient, isn't it? No need to manually handle any global tables, macro functions, module registration, etc.

PikaPython provides MDK projects, hence you can debug C modules with python scripts.

For details, see: I'm going to use the cheapest single-chip microcomputer to run python, and I also need to use MDK to develop it, what's the matter

In addition to pico, the portability of pikascript allows you to use it on a wide variety of platforms. For example: stm32g0, stm32f1, ch32, apm32, cm32, as well as Pingtou's w801, Boliu's bl-706…

The very popular ESP32C3, Godson architecture.

PikaPython supports both bare-metal but also RTOS enviroment, for example RT-Thread, VSF, and Linux.

In fact, PikaPython is deeply integrated with rt-thread, it supports rt-thread full series of BSP via software packages.

Let's see how to play PikaPython on Raspberry Pi pico using MDK:

https://github.com/pikastech/pikascript/tree/master/bsp/pico#pikascript-in-pico

If you can see the following information (or similar) on the Debug(printf) View, congrats!

Enjoy!

For getting technical support, please raise issues on github. Thank you.

2.8 ARM-2D based GUI simulation project

2.8.1 Preface

good news! The Arm2D module and simulation project of pikascript are preliminarily sorted out! pikaScript, ARM-2D, rt-thread work together, unlock new poses for python to play Arm2D! No hardware is needed, and it can be simulated directly, which is very convenient.

It is also very simple to deploy and run this simulation project on your own computer, just follow the steps below~

2.8.2 Get the simulation project

Go to the PikaPython official website: http://pikascript.com, then select `sumulation-rtt-qemu-arm2d` for the platform, and then click Start to generate the project.

2.8.3 Install the development environment

After you have the project, you also need to install the development environment. There are only two things that need to be installed. One is rt-thread studio, which is used as an IDE. rt-thread studio integrates qemu, which is very convenient for simulating mcu and gui. The other is the latest arm gcc toolchain.

rt-thread studio installation package link

https://download-sh-cmcc.rt-thread.org:9151/www/studio/download/RT-Thread Studio-v2.1.2-setup-x86_64_20210831-1200.exe

arm gcc installation package link

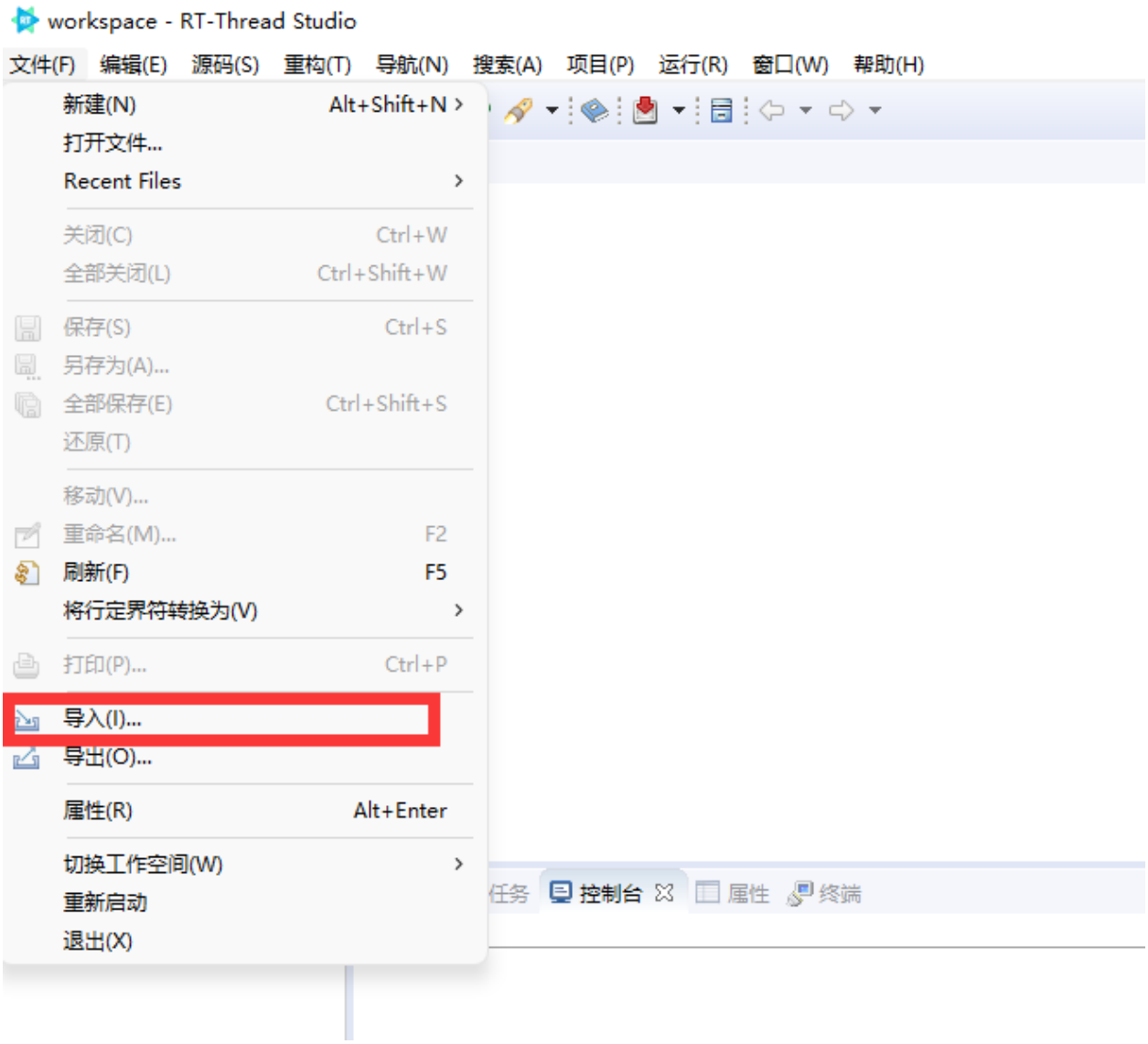
https://developer.arm.com/-/media/Files/downloads/gnu-rm/10.3-2021.10/gcc-arm-none-eabi-10.3-2021.10-win32.exe

You can install rt-thread studio where you like, arm gcc should be installed on the default c drive.

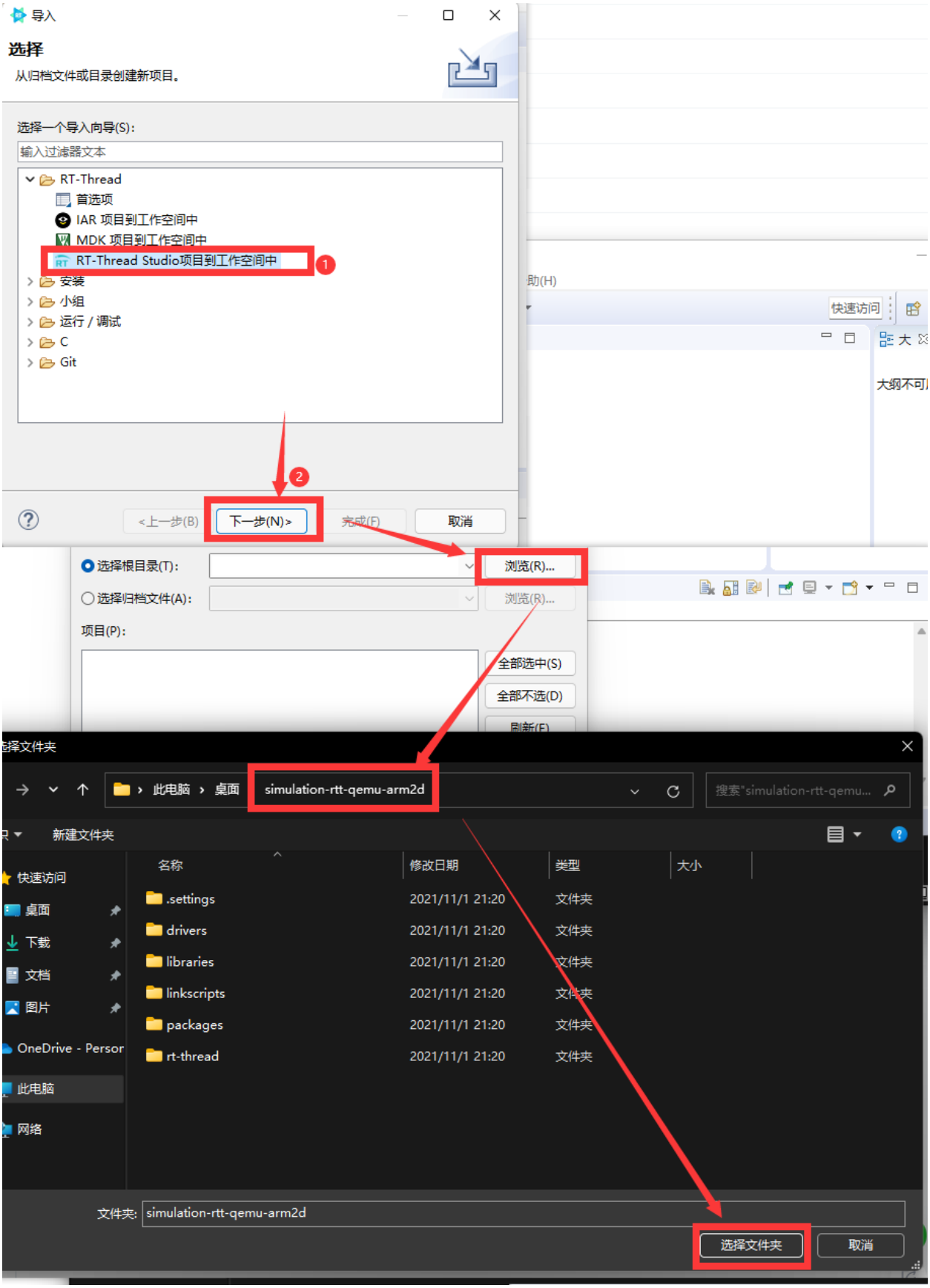
Once installed, you can start playing arm-2d with python.

2.8.4 run

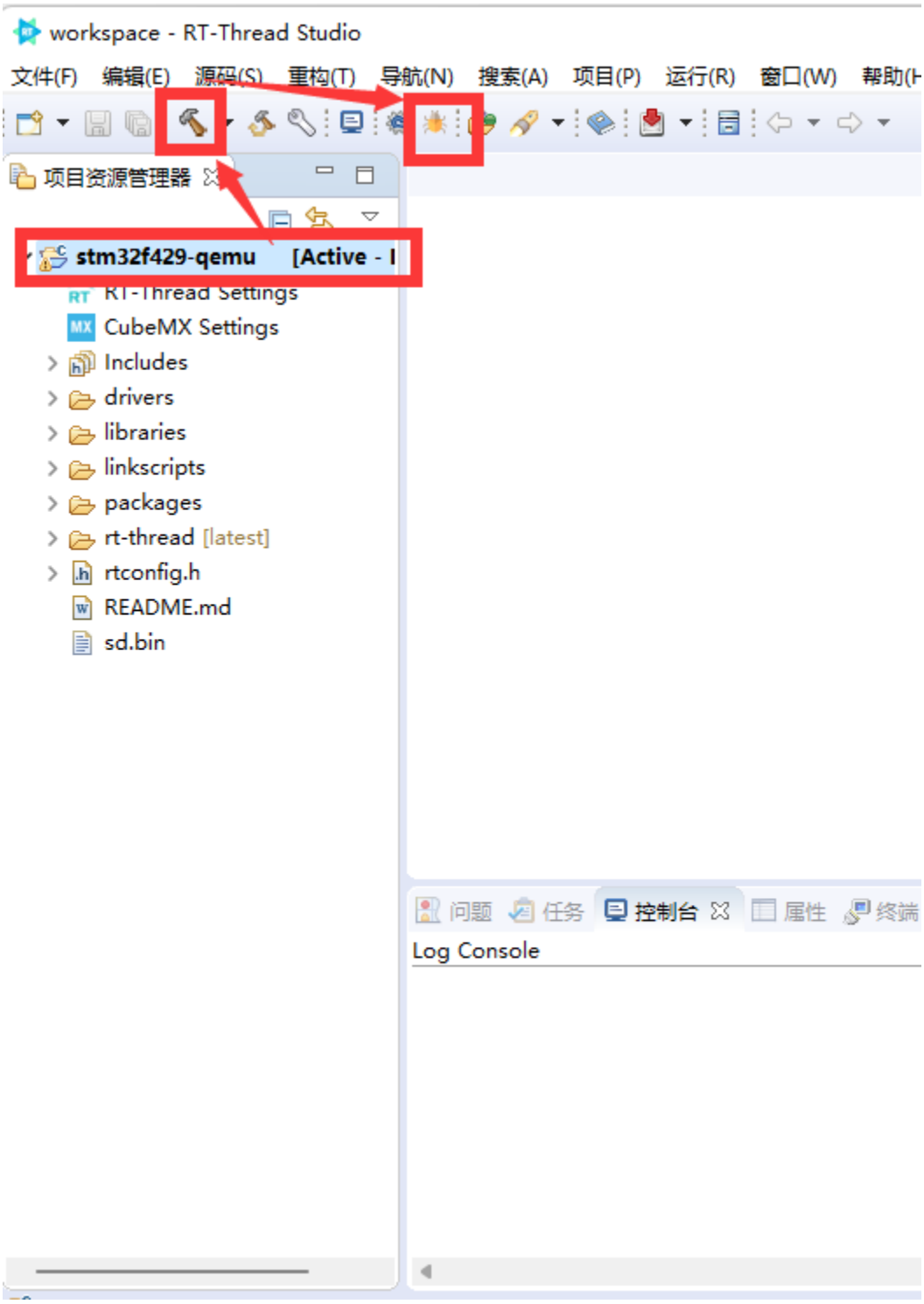
We open RT-Thread Studio and click Import



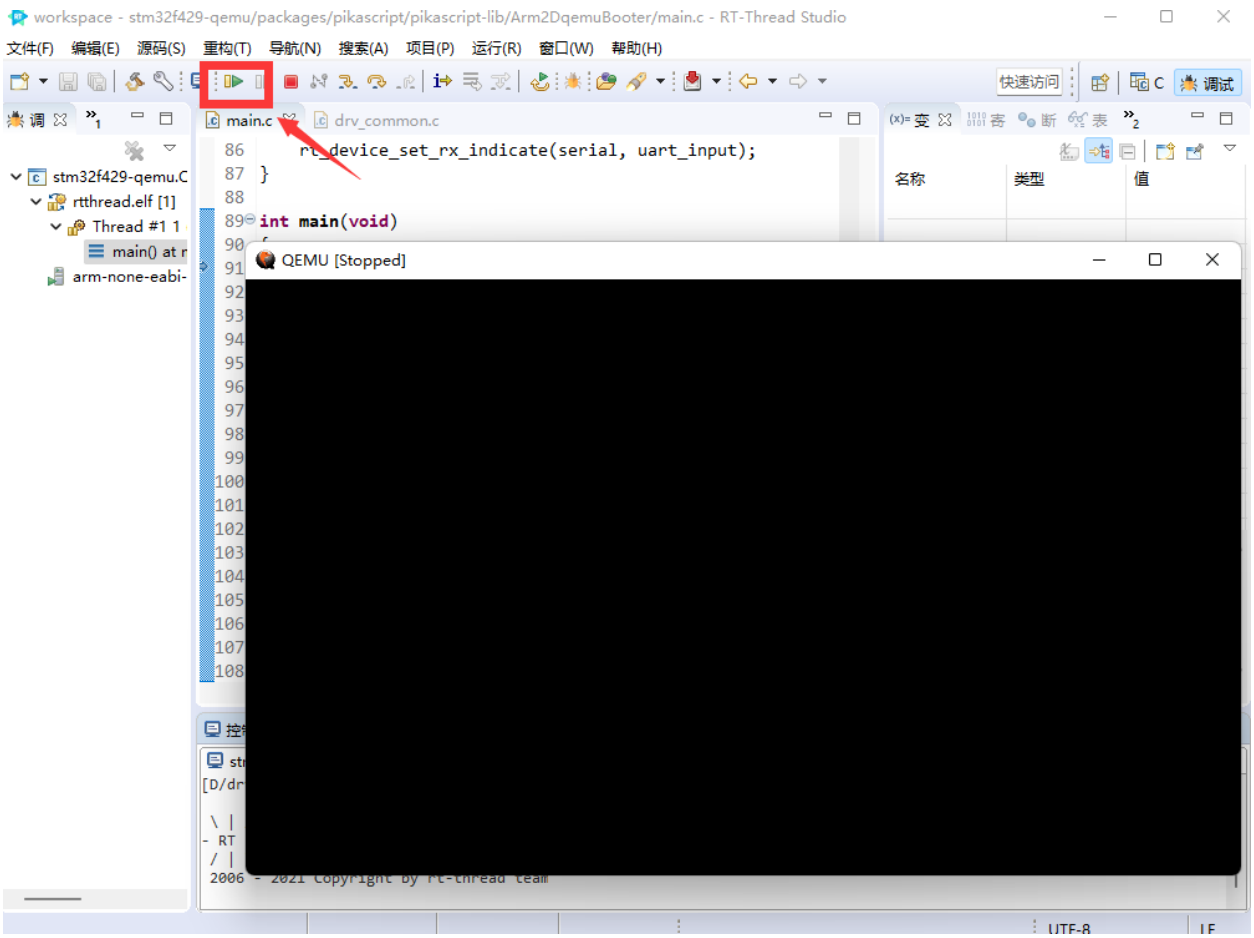
Then select the simulation-rtt-qemu-arm2d folder


导入
选择
从归档文件或目录创建新项目。
选择一个导入向导(S):
输入过滤器文本
RT-Thread
首选项
IAR 项目到工作空间中
MDK 项目到工作空间中
RT-Thread Studio项目到工作空间中
安装
小组
运行 / 调试
C
Git
1
2
<上一步(B)
下一步(N)>
完成(F)
取消
选择根目录(T):
选择归档文件(A):
浏览(R)...
项目(P):
全部选中(S)
全部不选(D)
刷新(F)
选择文件夹
此电脑 › 桌面
simulation-rtt-qemu-arm2d
搜索"simulation-rtt-qemu...
新建文件夹
名称
修改日期
类型
大小
.settings
drivers
libraries
linkscripts
packages
rt-thread
2021/11/1 21:20
文件夹
快速访问
桌面
下载
文档
图片
OneDrive - Persor
此电脑
网络
文件夹: simulation-rtt-qemu-arm2d
选择文件夹
取消

Select the project, then click the hammer to compile, and then click the bug to enter the simulation



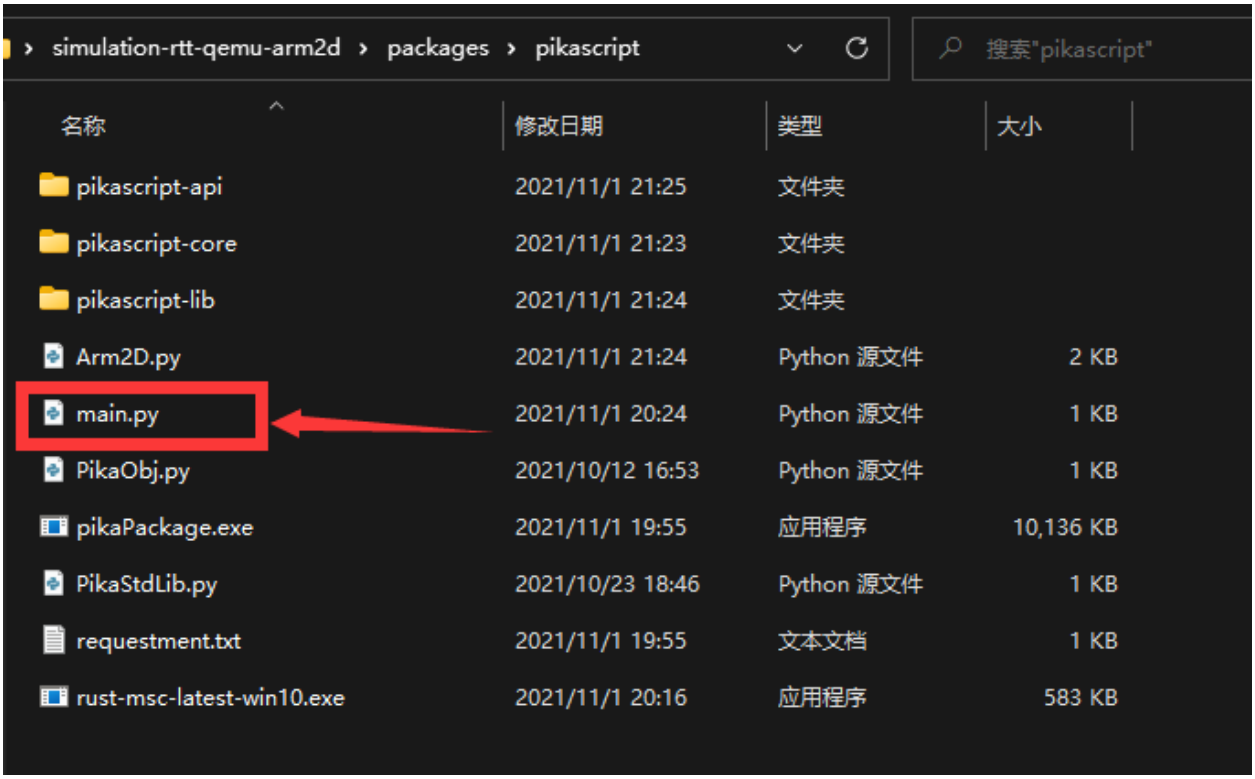
A QEMU box will pop up, then click Run.



If the operation is successful, you can see a small blue square on the white background. So far the deployment has been successful.

2.8.5 Modify the python code and try

The source code of python is in simulation-rtt-qemu-arm2d/packages/pikascript/main.py, you can open it and see~



The following is the content of main.py, create a new box object, and then set the color and position, you can try to change the color to 'white' or change the coordinates to see, you can also create another screen.elems.b2 try .

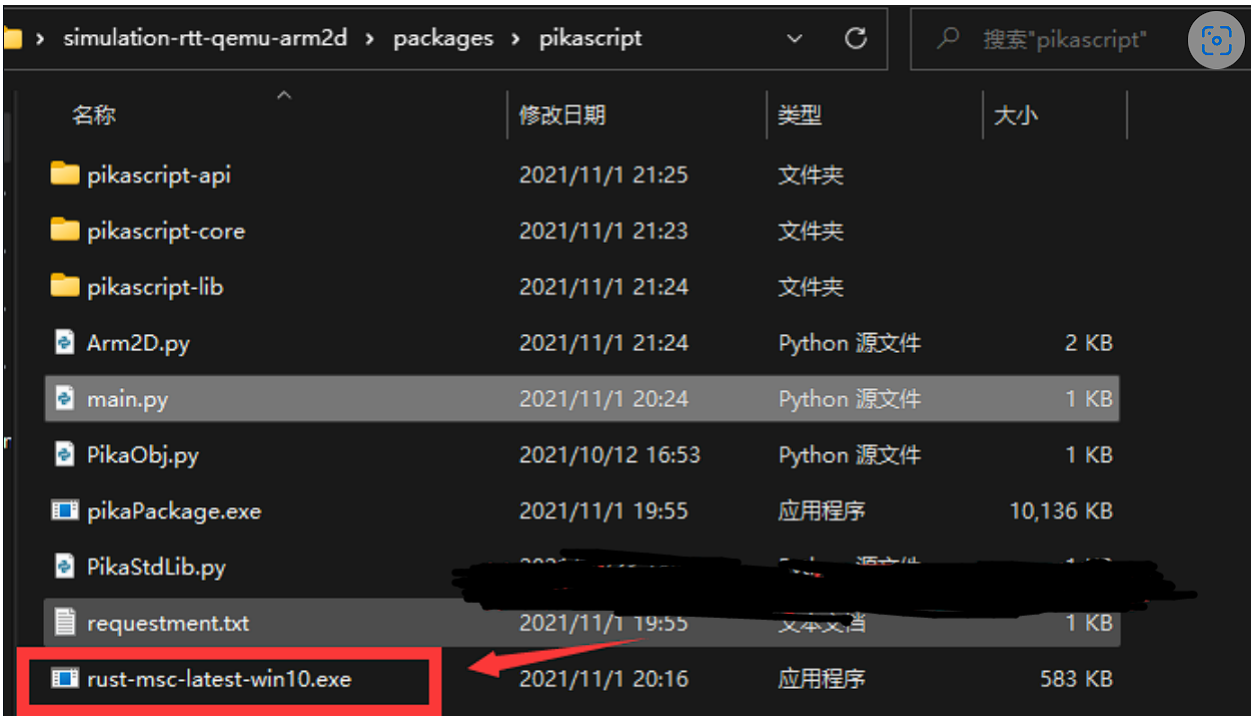
```
import PikaStdLib
import Arm2D

mem = PikaStdLib.MemChecker()

win = Arm2D.Window()
win.init()
win.background.setColor('white')

win.elems.b1 = Arm2D.Box()
win.elems.b1.init()
win.elems.b1.setColor('blue')
win.elems.b1.move(100, 100)
i = 0
x0 = 100
y0 = 100
sizeX0 = 50
sizeY0 = 50
alpha0 = 180
isIncrace = 1
loopTimes = 0

print('hello pikaScript')
print('mem used max:')
mem.max()
```

(continues on next page)

(continued from previous page)

```
print('mem used now:')
mem.now()
while True:
    win.elems.b1.move(x0 + i * 2, y0 + i * 1)
    win.elems.b1.setAlpha(alpha0 - i * 1)
    win.elems.b1.setSize(sizeX0 + i * 2, sizeY0 + i * 1)
    win.update()
    if isIncrace > 0:
        i = i + 1
        if i > 160:
            isIncrace = 0
    if isIncrace < 1:
        i = i - 1
        if i < 0:
            isIncrace = 1
            loopTimes = loopTimes + 1
```

Remember to precompile after each modification to convert python to .c file in the project

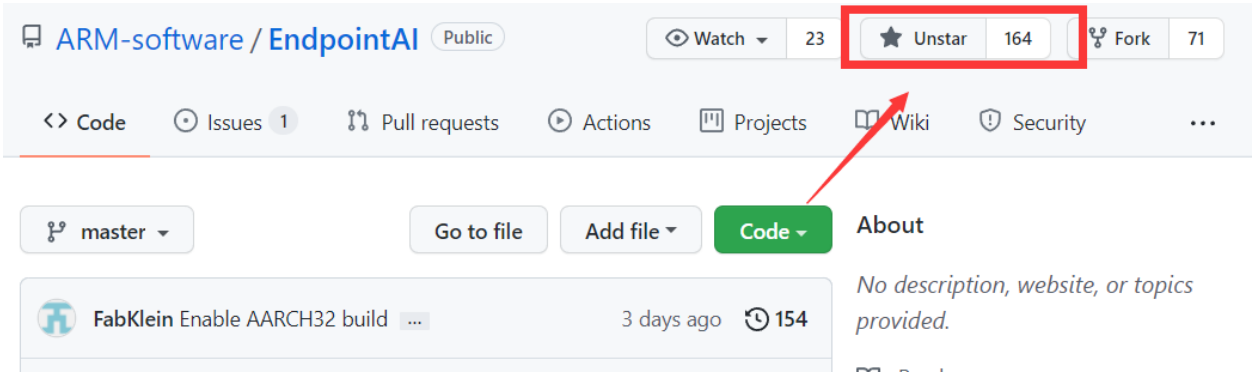


Then compile again, enter the simulation, and you can see the effect. This time I changed the square to red.

2.8.6 Conclusion

This is the Arm-2D warehouse~ Students who haven't starred remember to add a star~

https://github.com/ARM-software/Arm-2D



Thanks to liuduanfei for the rtt-Arm2d-qemu simulation project~ Here is the github homepage of liuduanfei

https://github.com/liuduanfei

CHAPTER

THREE

DEVELOPMENT BOARD

3.1 Pika Pie Development Board Quick Start

Today, we will not talk about the hard-core content of driver development and architectural principles. We will simply use the Pika Pie development board to play Python programming! Light up a "Life is too short, I use Python" achievement on the microcontroller!

Video link

3.1.1 Development board acquisition

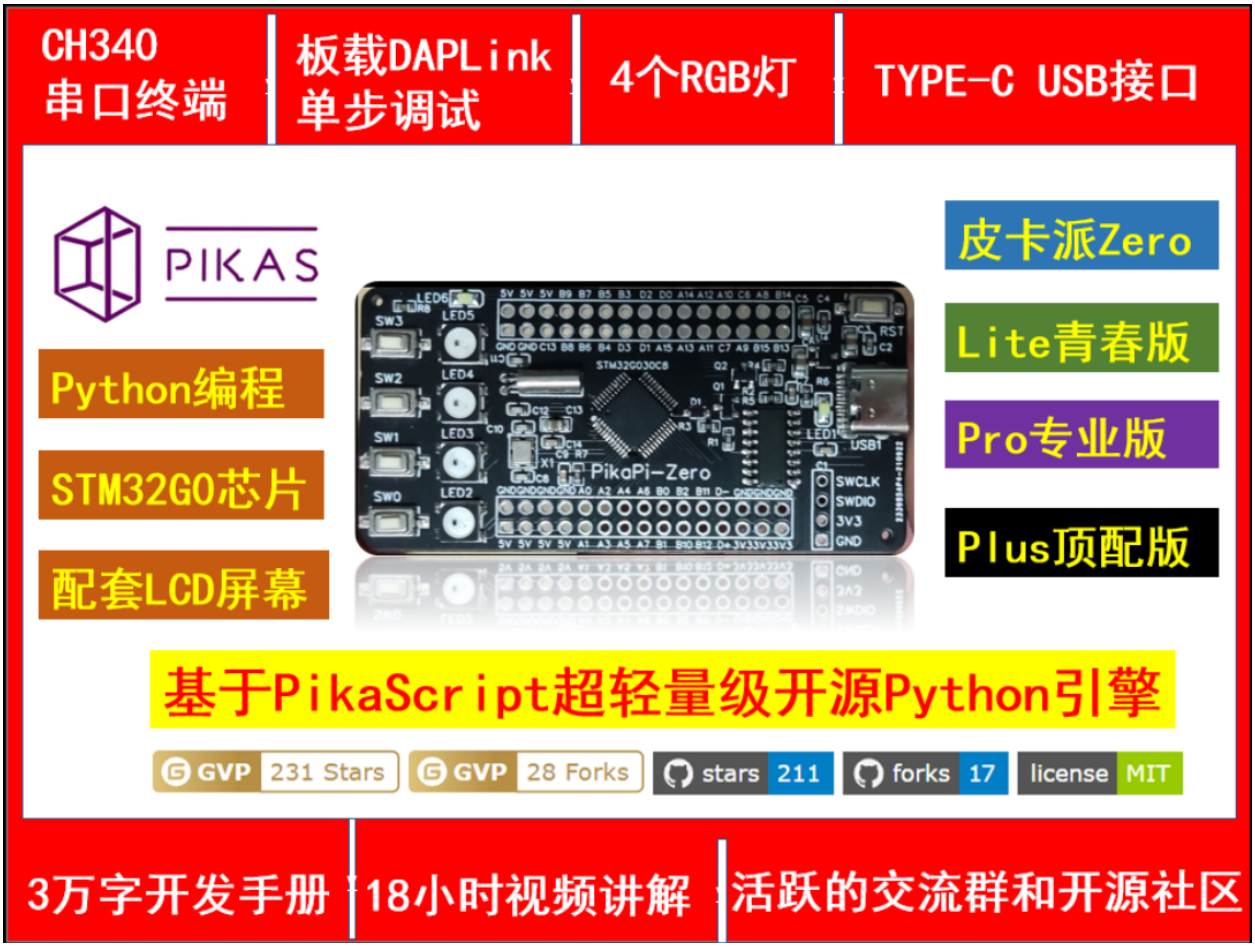
If you don't have a Pika Pie development board yet, you can buy it from the link below:

https://item.taobao.com/item.htm?spm=a21dvs.23580594.0.0.52de3d0dt7rqAx&ft=t&id=654947372034

The development board looks like this. It has an STM32G0 chip onboard with 4 colorful RGBs and a Type-C interface.

Optional:

- Lite Youth Edition: STM32G030 + CH340 serial port chip 64k flash 8k ram
- Pro version: STM32G030 + DAPLink debugger 64K flash 8k ram
- Plus top version: STM32G070 + DAPLink debugger 128k flash 32k ram



This development board is officially supported by the PikaPython project and continues to be updated continuously. The latest kernel and latest functions of PikaPython can be experienced on this development board.

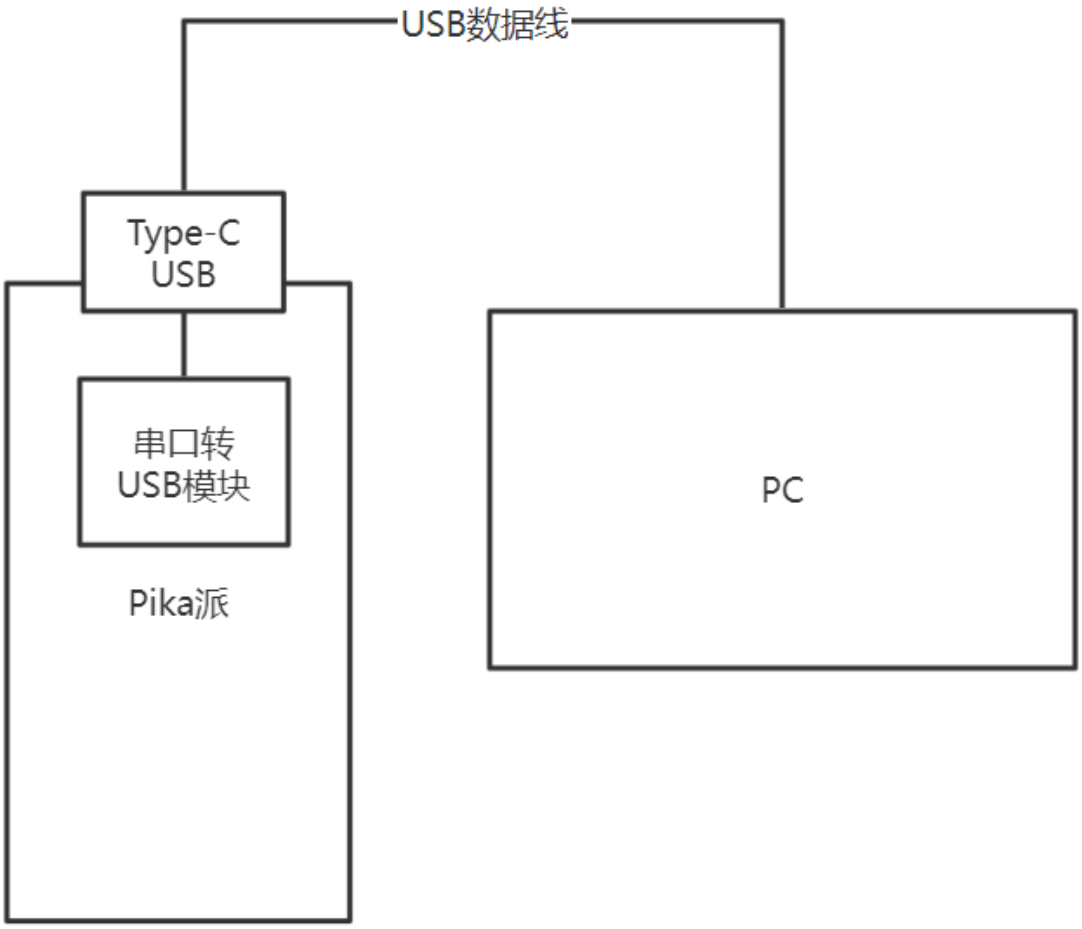
This development board has also been officially adapted by the project with a wealth of peripheral modules, including GPIO, TIME, ADC, IIC, LCD, KEY, PWM and other modules drivers have been developed and can be programmed directly with python.

3.1.2 Video tutorials

https://space.bilibili.com/5365336/channel/seriesdetail?sid=1034902

3.1.3 How to download the Python program for the microcontroller

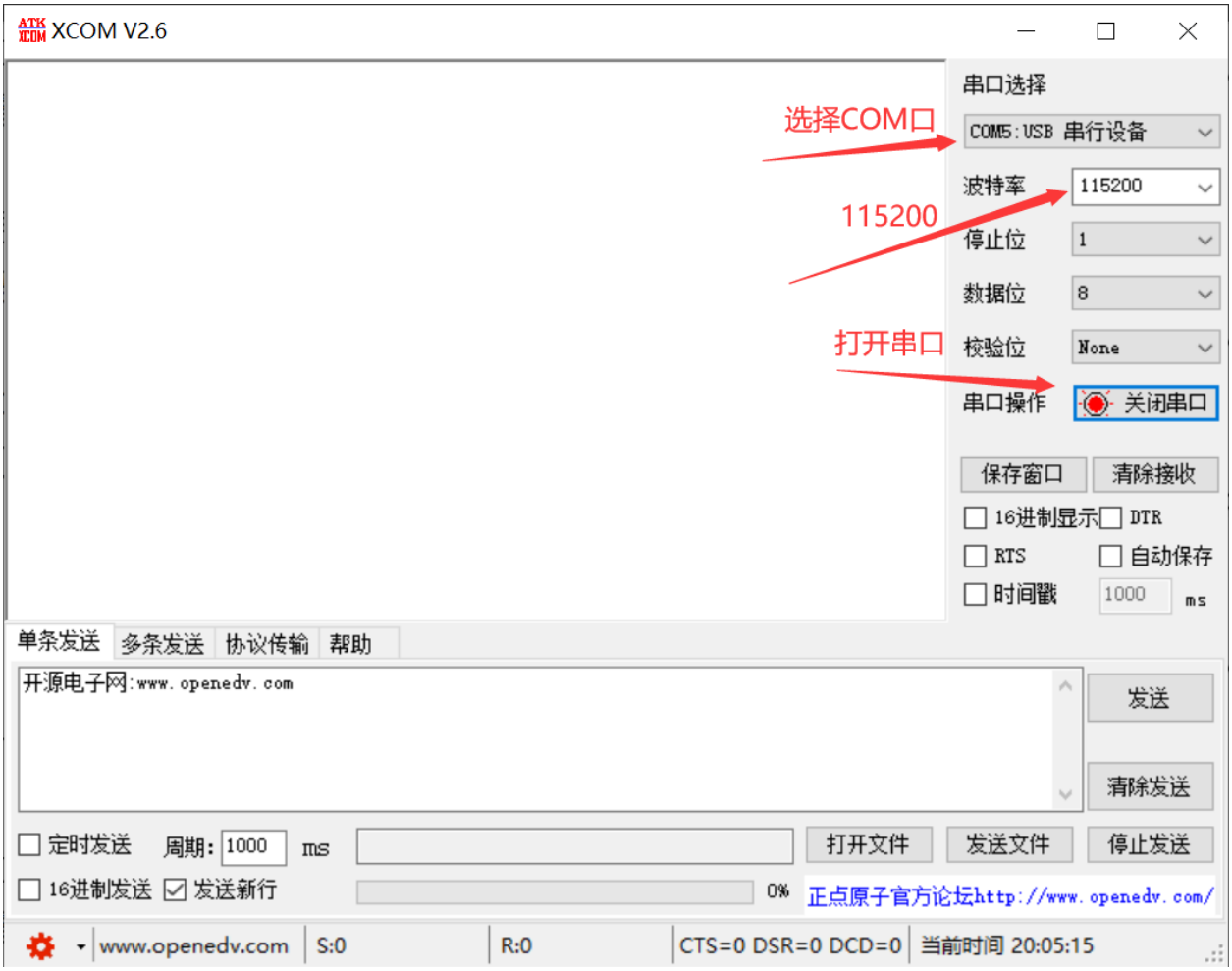
The download method is very simple, just connect the Type-C data cable.



We use a USB data cable to connect the computer and the Pika Pie development board, and we can download the program.

When downloading the program, you need to use a serial port assistant tool. We can use the XCOM assistant developed by Punctual Atom, which can be downloaded from the Punctual Atom forum.

http://www.openedv.com/thread-279749-1-1.html



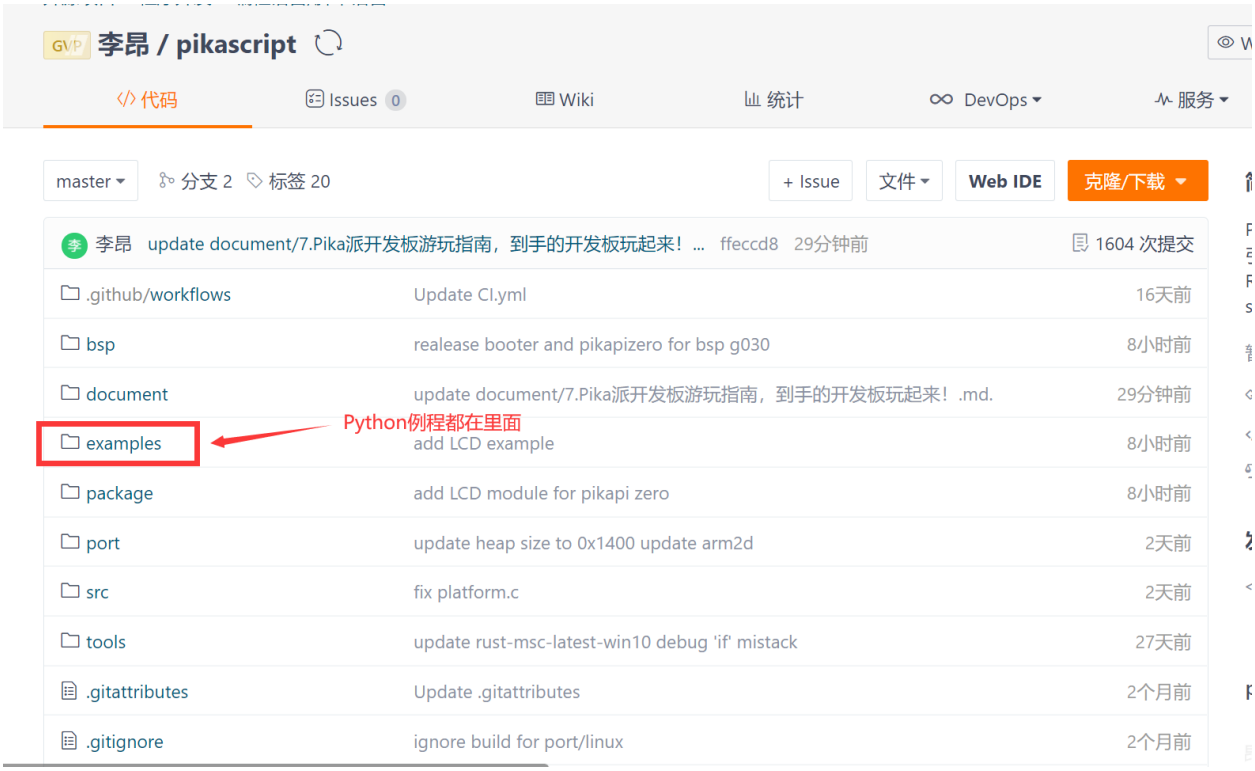
Select the COM port, then select the baud rate as 115200, and then click to open the serial port. At this time, it is connected to the Pika Pie. Simply send a Pthon script file to download the Python program to Pika Pie. To verify that the download was successful, we use the sample Python scripts in the PikaPython source repository.

We enter the code repository of PikaPython

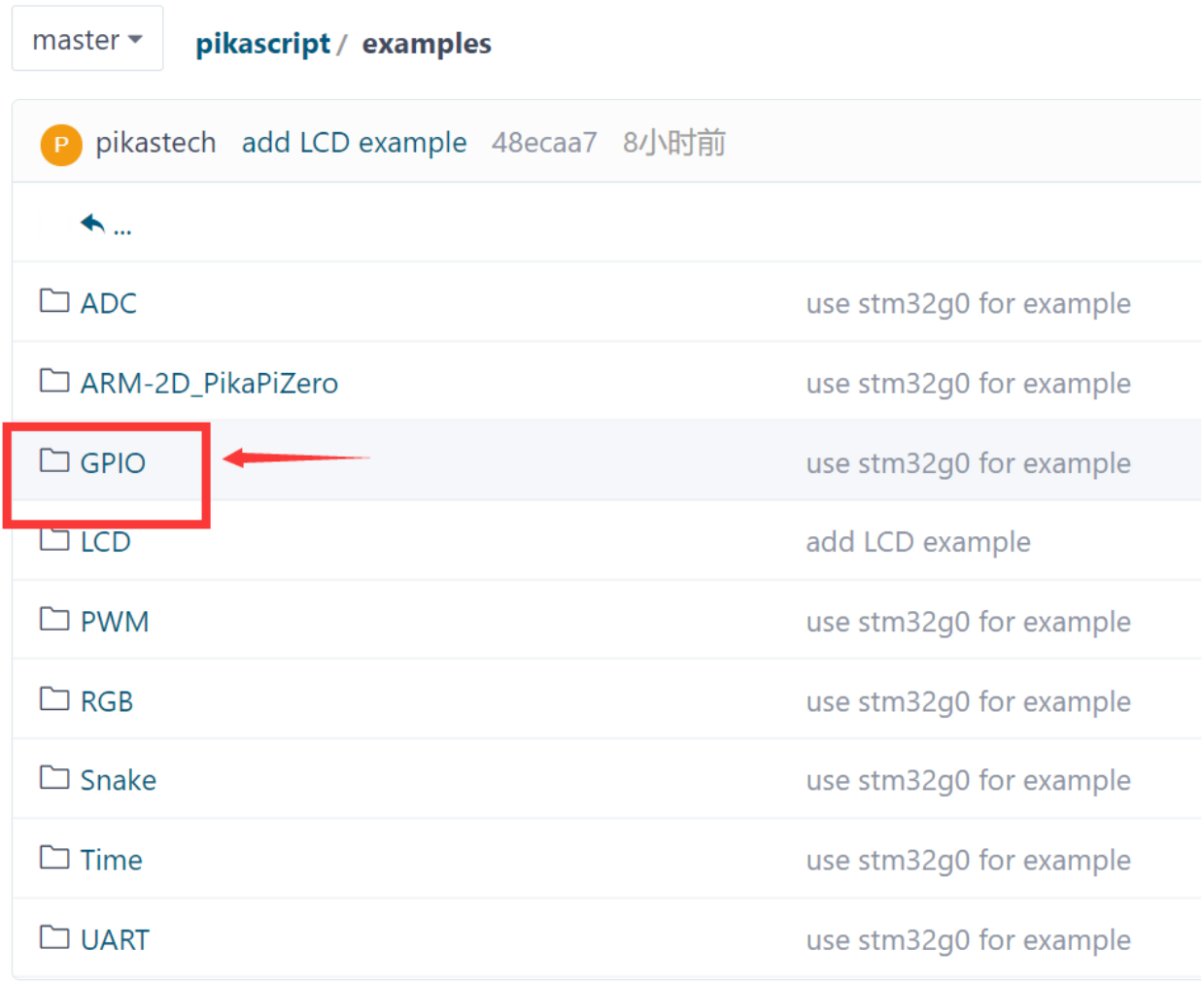
https://github.com/pikastech/pikascript

It is customary to click a Star~

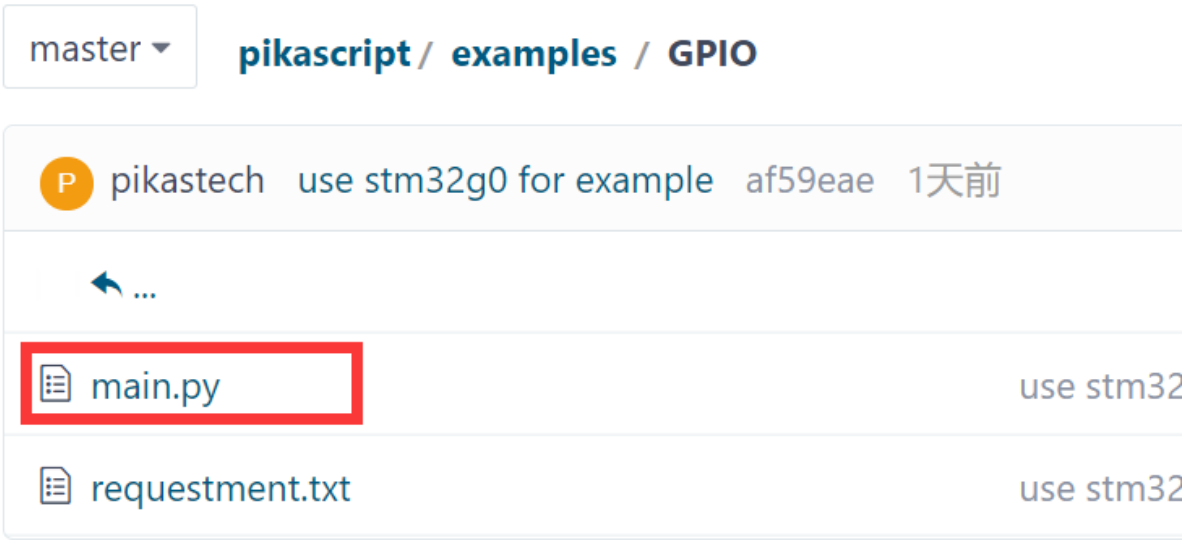
Then we click on the examples folder, which contains the Python routines that can be run.



Let's open the GPIO folder and light up the water lamp to see~



The main.py in the GPIO folder is the sample code for GPIO



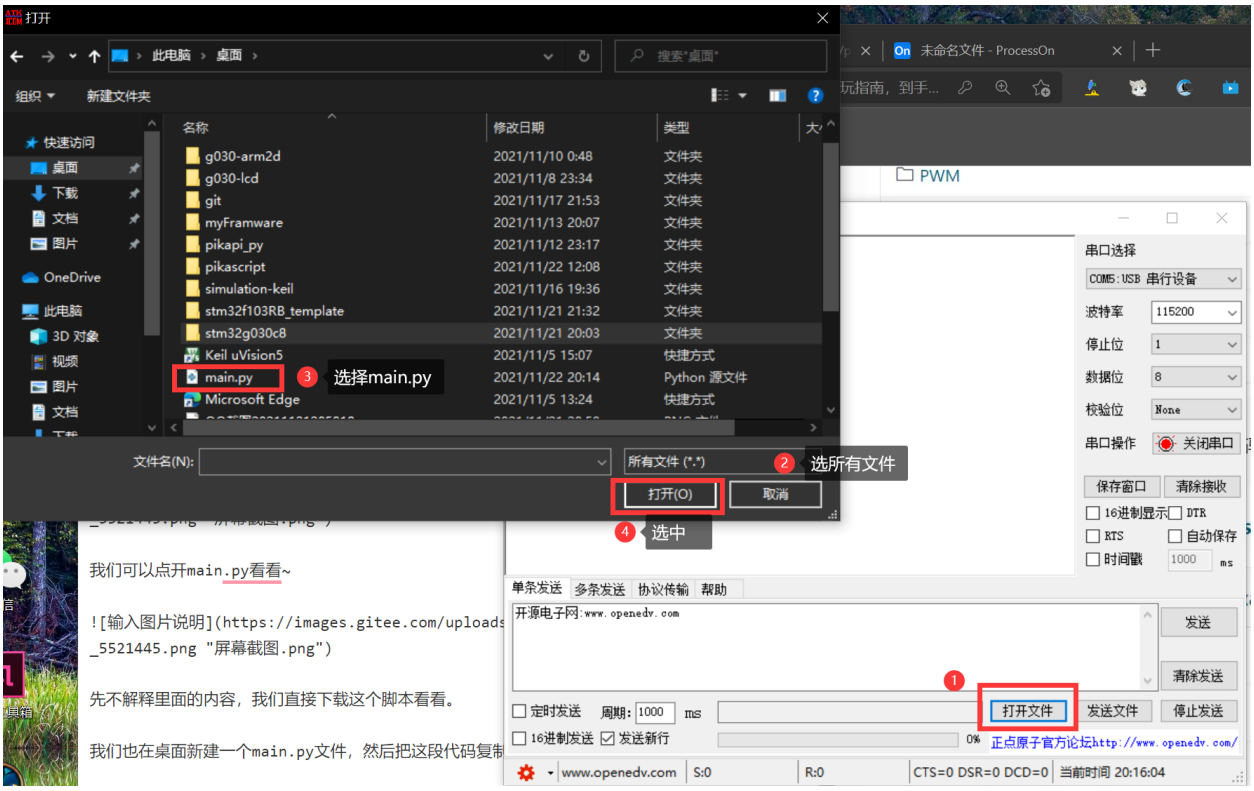
We can open main.py and see~

```
import PikaStdLib
import machine

mem = PikaStdLib.MemChecker()
io1 = machine.GPIO()
time = machine.Time()

io1.init()
io1.setPin('PA8')
io1.setMode('out')
io1.enable()
io1.low()

print('hello pikascript')
print('mem.max:')
mem.max()
print('mem.now:')
mem.now()

while True:
    io1.low()
    time.sleep_ms(500)
    io1.high()
    time.sleep_ms(500)
```

Without explaining the content inside, let's download this script directly.

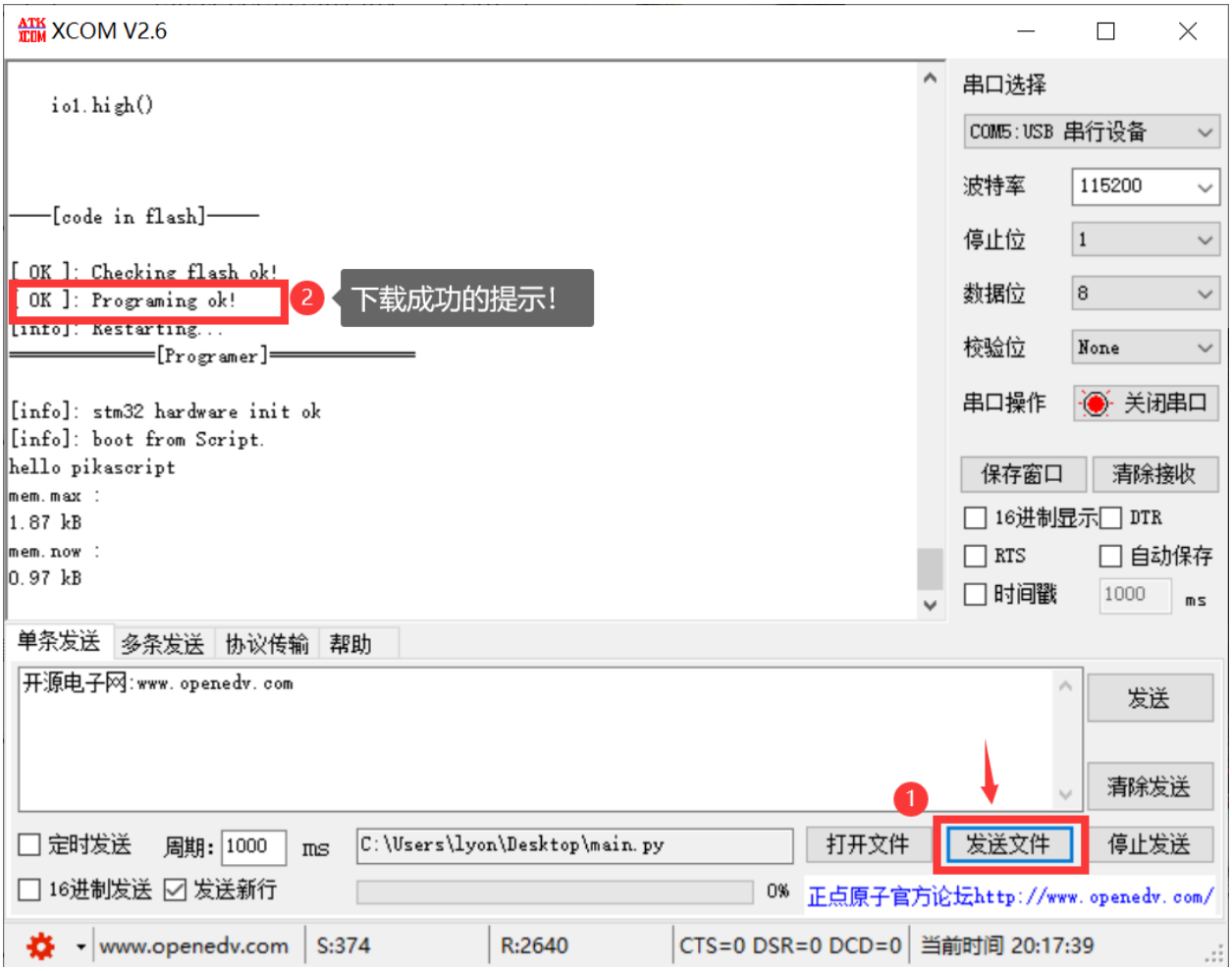
We also create a new main.py file on the desktop, and then copy this code into it.

We choose this main.py file

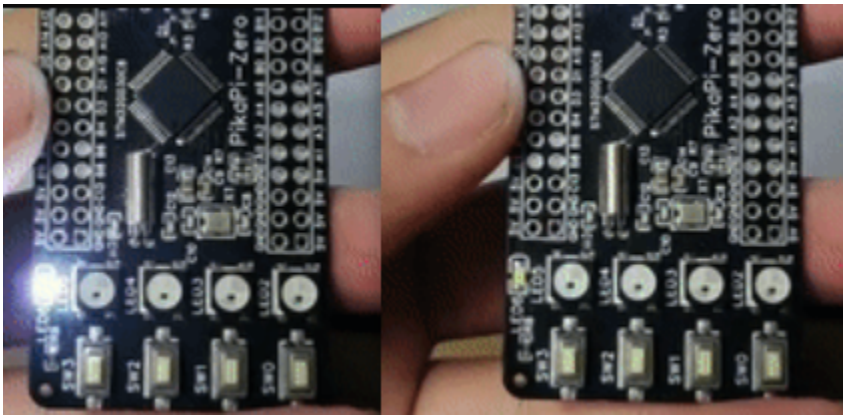


Then click “Send File” to download the script!

We can see the [OK]: Programing ok! prompt, which means the download is successful!



At this time, the LED on the development board will flash!



Congratulations on your achievement of playing Python with a microcontroller!

3.1.4 What is written in the GPIO script?

Let's parse this GPIO routine line by line.

```
import PikaStdLib
import machine
```

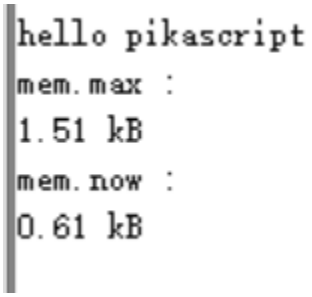
The first line is the first and second line, which means that two modules are imported, one is the `PikaStdLib` module and one is the `machine` module. PikaStdLib is the standard library of PikaPython, which has some system functions, such as checking the memory usage. In the fourth line, we create a new mem object whose class is PikaStdLib.MemChecker().

```
mem = PikaStdLib.MemChecker()
```

This class has a max() method and a now() method. Using these two methods, you can print out the memory size currently used by PikaPython.

```
print('hello pikascript')
print('mem.max:')
mem.max()
print('mem.now:')
mem.now()
```

We can look at the printout of the serial port, we can see that the maximum memory usage is 1.51kB, and the current memory usage is 0.61kB, is it very small!



screenshot.png

The time object is newly created through the Time() class of machine and can provide basic delay functions.

```
time = machine.Time()
```

Through the time.sleep_ms() method, you can delay in milliseconds. For example, the function of the following code is to delay 500ms.

```
time.sleep_ms(500)
```

io1 is our protagonist today. This is a GPIO object, which is newly created with the machine .GPIO() class.

```
io1 = machine.GPIO()
```

After creating a new io1 object, we need to initialize this io, init() is used for object initialization, used at the front, and then setPin('PA8') means using the PA8 port setMode('out') means using the output mode, And enable() means to start the hardware of io1, and low() pulls down the level of io1. A led light on the Pika Pie is connected to the PA8. As long as you control the level of the PA8, you can control the light on and off.

```
io1.init()
io1.setPin('PA8')
io1.setMode('out')
io1.enable()
io1.low()
```

In the main loop of the program, switch the high and low levels of io1 to make the LED flash~

```
while True:
    io1.low()
    time.sleep_ms(500)
    io1.high()
    time.sleep_ms(500)
```

3.1.5 Interpretation of other Python routines

ADC

Let's interpret other routines in examples, such as this ADC routine, which is to read the analog voltage value on the PA1 pin, and then print it out~

```
import PikaStdLib
import machine

time = machine.Time()
adc1 = machine.ADC() #Create a new ADC object

adc1.init() #Initialize ADC object
adc1.setPin('PA1') #Set the pin
adc1.enable() #Start the hardware

while True:
    val = adc1.read() #Read the value of ADC once and store it in the val variable
    print('adc1 value:') #Print what is read
    print(val)
    time.sleep_ms(500) #Wait for 0.5s
```

UART

The following is the routine of the serial port, the function is to read the received two bytes, and then print them out

```
import PikaStdLib
import machine

time = machine.Time()
uart = machine.UART() #Create a new serial port object
uart.init()
uart.setId(1) #Set the serial port number, use serial port 1
uart.setBaudRate(115200) #Set the baud rate
```

(continues on next page)

(continued from previous page)

```
uart.enable() #Start hardware

while True:
    time.sleep_ms(500)
    readBuff = uart.read(2) #read two characters
    print('read 2 char:')
    print(readBuff) # print out
```

PWM

The following is the PWM routine, you can specify the pin to output the PWM wave, you can set the frequency and duty cycle

```
import PikaStdLib
import machine

time = machine.Time()
pwm = machine.PWM()
pwm.setPin('PA8') #Set PWM output pin
pwm.setFrequency(2000) #Set the frequency
pwm.setDuty(0.5) #Set the duty cycle to 50%
pwm.enable()

while True:
    time.sleep_ms(500)
    pwm.setDuty(0.5)
    time.sleep_ms(500)
    pwm.setDuty(0.001) #Set the duty cycle to 0.1%
```

RGB

Then the following is the RGB routine~

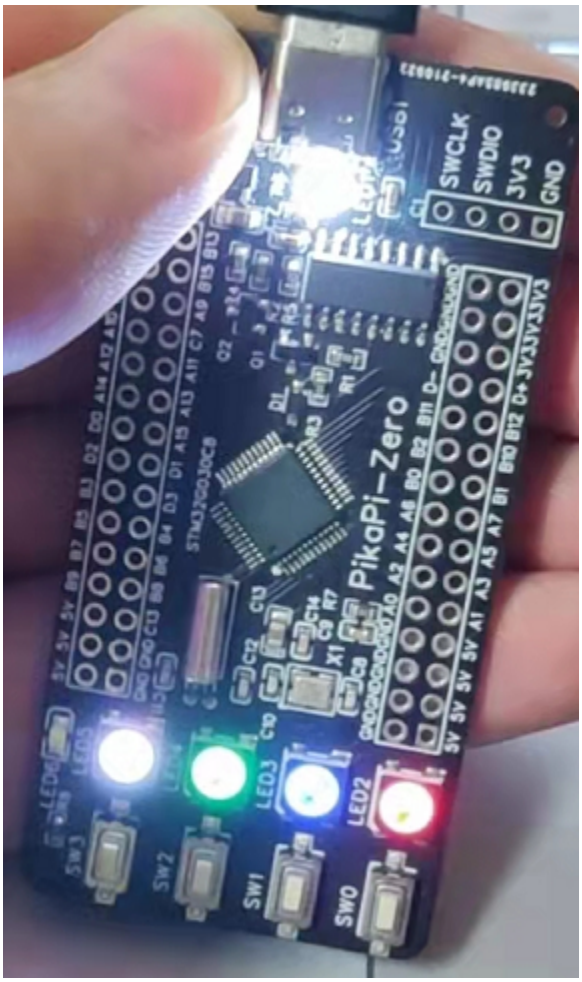
```
import machine

import PikaStdLib

time = machine.Time()
adc = machine.ADC()
pin = machine.GPIO()
pwm = machine.PWM()
uart = machine.UART()
rgb = machine.RGB() #Create a new RGB object
mem = PikaStdLib.MemChecker()

rgb.init() #Initialize the object
rgb.enable() #Start hardware

print('hello 2')
print('mem used max:')
```

(continues on next page)

(continued from previous page)

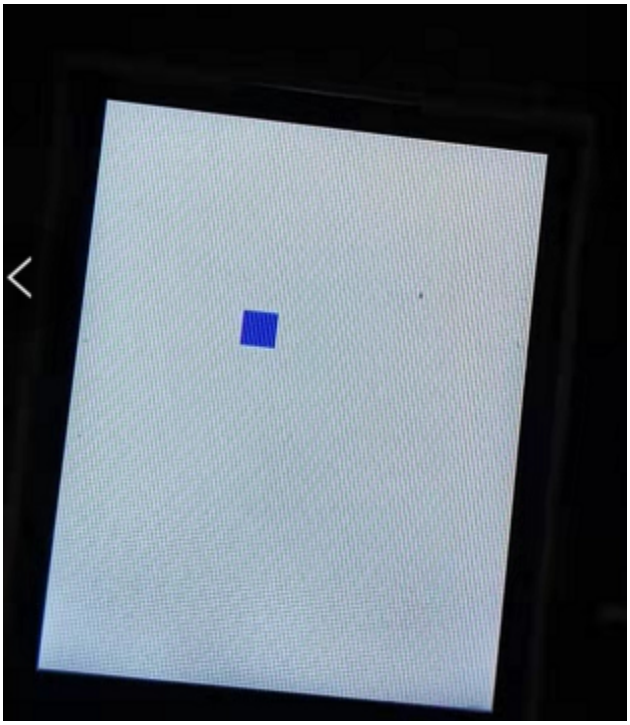
```
mem.max()

while True:
    print('flowing')
    rgb.flow() #RGB water light flow
```

This routine can drive the onboard 4 RGB water lights~



LCD

There is also an LCD routine that can display a small square on the LCD, and you can use the four onboard buttons to control the movement of the small square~



```
from PikaObj import *
import PikaStdLib

import machine

lcd = machine.LCD()
lcd.init()
lcd.clear('white') #Initialize LCD background fill with white
mem = PikaStdLib.MemChecker()
key = machine.KEY() #Create a new key object and get the onboard key input
key.init()
time = machine.Time()
h = 10
w = 10
x = 10
y = 10 # used to represent the height, width and coordinates of the small square
x_last = x
y_last = y #Record the last position for erasing
is_update = 0 #Control the flag variable that refreshes the screen
print('mem used max:')
mem.max()
lcd.fill(x, y, w, h, 'blue') # draw small blue squares
while True:
    key_val = key.get() # get the value of the key
    if key_val != -1:
```

(continues on next page)

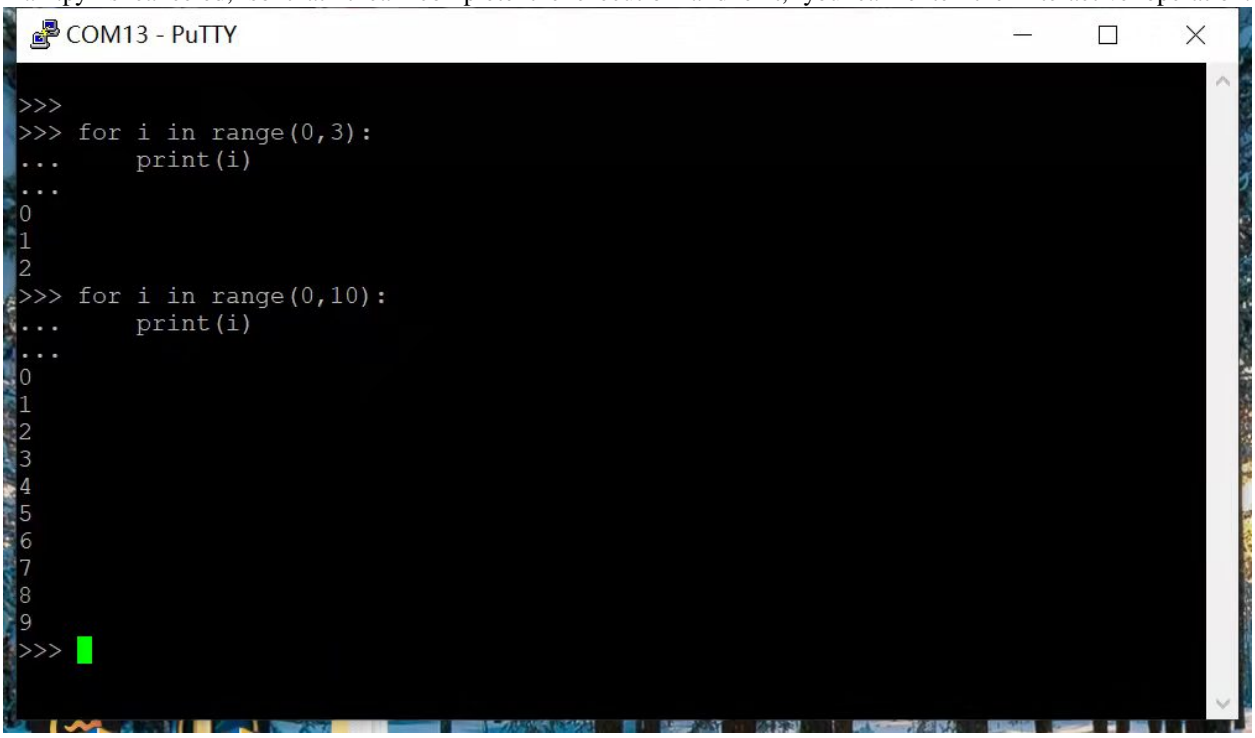
(continued from previous page)

```
        x_last = x
        y_last = y
        is_update = 1 #Start refresh
    if key_val == 0:
        x = x + 5 #change the coordinates of the small square
    if key_val == 1:
        y = y - 5
    if key_val == 2:
        y = y + 5
    if key_val == 3:
        x = x - 5
    if is_update: #Refresh the screen
        is_update = 0
        lcd.fill(x_last, y_last, w, h, 'white') #Erase the previous position
        lcd.fill(x, y, w, h, 'blue') # draw a new position
```

When you are familiar with the LCD driver, you can try to develop your own mini-games~

3.1.6 run interactively

After main.py is executed, it will enter the interactive operation, so as long as the `while True :` in main.py is canceled, so that it can complete the execution and exit, you can enter the interactive operation.



Interactive execution supports single-line and multi-line input, consistent with general Python usage. It is recommended to use PuTTY serial terminal. Entering `exit()` will directly restart the system. **Precautions**:

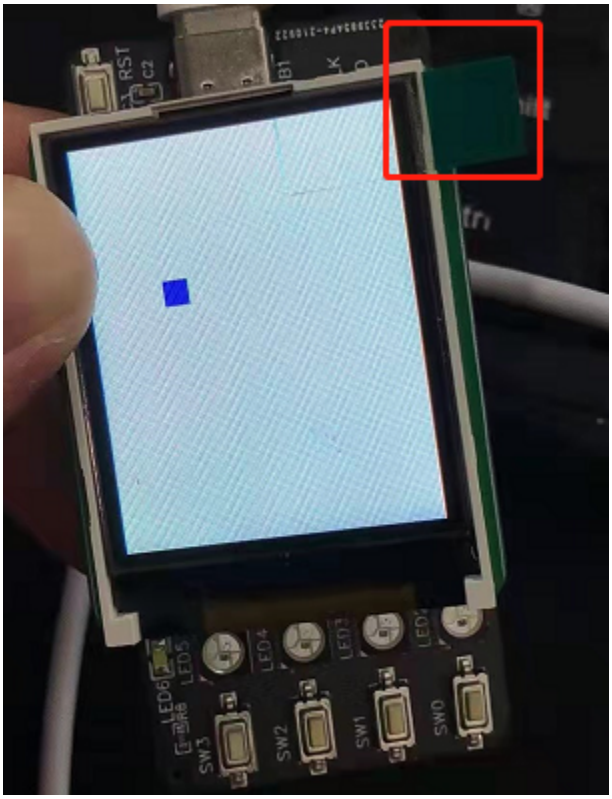
1. The firmware version needs to be no less than **v1.3.2. **
2. If using the PuTTY terminal does not work, use XCOM.
3. All English input methods should be used in the terminal.

4. Indent should use 4 spaces, do not use the TAB key.

3.1.7 LCD screen installation

1. Refer to the figure below to solder the long pin headers

1. Plug in the screen, refer to the direction of the green flag, if the screen can be lit, it means that the direction of the plug is correct, if it is plugged in reversely, it will not light up.



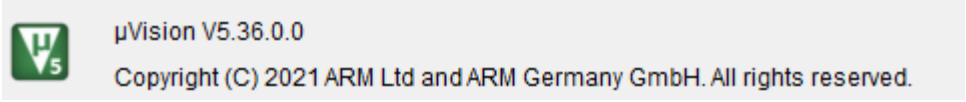
3.1.8 Firmware upgrade

The firmware of Pika Pie is updated on a rolling basis, and new firmware versions will be released continuously to provide new functions, and some new functions can only be played by upgrading the firmware, so it is also very important to learn to upgrade the firmware~

Compile the firmware yourself

The firmware is a Keil project and compilation is very simple. Download the firmware project: Enter pikascript official website http://pikascript.com The Lite and Pro versions use the stm32g030 platform. The Plus version uses the stm32g070 platform. Then click "Start Generation". (The default module will be automatically selected after selecting the platform)

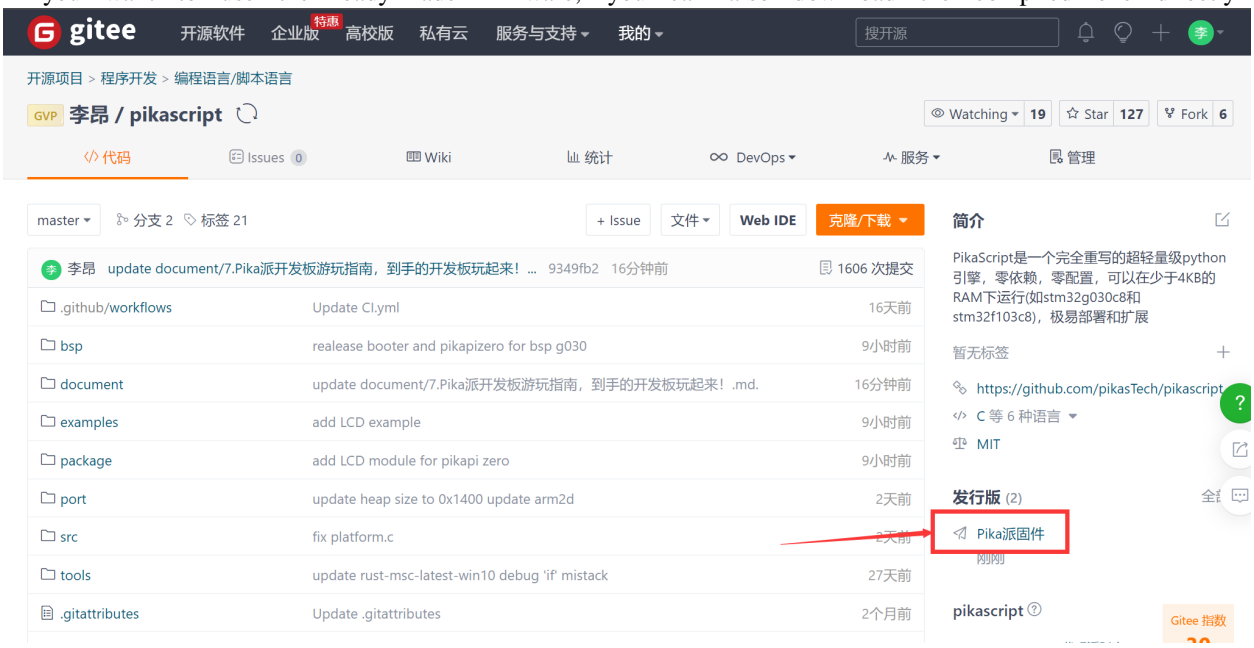
Just open the Keil project and compile it. When compiling, you need to use Keil not lower than V5.36, which needs

to be activated. 

The compiled .bin is in MDK/stm32g030c8/stm32g030c8.bin .

Download the compiled firmware directly

If you want to use the ready-made firmware, you can also download the compiled one directly~



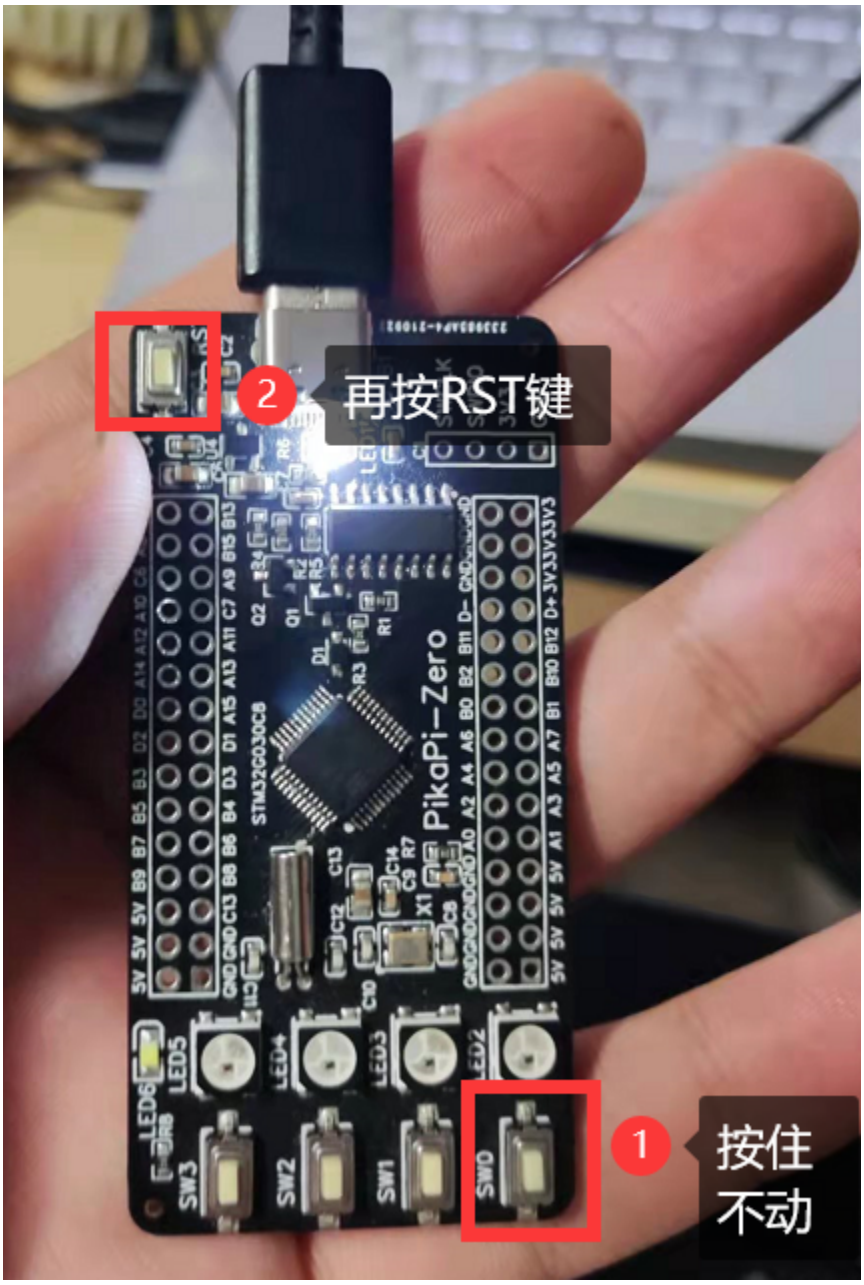
Click to download to get the latest firmware~

Serial Bootloader upgrade

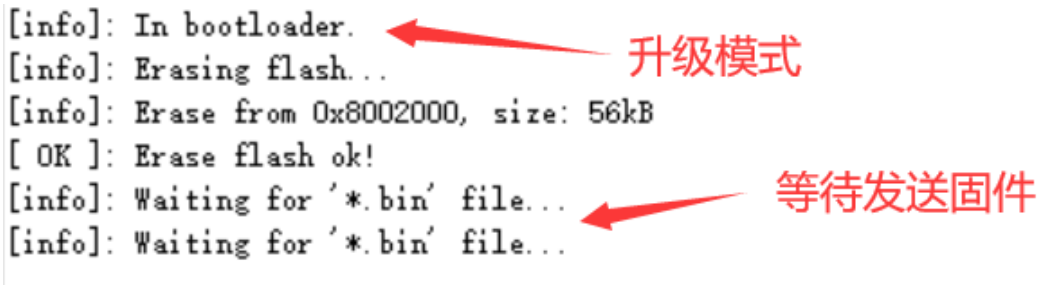
To upgrade the firmware, you can also use the serial port. When upgrading, you need to use the firmware compiled by yourself or the .bin firmware you downloaded directly. Currently, the versions that support serial bootloader upgrade are:

- Lite Youth Edition
- Pro Professional Edition

Next, we need to let the pika pie enter the upgrade mode. We press and hold the SW0 key on the development board and press the RST key at the same time to enter the upgrade mode.



In the upgrade mode, we can see the prompt information of the serial port



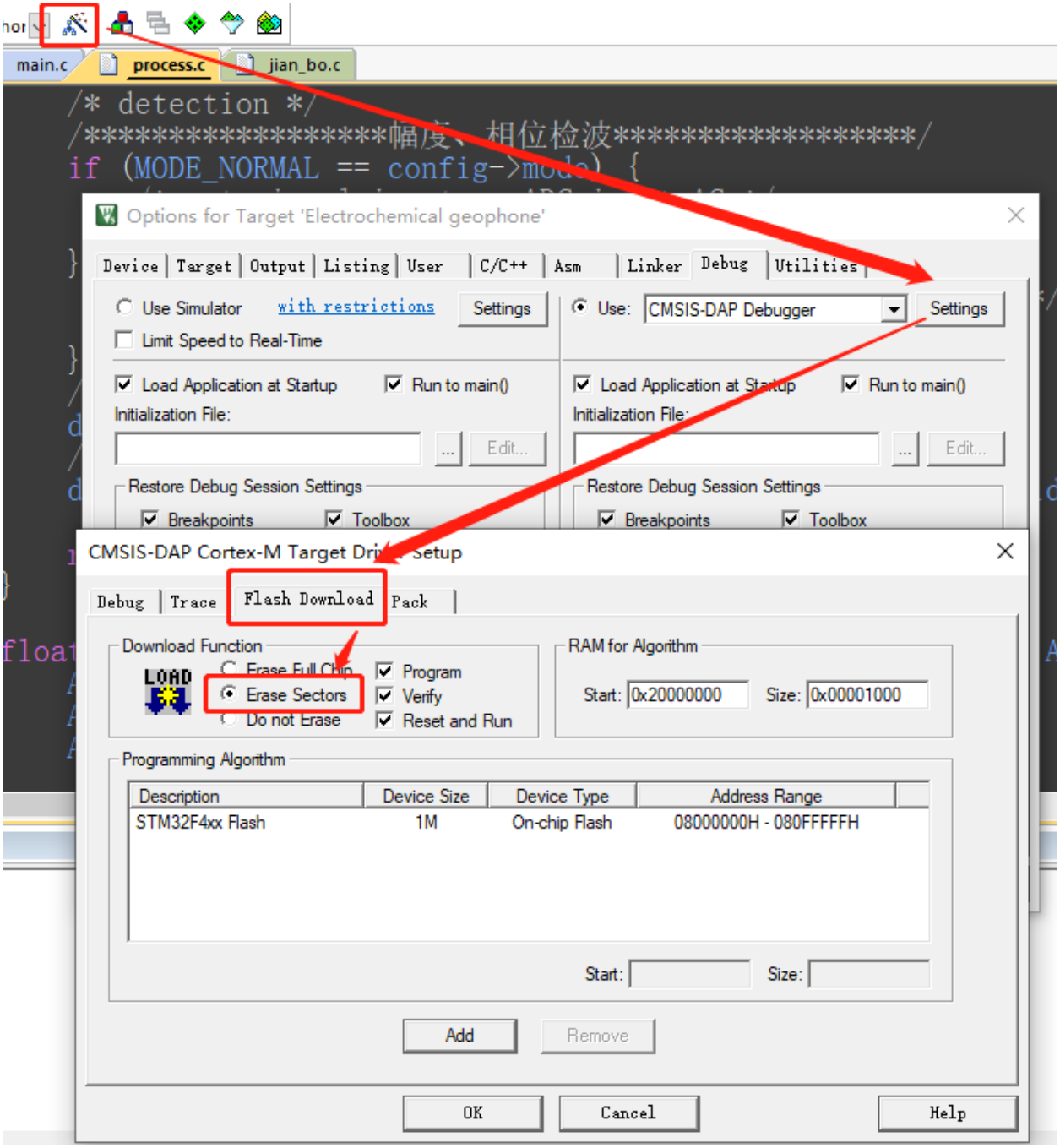
Then we use the serial port assistant to select the stm32g030c8.bin file just downloaded and send it through the serial port. After the firmware is recognized, Reciving....

```
[info]: Waiting for '*.bin' file...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
[info]: Reciving...
```

After sending, press the RST key to restart, and the upgrade is complete! If it can be started normally, then the upgrade is successful.

Upgrade using SWD

The Lite version can connect to J-Link \ DAP-Link \ ST-Link to upgrade SWD. The Pro version and Plus version have onboard DAP-Link, which can be upgraded by SWD directly by connecting to USB. The Lite and Pro versions use the bsp/stm32g030 project. The Plus version uses the bsp/stm32g070 project. When using SWD to upgrade, the download method of "Partial Erase" should be selected

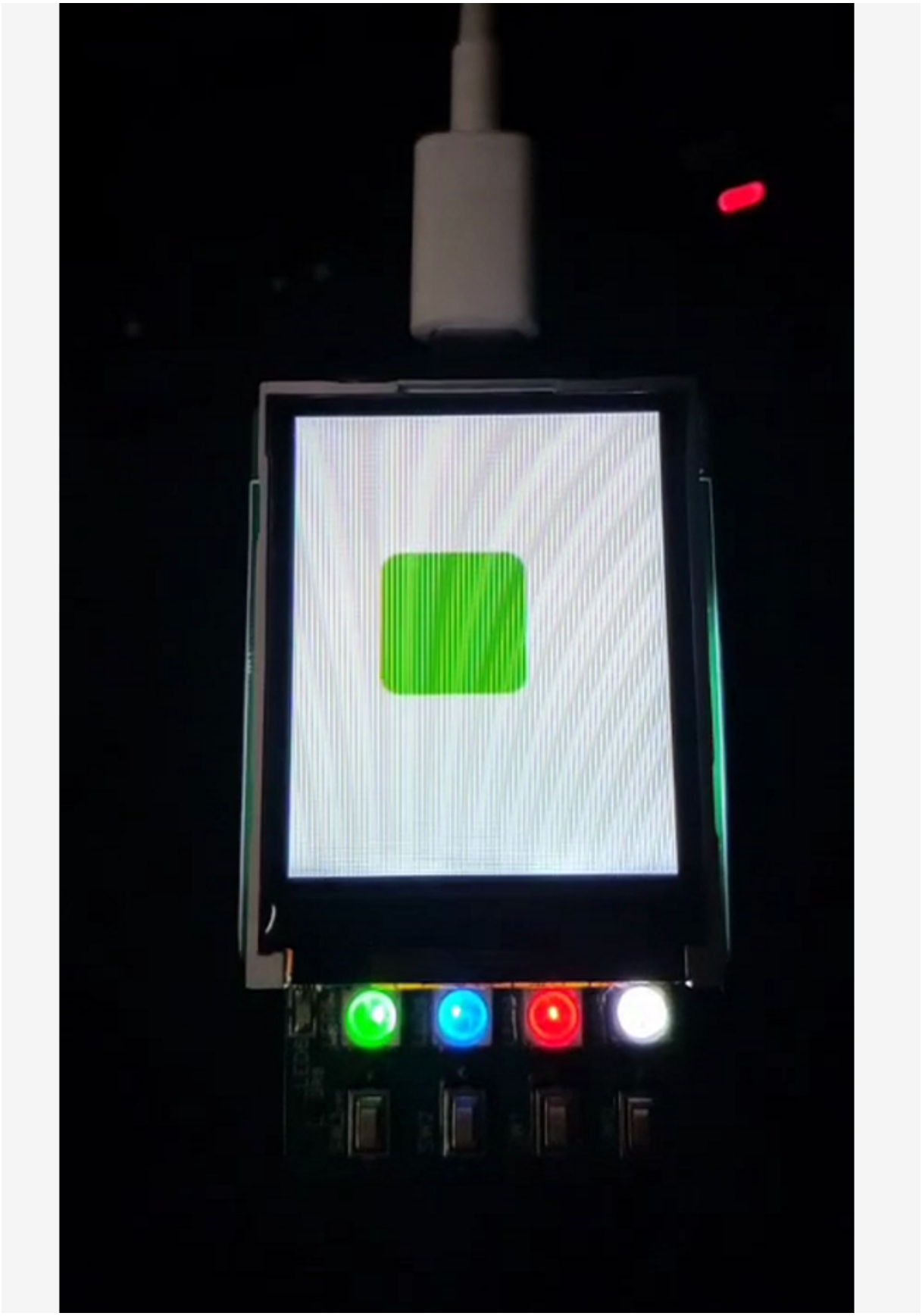

main.c
process.c
jian_bo.c
/* detection */
Options for Target 'Electrochemical geophone'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Use Simulator
with restrictions
Settings
Limit Speed to Real-Time
Use:
CMSIS-DAP Debugger
Settings
Load Application at Startup
Run to main()
Initialization File:
Edit...
Restore Debug Session Settings
Breakpoints
Toolbox
CMSIS-DAP Cortex-M Target Driver Setup
Debug
Trace
Flash Download
Pack
Download Function
LOAD
Erase Full Chip
Erase Sectors
Do not Erase
Program
Verify
Reset and Run
RAM for Algorithm
Start: 0x20000000
Size: 0x00001000
Programming Algorithm
Description
Device Size
Device Type
Address Range
STM32F4xx Flash
1M
On-chip Flash
08000000H - 080FFFFFH
Start:
Size:
Add
Remove
OK
Cancel
Help

Download Python program using firmware

The firmware loads pikascript/main.py as the default Python program when compiled. Before downloading the firmware, after pressing SW0 + RST to erase the flash, it will boot from the firmware Python program.

3.1.9 ARM-2D GUI engine

pika pie supports running ARM-2D GUI engine



Instructions:

1. Obtain the bsp/stm32g030 project.
2. Use the project files in examples/ARM-2D/PikaPiZero, replace main.py and requestment.txt.

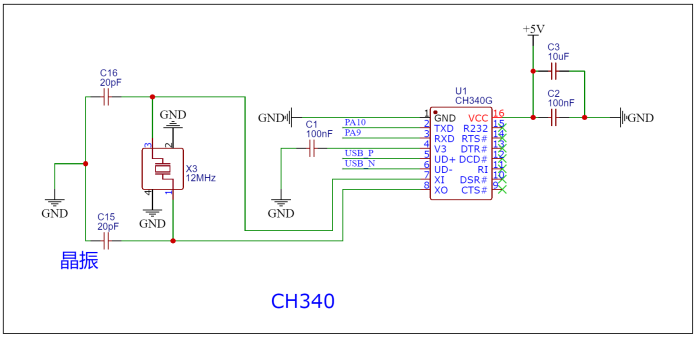
1. Press and hold the SW0 key on the development board and press the RST key at the same time to erase the flash.
2. Re-run the package manager, precompile, compile the project, and flash the project using SWD/Bootloader.

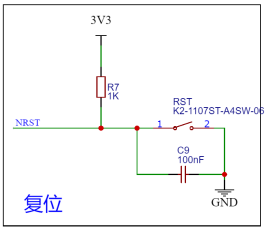
3.1.10 common problem

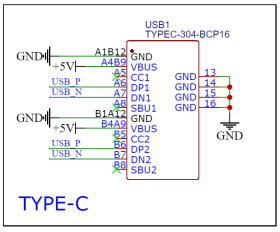
1 Press sw0 + rst to enter the upgrade mode: The first batch of boards shipped did not have the bootloader. You need to manually flash it once. Use jlink / stlink / DAPlink, etc., and flash into pikascript/bsp/pikapizero/bootloader. 2 Cannot enter the bootloader / Suspected to be stuck and unable to run: Check the serial port assistant, you cannot use dtr / rts control, it is recommended to use the xcom assistant of punctual atom. 3 Download python script stuck: When downloading the python program for the first time, do not download the LCD program, first download a gpio program, and then download the LCD program. In other cases, the download is stuck, and you can restart the download again. If it still doesn't work, re-flash the firmware and download it again. 4 Project compilation error, missing files: The project needs to pull modules and precompile remotely. You need to run pikascript/pikaPackage.exe and pikascript/rust-msc-win10-latest.exe before compiling the project.

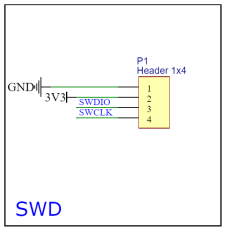
3.1.11 Schematic

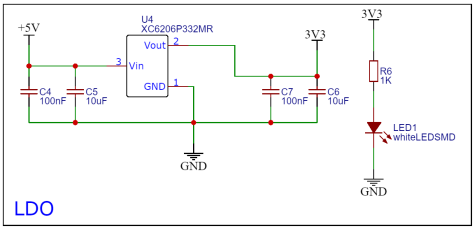
Lite Youth Edition

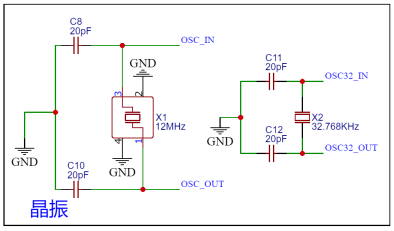


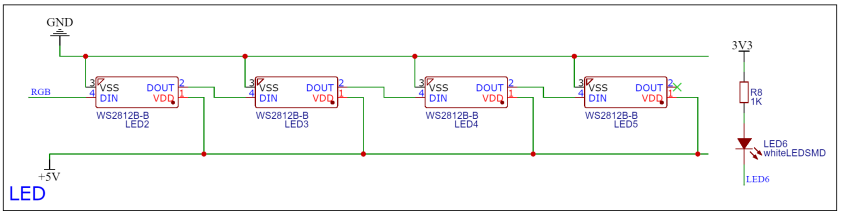


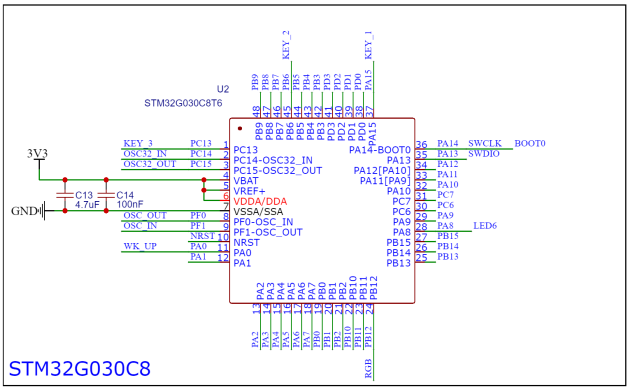


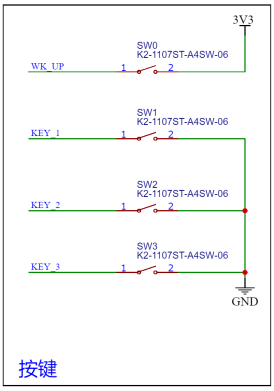


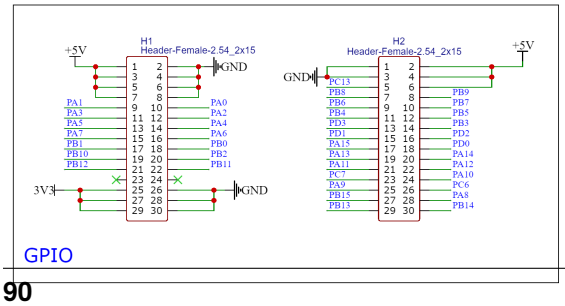




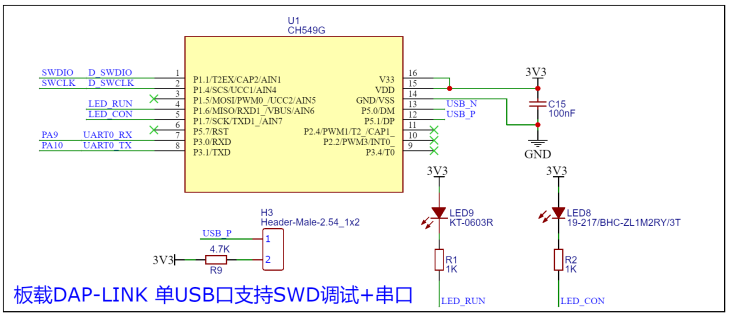


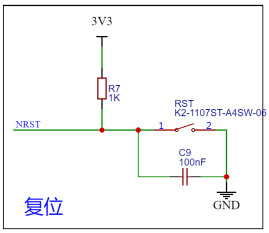


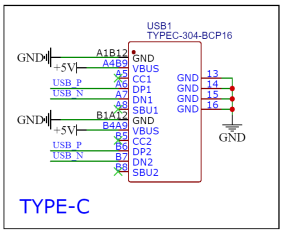


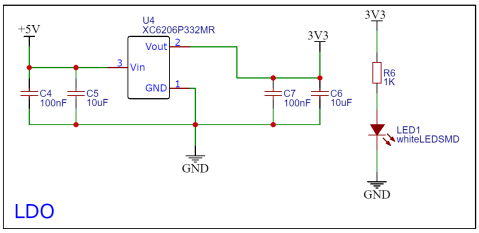


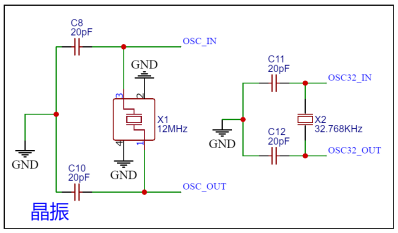
Pro Professional Edition

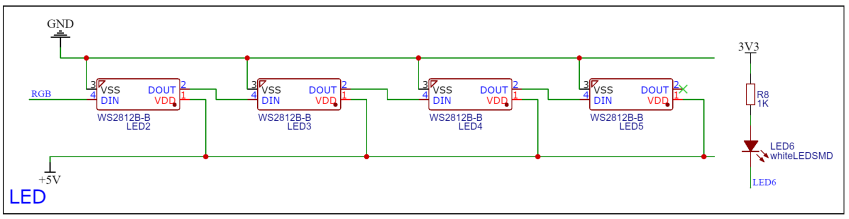


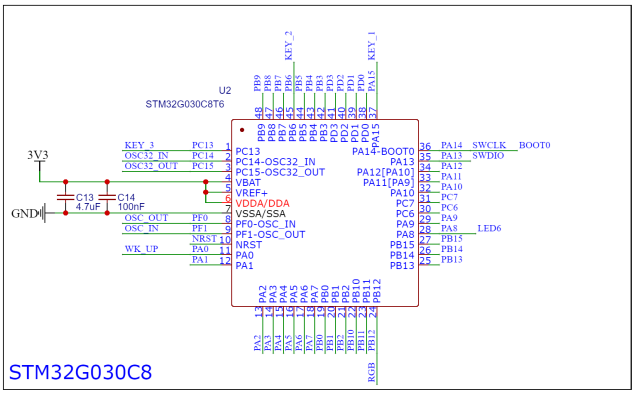


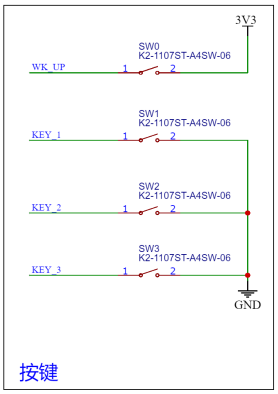


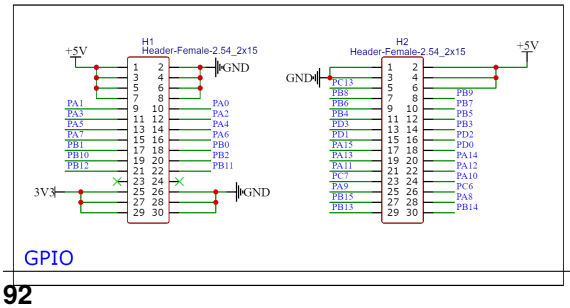




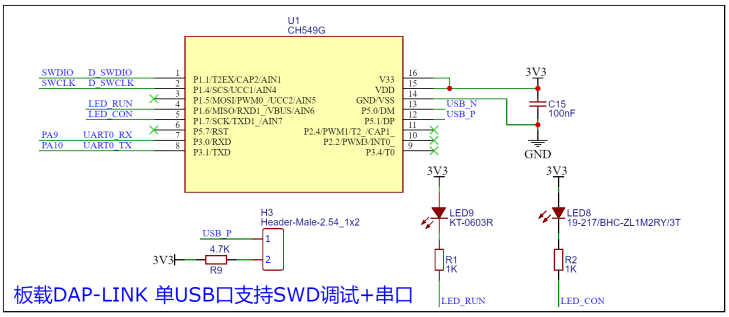


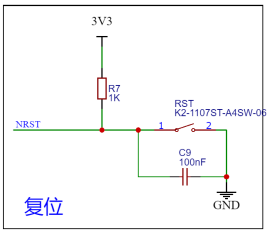


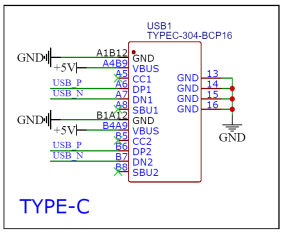


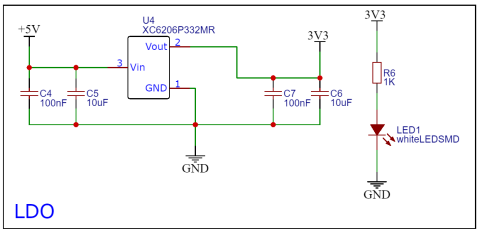


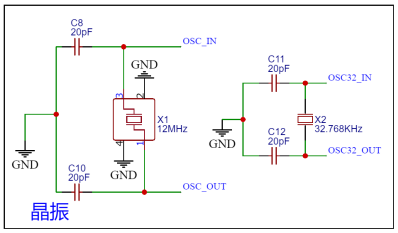
Plus top version

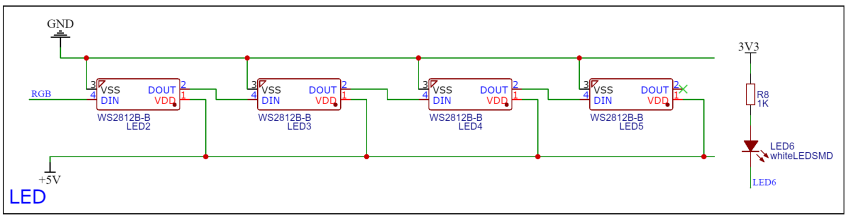


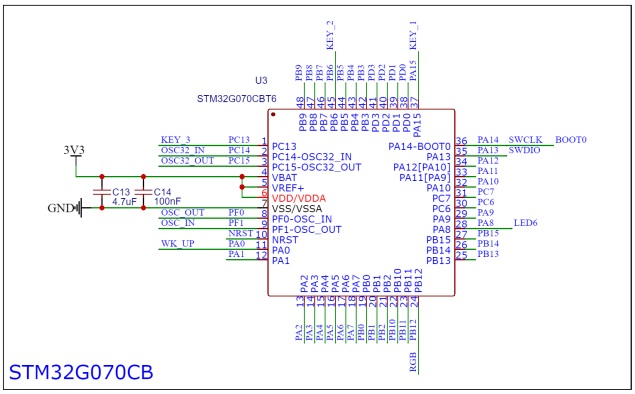


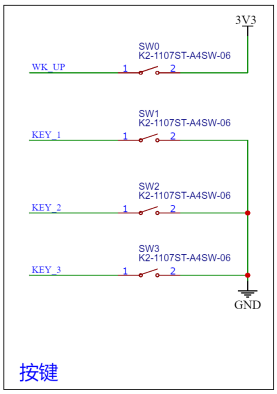


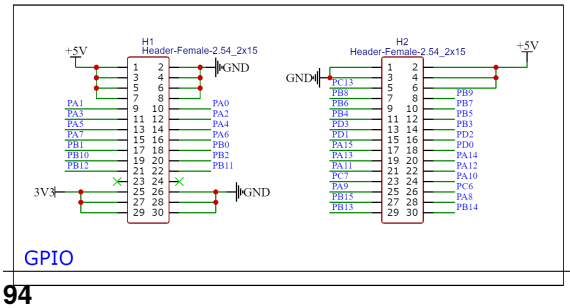




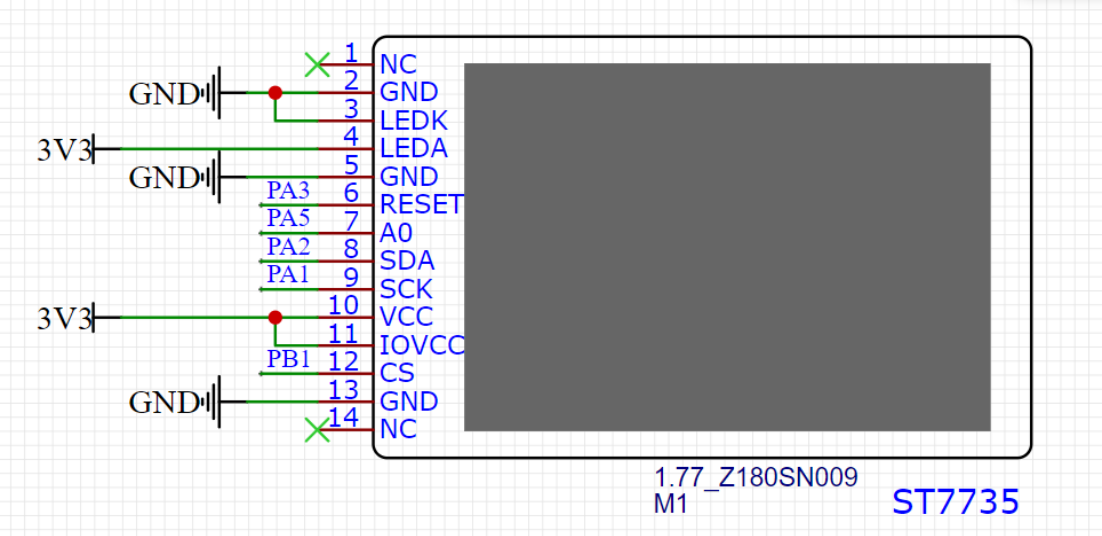








LCD


GND
3V3
GND
PA3
PA5
PA2
PA1
3V3
PB1
GND
1 NC
2 GND
3 LEDK
4 LEDA
5 GND
6 RESET
7 A0
8 SDA
9 SCK
10 VCC
11 IOVCC
12 CS
13 GND
14 NC
1.77_Z180SN009
M1
ST7735

CHAPTER

FOUR

PORTING

4.1 Deploy to new platform in ten minutes

4.1.1 How to choose a platform that can run pikascript

- PikaPython can run on all **bare metal** and **operating systems** that support libc.
- The compiler needs to be able to support the **C99 standard**.
- Supports **32bit/64bit** kernel, **does not support 8bit** kernel.
- resource occupancy
  - If it is an **arm** kernel, considering the resource consumption of the expansion module, it should be equipped with a minimum of **64k flash** and **8k ram**.
  - If it is a **risc-v** kernel, you need **128k flash** and **8k ram** , because of the gcc optimization of the risc-v kernel and the problem of code density, the code size is much larger than that of the arm kernel.
  - If it is **other kernel**, you can refer to the configuration requirements of risc-v.
- If it is a PC/server platform, linux/windows can be used.

4.1.2 Deployment operation process

In this document, we will describe how to deploy PikaPython for new platforms.

PikaPython has almost no global variables and macros, and only depends on the standard library, so it is very easy to deploy PikaPython for new platforms.

Here are the specific steps to deploy PikaPython

Prepare template project

Your template project just needs to include a serial port initialization that supports **printf**, and then you can happily use pikascript.

The usual script interpreters rely on the **operating system** and **file system**, and pikascript **does not need** these, if you have deployed other scripting engines, you will find that PikaScrip has **real super Lightweight** features.

Get PikaPython source code and toolset

To get PikaScsript, you can use the pika package manager (option 1), or use the project generator on the official website (option 2).

Option 2 is an automated version of Option 1. It is recommended that newbies use Option 1 when deploying for the first time to familiarize themselves with the package manager.

Download PikaPython Package Manager

PikaPython package manager can help you pull all **source code** and **tools** needed by pikascript, and provide **version selection** function, which is convenient for you to switch versions.

And the PikaPython package manager uses **gitee source**, which can be used smoothly in the mainland, **does not require** scientific Internet access.

Enter the PikaPython main repository

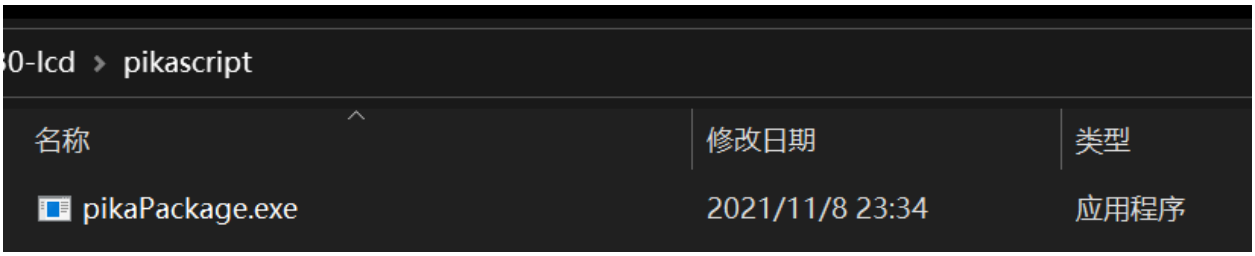
https://github.com/pikastech/pikascript

or:

https://gitee.com/lyon1998/pikascript

Download the PikaPython package manager PikaPackage.exe

Then open the project you want to deploy, create a new pikascript folder in the root directory of the project, and copy PikaPackage.exe into it.



Pull source code

Next, with the help of PikaPackage.exe, we can easily pull the source code and modules of the specified version.

Pull the source code and modules through a requestment.txt file.

If you are familiar with python's pip package manager, you will find that the requestment.txt file format of pikascript is the same as that of pip.

Create a new requestment.txt file in the pikascript folder of the project, and write the following content.

```
pikascript-core
PikaStdLib
```

The requestment.txt file indicates the installation of the `pikascript-core` interpreter kernel and the `PikaStdLib` standard library. The interpreter kernel and the standard library are mandatory, while the other modules can be added optionally, and only the kernel and the standard library should be **added** during the initial deployment to avoid compatibility issues.

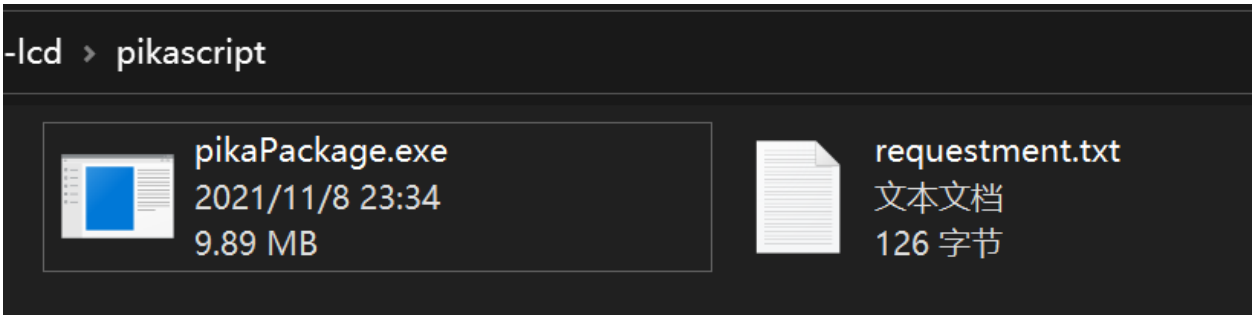
You can copy the requestment.txt kernel and standard library version of stm32g070, which is an officially supported development board The version used by Pika Pie-Zero.

And all optional versions can be viewed in the packages.toml file.

The lts2021 version refers to the long-term support version released at the end of 2021, and the support period is within 2022.

The usual version number is v1.x.x, the lts2021 version is based on v1.3.5 with stability patches.

The pikascript folder now has two files, pikaPackage.exe and requestment.txt. Double-click to run pikaPackage.exe, and the source code and modules specified in requestmemt.txt will be pulled down.



The pulled files are shown in the figure below, pikascript-core is the kernel source code, pikascript-lib is the module library, pikascript-api is the module API, and rust-msc-latest-win10.exe is the dedicated precompiler for pikascript.

After installation, the package manager will automatically lock the version and the requirement.txt will look like this

```
pikascript-core==v1.11.0
PikaStdLib==v1.11.0
```

If you want to upgrade the version, modify the version number in requestment.txt and run pikaPackage.exe again, the original version will be overwritten.

After pulling the source code, the next step is to write the python script that pikascript runs.

We create a new main.py file in the pikascript folder.

Then write:

```
import PikaStdLib

print('hello PikaPython!')
```

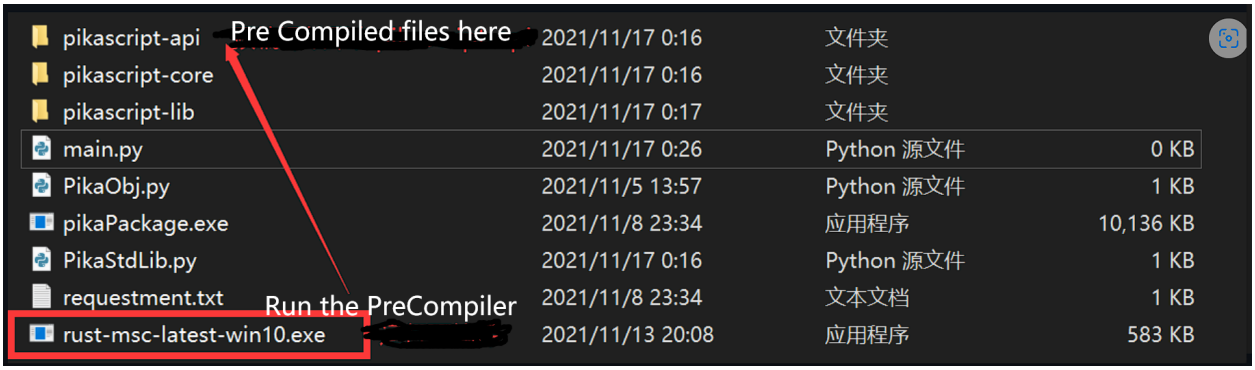
Among them, `import PikaStdLib` means importing the standard library.

The standard library must be imported, even if it is not used directly, and `print('hello PikaPython!')` is used to test whether pikascript is started normally.

Precompile modules

Next, run rust-msc-latest-win10.exe to precompile main.py and imported modules into pikascript api files.

The precompiled files are in the pikascript-api folder.

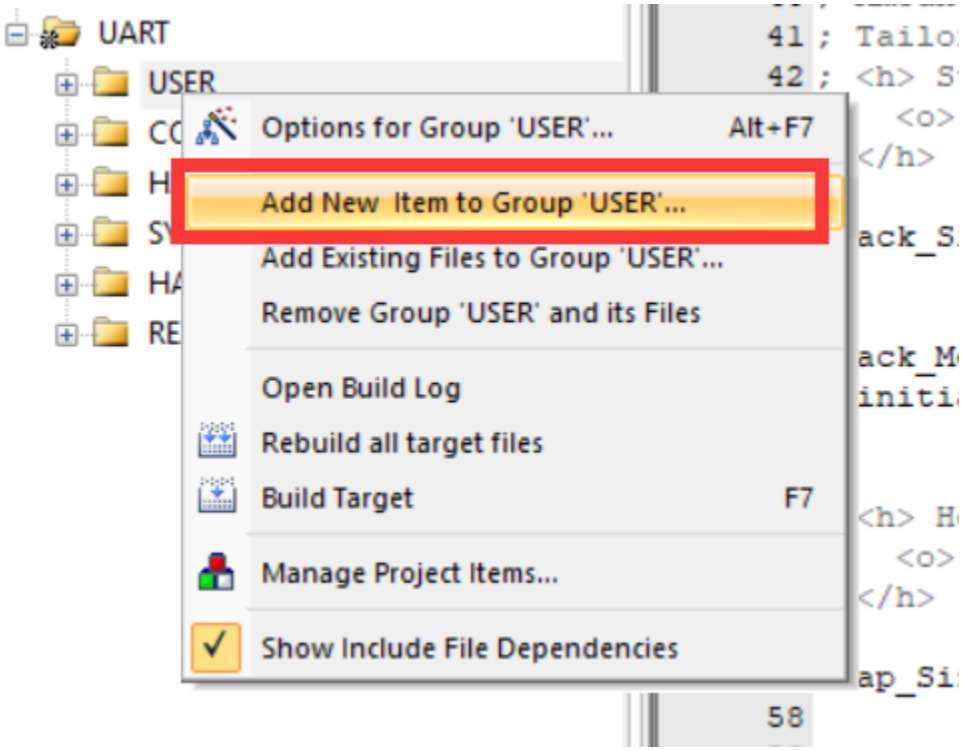


We open the pikascript-api folder and find that there are some .c and .h files in it, which means that the precompile is successful.

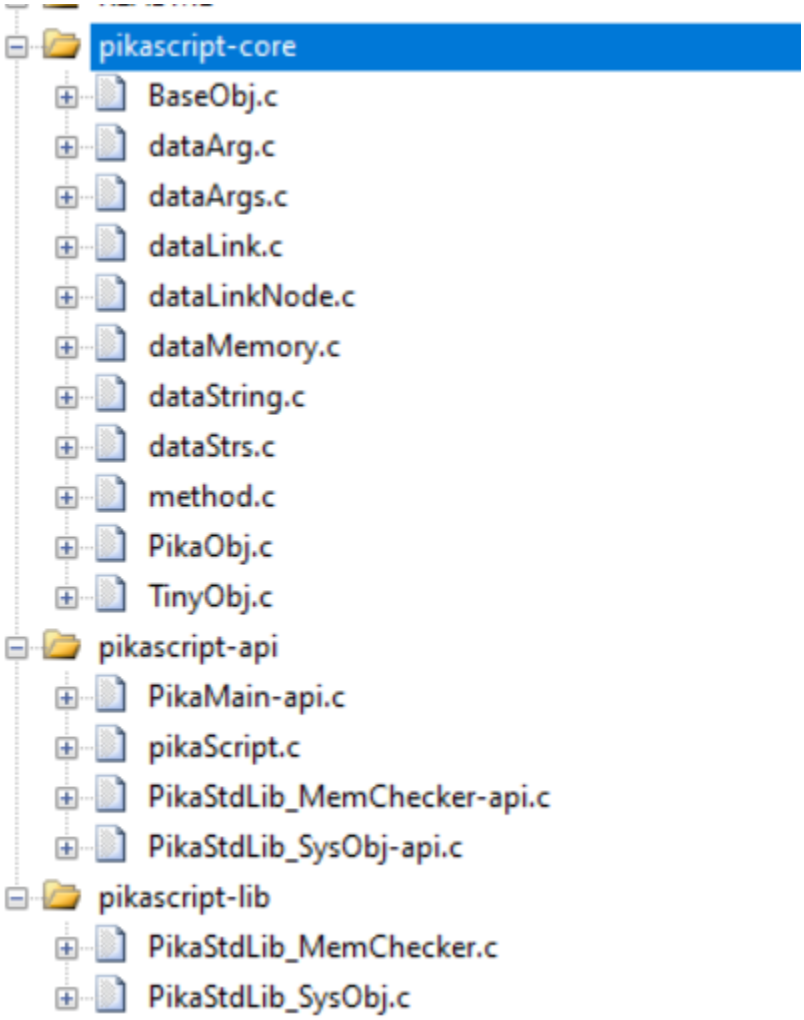
The pikascript precompiler can precompile **C modules** into .c and .h files.

Add source code

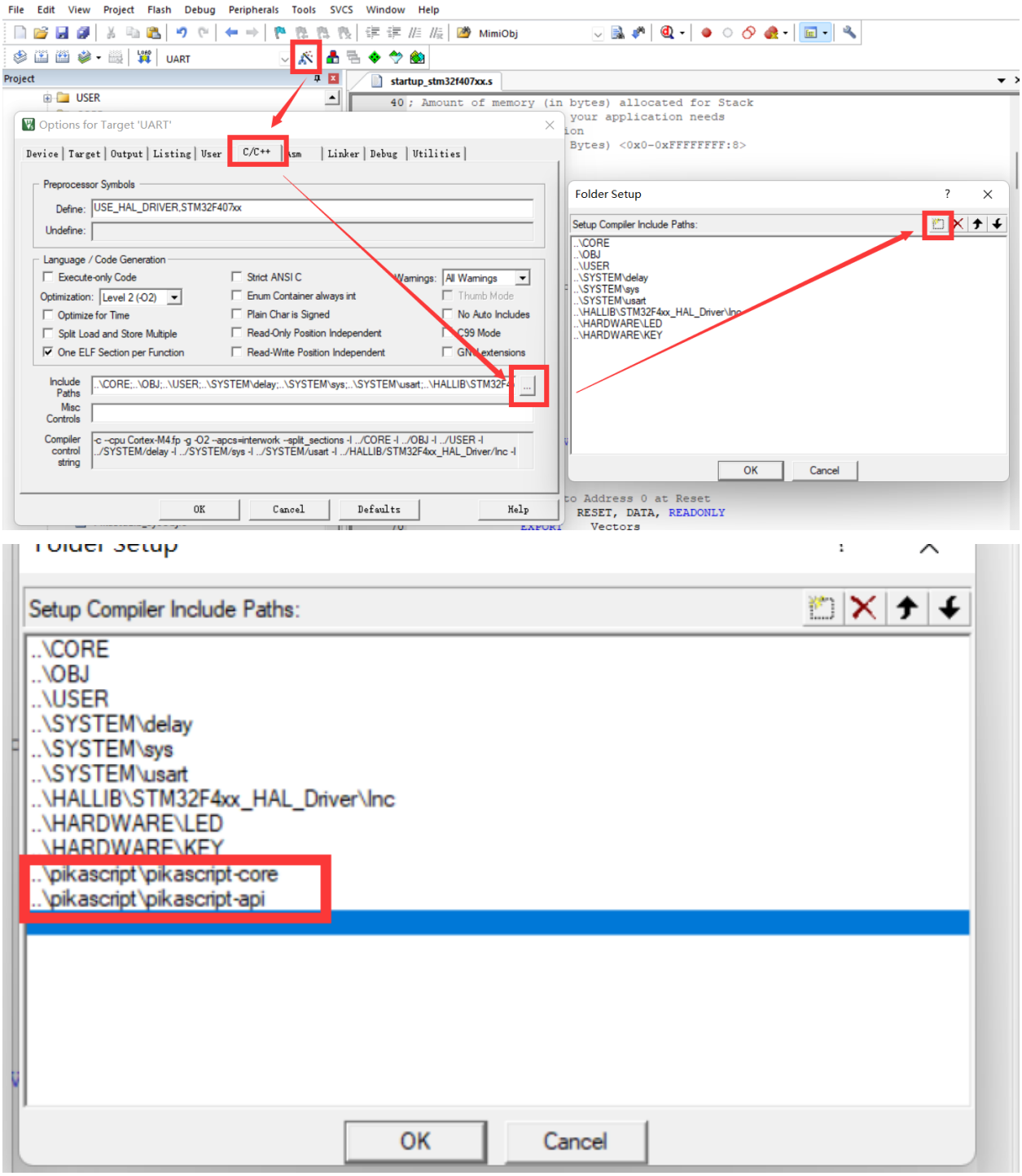
Create three new groups in Project, it is recommended to name them pikascript-core, pikascript-api and pikascript-lib



Then add all the .c files in the three subfolders of the pikascript folder (including the subfolder in pikascript-lib) to the keil project (the actual number of .c files may not match the screenshot, just add them all.)



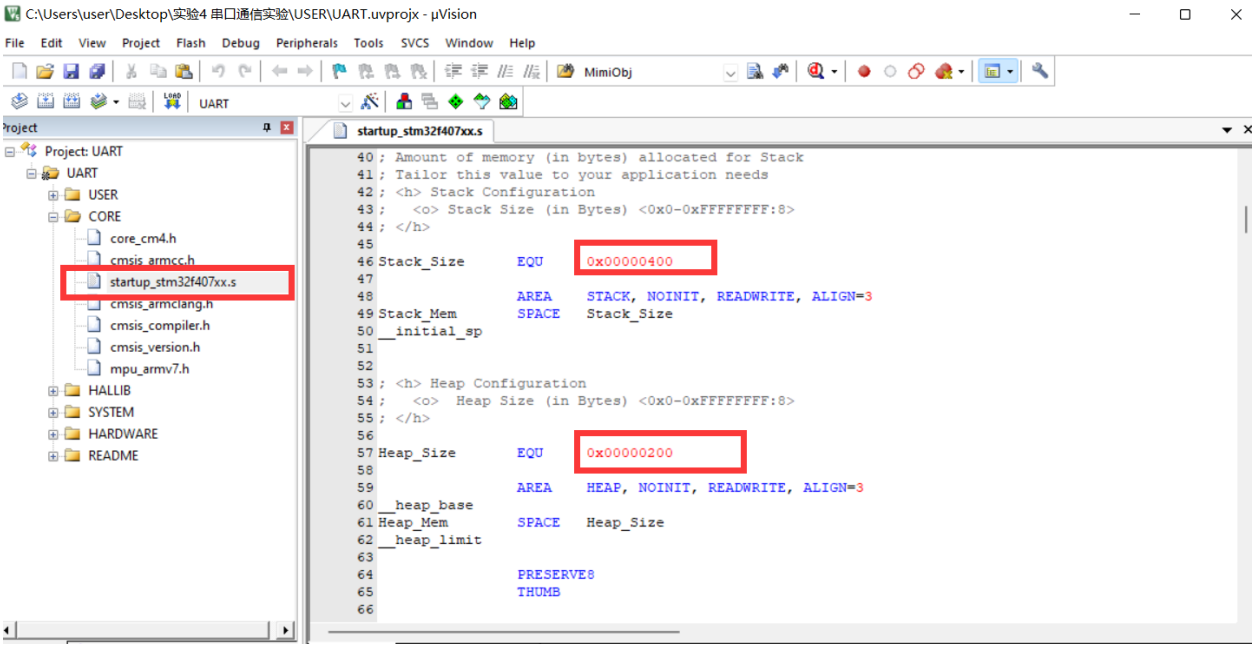
Then add include paths for pikascript-core and pikascript-api folders.


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
UART
Project
USER
startup_stm32f407xx.s
Options for Target 'UART'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Preprocessor Symbols
Define: USE_HAL_DRIVER,STM32F407xx
Undefine:
Language / Code Generation
Execute-only Code
Optimization: Level 2 (-O2)
Optimize for Time
Split Load and Store Multiple
One ELF Section per Function
Strict ANSI C
Enum Container always int
Plain Char is Signed
Read-Only Position Independent
Read-Write Position Independent
Warnings: All Warnings
Thumb Mode
No Auto Includes
C99 Mode
Include Paths
Misc Controls
Compiler control string
OK
Cancel
Defaults
Help
Folder Setup
Setup Compiler Include Paths:
..\CORE
..\OBJ
..\USER
..\SYSTEM\delay
..\SYSTEM\sys
..\SYSTEM\usart
..\HARDWARE\LED
..\HARDWARE\KEY
OK
Cancel

Setup Compiler Include Paths:
..\CORE
..\OBJ
..\USER
..\SYSTEM\delay
..\SYSTEM\sys
..\SYSTEM\usart
..\HALLIB\STM32F4xx_HAL_Driver\Inc
..\HARDWARE\LED
..\HARDWARE\KEY
..\pikascript\pikascript-core
..\pikascript\pikascript-api
OK
Cancel

Adjust stack

Open the project's startup file, in stm32, this is a startup_stm32xxxx.s file, and on other platforms, you have to figure out how to adjust the stack yourself.



It is recommended to allocate 4K stack space and 16K heap space, and at least 1K stack space and 4K heap space need to be allocated

4K stack space corresponds to 0x1000, 16K heap space corresponds to 0x4000, as shown in the following figure

```
46 Stack_Size       EQU     0x00001000
47
48                  AREA    STACK, NOINIT, READWRITE, ALIGN=3
49 Stack_Mem        SPACE   Stack_Size
50 __initial_sp
51
52
53 ; <h> Heap Configuration
54 ;   <o>  Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
55 ; </h>
56
57 Heap_Size       EQU     0x00004000
58
59                  AREA    HEAP, NOINIT, READWRITE, ALIGN=3
60 __heap_base
61 Heap_Mem        SPACE   Heap_Size
62 __heap_limit
```

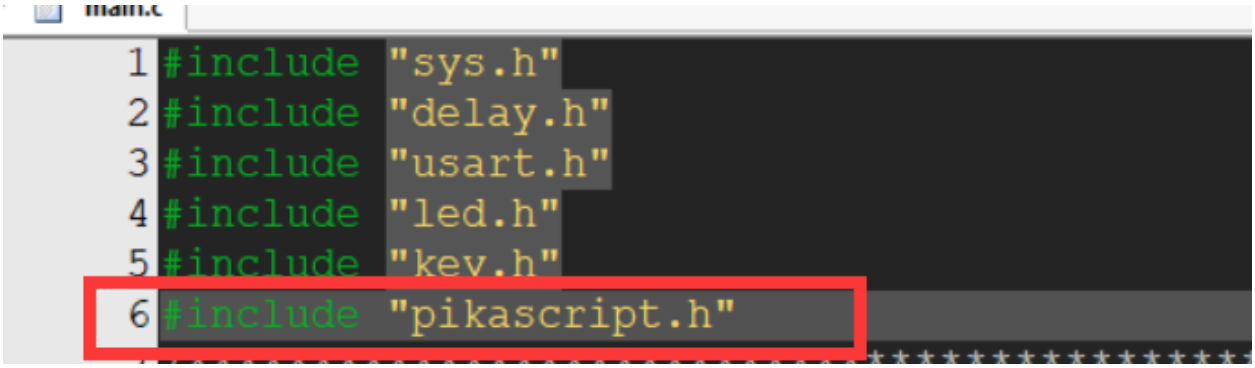
Start PikaPython

Add the startup code of PikaPython in the initialization code of main.c.

- add header files

add in header file

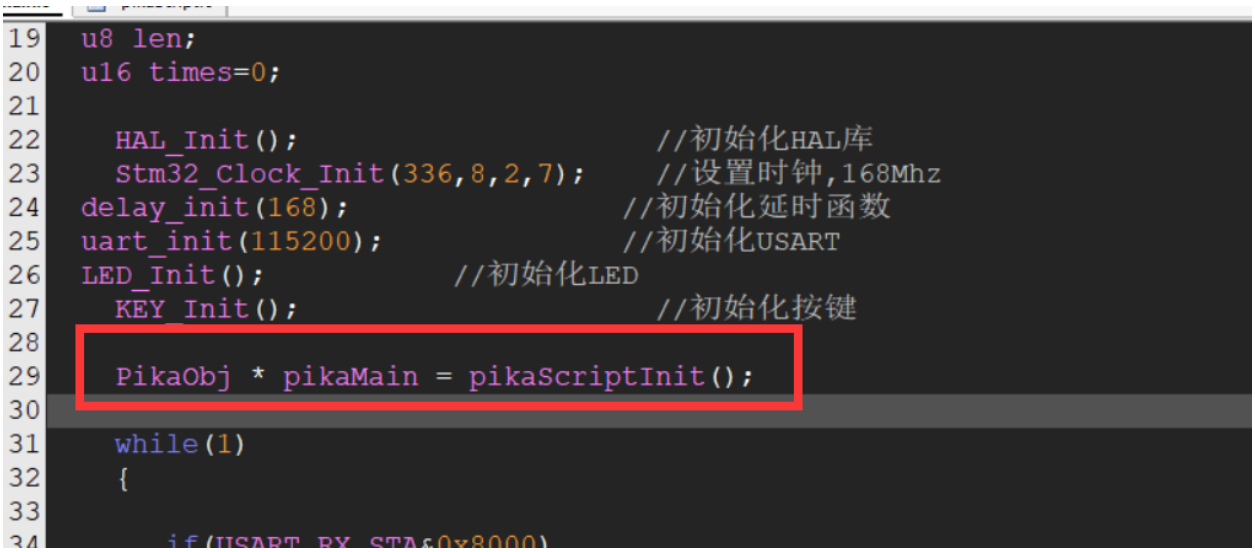
```
#include "pikascript.h"
```



- initialize pikaScript and get the pointer to the pikascript main object pikaMain

Add a startup code to the main function

```
PikaObj* pikaMain = pikaScriptInit();
```

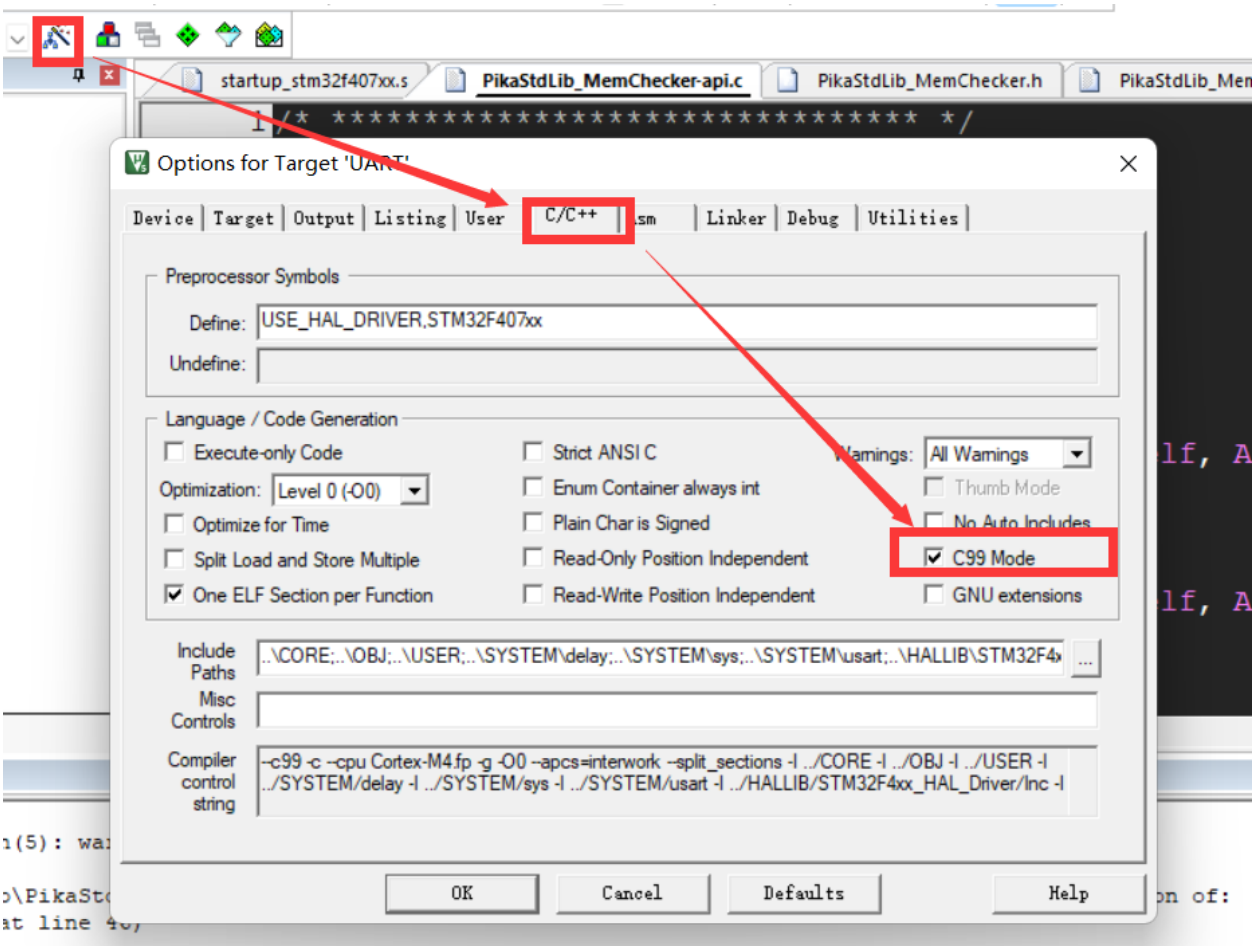


ended? Yes, it's over, it's that simple, isn't it amazing.

This is because the precompiler does a lot of auxiliary work behind the scenes, including the automatic generation of the `pikaScriptInit()` function.

compile source code

When compiling the source code, you need to check the C99 standard, and the compilation optimization level can be selected arbitrarily, and pikascript supports it.



Then you can compile it directly. Generally speaking, it can be passed directly.

You can use compiler version 5 or compiler version 6.

Contribute BSP

We sincerely appreciate your contribution, by contributing code, you can help PikaPython run on more platforms, and more developers will benefit from you.

Please see the operation method:

- *How to contribute to PikaPython BSP*

Add peripheral support

PikaPython manages peripherals through packages. To add peripheral support to the platform, please refer to the following documents:

- PikaPython Module Overview
- PikaPython Extension Module Development
- PikaPython Standard Device
- *How to contribute PikaPython modules*

4.2 Interactive Run

PikaPython supports reading strings directly to run Python scripts, so to support interactive operation, you only need to make a serial port receiving driver.

4.2.1 Option 1: Read and run by byte (recommended)

Implement a blocking byte read function

Interactive operation requires a low-level interface `__platform_getchar()` to read user input bytes. This interface is a weak function. Users need to implement a `__platform_getchar()` in their own code. to override this weak function. The weak function prototype is in PikaPlatform.c. If the user does not override it, an error will be reported when using the interactive runtime.

```
/* PikaPlatform.c */
PIKA_WEAK char __platform_getchar(void) {
    __platform_printf("[error]: __platform_getchar need impaltment!\r\n");
    while(1){
    }
}
```

Users can directly implement a `__platform_getchar()` in the main.c of the project. If the platform itself supports `getchar()`, you can directly access the platform's `getchar()`.

```
/* main.c */
char __platform_getchar(){
    return getchar();
}
```

If the platform does not support it, you need to implement it yourself, pay attention to implement a **blocking** `getchar()`, that is, when there is no serial input character, you need to use `__platform_getchar()` waits, and returns a character if there is input. E.g:

```
/* main.c */
char __platform_getchar(){
    char res = 0;
    while(rx_char == 0){
    };
    res = rx_char;
    rx_char = 0;
```

(continues on next page)

(continued from previous page)

```
    return res;
}
```

Start PikaPython Shell and run pikaScriptShell() directly to start interactive operation.

`pikaScriptShell()` The entry parameter is the root object of pika, and running `pikaScriptInit()` will create a root object.

```
pikaScriptShell(pikaScriptInit());
```

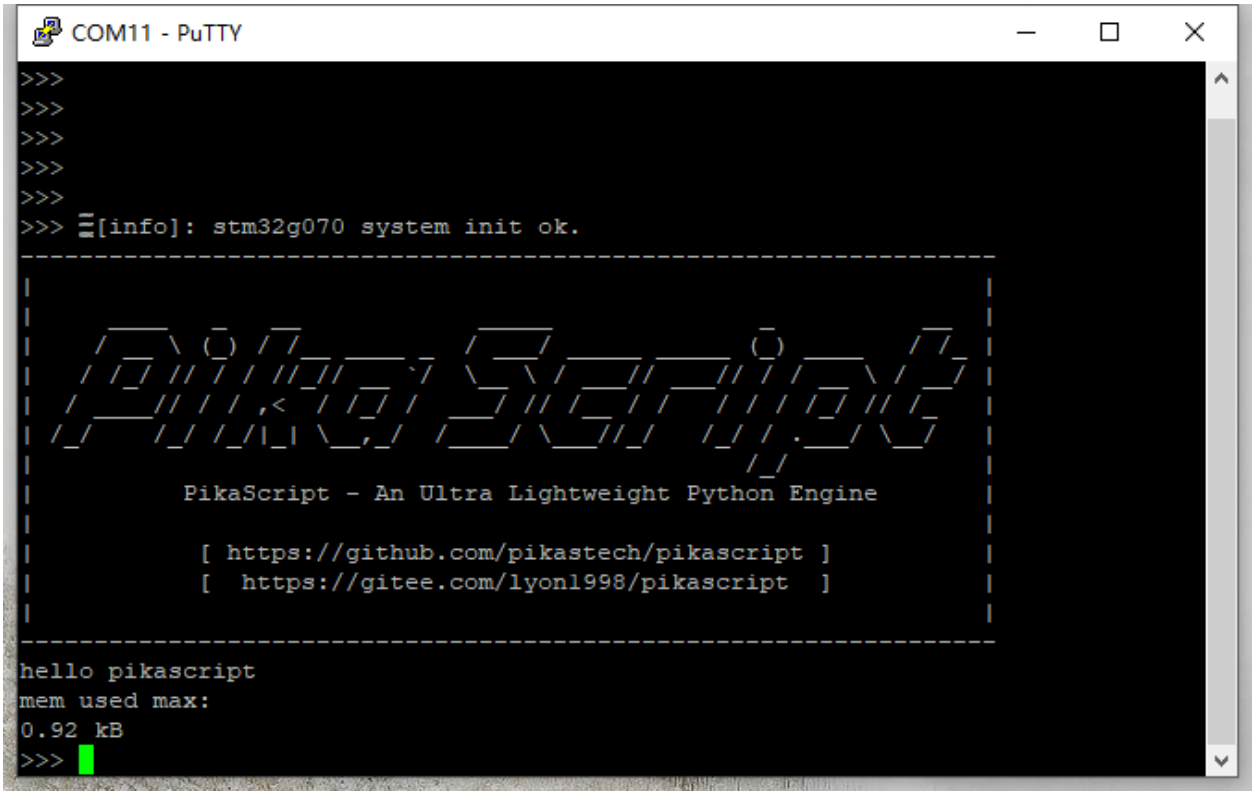
Sample code

stm32g070cb: https://github.com/pikastech/pikascript/blob/master/bsp/stm32g070cb/Booter/main.c

rt-thread: https://github.com/pikastech/pikascript/blob/master/package/pikaRTThread/rt_pika.c

Precautions:

- Kernel version needs to be at least v1.3.0
- It is strongly recommended to use putty as a serial terminal.



4.2.2 Option 2: Run by byte input

The `obj_runChar` kernel API can specify an object to execute a script with one byte of input.

You need to run `obj_runCharInit()` before you can use `obj_runChar`.

Example code.

```
PikaObj* pikaMain = pikaScriptInit();

obj_runCharInit(pikaMain);

while(1){
    char ch = my_get_char();
    obj_runChar(pikaMain, ch);
}
```

Caution.

Kernel version needs to be no less than v1.8.3

4.2.3 Option 3: Read and run the entire line

`obj_run` kernel API can specify an object to execute a script, and use this API to execute a **single-line** or **multi-line** script. The following is an example of the interactive running driver of CH32. This interactive running support is written in the main loop of the firmware and starts to execute after the `pikaScriptInit()` initialization script is executed.

```
    PikaObj *PikaMain = pikaScriptInit();
    printf(">>>");
    while(1)
    {
        if(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == SET)
        {
            is_Rx_start = 1;
            t_start = rt_tick_get();
            rxCh = USART_ReceiveData(USART1);
            if(rxCh < 128){
                RxBuffer[RxCnt++] = rxCh;
            }
        }
        if( (is_Rx_start == 1) && (rt_tick_get() - t_start > 10) ){
            is_Rx_start = 0;
            for(int i = 0; i< RxCnt; i ++)
            {
                USART_SendData(USART1, RxBuffer[i]);
                while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
            }
            obj_run(PikaMain, RxBuffer);
            printf(">>>");
            memset(RxBuffer, 0, 256);
            RxCnt = 0;
```

(continues on next page)

(continued from previous page)

```
        }
    }
```

Driven Content

- Poll to receive characters and store them in the buffer.
- A reception is considered complete when no new characters are received for more than 10ms. Using the idle time to determine the completion of the transmission of the string can support interactive running of multi-line scripts. If you only need to run the single-numbered script, you can use the newline character `'\n'` to determine the end of the string reception. When running a single-line script, the `'\n'` line break can be omitted, and a multi-line script needs to have a `'\n'` line break. Newlines of the form `"\r\n"` are also supported.
- Echo the received string after receiving.
- Execute scripts using the `obj_run` kernel API. The specified object is the root object created by the `pikaScriptInit()` init script, and the execution content is the received string.
- Clean up the receive buffer.

Notes:

- Kernel version needs to be at least v1.2.6
- When executing a multi-line script, you need to pass in a complete code block For example: the following script is a complete code block, especially the 4th line, which needs to have an indent of 0 to mark the end of the code block. and the last line needs to have a blank line, which means `print('the end')` with a newline at the end of the script.

```
while a < 10:
a = a + 1
    print(a)
print('the end')
```

The following example is also possible

```
while a < 10:
a = a + 1
    print(a)
```

The following example does not work

```
# Missing final newline
while a < 10:
a = a + 1
    print(a)
```

```
# The content of the while block is missing
while a < 10:
```

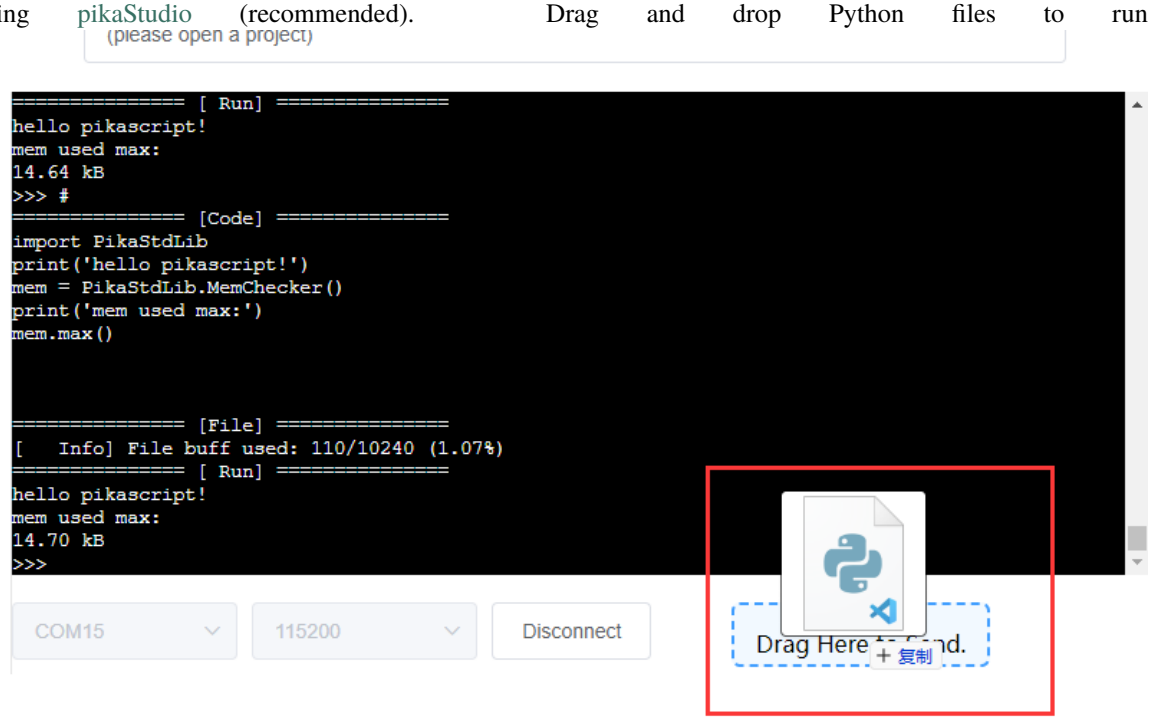
4.2.4 Quit Interaction

Type `exit()` to exit the interactive run.

4.2.5 Run temporary files

Run Python files

- Using pikaStudio (recommended). Drag and drop Python files to run



- Using other serial port tools (not recommended). Add `#!pika` to the first and last lines of the file you want to run, and then send the file directly through the serial port to run it. This file will be sent to RAM and run directly, and will be disabled after reboot. For example

```
#!pika
print('hello pikapython in file')
#!pika
```

Output.

```
>> #
========== [code] ==========
print('hello pikapython in file')
========== [code] ==========
hello pikapython in file
>>>>
```

Note that.

1. requires kernel version >= `v1.11.4`.
2. The first and last line of the temporary file must be `#!pika`, otherwise it will be treated as a normal string.

3.

4. `__platform_getchar()` is needed not to be too slow, otherwise the file will fail to be sent, or you can try to slow down the serial port baud rate.

Run the bytecode file

Send the xxx.py.o bytecode file to run Output.

```
=============== [Code] ===============
[ Info] Bytecode size: 305
=============== [ RUN] ===============
hello pikapython in file
```

Note that.

1. requires kernel version >= `v1.11.6`.

4.3 Docking with IDE

4.3.1 Overview

The toolset that PikaPython needs to interface with the IDE includes:

Package manager pikaPackage.exe

Refer to **package manager and module management** related documents

Precompiler rust-msc-latest-win10.exe

Refer to **module development** related documents

4.3.2 calling method

1. Start path:

1. [Bare metal project root directory]/pikascript path
2. [rtthread project root directory]/packages/pikascript-latest path

2. Package Manager

1. When pulling a module remotely from PikaSciprt for the first time, you need to run pikaPackge.exe
2. After modifying request.txt, you need to run pikaPackage.exe
3. If you use the latest version of the module, you need to run pikaPackage.exe when updating the module to the latest

3. Precompiler

a. run before each compilation **[Note]:** When running for the first time, use pikaPackage.exe to pull the precompiler first.

4.3.3 Project Files

1. After executing the package manager or precompiler, you need to add **all (including subfolders)** .c files and include paths under **pikascript-lib, pikascript-core, pikascript-api** .
2. Reset PikaPython project files: After deleting pikascript-lib, pikascript-core, and pikascript-api, re-run pikaPackage.exe and rust-msc-latest-win10.exe.

4.3.4 example

Automatic precompile script pikaBeforeBuild-keil.bat written for keil:

```
cd ../pikascript

if not exist pikascript-core (
    pikaPackage.exe
)
rust-msc-latest-win10.exe
```

4.4 Serial port download Python script

The serial port download Python script is very similar to the interactive running, and still uses the obj_run kernel API to run the script. Unlike interactive running, downloading a Python script also requires **storing** of the python script.

obj_run supports running Python scripts in the form of strings, so no matter how you store them, just pass the string of the Python script to obj_run at the end. So the possible storage methods are: **flash direct storage, file system, external storage** and so on.

PikaPython supports running Python script source code and parsed Pika bytecode.

4.4.1 Store Python source code

Storing the Python source code is very simple, just write the Python script string received by the serial port into Flash completely. Instead of using the pikaScriptInit() function at startup, manually create the pikaMain root object, and then use obj_run(pikaMain, code) to run the script, where code represents the stored python source code.

For specific code examples, please refer to:

1. https://github.com/pikastech/pikascript/blob/master/bsp/stm32g030c8/Booter/main.c
2. https://github.com/pikastech/pikascript/blob/master/bsp/stm32g030c8/Booter/pika_config.c
3. https://github.com/pikastech/pikascript/blob/master/bsp/stm32g030c8/Booter/pika_config.h

4.4.2 Store Pika bytecode

(to be improved) For specific code examples, please refer to:

1. bsp/stm32g030c8/Booter/main.c
2. bsp/stm32g030c8/Booter/pika_config.c
3. bsp/stm32g030c8/Booter/pika_config.h

4.5 Running Files Using the File System

When the MCU has a filesystem ported, you can use the file API to run Python script files directly.

[Note: requires kernel version >= `v1.10.0`.

The file API needs to be interfaced to the following file systems by overriding the WEAK function.

```
/* fopen */
PIKA_WEAK FILE* __platform_fopen(const char* filename, const char* modes);
/* fclose */
PIKA_WEAK int __platform_fclose(FILE* stream);
/* fwrite */
PIKA_WEAK size_t __platform_fwrite(const void* ptr,
                                   size_t size,
                                   size_t n,
                                   FILE* stream);
/* fread */
PIKA_WEAK size_t __platform_fread(void* ptr,
                                  size_t size,
                                  size_t n,
                                  FILE* stream);
```

Use `pikaVM_runSingleFile` to run a single Python file (no other files can be imported).

Function prototype.

```
VMParameters* pikaVM_runSingleFile(PikaObj* self, char* filename);
```

Use `pikaVM_runFile` to run Python files and their `import` files. A new `pikascript-api` folder needs to be created in the same level path as the running Python file to hold the intermediate files.

Function prototype.

```
VMParameters* pikaVM_runFile(PikaObj* self, char* file_name);
```

CHAPTER

FIVE

MODULE DEVELOPMENT

5.1 Module Import

The embedded environment is significantly different from the PC, in many cases the MCU doesn't even have a file system.

But don't worry, PikaPython already helps you to import modules easily with its official tools, all you need to do is to write a line `import`, just like you do with Python on PC.

The only difference with Python for PC is that you need to run the pre-compiler provided by PikaPython once (no complicated parameters and options, just double-click to run) before you can compile your PikaPython project with the compiler.

5.1.1 Importing Python modules

PikaPython supports importing multiple Python files as modules, and there is no need to port the filesystem inside the MCU (if you want to base it on a filesystem, you can, of course).

PikaPython's pre-compiler converts Python files into bytecode and packages them into a library right on the PC development machine, just like C.

This eliminates the need for a filesystem in a MCU with few resources (usually 20kB of ROM).

On the other hand, if you want to quickly try PikaPython on a new platform, you don't need to go through the effort of porting the filesystem for the new platform and then interfacing the filesystem with PikaPython.

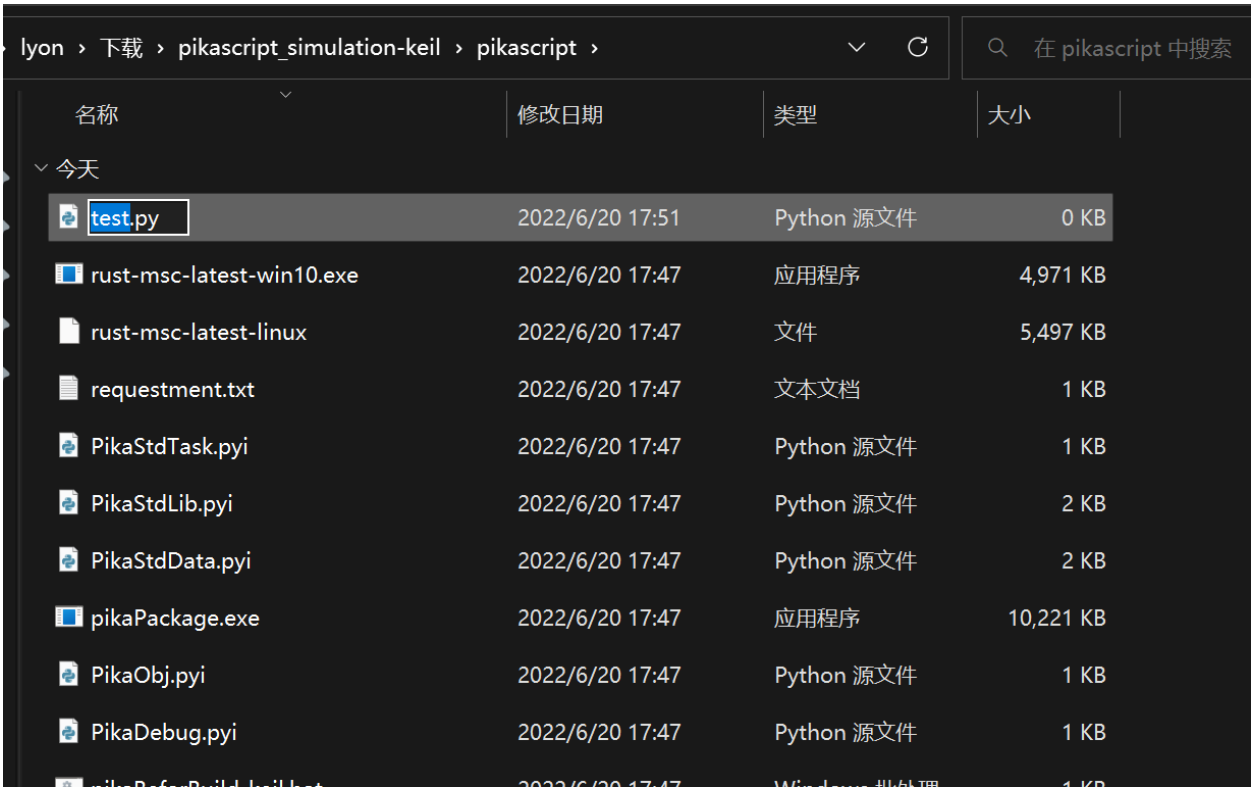
(Note that a kernel version of not less than v1.8.0 is required)

Experiment

We still use keil's emulation project as our experiment platform, so that we can experiment quickly without hardware.

First, refer to keil's emulation project documentation to get the project.

Then create a new Python file test.py in the pikascript_simulation-keil/pikascript/ directory (all Python modules should be placed in this directory).



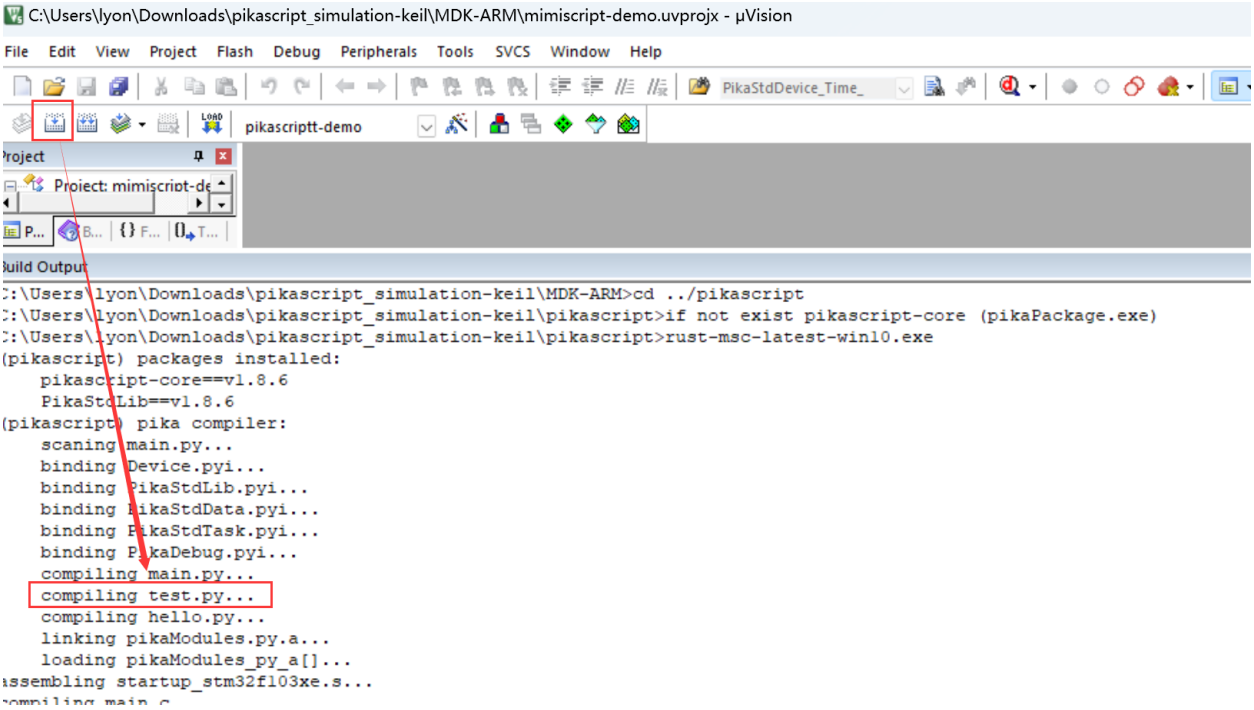
Then write the test code inside test.py as follows

```
# test.py
def mytest():
    print('hello from test.py!')

def add(a, b):
    return a + b
```

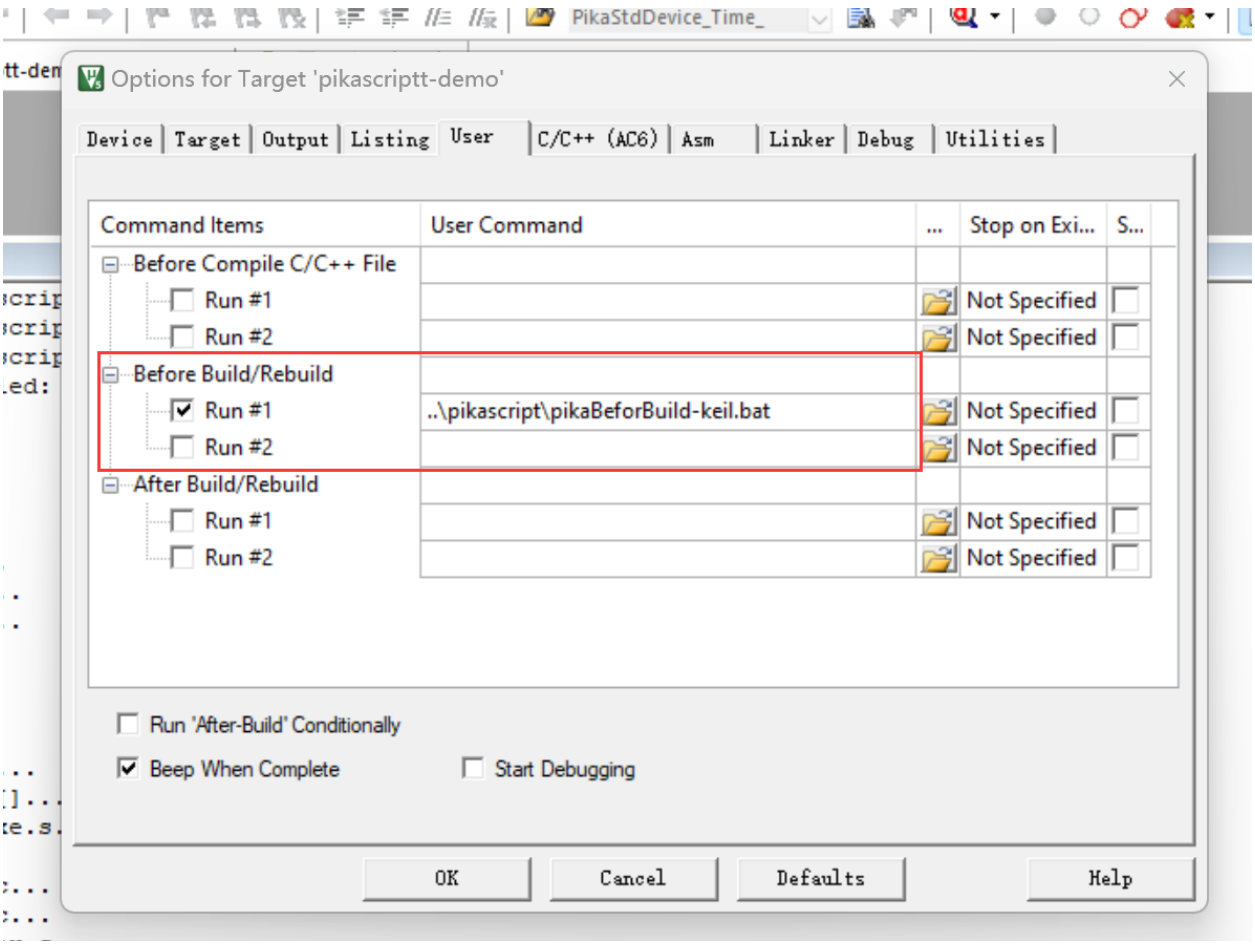
Next, introduce test.py inside main.py and test the functions mytest() and add() that we defined in test.py

```
import Device
import PikaStdLib
import PikaStdData
import hello

import test

print('test start...')

test.mytest()
print(test.add(3, 5))

print('test end...')
```

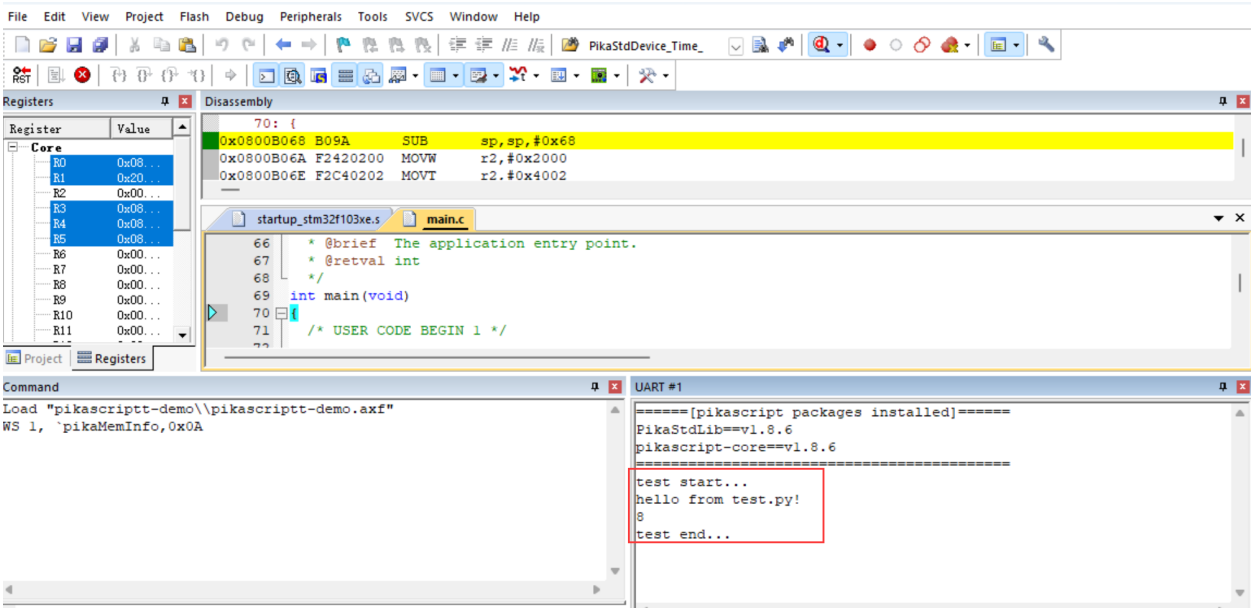
Then, if you compile directly inside the keil project, you will see that the PikaPython Compiler message appears before you start compiling the .c file, including the compiled test.py.



This is because the PikaPython precompiler has been automatically run, a Keil-supplied setting that executes a script before compilation begins, including running the PikaPython precompiler.



Then we start debugging the run and open the serial window to see the results



If you are interested in the principle, you can watch the explainer video.

5.1.2 Importing C modules

A C module is a module that is implemented in C at the bottom, but can still be called with Python.

A C module named <module> usually consists of a <module>.pyi file (a python interface file) and the pikascript-lib/<module> folder.

PikaPython imports C modules in the same way as Python modules, by directly `import` and then running a pre-compile.

After pre-compilation, some module linking files are automatically generated, all of them are in the pikascript-api folder. Therefore, after introducing the C module, you need to add the following files to the project for compilation.

- All .c files in the pikascript-lib/<module> folder
- All .c files in the pikascript-api folder

Experiment

We are still using the keil emulation project as our experimentation platform.

We introduce the PikaStdData.pyi C module in main.py.

We open PikaStdData.pyi to see the classes and functions provided by this C module.

```
# PikaStdData.pyi
class List:
    def __init__(self): ...
    # add an arg after the end of list
    def append(self, arg: any): ...
    # get an arg by the index
    def get(self, i: int) -> any: ...
    # set an arg by the index
    def set(self, i: int, arg: any): ...
    # get the length of list
    def len(self) -> int: ...
...
```

You can see that there is a `List` class inside.

Introduce `PikaStdData` in main.py and create a new object `list` with the `List` class, then test the `append()` method and the `get()` method of `List`.

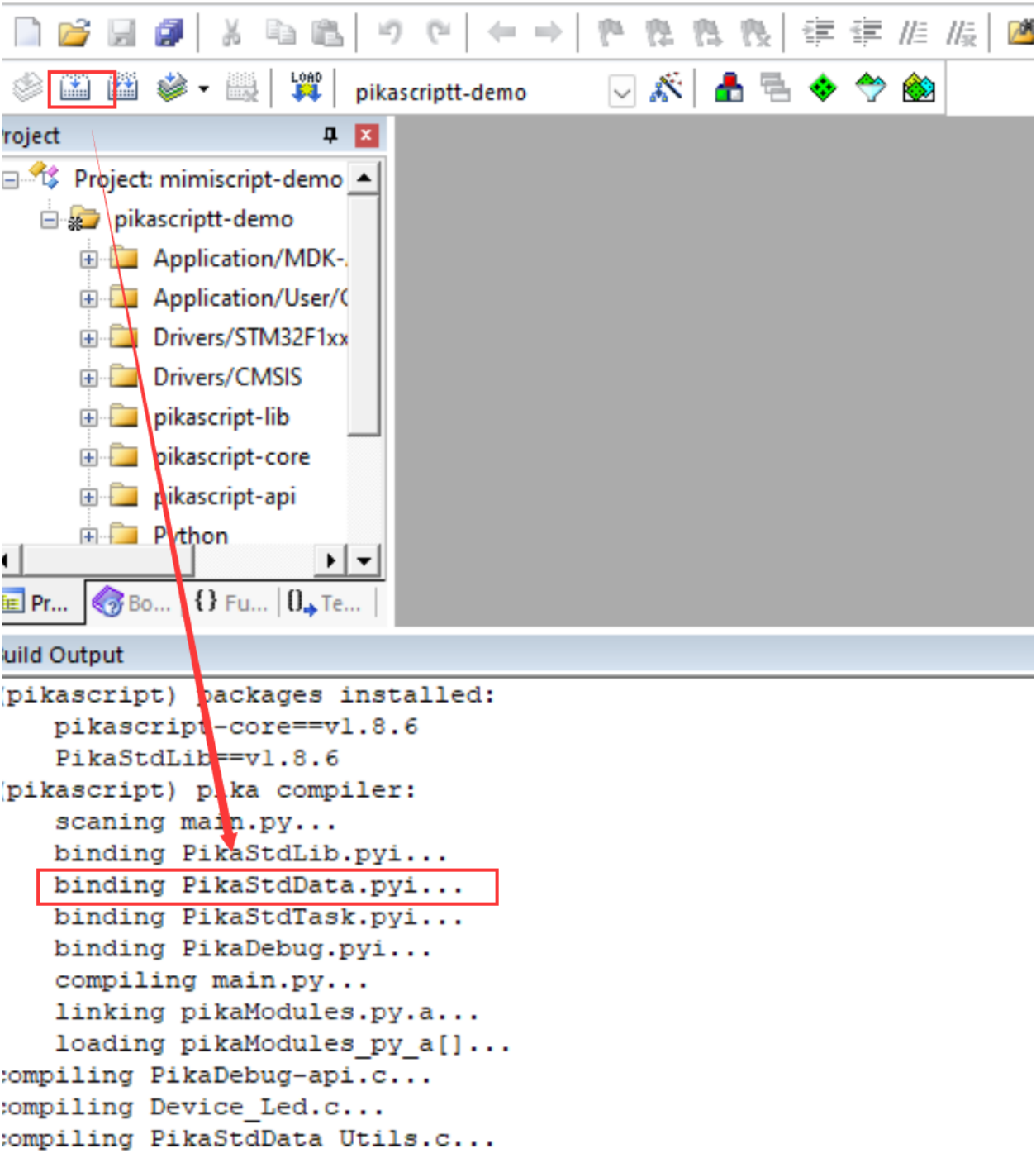
```
import PikaStdLib

import PikaStdData

print('test start...')

list = PikaStdData.List()
list.append(1)
list.append('test')
list.append(2.34)

print(list.get(0))
print(list.get(1))
print(list.get(2))
```

(continues on next page)

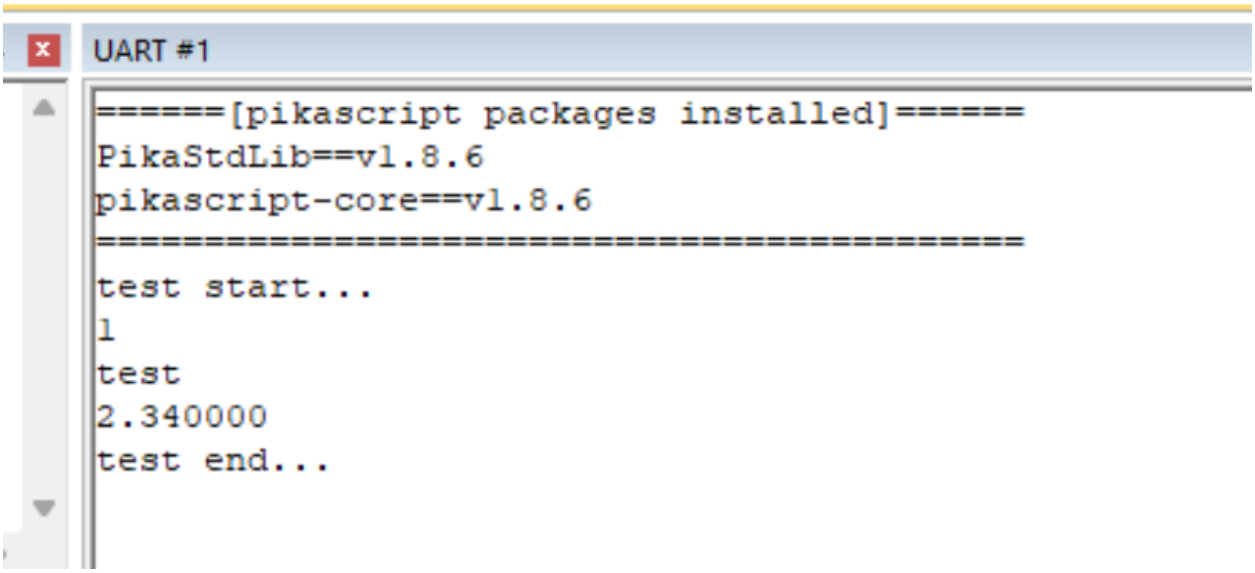
(continued from previous page)

```
print('test end...')
```

When compiling, you can see that the PikaPython pre-compiler binds the PikaStdData C module to the project.



Running the simulation you can see the result



You can also make your own C modules, all you need to do is write the <module>.pyi Python interface file and the .c implementation file inside pikascript-lib/<module>.

Please refer to the documentation for making C modules for details.

5.2 Package manager

5.2.1 Click to download Package Manager

5.2.2 PikaPackage package manager

PikaPython has an officially supported package manager, PikaPackage, which is used for module management. It can provide kernel, module download, module release, kernel, and module version switching functions, which is convenient for distributing developed modules and managing module versions.

PikaPackage is a monolithic application for the windows platform. Based on the development of the go language, it integrates the go-git component, which can realize the pull, update and version switching functions of the git repository without installing git.

5.2.3 Workflow of PikaPackage

PikaPackage will automatically complete the series of processes shown in the figure above.

- The first is to check whether the /tmp/pikascript folder exists. If not, it will clone a pikascript repository first.

The /tmp directory refers to the tmp folder in the root directory of the disk where pikaPackage.exe is currently located. For example, if pikaPackage.exe is on drive C, then /tmp is C:/tmp, and if it is on drive D, then /tmp is D:/tmp. The clone repository uses the gitee source, so don't worry about the network connection problem, and it is also very fast in China.

- Update repository to latest state.
- Read the modules in the current request.txt file.

Here is an example of a requestment.txt file, the format of this file is the same as the mainstream python pip package manager format, fill in the module name and version number to pull the corresponding module.

```
pikascript-core
PikaStdLib
PikaStdDevice==v1.6.0
STM32G0==v1.2.0
PikaPiZero==v1.1.3
```

You can write the module name directly, e.g. `pikascript-core`, `PikaStdLib`.

Or specify the version number, e.g. `PikaStdDeivce==v1.6.0`, currently only the == symbol is supported, which means the version number is strictly matched.

There is also a special version `latest`, which means pulling the latest module, which refers to the latest version of the master branch in the pikascript repository

If you are a module user and not a developer, please be careful to avoid using the latest version at all costs. Because the latest version is constantly changing, newer versions of the module may cause compatibility issues.

- pikaPackage.exe checks /tmp/pikascript/packages.toml file, which is a module description file in a repository, this file describes all available modules and their versions. The following is the intercepted part of the packages.toml file. In this file, there are four modules, pikascript-core, PikaStdLib, PikaStdDevice, and STM32, which can be pulled, and the release section under each module describes the version that can be used.

The format of the module version is " ". Fill in the corresponding version name in request.txt to pull the corresponding version of the module.

If you also want to publish the module, you can fill in the packages.toml file in the same way, and the package manager can recognize the module you published.

```
[[packages]]
name = "pikascript-core"
releases = [
  "v0.8.1 af42fd61884dc7048628b0d3bafaa42697f6e8ea",
  "v0.8.2 af42fd61884dc7048628b0d3bafaa42697f6e8ea",
  "v0.8.3 af42fd61884dc7048628b0d3bafaa42697f6e8ea",
  "v0.9.0 332ef8afb0692cddd194782a07e30f2688d0f813",
  "v0.10.0 c86eaefa4516dd82b1050fa74a7d85399459d5ed",
  "v1.0.0 7b816b1546ef91a03f77760d4b10806ab956d452",
  "v1.1.0 845d1fc6520237e2238087800f72608dcb81afa6",
  "v1.1.1 c77e42450ed0eb09fcd7bb2b7d7c2b7eeeb55a2e",
  "v1.1.2 f6ad2c78f49162ab3f898abc6a0a4d87777ce655",
  "v1.1.3 6539072bf7bebb242ea40f8595bfb5c9aae3de7f",
  "v1.2.0 ce3df083b68fbfc85e64e6793fe07a6736d6f29f",
  "v1.2.1 e29a77527fd753c4eb811b047899534472bfc8ec",
  "v1.2.2 5316ede928b01a20571103616a64666abbc40e0a",
  "v1.2.3 5ae86929851ff6a62342a7072b77e9cf5be85f1c",
  "v1.2.4 b7ac057d75e88736cc844de0bafb447a48f2fb6d",
  "v1.2.5 db51f0520a673074a14ef0f5c4434da0d5c3425f",
  "v1.2.6 044a2a8f0905c6ca90c633759f397323ce57eefd",
]

[[packages]]
name = "PikaStdLib"
releases = [
  "v1.0.1 af42fd61884dc7048628b0d3bafaa42697f6e8ea",
```

(continues on next page)

(continued from previous page)

```
  "v1.1.0 0b3b866dbacc363c7b6b3899faa0cbcaccd59d5e",
  "v1.2.0 ca29e112687525ee7511bd30418d368754627a00",
  "v1.2.1 5ae86929851ff6a62342a7072b77e9cf5be85f1c",
  "v1.2.2 b7ac057d75e88736cc844de0bafb447a48f2fb6d",
]

[[packages]]
name = "PikaStdDevice"
releases = [
  "v1.3.0 af42fd61884dc7048628b0d3bafaa42697f6e8ea",
  "v1.4.0 29c3c5b3b0cb4d3e41e6a2a0aef9e2826bc6f7ba",
  "v1.4.1 6539072bf7bebb242ea40f8595bfb5c9aae3de7f",
  "v1.4.2 5ae86929851ff6a62342a7072b77e9cf5be85f1c",
]

[[packages]]
name = "STM32"
releases = [
  "v1.0.0 af42fd61884dc7048628b0d3bafaa42697f6e8ea",
  "v1.0.1 af42fd61884dc7048628b0d3bafaa42697f6e8ea",
  "v1.0.2 af42fd61884dc7048628b0d3bafaa42697f6e8ea",
  "v1.1.0 a18910b5dc349c64297bba3a13b7044f41d48e5f",
  "v1.1.1 91818aab0fa87b007e84866d479af5ac507339fe",
  "v1.2.0 6bd4aac6e9aba2a603da602be8583021da1272c0",
  "v1.3.0 7b816b1546ef91a03f77760d4b10806ab956d452",
  "v1.4.0 29c3c5b3b0cb4d3e41e6a2a0aef9e2826bc6f7ba",
  "v1.4.1 6539072bf7bebb242ea40f8595bfb5c9aae3de7f",
  "v1.4.2 8866710f653ad005f5c3edc5e6417ad31075b7d5",
  "v2.0.0 e29a77527fd753c4eb811b047899534472bfc8ec",
  "v2.0.1 5ae86929851ff6a62342a7072b77e9cf5be85f1c",
]
```

- pikaPackage.exe go to the /tmp/pikascript/pacakge folder to find the folder with the same name as packages.toml, then switch to the specified commit id, and then copy the folder to the current pikascript-lib folder.

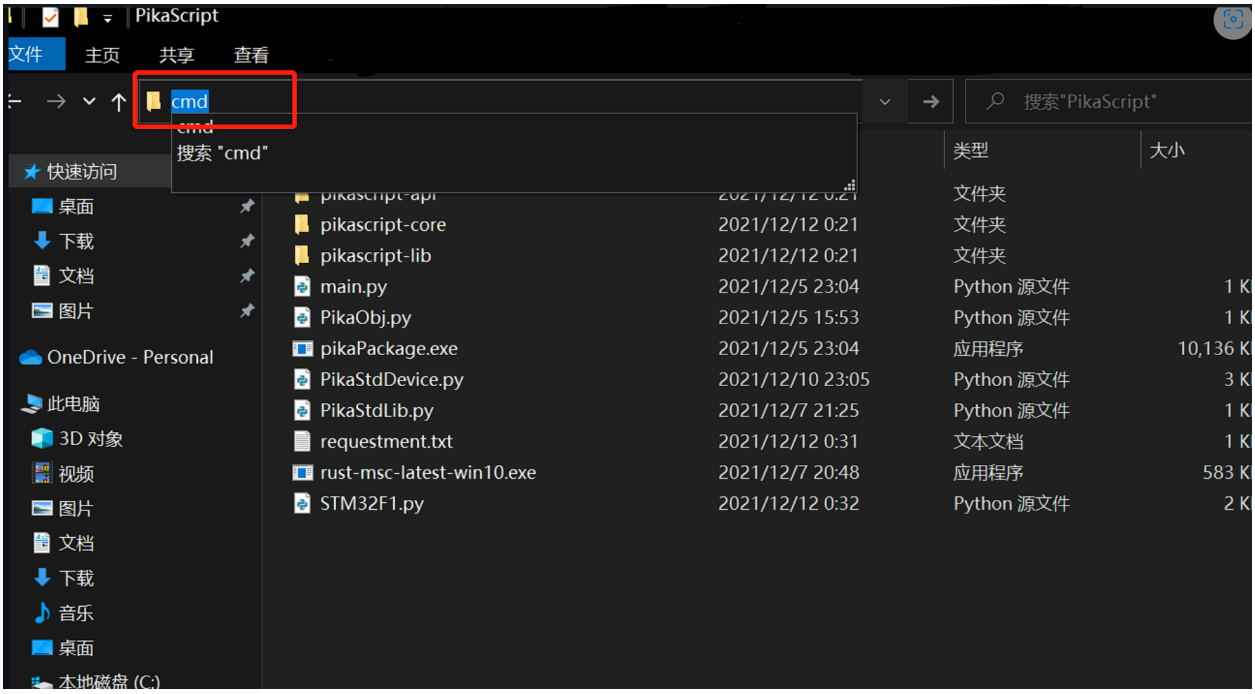
If you want to publish a module, create a new folder with the same name as the module in the pikascript/package directory, and then copy all the files contained in the module to this folder. After submitting the folder and obtaining the commit id, fill in the commit id into the packages.toml description file.

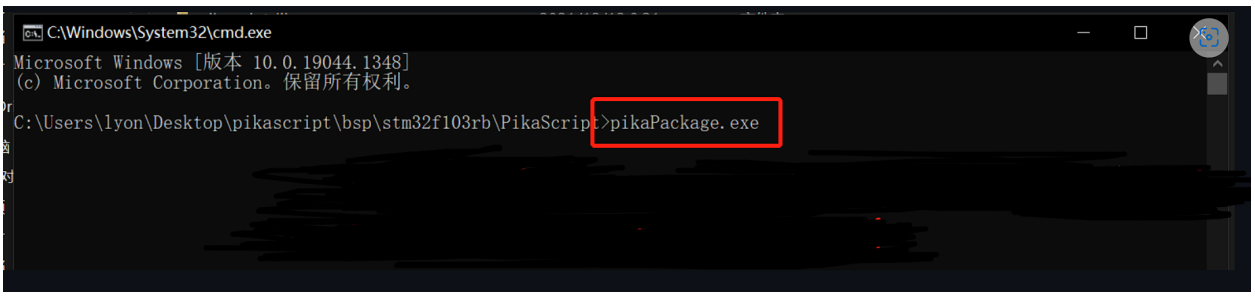
Note: To limit the complexity of modules and improve maintainability, nested folders are not supported in module folders.

- The *.py and *.pyi files contained in the module folder will be automatically copied to the current folder, in order to be able to recognize the python module (python only recognizes the module files in the current folder when importing a module.)

5.2.4 Error troubleshooting

If you are suspected of encountering problems during use, you can run pikaPackage.exe in cmd to view the complete log information.





CHAPTER

SIX

STANDARD LIBRARY

6.1 PikaStdLib standard library

PikaStdLib is a built-in library of PikaPython, which must be installed. It includes memory checking tools and system objects.

6.1.1 Install

Add the dependency of PikaStdLib to requestment.txt. The version number of PikaStdLib should be the same as the version number of the kernel.

```
PikaStdLib
```

Run pikaPackage.exe

6.1.2 import

Add in main.py

```
#main.py
import PikaStdLib
```

6.1.3 class MemChecker()

MemChecker provides PikaPython's memory monitoring capabilities. Can be used to view memory usage and check for memory leaks.

```
def max(self):
```

Print the maximum memory footprint value.

```
def now(self):
```

Print the current memory usage value.

```
def getMax(self)->float:
```

Returns the largest memory footprint

```
def getNow(self)->float
```

Returns the current memory usage value.

```
def resetMax(self)
```

Reset the maximum memory usage value **Example:**

```
# main.py
import PikaStdLib
mem = PikaStdLib.MemChecker()
print('mem used max:')
mem.max()
print('mem used now:')
mem.resetMax()
print('mem used max:' + str(mem.getMax()))
print('mem used now:' + str(mem.getNow()))
```

6.1.4 class SysObj()

SysObj is used to provide built-in functions, the scripts executed in main.py are executed by the root object, and the root object is created by the SysObj class, so the methods in the SysObj class are built-in functions.

```
def type(arg: any):
```

print variable type

```
def remove(argPath: str):
```

To remove a variable/object, use a string when removing, e.g. `remove('a')`.

```
def int(arg: any) -> int:
def float(arg: any) -> float:
def str(arg: any) -> str:
```

for type conversion

```
def print(arg:any):
```

Inherited from BaseObj, provides print output. Formatted output is not currently supported.

6.2 PikaStdDevice Standard Device

PikaStdDevice is an abstract device model that provides a unified API for peripherals across platforms.

6.2.1 Installation

- Add the PikaStdDevice dependency to requestment.txt.

```
PikaStdDevice
```

- Run pikaPackage.exe

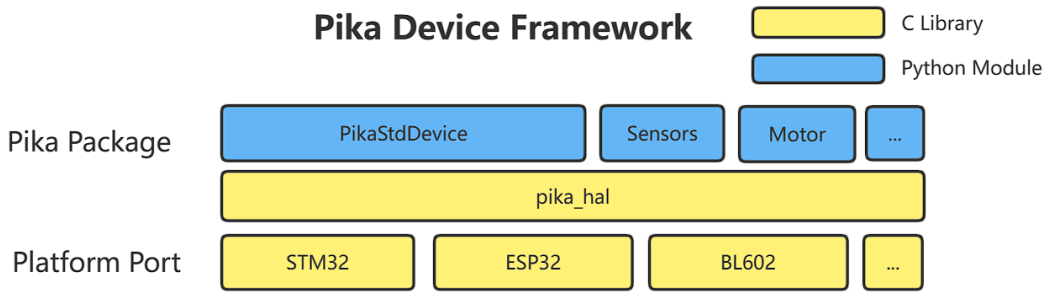
6.2.2 Why do we need a standard device module

What is a standard device module? Let's start with other scripting technologies, such as MicroPython, there is no unified API for peripheral calls, which makes users need to relearn the API when using different platforms, for example, the following is the code for MicroPython to drive GPIO on STM32F4 platform.

This is the ESP8266

It is obvious that when selecting the pin, one uses a string, while the other uses an integer, and when controlling the level, one uses the `high()` and `low()` methods, while the other uses the `on()` and `off()` methods. Is there any way to unify the APIs of peripherals, so that users only need to be familiar with a set of APIs, they can be common in any platform? There is a way, and it is the PikaStdDevice standard device driver module.

6.2.3 Module structure



- The `PikaStdDevice` module provides the basic peripheral Python modules for `GPIO`, `IIC`, `PWM`, etc.
- `PikaStdDevice` is based on the `pika_hal` device abstraction layer. `pika_hal` is a pure `c` language device abstraction layer that unifies peripheral operations of different platforms into the same `API` for `PikaStdDevice` to call, so that different platforms (STM32, ESP32, BL602) etc. can use common `Python` code to control the device.
- The `pika_hal` device abstraction layer needs to be adapted in different platforms (Platform Port), by rewriting the `WEAK` function like `pika_hal_platform_xxxx()` in different platforms, it is possible to provide support for different platforms.
- Besides `PikaStdDevice` modules, there are also `Python` modules like `sensor` / `motor`, which are based on `pika_hal`. These modules use `GPIO`, `IIC`, `PWM` and other adapted functions of `pika_hal`, so no additional adaptations are needed besides `pika_hal` to use them.

6.2.4 PikaStdDevice module example

Using the GPIO module as an example, here is the user API defined by PikaStdDevice

```
class GPIO:
    def __init__(self):
        pass

    def init(self):
        pass

    def setPin(self, pinName: str):
        pass

    def setId(self, id: int):
        pass

    def getId(self) -> int:
        pass

    def getPin(self) -> str:
        pass

    def setMode(self, mode: str):
        pass

    def getMode(self) -> str:
        pass

    def setPull(self, pull: str):
        pass

    def enable(self):
        pass

    def disable(self):
        pass

    def high(self):
        pass

    def low(self):
        pass

    def read(self) -> int:
        pass
```

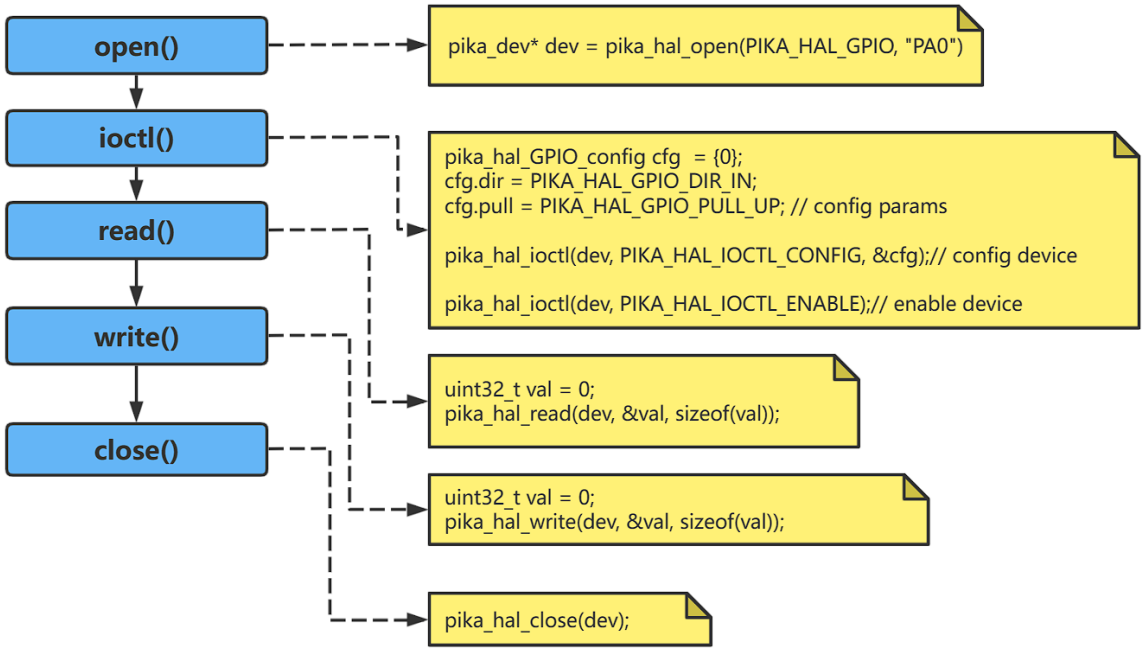
The sample code for the PikaStdDevice module is under the https://gitee.com/Lyon1998/pikascript/tree/master/examples/Device path. The machine module in the example is a simple renaming of the PikaStdDevice module.

6.2.5 `pika_hal` device abstraction layer

Design philosophy

- Efficient. Pure C implementation, with streamlined internal links.
- Standard. `linux`-like design, all types of device operations have and only have 5 standard file-like APIs: `open()`, `close()`, `write()`, `read()`, `ioctl()`.

Programming model



All devices follow the linux-like file programming model, all types of devices use the `pika_dev` structure as a device handle, and all types of devices have and only have the following five control APIs.

`open()`

- Overview

  The `open()` function is used to open a device and is called first.

- The function prototype

```
pika_dev* pika_hal_open(PIKA_HAL_DEV_TYPE dev_type, char* name);
```

- Parameter

close()

- Overview

  The `close()` function is used to close a device, and is called last, to avoid memory leaks.

- Function prototype

```
int pika_hal_close(pika_dev* dev);
```

- Parameters

ioctl()

- Overview

  The `ioctl()` function is used to control the device, including

  - config
  - enable
  - disable - disable

- Function prototypes

```
int pika_hal_ioctl(pika_dev* dev, PIKA_HAL_IOCTL_CMD cmd, ...) ;
```

- Parameter

read()

- Overview

  The `read()` function is used to read data from a device.

- Function prototype

```
int pika_hal_write(pika_dev* dev, void* buf, size_t len);
```

- Parameters

write()

- Overview

  The `write()` function is used to write data to a device.

- The function prototype

```
int pika_hal_write(pika_dev* dev, void* buf, size_t len);
```

- Parameter

Driver adaptation

Adapt pika_hal to the platform by rewriting the following pika_hal_platform_XXXX prefixed WEAK functions for the device, where XXXX is the device type name, such as GPIO, PWM, etc.

```
PIKA_WEAK int pika_hal_platform_XXXX_open(pika_dev* dev, char* name);
PIKA_WEAK int pika_hal_platform_XXXX_close(pika_dev* dev);
PIKA_WEAK int pika_hal_platform_XXXX_read(pika_dev* dev, void* buf, size_t count);
PIKA_WEAK int pika_hal_platform_XXXX_write(pika_dev* dev, void* buf, size_t count);
PIKA_WEAK int pika_hal_platform_XXXX_ioctl_enable(pika_dev* dev);
PIKA_WEAK int pika_hal_platform_XXXX_ioctl_disable(pika_dev* dev);
PIKA_WEAK int pika_hal_platform_XXXX_ioctl_config(pika_dev* dev, pika_hal_XXXX_config*␣
↪cfg);
```

Reference adaptation code.

https://gitee.com/Lyon1998/pikapython/tree/master/package/BLIOT https://gitee.com/Lyon1998/pikapython/tree/master/package/STM32G0 https://gitee.com/Lyon1998/pikapython/tree/master/package/ESP32

Case Tutorial 1 - Adaptation of WIFI devices on ESP32

source link

First, we need to include some necessary header files such as pika_hal.h, esp_wifi.h, esp_event.h, etc. These header files provide the definitions and functions related to pika_hal and esp32.

```
#include "... /pikascript-lib/pikastddevice/pika_hal.h"
#include "esp_event.h"
#include "esp_mac.h"
#include "esp_netif.h"
#include "esp_wifi.h"
#include "freertos/freertos.h"
#include "freertos/event_groups.h"
#include "freertos/task.h"
#include "nvs_flash.h"
```

Then, we define some global variables and constants to record the status and configuration information of WIFI. For example, wifi_started indicates whether WIFI has been started, wifi_sta_connect_requested indicates whether a connection to a WIFI hotspot was requested, wifi_sta_disconn_reason indicates the reason for a failed connection, etc.

```
static volatile pika_bool wifi_started = pika_false;
static volatile pika_bool wifi_sta_connect_requested = pika_false;
static volatile pika_bool wifi_sta_connected = pika_false;
static volatile pika_hal_wifi_status wifi_sta_disconn_reason = pika_false; static␣
↪volatile pika_hal_wifi_status wifi_sta_disconn_reason =
    pika_hal_wifi_status_idle;
static eventgrouphandle_t wifi_event_group;
static esp_netif_t* sta_netif = null;
static esp_netif_t* ap_netif = null;
```

Next, we define a helper function _ip_str2u32 that converts the IP address in string form to a value of type uint32_t. This function iterates over each number in the string and stores it in an array of type uint8_t, then returns the uint32_t value represented by this array.

```
uint32_t _ip_str2u32(char* ip_str) {
    uint32_t ip = 0;
    uint8_t* ip_u8 = (uint8_t*)&ip;
    char* p = ip_str;
    for (int i = 0; i < 4; i++) {
        ip_u8[i] = atoi(p);
        p = strchr(p, '.') ;
        if (p == null) {
            break;
        }
        p++;
    }
    return ip;
}
```

Immediately afterwards, we define an event handler function event_handler to respond to events of different types and IDs and to perform the corresponding actions based on the event data. For example, in the WIFI_EVENT_STA_START event, the esp_wifi_connect function is called if a connection to a hotspot is requested; in the IP_EVENT_STA_GOT_IP event, the wifi_sta_connected is set to PIKA_TRUE and the wifi_sta_disconn_reason is set to PIKA_TRUE. disconn_reason to PIKA_HAL_WIFI_STATUS_GOT_IP, etc.

```
static void event_handler(void* event_handler_arg,
                          esp_event_base_t event_base,
                          int32_t event_id,
                          void* event_data) {
    // ...
}
```

Then, we implement several main device manipulation functions corresponding to turning on, turning off, configuring and controlling the WIFI device. Each of these functions requires passing a pointer to a device object (pika_dev) and returns the corresponding result or error code, depending on the case.

- The `pika_hal_platform_WIFI_open` function is used to initialize the NVS (non-volatile storage), the network interface and the event loop, and to create an event group.
- The `pika_hal_platform_WIFI_close` function is used to deinitialize the NVS, the network interface and the event loop, and to delete the event group.
- The `pika_hal_platform_WIFI_ioctl_config` function is used to configure the WIFI mode, hotspot information, etc. based on the ioctl_config field (type pika_hal_WIFI_config) in the device object. In case of STA mode, the configuration is not supported; in case of AP mode, the esp_wifi_set_config function is called to set the SSID, password, channel, authentication mode and maximum number of connections of the hotspot, etc.
- The `pika_hal_platform_WIFI_ioctl_enable` function is used to start or stop the WIFI. first, the mode of the WIFI is determined according to the mode field in the ioctl_config field, and then the esp_wifi_set_mode function is called to set the mode. If WIFI is not yet started, you also need to register the event handler function, create the default network interface, and call the esp_wifi_start function to start WIFI and set wifi_started to PIKA_TRUE; otherwise, you just need to set the mode.
- The `pika_hal_platform_WIFI_ioctl_disable` function is used to stop or deinitialize WIFI. if WIFI is already started, call the esp_wifi_stop and esp_wifi_deinit functions to stop and deinitialize WIFI and set wifi_started to PIKA_FALSE; otherwise, -1 is returned to indicate an error.
- The `pika_hal_platform_WIFI_ioctl_others` function is used to handle other types of control commands, such as getting the status of the WIFI, whether it is active or not, scanning for nearby hotspots, etc. These commands are specified by the cmd parameter and data is passed or returned by the arg parameter. For example, in the PIKA_HAL_IOCTL_WIFI_GET_STATUS command, the current connection status is determined based

on variables like wifi_sta_connect_requested and wifi_sta_connected and assigned to the pika_hal_wifi_ status variable pointed to by arg. status variable.

6.2.6 Contribute

Please refer to the documentation in the *Contribute to the community -> Contribute module* section of the documentation to post the module you have written.

6.3 PikaStdData data structure

PikaStdData data structure library provides List (list), Dict (dictionary) data structure.

6.3.1 Install

Add the dependency of PikaStdLib to requestment.txt. The version number of PikaStdLib should be the same as the version number of the kernel.

```
PikaStdLib
```

Run pikaPackage.exe

6.3.2 import

Add in main.py

```
#main.py
import PikaStdData
```

6.3.3 class List():

The List class provides the List list function. By creating an object of the List class, a list can be created. Such as:

```
import PikaStdData
list = PikaStdData.List()
```

Methods of the List class

```
    # add an arg after the end of list
    def append(self, arg: any):
        pass

    # get an arg by the index
    def __getitem__(self, i: int) -> any:
        pass

    # set an arg by the index
    def __setitem__(self, i: int, arg: any):
```

(continues on next page)

(continued from previous page)

```
        pass

    # get the length of list
    def len(self) -> int:
        pass
```

Note that the index of the `__setitem__()` method cannot exceed the length of the List. If you want to add members of the list, you need to use the `append()` method.

Use '[]' brackets to index the list

List objects can be indexed using '[]'. `list[1] = a` is equivalent to `list.__setitem__(1, a)`, and `a = list[1]` is equivalent to `a = list.__getitem__(1)`.

Use for loop to iterate over List

List objects support for loop traversal

example:

```
import PikaStdData
list = PikaStdData.List()
list.append(1)
list.append('eee')
list.append(23.44)
for item in list:
    print(item)
```

6.3.4 class Dict():

The Dict class provides the Dict dictionary function, and a dictionary can be created by creating an object of the Dict class. Such as:

```
import PikaStdData
dict = PikaStdData.Dict()
```

Dict class methods

```
    # get an arg by the key
    def __getitem__(self, key: str) -> any:
        pass

    # set an arg by the key
    def __setitem__(self, key: str, arg: any):
        pass

    # remove an arg by the key
    def remove(self, key: str):
        pass
```

Index dictionary using '[]' brackets

Dict objects can be indexed using '[]'. `dict['x'] = a` is equivalent to `dict.set('x', a)` and `a = dict['x']` is equivalent to `a = dict.__getitem__('x')` .

Using a for loop to iterate over a Dict

Dict objects support for loop traversal

example:

```
import PikaStdData
dict = PikaStdData.Dict()
dict['a'] = 1
dict['b'] = 'eee'
dict['c'] = 23.44
for item in dict:
    print(item)
```

6.3.5 class ByteArray(List)

[Note]: The version of PikaStdData requires at least v1.5.3

The ByteArray class provides the ByteArray byte array function. By creating an object of the ByteArray class, a byte array can be created.

Such as:

```
import PikaStdData
bytes = PikaStdData.ByteArray()
```

The ByteArray class inherits from the List class and can use the methods of the List class.

Example:

```
>>> bytes = PikaStdData.ByteArray(b'test')
>>> for byte in bytes:
...     print(byte)
...
116
101
115
116
>>> bytes.append(0xff)
>>> bytes.append(0x0f)
>>> print(bytes[4])
255
>>> print(bytes[5])
15
```

6.4 PikaStdTask multitasking

The PikaStdTask multitasking library provides asynchronous multitasking capabilities of Task (task loop).

6.4.1 Install

Add the dependency of PikaStdLib to requestment.txt. The version number of PikaStdLib should be the same as the version number of the kernel.

```
PikaStdLib
```

Run pikaPackage.exe

6.4.2 class Task():

The Task class provides the task loop function, and a task loop can be created by creating an object of the Task class.

Methods of the Task class

```
import PikaStdData


class Task:
    calls = PikaStdData.List()

    def __init__(self):
        pass

    # regist a function to be called always
    def call_always(self, fun_todo: any):
        pass

    # regist a function to be called when fun_when() return 'True'
    def call_when(self, fun_todo: any, fun_when: any):
        pass

    # register a function to be called periodically
    def call_period_ms(self, fun_todo: any, period_ms: int):
        pass

    # run all registered function once
    def run_once(self):
        pass

    # run all registered function forever
    def run_forever(self):
        pass

    # run all registered function until time is up
    def run_until_ms(self, until_ms: int):
```

(continues on next page)

(continued from previous page)

```
        pass

    # need be overried to supply the system tick
    def platformGetTick(self):
        pass
```

Instructions:

Use the `call_xxx()` method to specify the calling method, and register the function to be executed in the task object.

Use the `run_xxx()` methods to specify how the task loops and execute all functions in the `task` object.

Time-related functions, such as `call_period_ms()` and `run_until_ms()`, need to provide the system clock by creating a new class that inherits from `PikaStdTask`, and then override the `platformGetTick()` method.

Notice:

All registered functions should be **non-blocking**, otherwise the entire task loop will be blocked.

The task loop is not real-time.

Example:

Create a new class that inherits from `PikaStdTask`.

```
# STM32G0.py
class Task(PikaStdTask.Task):
    # override
    def platformGetTick():
        pass
```

Override the `platformGetTick()` method.

```
/* STM32G0_Task.c */

void STM32G0_Task_platformGetTick(PikaObj* self) {
    obj_setInt(self, "tick", HAL_GetTick());
}
```

Python use cases

```
import STM32G0
import PikaPiZero
import PikaStdLib

pin = STM32G0.GPIO()
rgb = PikaPiZero.RGB()
mem = PikaStdLib.MemChecker()

pin.setPin('PA8')
pin.setMode('out')
```

(continues on next page)

(continued from previous page)

```
pin.enalbe()

rgb.init()
rgb.enable()

print('task demo')
print('mem used max:')
mem.max()


def rgb_task():
    rgb.flow()


def led_task():
    if pin.read():
        pin.low()
    else:
        pin.high()


task = STM32G0.Task()

task.call_period_ms(rgb_task, 50)
task.call_period_ms(led_task, 500)

task.run_forever()
```

6.5 PikaDebug debugger

The PikaDebug debugger module provides features such as breakpoint debugging.

6.5.1 Install

Add the dependency of PikaStdLib to requestment.txt. The version number of PikaStdLib should be the same as the version number of the kernel.

```
PikaStdLib
```

Run pikaPackage.exe

6.5.2 class Debuger():

The Debuger class provides the debugger function. By creating an object of the Debuger class, a debugger can be created.

Debuger class methods

```
class Debuger:
    def __init__(self):
        pass

    def set_trace(self):
        pass
```

The `__init__()` method is the method executed when the object is created, and the user does not need to know about it. The `set_trace()` method can place a breakpoint in the code. When the code execution reaches the breakpoint, it will stop and open the `(pika-debug)` terminal. The user can enter commands in the terminal (c : continue running, q : to end debugging), or a python interactive call (`printf(i)`, `i = 10`).

Example:

```
import PikaDebug

pkdb = PikaDebug.Debuger()

i = 0
while i < 10:
    i = i + 1
    print('i:' + str(i))
    # set a breakpoint here
    pkdb.set_trace()
```

Command example:

n: (next) continue to run to the next breakpoint.

q: (quit) to exit debug mode and continue running.

p: (print) print variable, `p i` means print variable `i`.

Interactive run: Directly execute interactive commands, such as `print(i)`, `i = 2`, etc.

```
# Debug logging example
i : 1
(pika-debug) n
i : 2
(pika-debug) n
i : 3
(pika-debug) n
i : 4
(pika-debug) p i
4
(pika-debug) print(i)
4
```

(continues on next page)

(continued from previous page)

```
(pika-debug) i = 2
(pika-debug) n
i : 3
(pika-debug) n
i : 4
(pika-debug) i = 9
(pika-debug) n
i : 10
(pika-debug) i = 2
(pika-debug) n
i : 3
(pika-debug) q
i : 4
i : 5
i:6
i : 7
i :8
i :9
i : 10
```

6.6 PikaCV Image Processing Libraries

PikaCV implements some commonly used image processing algorithms.

6.6.1 Install

1. Add the dependency of PikaCV to requestment.txt.

   ```
   PikaCV
   ```

2. Run pikaPackage.exe

6.6.2 Import

Add in main.py

```
#main.py
import PikaCV as cv
```

6.6.3 class Image():

The Image class is the basis of the PikaCV,and subsequent image processing algorithms are based on the Image class.By creating an object of the Image class,an empty image can be created.Such as:

```
import PikaCV
img = cv.Image()
```

Image write and read

PikaCV can read Jpeg format files and write bmp format files.

```
    def read(self, path: str):
        """Read the image from the specified path,
        Need implement the   `__platform_fopen()`, `__platform_fread()`
          and `__platform_fclose()`"""
        ...

    def write(self, path: str):
        """Write the image to the specified path,
        Need implement the   `__platform_fopen()`, `__platform_fwrite()`
        and `__platform_fclose()`"""
        ...

    def loadJpeg(self, bytes: any):
        """Load the image from bytes"""

    def loadRGB888(self, width: int, height: int, bytes: bytes):
        """Load the image from bytes"""

    def loadRGB565(self, width: int, hight: int, bytes: bytes):
        """Load the image from bytes"""

    def loadGray(self, width: int, hight: int, bytes: bytes):
        """Load the image from bytes"""
```

Image properties

The `size` of an image is `width * hight * channel`

```
    def width(self) -> int:
        """Get the width of the image"""

    def hight(self) -> int:
        """Get the hight of the image"""

    def format(self) -> int:
        """Get the format of the image.
        The format is one of the `ImageFormat` enum,
        like `ImageFormat.RGB888`"""
            def data(self) -> bytes:
```

(continues on next page)

(continued from previous page)

```
        """Get the data of the image"""

    def getPixel(self, x: int, y: int, channel: int) -> int:
        """Get the pixel value of the specified channel.
        For example, if the format of image is `RGB888`,
        the channel `0`, `1`, `2`, means `R`, `G`, `B`,
        and for the format of `GRAY8`, the channel is `0`
        """

    def setPixel(self, x: int, y: int, channel: int, value: int):
        """Set the pixel value of the specified channel.
        For example, if the format of image is `RGB888`,
        the channel `0`, `1`, `2`, means `R`, `G`, `B`,
        and for the format of `GRAY8`, the channel is `0`
        """

    def size(self) -> int:
        """Get the size of the image by bytes"""
```

Image operations

1. `add()` and `minus()` is pixel-by-pixel operation.When the result of the operation exceeds 255, it is classified as 255, and when it is below 0, it is classified as 0.
2. The channel order is RGB in `merge()` and `split()`

6.6.4 class Converter():

Converter class mainly implements the conversion between image formats, and currently Converter supports the following image storage formats and conversions

- means no action* means that an intermediate transformation is required means that it can be converted directly

An example of an image format conversion operation is as follows:

```
cv.Converter.toBMP(img)
```

6.6.5 class Transforms():

The Transforms class mainly implements image transformation algorithms, and the transformation algorithms that have been implemented so far are:

1. `rotateDown(image: Image)`

   This function can rotates the image by 180 degrees.

2. `threshold(image:Image,thre:int,maxval:int,thresholdType:int)`

   This function is used to convert an image to a binary image.

   `thre`:When the value of the thresholdType is 0-4, thre is used as the demarcation threshold for the image

   `thresholdType`: Threshold type, which means as follows:

thresholdType	Corresponding method	Formula
0	THRESH_BINARY	$dst(x,y) = \begin{cases} maxval, & src(x,y) > thre \\ 0, & otherwise \end{cases}$
1	THRESH_BINARY_INV	$dst(x,y) = \begin{cases} 0, & src(x,y) > thre \\ maxval, & otherwise \end{cases}$
2	THRESH_TRUNC	$dst(x,y) = \begin{cases} thre, & src(x,y) > thre \\ maxval, & otherwise \end{cases}$
3	THRESH_TOZERO	$dst(x,y) = \begin{cases} src(x,y), & if \ \ src(x,y) > thre \\ 0, & otherwise \end{cases}$
4	THRESH_TOZERO_INV	$dst(x,y) = \begin{cases} 0, & if \ \ src(x,y) > thre \\ src(x,y), & otherwise \end{cases}$
5	THRESH_OTSU	Use `getOTSUthre()`

3. `setROI(image:Image,x:int,y:int,w:int,h:int)`

   This function is used to select a ROI from an image, the definition of the area is xywh, `x` and `y` represent the upper left vertex coordinates of the region, `w` represents the width of the area, and `h` represents the height of the area.

4. `getOTSUthre(image:Image) -> int`

   This function implements OTSUFor the specific principle, please participate in the paper, the return value of the function is the threshold calculated by the OTSU method.

5. `setOTSU(image:Image)`

   This function uses the OTSU algorithm to binaryize the image.

6. `resize(image:Image,x:int,y:int,resizeType:int)`

   This function implements the scaling of the image, with x and y being the target size of the image

   `resizeType`:The scaling method of the image. 0 represents the nearest neighbor algorithm.

7. `adaptiveThreshold(image:Image,maxval:int,subsize:int,c:int,method:int)`

   `method`An algorithm used to calculate the threshold within a neighborhood. 0 represents mean filtering, 1 represents median filtering.

   `c`:offset value

   `subsize`: Convolutional kernel size

6.6.6 class Filter

The Filter class implements some commonly used image filtering algorithms, and the algorithms that have been implemented so far are:

1. `meanFilter(image:Image,ksizex:int,ksizey:int)`

   Mean filtering, ksizex and ksizey are the size of x and y of the convolutional kernels, respectively. There is currently no support for pads, so the size of the image after filtering equal `W-F+1` when `ksizex=ksizey`.

2. `medianFilter(image:Image)`

   Median filtering, currently only supports convolutional kernels with a size of 3*3.

6.7 requests module declaration

Author: Onceday Date: 20221210

6.7.1 Module basic information

1. Based on webclient.c development, temporarily support the simplest get request and post request.
2. Additional support for simple URL concatenation on get requests.
3. Ability to specify additional request header keywords.
4. The returned data includes the status code, payload length, and payload content.

6.7.2 install

requestment.txt join

```
requests
```

6.7.3 usage

1. Import module first

   ```
   import requests
   ```

2. Then enter the method and url address

   ```
   result = requests.request("GET", "http://pikascript.com/")
   ```

3. If everything succeeds, the result will contain the following information

   ```
   content_length: int        Returns the length of the text content
   text: str                             The text content returned
   state_code: int               get Indicates the status code of the request
   headers: str                       The response header returned
   url: str                               get Indicates the url of the request
   ```

   text is the core returned data. For the request in (2), it can be shown as follows:

   ```
   print(result.text)
   ```

   So that's the web page `http://pikascript.com/`

4. If this `request` Failure, result will be an empty object, so you need to determine whether result is empty.

`request` The available parameters of the module are as follows:

```
request(method: str, url: str, params=None, headers=None, data=None) -> Response:
```

- `method`, optional `GET`POSTThe two most basic operations
- `url`, that is, the standard url field. Note that the length of the field is limited. It is recommended that the field not exceed 2Kb.

- paramsOptional. Used to concatenate parameters after a given url field. The characters are automatically escaped, or you can concatenate parameters in the url manually.
- headersThis parameter is optional. The keyword, such asHost , is used to specify the request header. This parameter is optional.
- dataLoad data used to transmit in POST, note that it is of string type.
- ResponseThe returned response object is returned only when the response to the sent request is successful. Otherwise, it is None

6.7.4 Concatenate URL

The extra support for the get method is to concatenate urls, which also involves some character conversions. Because there are some special characters in the URL that cannot be displayed directly, they must be escaped. It is simple to use, as follows:

```
result = requests.request("GET", "http://pikascript.com/package", params = {"name":"get-
↪test", "id":"23"})
```

The url is concatenated as follows:

```
http://pikascript.com/package?name=get-test&id=23
```

Then use this to send an http request. There is no complicated operation here, but simply concatenate the parameters in the dictionary. If the returned data is displayed below, the result will not be empty until the response is successfully received. Therefore, it is necessary to determine:

```
if result not None:
        print(result.status_code)
        print(result.content_length)
        print(result.text)
```

6.7.5 post port

This interface is primitive, and if you want to upload the data yourself, you need to manually concatenate the content. Here's why:

1. The http protocol is very complex and there is no need to implement it again.
2. Embedded requirements are fixed. Here is a typical use:

```
import requests
print("test")
form_data = '------WebKitFormBoundaryrEPACvZYkAbE4bYB\r\nContent-Disposition: form-data;␣
↪name="file"; filename="test_file.txt"\r\nContent-Type: text/plain\r\n\r\nhello,␣
↪pikascript!\r\n------WebKitFormBoundaryrEPACvZYkAbE4bYB\r\nContent-Disposition: form-
↪data; name="id"\r\n\r\n1670666272201\r\n------WebKitFormBoundaryrEPACvZYkAbE4bYB\r\
↪nContent-Disposition: form-data; name="uploadFileNum"\r\n\r\n1\r\n------
↪WebKitFormBoundaryrEPACvZYkAbE4bYB--\r\n'

print(form_data)
header = {"Content-Type": "multipart/form-data; boundary=----
↪WebKitFormBoundaryrEPACvZYkAbE4bYB"}
```

(continues on next page)

(continued from previous page)

```
a = requests.request("POST", "http://pikascript.com/upload", headers=header, data=form_
↪data)

if a not None:
        print(a.headers)
        print(a.content_length)
        print(a.text)
```

The normal output is as follows:

```
HTTP/1.1 200 OK
309
{"files":{"file":[{"fieldName":"file","originalFilename":"test_file.txt","path":"html/
↪upload/pJwNgC8fobWOLN_l9qmAk-Oi.txt","headers":{"content-disposition":"form-data;␣
↪name=\"file\"; filename=\"test_file.txt\"","content-type":"text/plain"},"size":18}]},
↪"fields":{"id":["1670666272201"],"uploadFileNum":["1"]}}
```

Here's an explanation of the above behavior:

1. First specify additional header keywords, `Content-Type` indicates the type of load, `multipart/form-data` is a common form type, and `boundary` specifies the dividing line between different parts of the form.

   ```
   Content-Type:multipart/form-data; boundary=----WebKitFormBoundaryrEPACvZYkAbE4bYB
   ```

   `----WebKitFormBoundaryrEPACvZYkAbE4bYB` Is used to split the load content, this is simply some random characters, so can be mixed with some identifiers `WebKitFormBoundaryr` Inside.

You will notice that this separator may duplicate the content of the transmission! Yes, it could be repeated, which would cause the server to fail parsing and then have to pass it again. So it's a random string every time, and the probability of repeating it many times in a row is very low. The above form data is the encoded string, which contains three kinds of data:

1. File name, file type, and file content "hello, pikascript!" .
2. The mapping ID.
3. Number of uploaded files.

The POST key requires the following two request keywords:

```
header buffer:POST http://pikascript.com/upload HTTP/1.1
Content-Type:multipart/form-data; boundary=----WebKitFormBoundaryrEPACvZYkAbE4bYB
Content-Length: 408
```

For a post, the request header is as simple as the above. 3. The return code is 200, indicating that the post request was successful. Of course, the post response also carries some information about the uploaded content.

The most direct post transmission, only need the following call can.

```
a = requests.request("POST", "http://pikascript.com/upload", data=binary_data)
```

This transfers binary data, which is populated by default with the following:

```
Content-Type: application/octet-stream
Content-Length: (sizeof(data))
```

But this requires server corresponding analytical support, it is obvious that `http://pikascript.com/upload` cannot parse the data.

6.7.6 Running process

The entire request code was developed based on webclient with simple changes. When the following statement is run, the following process actually takes place:

```
result = requests.request("GET", "http://pikascript.com/package", params = {"name":"get-
↪test"}, headers = {"Connection":"keep-alive"})
```

- First create a session object and apply for a 4Kb buffer to store the request headers.
- Write 'GET' to buffer.
- write "http://pikascript.com/package" into the buffer,
- Writes the concatenated part of the url 'params = {"name":"get-test"}' into buffer, then fills in any other characters as necessary.
- Writes the specified keyword 'headers = {"Connection":"keep-alive"}' to buffer.
- For POST, additional 'Content-Type' and 'Content-Length' contents are written.
- **This will start resolving URL addresses, such as domain names to actual IP addresses**
- Write the default standard header section keywords, including
  1. `Host: ()`
  2. `User-Agent: PikaPython HTTP Agent`
  3. `Accept: */*`
- **Create a socket connection and start communication** -Send the request header portion before sending the data (for POST). -Then wait to receive data, this time there is a possibility of timeout. -Parses the data, writing content_length, text, header, status_code.

Finally, you can view the above four data through the returned result object.

6.8 PIKA-MQTT libary

6.8.1 `__init__()`

introduce

Instantiate one MQTT Client

args

returned value

give a typical example

```
# Minimalist creation
c = MQTT("broker-cn.emqx.io")
# Create a custom port
c = MQTT("broker-cn.emqx.io", 1111)
# Create a custom clientID
```

(continues on next page)

(continued from previous page)

```
c = MQTT("broker-cn.emqx.io", clientID="pikascript")
# Requires the creation of a user name and password
c = MQTT("broker-cn.emqx.io", username="pikascript123", password="123456")
# The creation of an encrypted mqtt
c = MQTT("broker-cn.emqx.io", 8883, username="pikascript123", password="123456",␣
↪ca=open(cert_file).read())
```

6.8.2 setClientID()

introduce:

Setting clientID overrides the parameters at instantiation time **parameter:**

---

give a typical example

```
c.setClientID("pikascript")
```

6.8.3 setUsername()

introduce

Setting usrname overrides the instantiation parameters

parameter

returned value

give a typical example

```
c.setUsername("pikascript")
```

6.8.4 setPassword()

introduce

Setting password overrides the parameter used during instantiation

parameter

returned value

give a typical example

```
c.setPassword("pikascript")
```

6.8.5 setVersion()

introduce

Setting the mqtt version overrides the parameter at instantiation time

parameter

returned value

give a typical example

```
# choosable "3.1" "3.1.1"
c.setVersion("3.1")
```

6.8.6 setCa()

introduce

Setting the ssl certificate overrides the parameter during instantiation Once this parameter is in effect, an ssl connection is forced

parameter

returned value

give a typical example

```
c.setCa(open(cert_file).read())
```

6.8.7 setKeepAlive()

introduce

Setting the keepalive time overrides the parameter at instantiation time

parameter

returned value

give a typical example

```
# cgs unit
c.setKeepAlive(120)
```

6.8.8 setWill()

introduce

testamentary

parameter

returned value

give a typical example

```
c.setWill("/device/will", "{"name":"pikascript"}")
```

6.8.9 setDisconnectHandler()

introduce

Set the disconnection callback

parameter

returned value

give a typical example

```
def disconnect_cb():
    print("mqtt disconnect")

c.setDisconnectHandler(disconnect_cb)
```

6.8.10 connect()

introduce

connect to server

parameter

null

returned value

give a typical example

```
c.connect()
```

6.8.11 disconnect()

introduce

Disconnect server

parameter

null

returned value

give a typical example

```
c.disconnect()
```

6.8.12 subscribe()

introduce

Subscribe to a subject

parameter

returned value

give a typical example

```
def sub_cb(evt):
    print(evt.msg, evt.topic)

c.subscribe("/topic/sub", sub_cb)
```

6.8.13 unsubscribe()

introduce

Unsubscribe to a topic

parameter

returned value

give a typical example

```
c.unsubscribe("/topic/sub")
```

6.8.14 listSubscribeTopics()

introduce

Lists the topics to which you are currently subscribed

parameter:

null

returned value

example:

```
t = c.listSubscribeTopics()
print(t)
```

6.8.15 publish()

introduce:

publish the message

parameters:

return values

example

```
c.publish("/topic/pub", 0, "{"msg":"hello pikascript"}")
```

6.8.16 Attachment 1: Error code

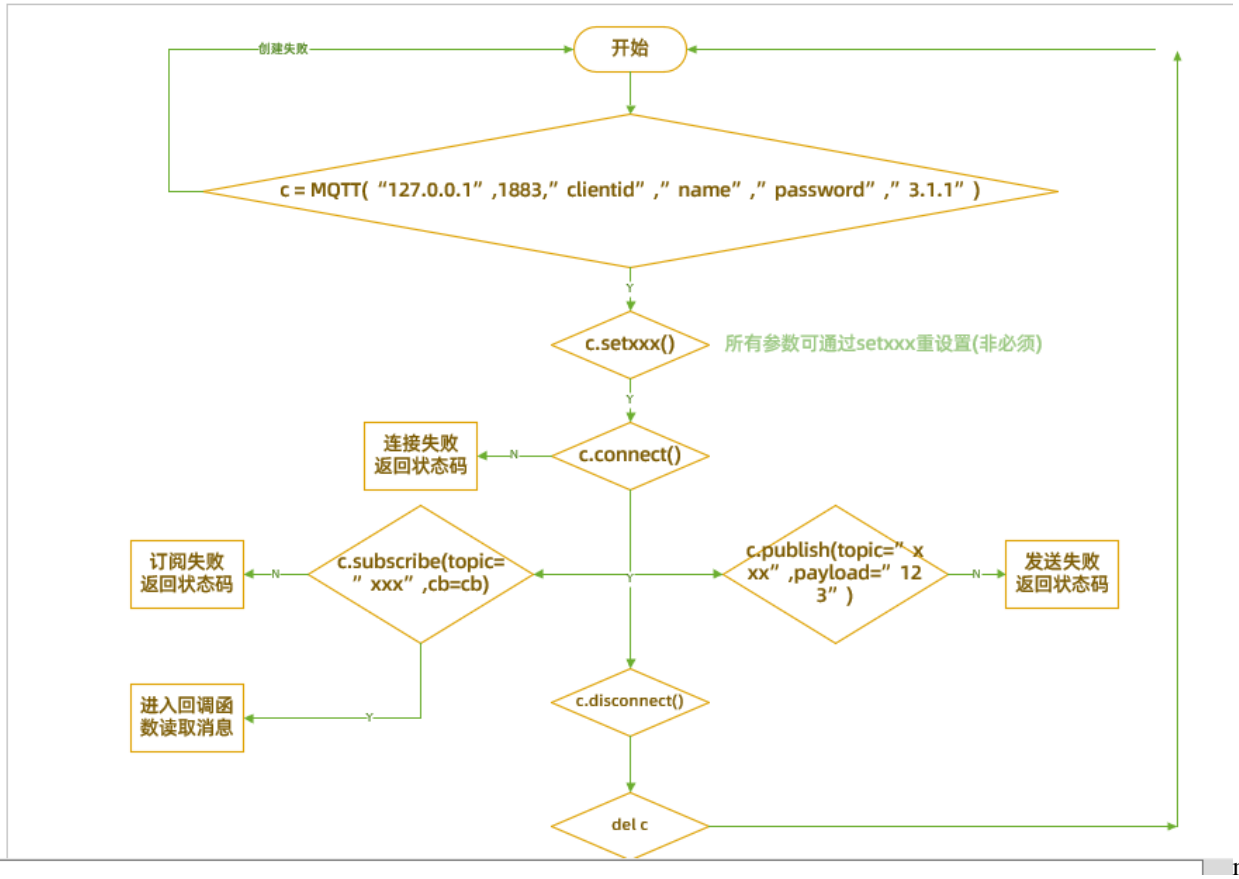
6.8.17 Second:Comprehensive example

```
# coding=utf-8

def sub_test1_cb(evt):
    print("sub test1 message:", evt.msg)


def sub_test2_cb(evt):
    print("sub test2 message:", evt.msg)


def sub_test3_cb(evt):
    print("sub test3 message:", evt.msg)


def disconnect_cb():
```

(continues on next page)

(continued from previous page)

```
    print("mqtt disconnect")


def test():
    # MQTTS
    c = MQTT("broker-cn.emqx.io", 8883,  clientID="pikaone", username="pikascript123",␣
↪password="123456", ca=open("/ca.crt").read())

    # Test a mentary
    c.setWill("/device/will", "{"name":"pikascript"}", 0, True)

        # Set the dis connection callback
        c.setDisconnectHandler(disconnect_cb)

    # connect to server
    result = c.connect()
    if result == 0:
        print("connect success!")
    else:
        print("connect faild id={}".format(result))
        return

    # Subscribe to a subject
    c.subscribe("/pikascript/test1", sub_test1_cb, 0)
    c.subscribe("/pikascript/test2", sub_test2_cb, 1)
    c.subscribe("/pikascript/test3", sub_test3_cb, 2)

    print("list subscribe topics:")
    print(c.listSubscribeTopics())

    print("start publish")

    # Send topic message
    c.publish("/pikascript/test1", "{'msg'}:'hello from test1'", 0)
    c.publish("/pikascript/test2", "{'msg'}:'hello from test2'", 1)
    c.publish("/pikascript/test3", "{'msg'}:'hello from test3'", 2)

    print("end publish")

    # Discount
    result = c.disconnect()
    if result == 0:
        print("disconnect success!")
    else:
        print("disconnect faild id={}".format(result))
        return

# run
test()
```



flow chart mqtt

CHAPTER

SEVEN

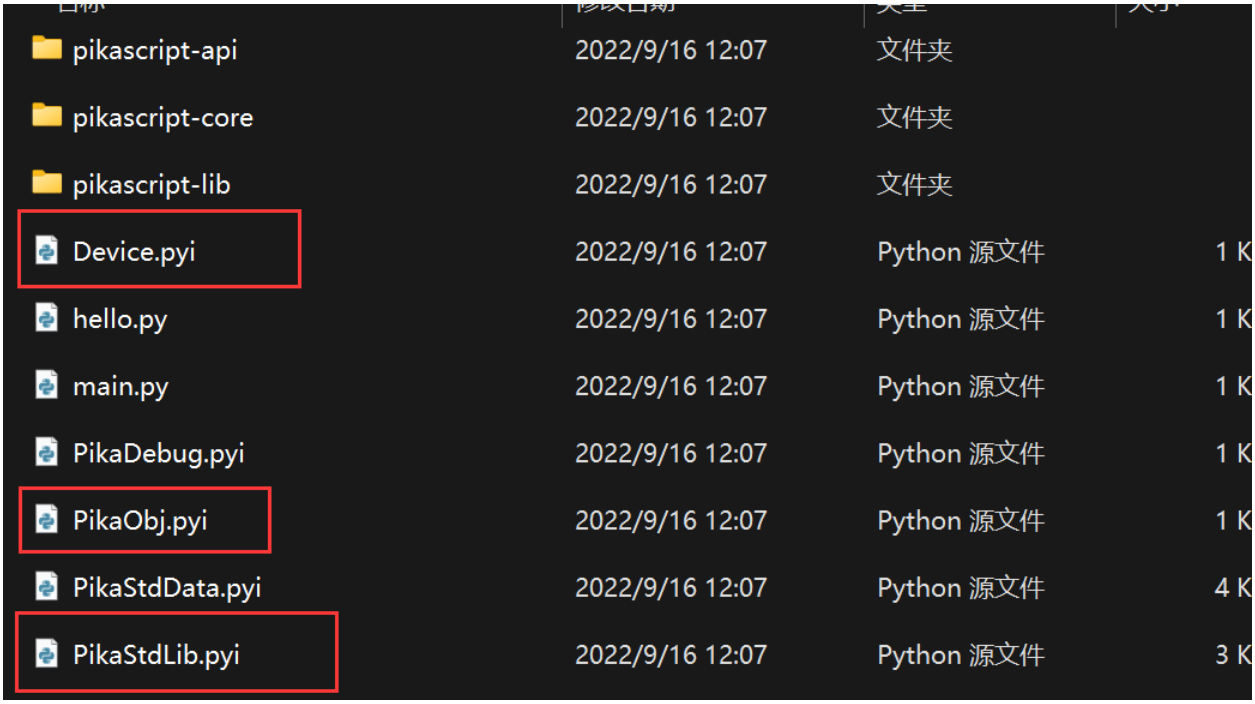
C MODULE - BIND C CODE TO PYTHON MODULE

7.1 PikaPython C module overview

We still use keil's simulation project as an example, if you don't get the simulation project yet, please refer to 1. Three minutes to get started quickly

7.1.1 PikaPython module and module interface

We open the pikascript folder and find that in addition to main.py, there are Device.pyi, PikaObj.pyi and PikaStdLib.pyi in the root of the folder, which correspond to three PikaPython **C modules** (class package), each .pyi file itself is called the **module interface** (package interface). A C module can contain several classes that are more related.



Each PikaPython **C module** consists of two parts: **module interface** and **module implementation** (package implement). Let's start by opening Device.pyi to see the contents, we will call Device.pyi the **Device module interface** in the subsequent documentation. Here is the entire contents of Device.pyi.

```
# Device.pyi
```

(continues on next page)

(continued from previous page)

```
class LED:
    def on(self):
        pass
    def off(self):
        pass

class Uart:
    def send(self, data:str):
        pass
    def setName(self, name:str):
        pass
    def printName(self):
        pass
```

As you can see, there are two classes defined in Device.pyi using pyhon standard syntax, the `LED` class and the `Uart` class.

The LED class defines two methods, the `on()` method and the `off()` method, while the `Uart` class defines the `send(data:str)` method, the `setName(name:str)` method, and the `printName()` method.

As you can see, all these methods have the feature that instead of being **definitions** of methods, they are **declarations** (annotations) of methods, because all method implementations are passed out and none of them are written for implementation. And the method's entry parameters are all with **type declarations**. For example, `data:str` means a `data` parameter with the type `str`, i.e. a string type.

This is because the module implementation of this module is written in C, i.e. the C modules of PikaPython are written with declarations in python syntax and implementations in C. PikaPython's module development is a **hybrid programming** technique for **interface-oriented** programming.

However, when using an existing module, it is not necessary to know the module implementation, but only the module interface, in order to use the module.

7.1.2 Importing and calling modules

Let's see how to use this module.

Let's open main.py in the project, see the name, this file is the entry file for PikaPython.

The content of main.py is as follows

```
# main.py
import Device
import PikaStdLib

led = Device.LED()
uart = Device.Uart()
mem = PikaStdLib.MemChecker()

print('hello wrold')
uart.setName('com1')
uart.send('My name is:')
uart.printName()
print('mem used max:')
mem.max()
```

(continues on next page)

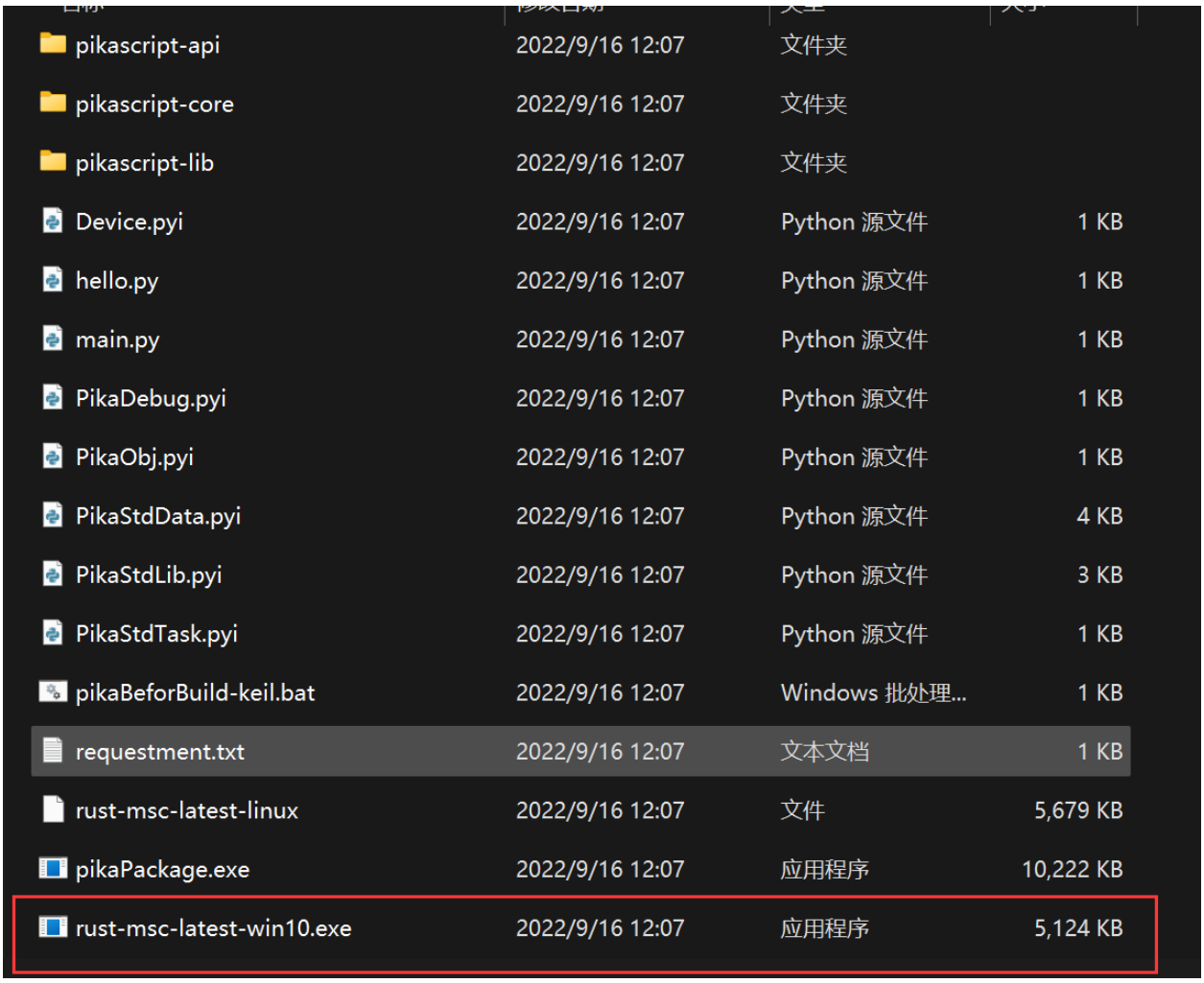
(continued from previous page)

```
print('mem used now:')
mem.now()
```

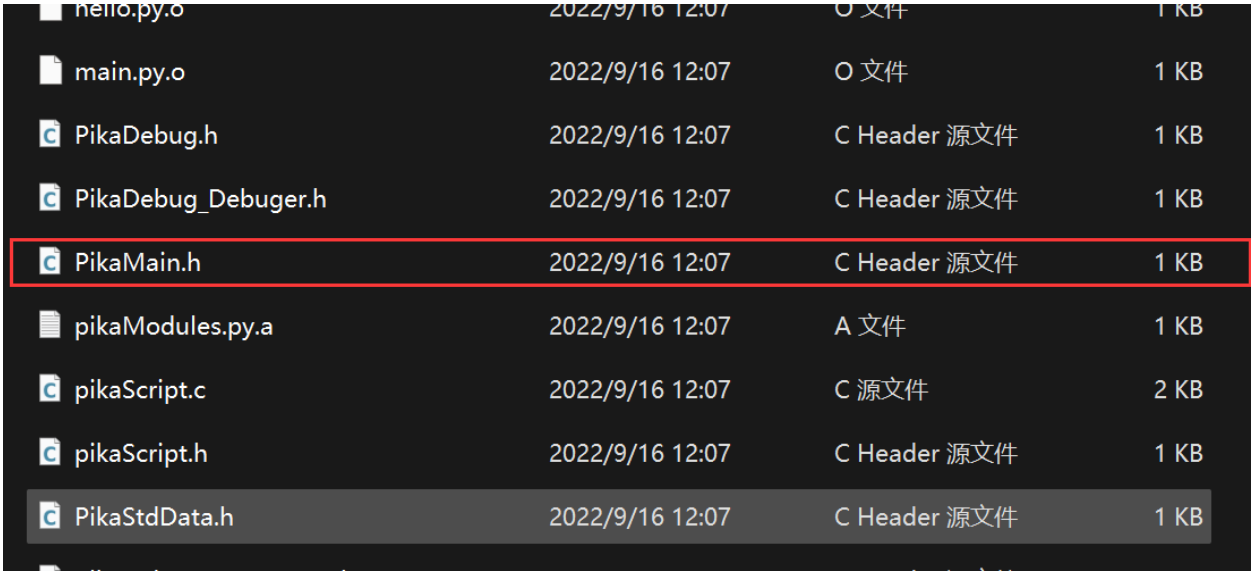
Importing an already written C module is very simple, for example, to import the Device module, you just need to `import Device`, and note that all .py and .pyi files should be placed in the root directory of the pikascript fileshelf.

The call method uses the form `uart.setName('com')`, which is standard Python syntax and does not need much introduction.

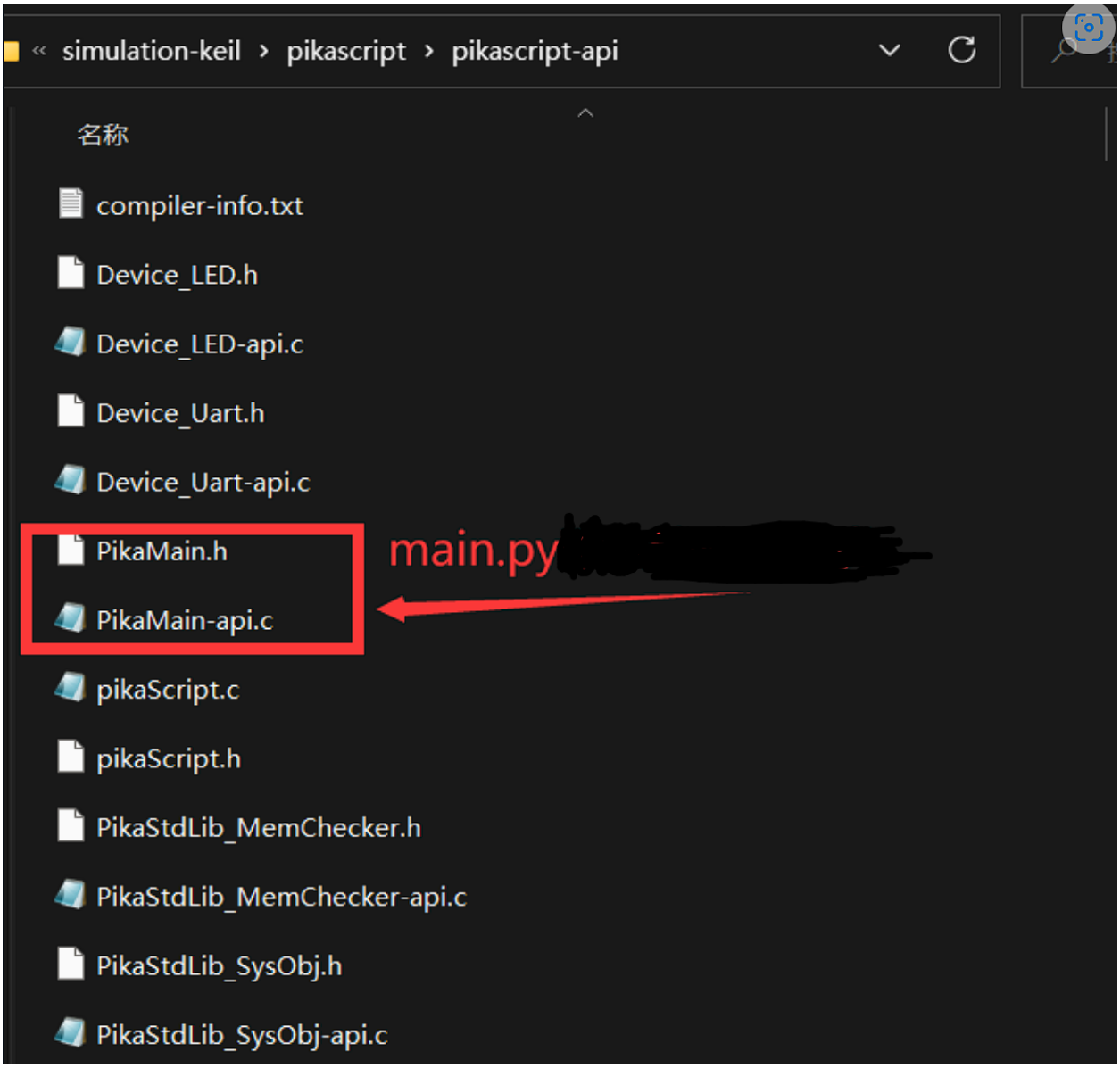
After writing the module calls in main.py, double-click on rust-msc-v0.5.0.exe to pre-compile the PikaPython project, the pre-compiled output file is in the pikascrip-api folder.



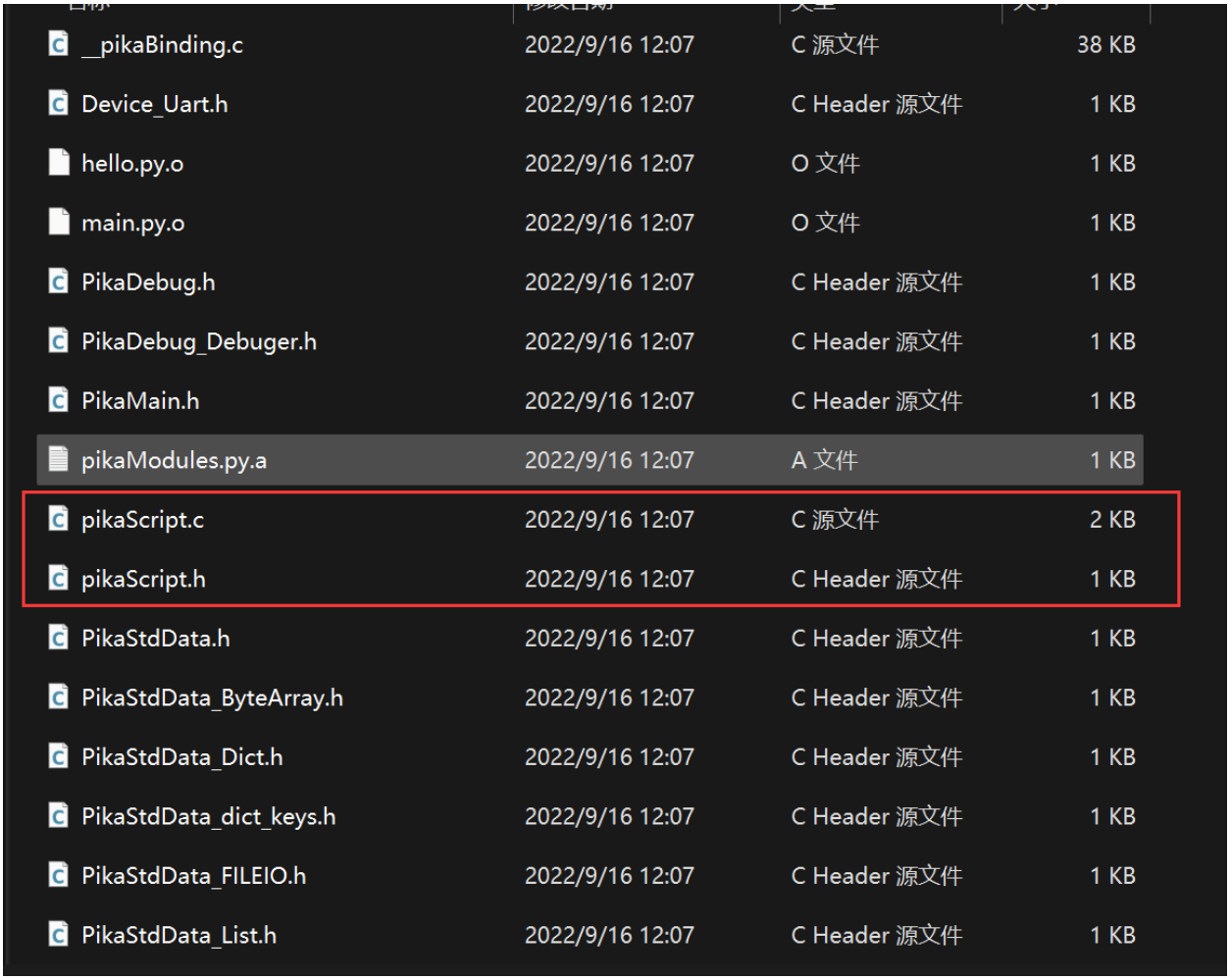
The pika pre-compiler generates .h declaration files for the imported modules. The filenames start with the module name and each class corresponds to one .h file.



And PikaMain.h correspond to a special class that is the main PikaPython class, compiled from main.py.



pikaScript.c and pikaScript.h, on the other hand, are initialization functions compiled from main.py. When the initialization functions are run, the startup script is automatically executed.



In the current main.py, the startup script is written in the outermost method call, which is:

```
led = Device.LED()
uart = Device.Uart()
mem = PikaStdLib.MemChecker()

print('hello wrold')
uart.setName('com1')
uart.send('My name is:')
uart.printName()
print('mem used max:')
mem.max()
print('mem used now:')
mem.now()
```

The compiled pikaScriptInit() initialization function corresponds to:

```
PikaObj * pikaScriptInit(){
    PikaObj * pikaMain = newRootObj("pikaMain", New_PikaMain);
    obj_run(pikaMain,
            "\n"
            "led = Device.LED()\n"
```

(continues on next page)

(continued from previous page)

```
            "uart = Device.Uart()\n"
            "mem = PikaStdLib.MemChecker()\n"
            "\n"
            "print('hello wrold')\n"
            "uart.setName('com1')\n"
            "uart.send('My name is:')\n"
            "uart.printName()\n"
            "print('mem used max:')\n"
            "mem.max()\n"
            "print('mem used now:')\n"
            "mem.now()\n"
            "\n"
            "\n");
    return pikaMain;
}
```

7.2 PikaPython C module development process

We still use keil's simulation project as an example, if you haven't got the simulation project yet, please refer to 1. Three minutes to get started quickly

7.2.1 New module interface

To write a new module, you first need to write a module interface file, for example, to write a math calculation module Math, the first step is to write Math.pyi.

7.2.2 Writing class interfaces

Now we can create new classes inside Math.pyi. For example, if we want to create a new `Adder` class to implement the relevant addition operations, we can add the Adder class inside Math.pyi.

Then we want Adder to provide addition operations for plastic and floating-point data, so we can add the byInt and byFloat methods.

```
# Math.pyi
class Adder:
    def byInt(self, a:int, b:int)->int:
        pass
    def byFloat(self, a:float, b:float)->float:
        pass
```

Use `...` to replace `pass` is also avaliablefor example:

```
# Math.pyi
class Addr:
    def byInt(self, a:int, b:int)->int:...
    def byFloat(self, a:float, b:float)->float:...
```

The above code defines the `Adder` class and adds two method declarations, `byInt(self, a:int, b:int)->int`, indicating that the method name is `byInt`, the input parameters are `a` and `b`, the type of `a` and `b` are both `int`, and the

return value is also `int`. and the return value is determined by `->int`, which is the standard python syntax for writing with type annotations.

The first argument of a method of a class in python is `self`, which is required by python syntax.

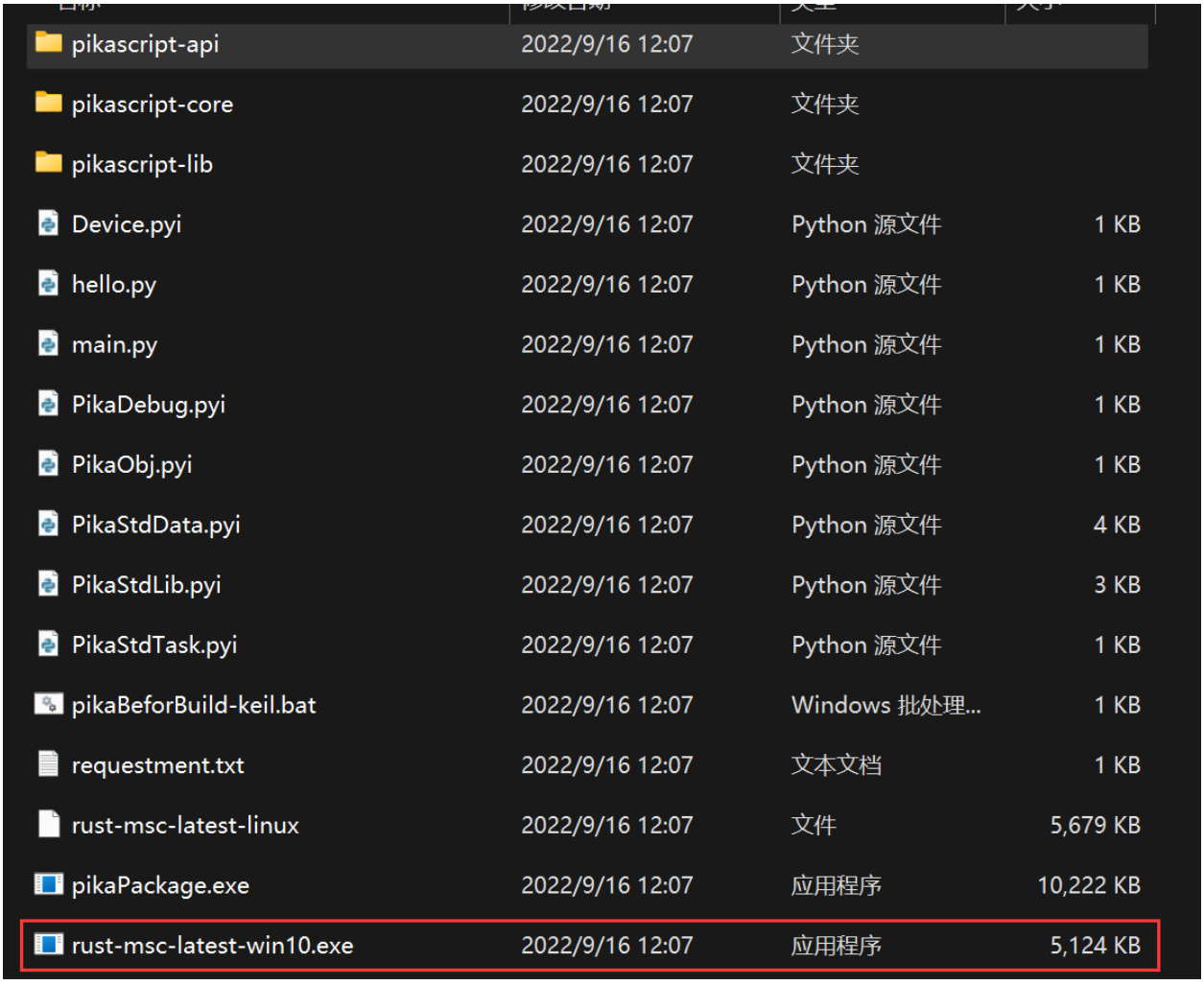
We add a Multiplier class to math.py to implement multiplication, which is written as follows.

```
# Math.pyi
class Multiplier:
    def byInt(self, a:int, b:int)->int:
        pass
    def byFloat(self, a:float, b:float)->float:
        pass
```

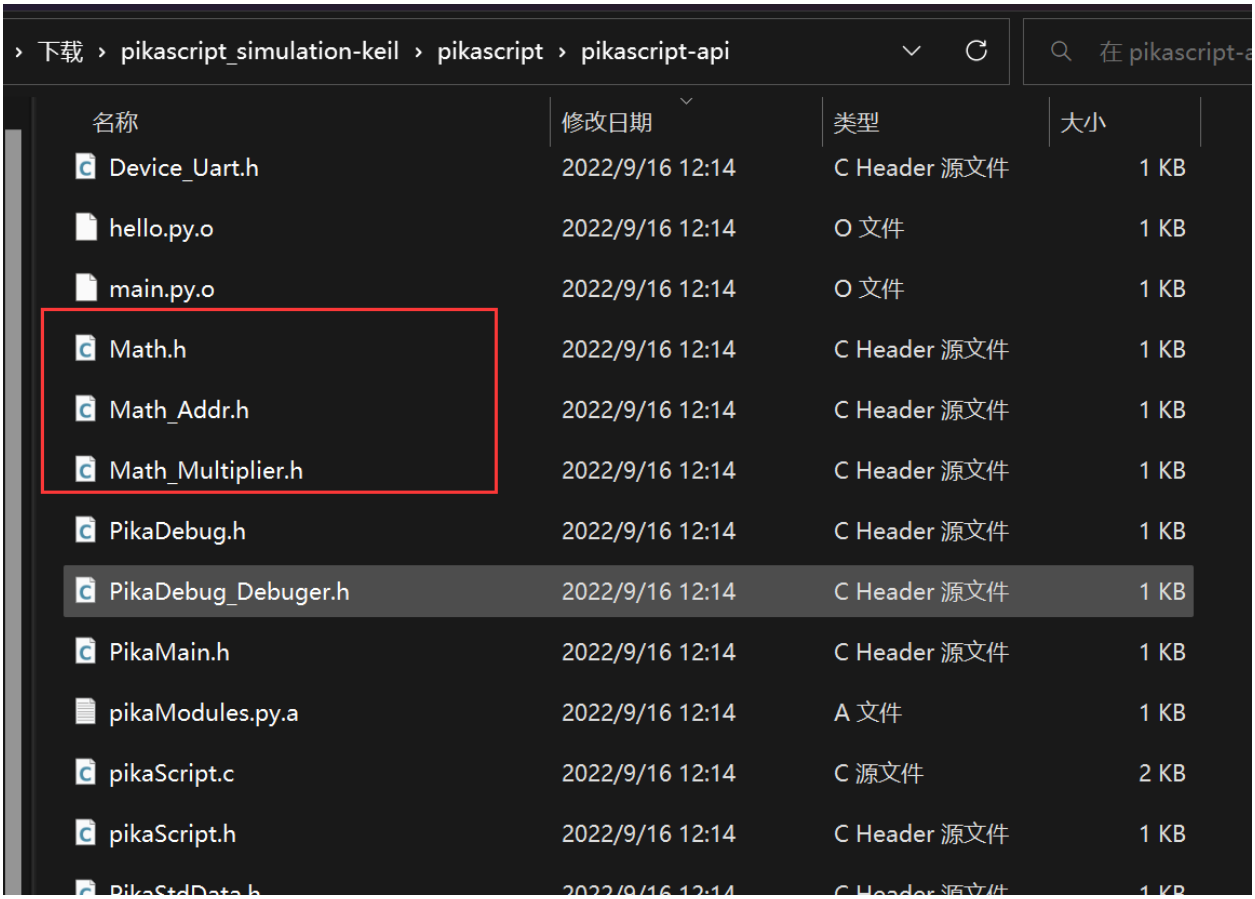
This is the end of the interface. We introduce the Math module in main.py so that the Pika precompiler will go ahead and precompile the Math module.

```
# main.py
import Math
```

Double-click to run the pika precompiler.

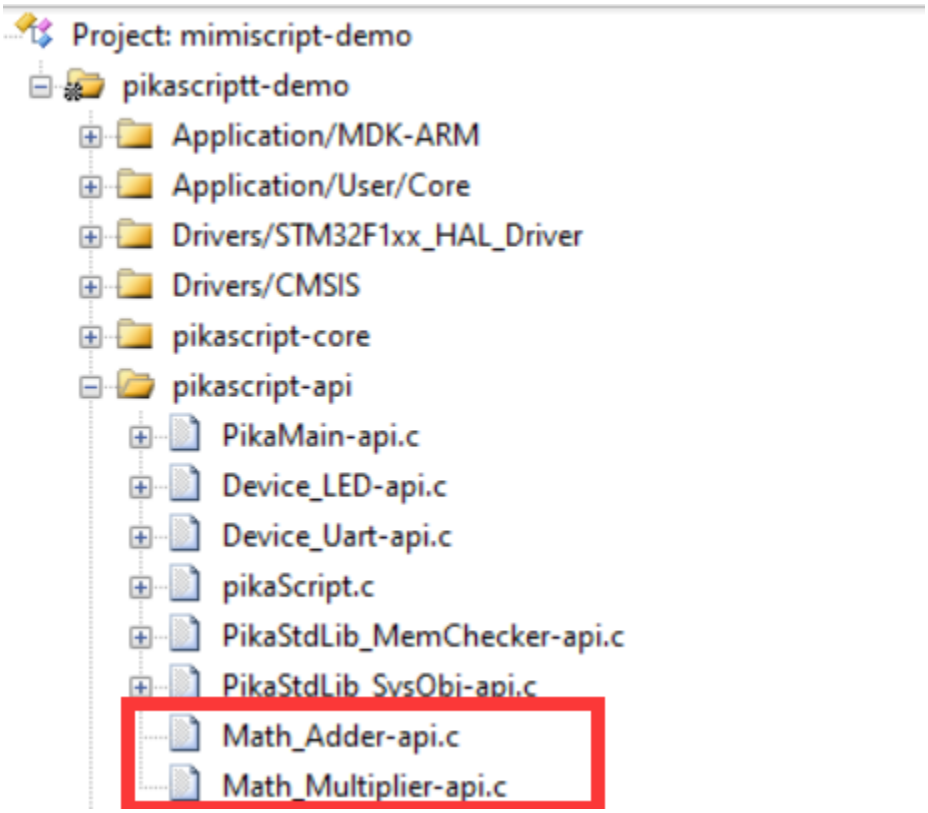


Opening the pikascript-api folder shows that our newly written module interface is ready to be compiled.



7.2.3 Writing the class implementation

Try compiling them.

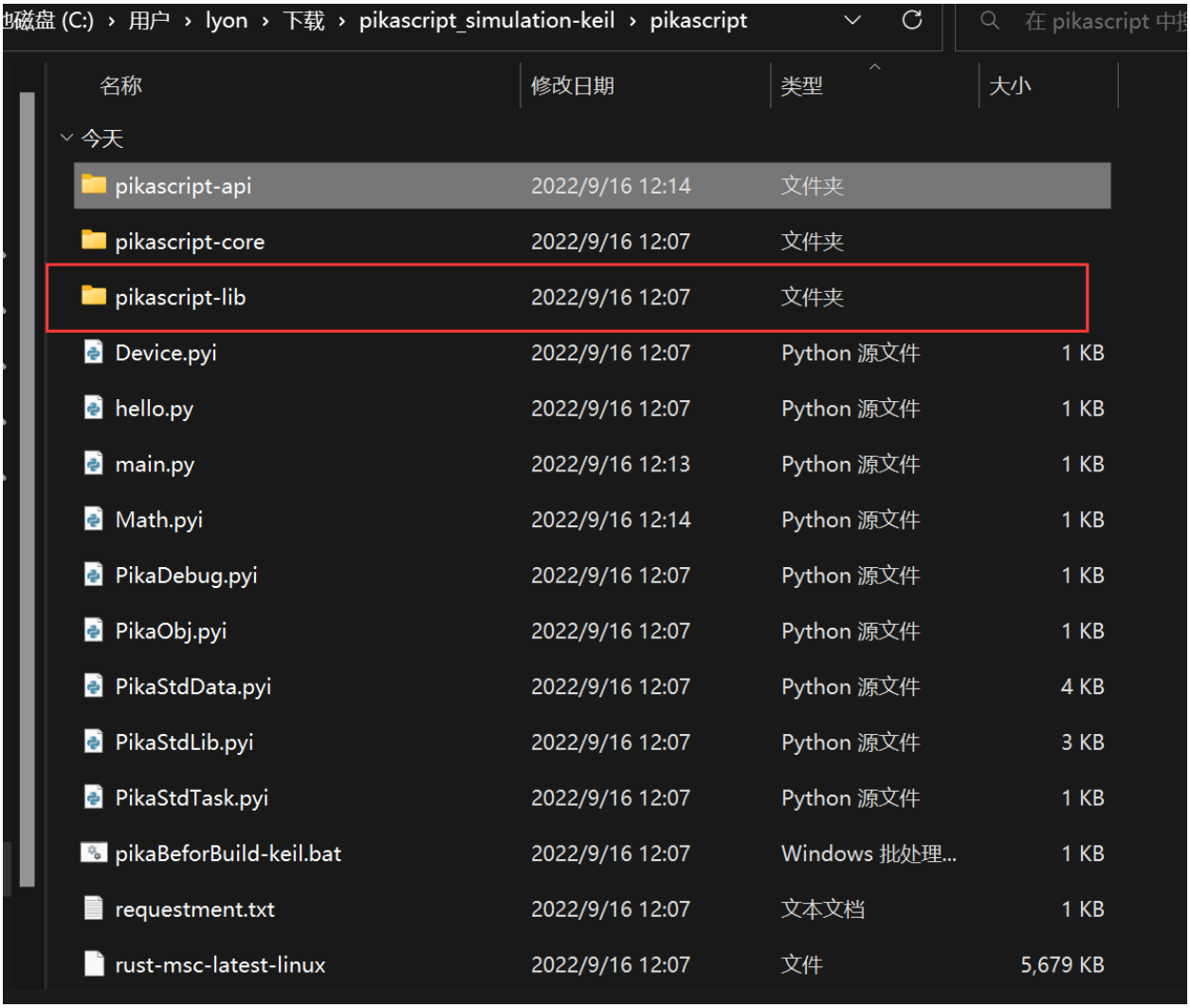


found that the compilation reported an error, suggesting that there are four functions not found in the definition.

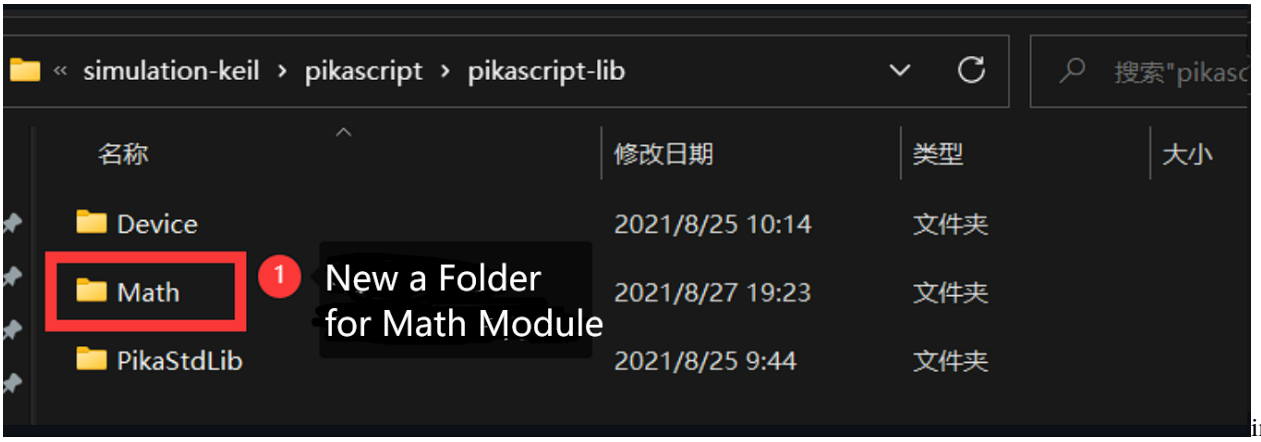
```
linking...
pikascriptt-demo\pikascriptt-demo.axf: Error: L6218E: Undefined symbol Math_Adder_byFloat (referred from math_adder-api.o).
pikascriptt-demo\pikascriptt-demo.axf: Error: L6218E: Undefined symbol Math_Adder_byInt (referred from math_adder-api.o).
pikascriptt-demo\pikascriptt-demo.axf: Error: L6218E: Undefined symbol Math_Multiplier_byFloat (referred from math_multiplier-api.o).
pikascriptt-demo\pikascriptt-demo.axf: Error: L6218E: Undefined symbol Math_Multiplier_byInt (referred from math_multiplier-api.o).
Not enough information to list image symbols.
Not enough information to list load addresses in the image map.
Finished: 2 information, 0 warning and 4 error messages.
"pikascriptt-demo\pikascriptt-demo.axf" - 4 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed:  00:00:03
```

This is normal because we did not write implementations for the classes of the Math module before, and we will write implementations for those classes below.

For the convenience of module management, we put all the implementation files in the pikascript-lib folder.

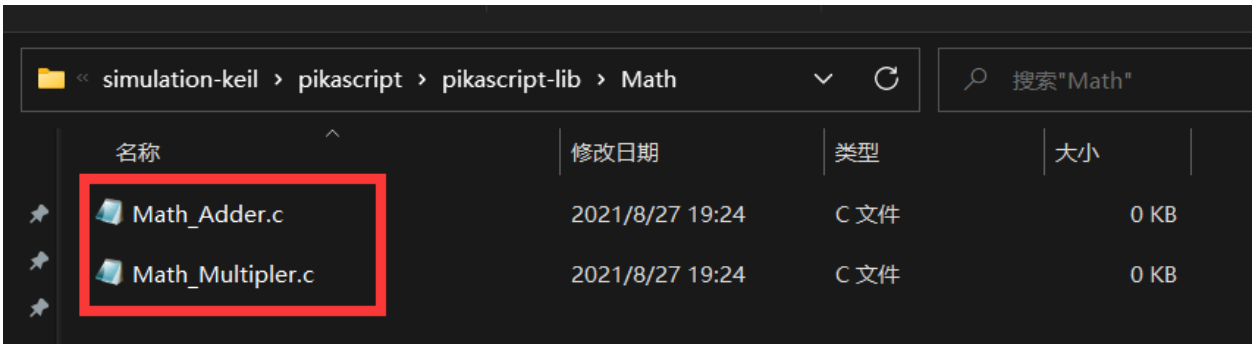


Under the pikascript-lib folder, create a new Math folder to hold the implementation code for the Math module.



image

Then create a new .c file in the Math folder. It is recommended to use the naming scheme "module_class_name.c" to create a new .c file for each class to improve the clarity of the code.



Then we write the method implementation of the class inside these two .c files. So the question arises, how do we know which implementations should be written?

This is easy, we open Math_Multiplier.h and Math_Adder.h to find that the implementation functions we need to write have already been declared.

```c
/* Math_Multiplier.h */
/* ******************************** */
/* Warning! Don't modify this file!
/* ******************************** */
#ifndef __Math_Multiplier__H
#define __Math_Multiplier__H
#include <stdio.h>
#include <stdlib.h>
#include "PikaObj.h"

PikaObj *New_Math_Multiplier(Args *args);

double Math_Multiplier_byFloat(PikaObj *self, double a, doutlb b);
int Math_Multiplier_byInt(PikaObj *self, int a, int b);

#endif
```

```c
/* Math_Adder.h */
/* ******************************** */
/* Warning! Don't modify this file!
/* ******************************** */
#ifndef __Math_Adder__H
#define __Math_Adder__H
#include <stdio.h>
#include <stdlib.h>
#include "PikaObj.h"

PikaObj *New_Math_Adder(Args *args);

double Math_Adder_byFloat(PikaObj *self, double a, double b);
int Math_Adder_byInt(PikaObj *self, int a, int b);

#endif
```

Then we directly implement these four functions in Math_Adder.c and Math_Multipler.c and we're good to go.

```c
/* Math_Adder.c */
#include "pikaScript.h"

double Math_Adder_byFloat(PikaObj *self, double a, double b)
{
        return a + b;
}

int Math_Adder_byInt(PikaObj *self, int a, int b)
{
        return a + b;
}
```

```c
/* Math_Multipler.c */
#include "pikaScript.h"

double Math_Multiplier_byFloat(PikaObj *self, double a, double b)
{
        return a * b;
}

int Math_Multiplier_byInt(PikaObj *self, int a, int b)
{
        return a * b;
}
```

At this point,compile the project again and it will pass.

7.2.4 Test the effect

Let's test our newly written module with the following main.py

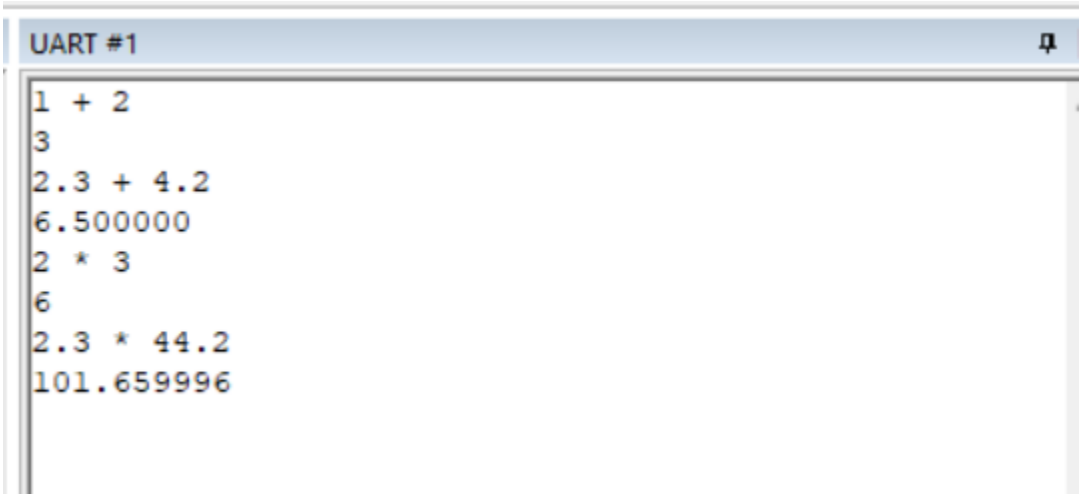
```python
# main.py
import Math

adder = Math.Adder()
muler = Math.Multiplier()

res1 = adder.byInt(1, 2)
print('1 + 2')
print(res1)

res2 = adder.byFloat(2.3, 4.2)
print('2.3 + 4.2')
print(res2)

res3 = muler.byInt(2, 3)
print('2 * 3')
print(res3)

res4 = muler.byFloat(2.3, 44.2)
```

(continues on next page)

(continued from previous page)

```
print('2.3 * 44.2')
print(res4)
```

The result of the run is as follows.



This shows that the module we wrote is working correctly.

7.2.5 Available type annotations

The following table lists all the type declarations supported by PikaPython, and how they correspond to the native types of the C language.

Note

1. `str` is returned as `char*` in c. If the string to be returned is a local variable in the function, it needs to be cached with `obj_cacheStr` to avoid dangling references when it goes out of the function scope, e.g.: `return obj_cacheStr(self, res);`.
2. `bytes` as return value returns `Arg*` in c. This is because `bytes` needs to specify the length and returning `uint8_t*` does not meet the requirement. The correct way to return is: `return arg_newBytes(bytes, len);`.

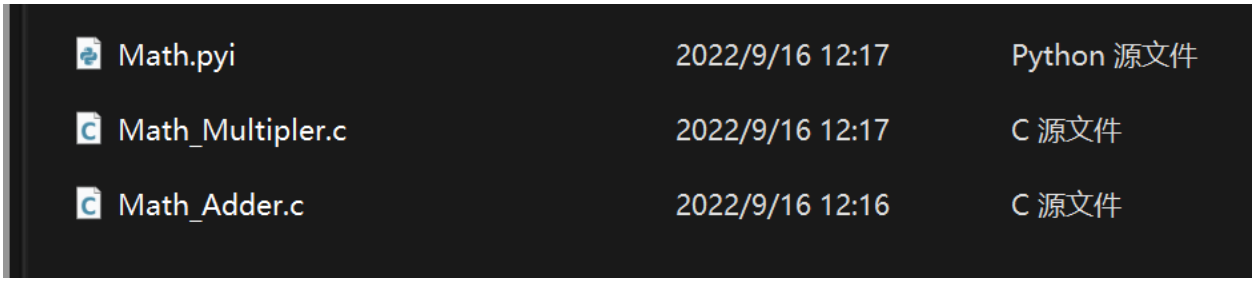
Translated with www.DeepL.com/Translator (free version)

7.2.6 Publishing modules

In the spirit of open source, it is very cool and exciting to publish your own modules.

All you need to do to publish a module is to publish the class interface and class implementation files.

For example, to publish the newly written Math module, you publish the Math.pyi file and the files in pikascript-lib/Math folder.



Please refer to the documentation in the **Participate in Community Contributions** section to distribute the modules you write.

7.3 C module variable parameters

The C module supports variable arguments, just use `*xxx` input arguments, any number of arguments will be packed into the `PikaTuple` data type at the C level, use `pikaTuple_getSize()` to get the number of variable arguments, use `pikaTuple_getArg()` to get `arg` based on the position of the variable arguments. The `pikaTuple_get<Type>` api is also supported to get the value of the specified type directly.

[Note]

- Requires kernel version >= `v1.11.7`
- Variable arguments must be placed after positional arguments

Example.

```
# test.pyi
def vals(a:int, *val):...
```

```
// test.c
void test_vals(PikaObj* self, int a, PikaTuple* val){
    printf("a: %d\n", a);
    for(int i =0; i< pikaTuple_getSize(val); i++){
        Arg* arg_i = pikaTuple_getArg(val, i);
        printf("val[%d]: %d\n", i, arg_getInt(arg_i));
    }
}
```

Output the result:

```
>>> test.vals(1, 2, 3, 4)
a: 1
val[0]: 2
val[1]: 3
val[2]: 4
>>>
```

7.4 C module keyword parameters

C module supports keyword arguments, just use `**xxx` input arguments, any number of arguments will be packed into `PikaDict` data type at C level, use `pikaDict_getArg()` to get `arg` based on keyword. The `pikaDict_get<Type>()` api is also supported to get the value of the specified type directly.

[Note]

- Requires kernel version >= `v1.11.7`
- Keyword arguments must be placed after positional and variable arguments

Example.

```
# test.pyi
def keys(a:int, **keys):...
```

```
// test.c
void test_keys(PikaObj* self, int a, PikaDict* keys){
    printf("a: %d\n", a);
    printf("keys['b']: %d\n", i, pikaDict_getInt(keys, "b"));
    printf("keys['c']: %d\n", i, pikaDict_getInt(keys, "c"));
}
```

Output result:

```
>>> test.keys(1, b=2, c=3)
a: 1
keys['b']: 2
keys['c']: 3
>>>
```

7.5 C module returns List/Dict

7.5.1 List

```
# test.pyi
def test_list()->list: ...
```

```
// test.c
#include "PikaStdData_List.h"
PikaObj* test_test_list(PikaObj* self){
    /* Create list object */
    PikaObj* list = newNormalObj(New_PikaStdData_List).
    /* Initialize the list */
    PikaStdData_List___init__(list).
    /* Create arg */ with api of arg_new<type>.
    Arg* str_arg1 = arg_newStr("aaa");
    /* Add to list object */
    PikaStdData_List_append(list, str_arg1).
    /* destroy arg */
    arg_deinit(str_arg1);
```

(continues on next page)

(continued from previous page)

```
    /* Return the list */
    Returns the list.
}
```

7.5.2 Dict

Note: requires kernel version >= `v1.10.8`.

```
## test.pyi
def test_dict()->dict: ...
```

```
// test.c
#include "PikaStdData_Dict.h"
PikaObj* test_test_dict(PikaObj* self){
    PikaObj* dict = newNormalObj(New_PikaStdData_Dict).
    PikaStdData_Dict___init__(dict).
    Arg* para1 = arg_newInt(1);
    Arg* para2 = arg_newInt(2);
    PikaStdData_Dict_set(dict, "para1", para1).
    PikaStdData_Dict_set(dict, "para2", para2);
    arg_deinit(para1).
    arg_deinit(para2).
    Return dict.
}
```

7.6 C module constants

C modules support adding constants to classes or modules, either using the `val:type` syntax. These constants need to be assigned at initialization time, so the `__init__()` method needs to be defined, e.g.

```
class cJSON:
    cJSON_Invalid: int
    cJSON_False: int
    def __init__(self):...
    ...
```

```
void pika_cjson_cJSON___init__(PikaObj* self) {
    /* const value */
    obj_setInt(self, "cJSON_Invalid", cJSON_Invalid);
    obj_setInt(self, "cJSON_False", cJSON_False);
        ...
}
```

These constants can be used directly without creating an object, i.e. as class properties.

```
print(cJSON.cJSON_Invalid)
```

Note that PikaPython class properties are read-only, and all modifications to class properties are invalid.

7.7 C module initialization

Define `__init__()` function directly in .pyi to perform module initialization, which will be triggered when the module is loaded, PikaPython has a delayed module loading mechanism, `import` will not trigger module loading directly, but only when the module is actually used for the first time.

For example:

```
# test.pyi
def __init__():...
def hello():...
```

```
//test.c
void test___init__(PikaObj* self){
    printf("now loading module test...\n");
}

void test_hello(PikaObj* self){
    printf("hello!\n");
};
```

```
# main.py
import test
print('before run test.hello()')
test.hello()
print('after run test.hello()')
```

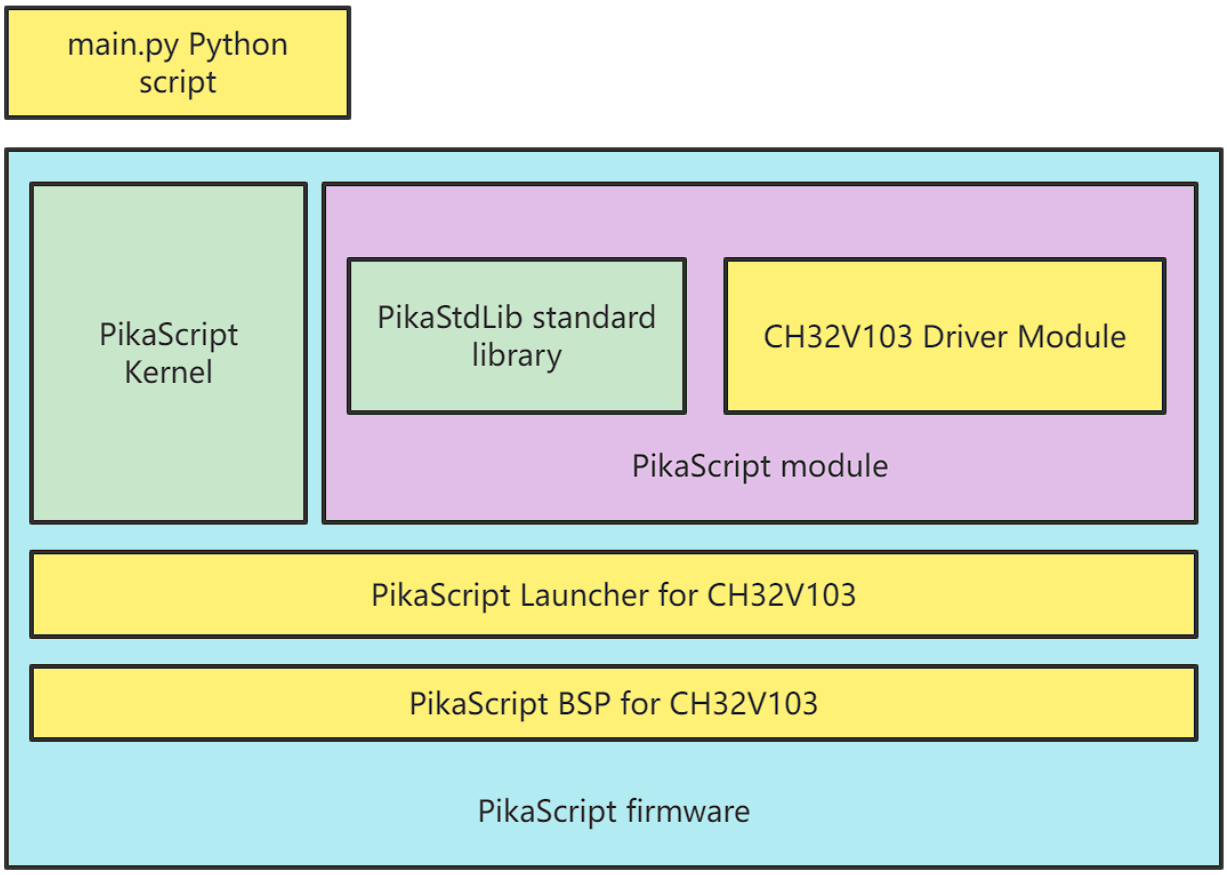
Output.

```
before run test.hello()
now loading module test...
hello!
after run test.hello()
```

7.8 Module clipping

PikaPython module, except PikaStdLib standard library, all other **modules support one-click cropping**

As shown in the CH32V103 driver module in the figure below, the modules that are not needed can be directly cut out. If there are several classes in a module that need to be used, can fine-cutting be done by class? This is also possible, which will be introduced later.

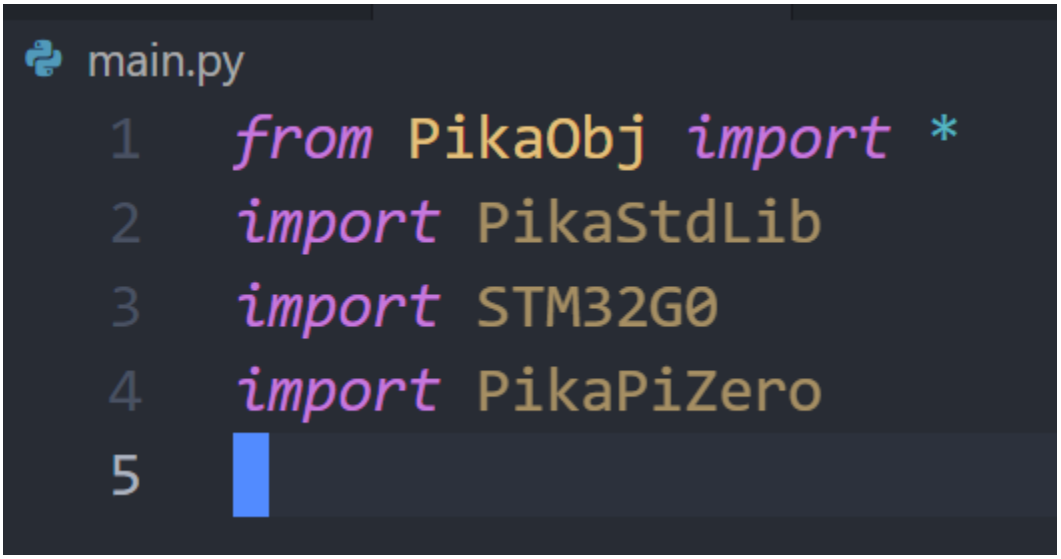


image

7.8.1 Cut by module

It is very simple to trim according to the module. Just delete the import statement in main.py, and the modules that are not imported will be automatically trimmed by the precompiler.

Taking the stm32g030c8 project as an example, the default main.py is as follows:



The first line is to import the base object. The base object is provided by the kernel, does not occupy the module space, and does not need to be clipped. The second line is the standard library and cannot be trimmed. The third row of STM32G0 chip modules and the fourth row of the on-board resource modules of the PikaPiZero development board can be cut. Compile and run, and see that the code size is 48k+3k, about **51K**.

```
linking...
Program Size: Code=48576 RO-data=3324 RW-data=8 ZI-data=7968
```

Unimport the PikaPiZero module

```
main.py
1  from PikaObj import *
2  import PikaStdLib
3  import STM32G0
```

Then precompile and compile the result: It can be seen that the code size has been reduced to **46K**, indicating that the module has been successfully cut.

```
linking...
Program Size: Code=43688 RO-data=3308 RW-data=8 ZI-data=7872
```

Then cancel the import of the STM32G0 module

```
main.py
1  from PikaObj import *
2  import PikaStdLib
3
```

Code size reduced to **36K**

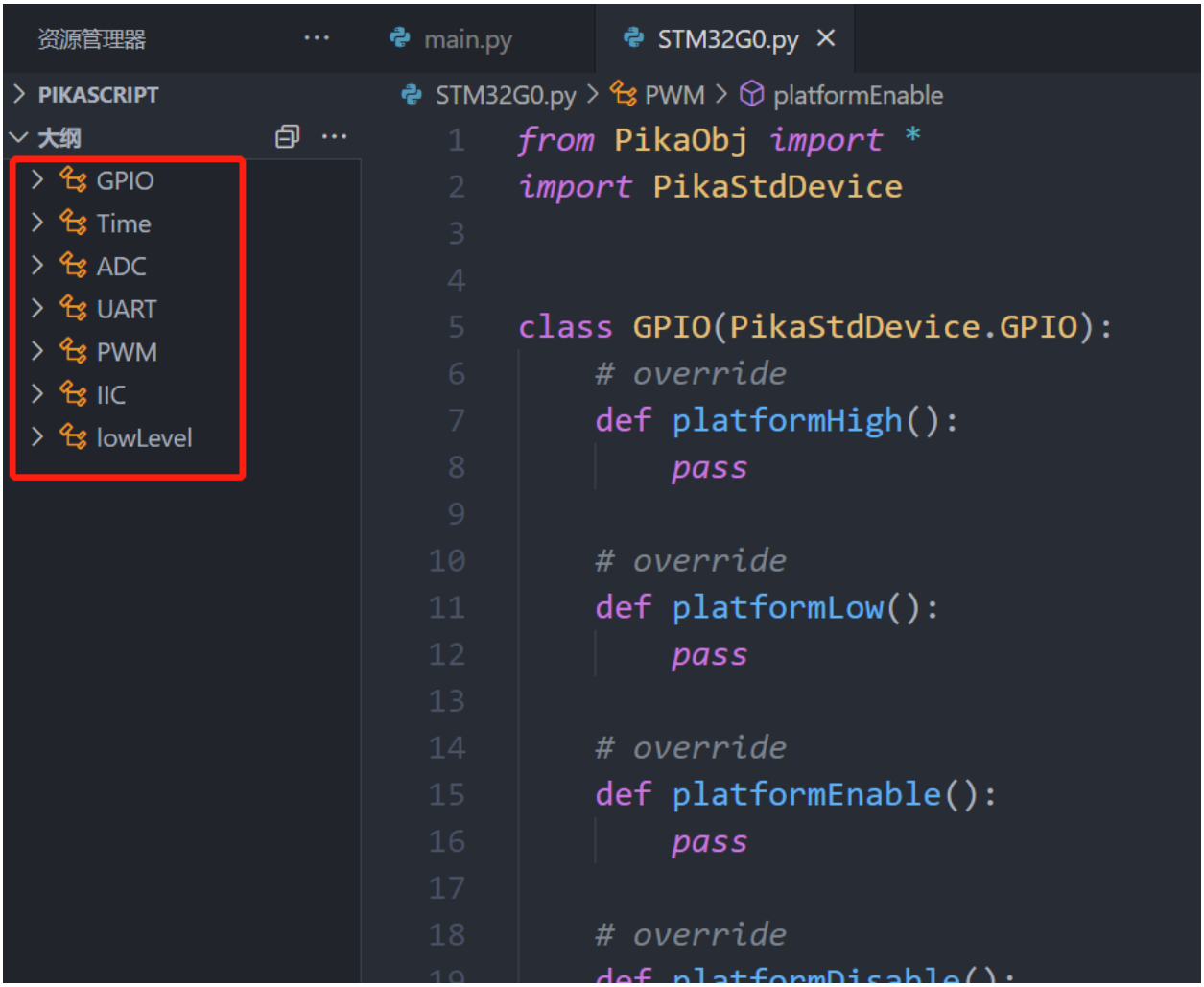
```
linking...
Program Size: Code=33072 RO-data=2700 RW-data=8 ZI-data=7872
FromELF: creating hex file...
```

7.8.2 Cut by class

Using the **inheritance** function of the module, you can fine-tune according to the class. Modules that are **directly imported** in main.py are in a **runtime ready** state, so **all** classes will be added to the project. For modules that are **indirectly imported** by other files, precompiled **can determine which ones will not be used**, so only **used** classes will be added to the project.

In this way, we can **create a new module**, **inherit** the required classes from the modules that need to be used, and then only import the newly created module, you can cut out the classes that are not needed in the module. .

For example, there are 7 classes of GPIO, Time, ADC, UART, PWM, IIC, and lowLevel in STM32G0, and I only use the GPIO class.



You can create a new myDevice module, and then inherit only the GPIO class from STM32G0.

main.py STM32G0.py myDevice.py

myDevice.py > GPIO

```
1  import STM32G0
2
3  class GPIO(STM32G0.GPIO):
4      pass
```

Then change import STM32G0 in main.py to import myDevice

main.py

```
1  from PikaObj import *
2  import PikaStdLib
3  import myDevice
4  import PikaPiZero
5
```

It can be seen that compared to using the complete STM32G0 module, the code size is reduced to **43K**

```
linking...
Program Size: Code=40648 RO-data=3008 RW-data=8 ZI-data=7968
FromELF: creating hex file...
```

CHAPTER

EIGHT

KERNAL API

8.1 Pika object PikaObj

8.1.1 head File

```
#include "PikaObj.h"
```

8.1.2 Overview

- The object API is a series of functions prefixed with **obj_**.
- The Object API provides a series of interfaces for accessing Python objects in C. **Most frequently used in module development.**
- The object API itself is also designed using object-oriented ideas. The first entry parameters of these functions are pointers to the objects to be operated.
- An object consists of two parts: properties and methods, so the object API is also divided into two parts: properties and methods.

8.1.3 type of data

The data type of the object itself is PikaObj, which is used by all Python objects when accessed in C.

```
struct PikaObj_t {
    /* list */
    Args* list;
};
typedef struct PikaObj_t PikaObj;
```

PikaObj internally maintains a parameter table, which contains attribute information, class information, method information, etc. **Be careful not to directly access the parameter table inside PikaObj**, please use the object API to access PikaObj. This is because the object API, as an external interface, is stable for a long time, and the internal implementation will change frequently with the iteration of the kernel code. Directly operating the interior of PikaObj will greatly lose backward compatibility.

8.1.4 Object Properties API

This part of the API provides access to Python object properties.

Attributes of primitive types

PikaObj supports **integer, floating point, pointer, string** four basic types of attributes. Use the set and get methods to read and write properties of an object.

PikaObj objects are **dynamic**, so new properties can be added to the object at any time (the properties of static objects are determined at construction time).

The APIs for primitive type properties are as follows:

```
/* set API */
int32_t obj_setInt(PikaObj* self, char* argPath, int64_t val);
int32_t obj_setPtr(PikaObj* self, char* argPath, void* pointer);
int32_t obj_setFloat(PikaObj* self, char* argPath, float value);
int32_t obj_setStr(PikaObj* self, char* argPath, char* str);
/* get API */
void* obj_getPtr(PikaObj* self, char* argPath);
float obj_getFloat(PikaObj* self, char* argPath);
char* obj_getStr(PikaObj* self, char* argPath);
int64_t obj_getInt(PikaObj* self, char* argPath);
```

Primitive type properties are named as obj_set[Type] and obj_get[Type].

The first input parameter is the object pointer to be manipulated. The second input parameter is attribute name/attribute address.

PikaObj supports object nesting and can access properties of sub-objects. When accessing properties of sub-objects, the second parameter is the property address, and when accessing properties of this object, the second value is property name.

```
// set an Int type arg, the arg name is "a".
obj_setInt(self, "a", 1);
// set an Int type arg for subObjcet , the arg path is "subObj.a".
obj_setInt(self, "subObj.a", 1);
```

The third input parameter of the set method is the written property value, and the return value of the get method is the read property value. The return value of the set method is an error code, 0 means no error occurred.

Generic properties

PikaObj supports generic properties and also provides set and get methods. Input parameters and return values are similar to primitive types.

```
int32_t obj_setArg(PikaObj* self, char* argPath, Arg* arg);
Arg* obj_getArg(PikaObj* self, char* argPath);
```

Generic properties need to be converted to primitive types when used.

Use the following API to determine the current type of a generic property.

```
ArgType arg_getType(Arg* self);
```

Use the following API to convert generic properties to primitive types.

```
int64_t arg_getInt(Arg* self);
float arg_getFloat(Arg* self);
void* arg_getPtr(Arg* self);
char* arg_getStr(Arg* self);
```

Property management

- Determine whether an attribute exists, and the return value is 1 to indicate existence.

```
int32_t obj_isArgExist(PikaObj* self, char* argPath);
```

- Delete an attribute

```
int32_t obj_removeArg(PikaObj* self, char* argPath);
```

The return value is an error code, 0 means success.

8.1.5 Object method API

The object method API is divided into two parts: method registration and method invocation. The **method registration part is proxied by the precompiler**, and the module developer only needs to use the method to call the API.

Method call API

```
void obj_run(PikaObj* self, char* cmd);
```

obj_run is a versatile API that can directly run Python scripts and supports multi-line scripts. The first entry parameter is a pointer to the object, and the second entry parameter is the Python script as a string. Note that when passing in a multi-line script, you should pass in a complete block of code.

8.1.6 Throw an exception

An exception can be thrown using obj_setErrorCode in the module, and the user can customize the exception handling method (continue running or stop running). Throwing an exception is usually used in the method of the C module, just pass in the self object pointer of the current method, and set errCode to non-zero to trigger the exception. The obj_setSysOut method is often used in conjunction with the obj_setErrorCode method to provide debugging information, which will be displayed on the terminal when the exception is triggered.

```
/* set Error Code, if the errCode is not 0, an exaption would be throw out */
void obj_setErrorCode(PikaObj* self, int32_t errCode);
/* print out exaption infomation */
void obj_setSysOut(PikaObj* self, char* str);
```

8.2 Parameter list Args

8.2.1 head File

```
#include "dataArgs.h"
```

8.2.2 Overview

1. The Args parameter table API is a series of functions prefixed with args_.
2. The Args parameter table API is designed using object-oriented ideas. The first entry parameters of these functions are pointers to the parameter table to be manipulated.
3. The Args parameter table uses the **key-value pair (Map)** data model, or **dictionary (Dist)**.
4. A parameter table can contain **any number of** parameters, each parameter is indexed by **parameter name (key)**.
5. The parameters obtained by indexing can be **basic data types (int, float, pointer, string)** or **generic** parameters (Arg).
6. The Args parameter table supports adding, deleting, modifying and searching parameters dynamically.
7. Args parameter table **does not support nesting** (the main difference from the PikaObj attribute).

8.2.3 type of data

The data type of the parameter table is Args.

```
typedef Link Args;
```

The parameter table is internally implemented based on a linked list (Link). **Be careful not to directly access the linked list inside Args**, please use the Args API to access Args. For **maximum backward compatibility**.

8.2.4 Create and destroy the parameter table

1. Create a new parameter table, create a new parameter table from the heap, and return the pointer of the parameter table. **Note that the newly created parameter table needs to be destroyed manually to reclaim the memory. Constantly creating new parameter tables without destroying them can lead to memory leaks.**

[Note] To avoid memory leaks, please develop under docker development environment and ensure sufficient unit tests and memory checks.

```
Args* New_args(Args* args);
```

The parameter passed in when creating a new parameter table is a reserved auxiliary parameter table, which is usually filled with NULL.

1. Destroy the parameter table. When a parameter table is destroyed, all parameters inside the parameter table will also be automatically destroyed.

```
void args_deinit(Args* self);
```

The pointer to the parameter table is passed in, and the parameter table is destroyed.

8.2.5 CRUD API

This part of the API provides the addition, deletion, modification and query of the parameter table.

Basic types of additions, deletions, modifications and inspections

Args parameter table supports **integer, floating point, pointer, string** four basic types of parameters. Use the set and get methods to read and write parameters in a parameter table.

The Args parameter table is **dynamic**, so new parameters can be added to the parameter table at any time.

The API for primitive type properties is as follows, which is similar to the parameter API for objects, but **does not support nesting**:

```
/* set API */
int32_t args_setInt(Args* self, char* name, int64_t int64In);
int32_t args_setFloat(Args* self, char* name, float argFloat);
int32_t args_setPtr(Args* self, char* name, void* argPointer);
int32_t args_setStr(Args* self, char* name, char* strIn);

/* get API */
int64_t args_getInt(Args* self, char* name);
float args_getFloat(Args* self, char* name);
void* args_getPtr(Args* self, char* name);
char* args_getStr(Args* self, char* name);
```

Primitive type attributes are named args_set[Type] and args_get[Type].

1. The first input parameter is the pointer to the parameter table to be manipulated.
2. The second input parameter is the parameter name
3. The third input parameter of the set method is the written parameter value, and the return value of the get method is the read parameter value.
4. The return value of the set method is an error code, 0 means no error occurred.

Generic parameters

args supports generic parameters and also provides set and get methods. Input parameters and return values are similar to primitive types. args_getType can get the type of the argument.

```
int32_t args_setArg(Args* self, Arg* arg);
Arg* args_getArg(Args* self, char* name);
ArgType args_getType(Args* self, char* name);
```

Generic parameters need to be converted to primitive types when used.

Use the following API to determine the current type of a generic parameter.

```
ArgType arg_getType(Arg* self);
```

Generic parameters can be converted to primitive types using the following API.

```
int64_t arg_getInt(Arg* self);
float arg_getFloat(Arg* self);
void* arg_getPtr(Arg* self);
char* arg_getStr(Arg* self);
```

Parameter management

1. Use the parameter name hash or parameter name to determine whether a parameter exists. The return value is 1 to indicate that it exists, and the times33 algorithm is used to obtain the parameter name hash.

```
int32_t args_isArgExist_hash(Args* self, Hash nameHash);
int32_t args_isArgExist(Args* self, char* name);
Hash hash_time33(char* str);
```

1. Delete a parameter using a pointer to a generic parameter

```
int32_t args_removeArg(Args* self, Arg* argNow);
```

The return value is an error code, and 0 indicates success.

8.2.6 traversal of parameter list

A parameter table can be traversed using the following API.

1. The first entry parameter is a pointer to the parameter table.
2. The second parameter is the function pointer of the callback function when traversing the parameters
3. The third parameter is an auxiliary parameter table, which is used to pass auxiliary parameters. When auxiliary parameters are not used, the third input parameter can be filled with NULL.

```
int32_t args_foreach(Args* self,
                     int32_t (*eachHandle)(Arg* argEach, Args* handleArgs),
                     Args* handleArgs);
```

8.3 Generic parameters Arg

8.3.1 Header file

```
#include "dataArg.h"
```

8.3.2 Overview

1. arg The generic argument API is a set of functions prefixed with arg_.
2. arg can hold a value of any type in it. The types supported by arg are: int, float, pointer, string, null, bytes. 1.
3. arg can be put into an object and the value of arg can be accessed directly in the python script.

8.3.3 Data types

The data type of a generic argument is Arg.

```
typedef struct Arg Arg;
struct Arg {
    Arg* next;
    uint32_t size;
    uint8_t type;
    Hash name_hash;
    uint8_t content[];
};
```

The generic arguments internally include header information (size, type, name_hash), the data body (content), and a pointer (next) used to form the chain.

Be careful not to access the internal members of arg directly, use the arg API to access arg. for **maximum backward compatibility**.

8.3.4 Creating and destroying generic arguments

- Creating a new generic argument

Creates a new generic argument from the heap and returns a pointer to the generic argument.

**Note that newly created generic parameters need to be manually destroyed to reclaim memory. Constantly creating new generic parameters but not destroying them can lead to memory leaks. **

[Note] The following api requires a kernel version of at least v1.9.2

```
Arg* arg_newInt(int val);
Arg* arg_newFloat(double val);
Arg* arg_newPtr(ArgType type, void* pointer);
Arg* arg_newStr(char* val);
Arg* arg_newNull(void);
Arg* arg_newBytes(uint8_t* src, size_t size);
```

New arg The argument passed in is the value of arg.

- Destroy generic arguments.

```
void arg_deinit(Arg* self);
```

- Copy generic arguments

```
Arg* arg_copy(Arg* self);
```

Pass in a pointer to the generic argument and destroy the generic argument.

8.3.5 Getting the value of a generic argument

Use the following API to determine the current type of a generic argument.

```
ArgType arg_getType(Arg* self);
```

Use the following API to convert a generic argument to a basic type.

```
int64_t arg_getInt(Arg* self);
float arg_getFloat(Arg* self);
void* arg_getPtr(Arg* self);
char* arg_getStr(Arg* self);
uint8_t* arg_getBytes(Arg* self);
size_t arg_getBytesSize(Arg* self);
```

8.3.6 Important Notes

Direct use of the arg_new<Type>() api **is highly likely to cause **memory leaks or dangling references, resulting in **fatal flaws**.

Please develop under docker development environment to ensure sufficient unit testing and memory checking.

8.3.7 Case

Build a list of strings using arg

```
## Include "PikaStdData_List.h"
...
    /* Create a list object */
    PikaObj* list = newNormalObj(New_PikaStdData_List);
    /* initialize list */
    PikaStdData_List___init__(list);
    /* Create arg with api of arg_new<type> */
    Arg* str_arg1 = arg_newStr("aaa");
    /* add to list object */
    PikaStdData_List_append(list, str_arg1);
    /* destroy arg */
    arg_deinit(str_arg1);
...
```

8.4 String pool Strs

8.4.1 head File

```
#include "dataStrs.h"
```

8.4.2 Overview

1. The Strs string pool API is a series of functions prefixed with strs.
2. The string pool provides **dynamic memory space** for strings, supports **any length** strings, and a string pool can store **any number of** strings.
3. Provide **convenient memory management**, when destroying the string pool, all the string memory in the pool will be automatically **batch destroyed**.
4. Provide a **safe operation method**, when using the strs API, the quoted string **will not be modified**. All modifications are made in the **newly allocated memory area**. Therefore, there will be no serious security problems such as dangling pointers and tampered strings.
5. The Strs string pool API is designed using object-oriented ideas. The first entry parameters of these functions are pointers to the operated string pool.

8.4.3 type of data

The data type of Strs is Args, and a parameter table is maintained internally.

```
typedef Link Args;
```

Be careful not to directly access the string pool's parameter table, use the Strs API to access Strs for **memory safety** and **maximum backward compatibility**.

CHAPTER

NINE

CONFIGURATION AND ADVANCED FEATURES

9.1 PikaPython configuration manual

9.1.1 When to configure

PikaPython itself is **configuration-free**, so usually you don't need to know this part.

You can consider configuring PikaPython when you have the following requirements:

- faster speed
- Smaller memory footprint
- Replace dependencies (libc, pinrtf, etc.)
- Replace memory management algorithm (malloc)
- Safer interruption protection

9.1.2 Optimization

[Note]: For optimized configuration, the kernel version needs to be at least `v1.5.4`

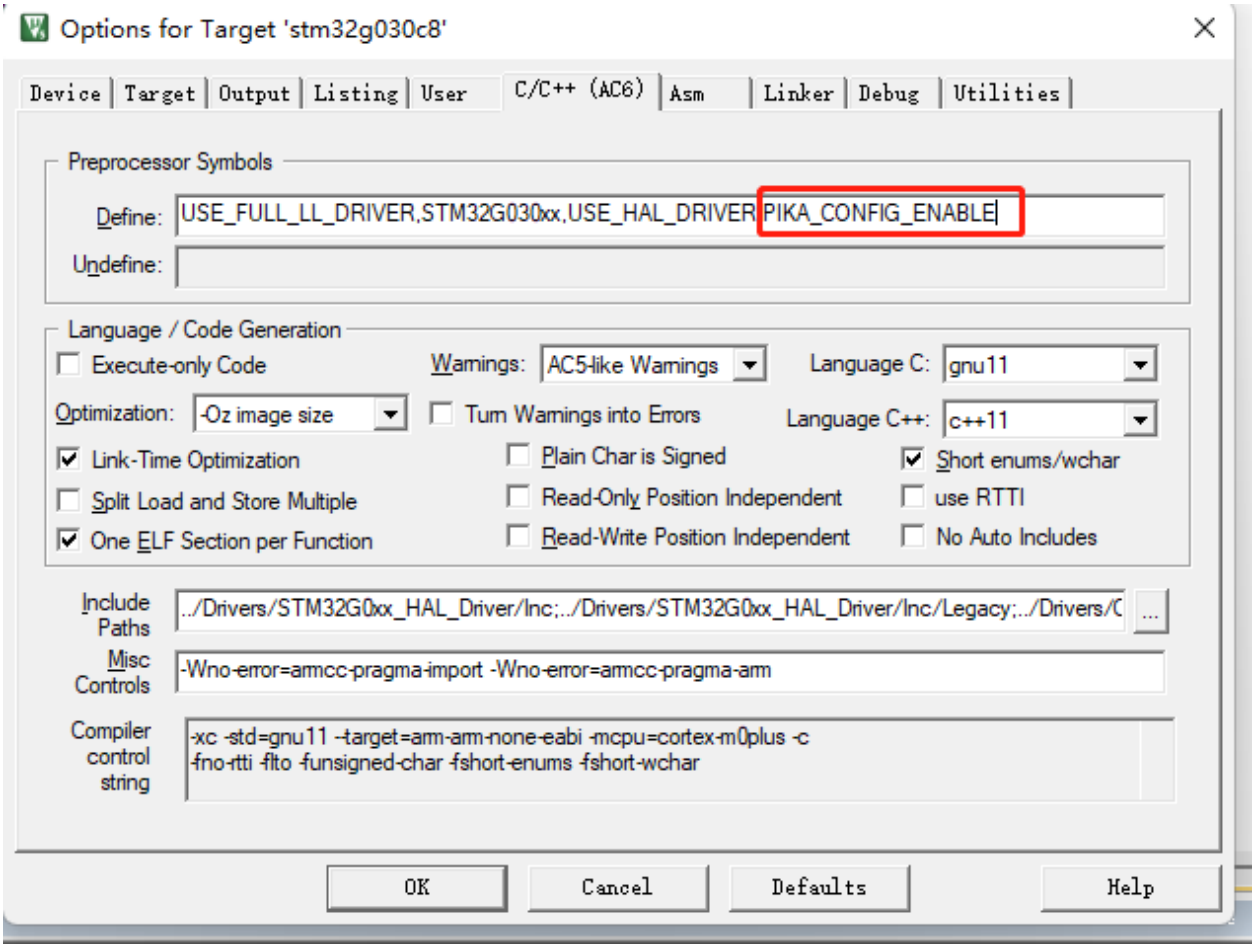
Similar to GCC, PikaPython also provides different optimization modes. The currently available optimization modes are:

- `PIKA_OPTIMIZE_SIZE` volume mode minimizes running memory
- `PIKA_OPTIMIZE_SPEED` performance mode maximizes running speed

Enable user configuration

User configuration is not enabled by default. The way to enable user configuration is to add compile-time macro definition `PIKA_CONFIG_ENABLE`. Then create the `pika_config.h` header file.

It should be noted that the `PIKA_CONFIG_ENABLE` macro should be added to the compile options, such as keil:



Configuration items

Available configuration items and default configuration are in the `pika_config_valid.h` header file.

https://github.com/pikastech/pikascript/blob/master/src/pika_config_valid.h

Intercept the important part for explanation:

```
/* optimize options */
    #define PIKA_OPTIMIZE_SIZE 0
    #define PIKA_OPTIMIZE_SPEED 1

/* syntax support level */
    #define PIKA_SYNTAX_LEVEL_MINIMAL 0
    #define PIKA_SYNTAX_LEVEL_MAXIMAL 1

/* use user config */
#ifdef PIKA_CONFIG_ENABLE
    #include "pika_config.h"
#endif

/* default optimize */
#ifndef PIKA_OPTIMIZE
```

(continues on next page)

(continued from previous page)

```
        #define PIKA_OPTIMIZE PIKA_OPTIMIZE_SIZE
    #endif

    /* default syntax support level */
    #ifndef PIKA_SYNTAX_LEVEL
        #define PIKA_SYNTAX_LEVEL PIKA_SYNTAX_LEVEL_MAXIMAL
    #endif

...

    /* default configuration  */

        #ifndef PIKA_STACK_BUFF_SIZE
        #define PIKA_STACK_BUFF_SIZE 256
    #endif
```

`default configuration` is the default value of the configuration item. When the `PIKA_CONFIG_ENABLE` macro is defined, `pika_config_valid.h` will import `pika_config.h`, so User can override the above default configuration in `pika_config.h`.

For example, if you want to increase the runtime stack of the PikaPython virtual machine, you can write in `pika_config.h`

```
#define PIKA_STACK_BUFF_SIZE 512
```

As can be seen from `pika_config_valid.h`, the default optimization option of PikaPython `PIKA_OPTIMIZE` is `PIKA_OPTIMIZE_SIZE`, if you need to switch to speed optimization, you can write in `pika_config.h`

```
#define PIKA_OPTIMIZE PIKA_OPTIMIZE_SPEED
```

Sample code

https://github.com/pikastech/pikascript/blob/master/bsp/stm32g070cb/Booter/pika_config.h

9.1.3 Dependency configuration

PikaPython can be configured by creating `pika_config.c`, rewriting the weak functions in PikaPlagform.h 's dependencies.

```
/* interrupt config */
void __platform_enable_irq_handle(void);
void __platform_disable_irq_handle(void);

/* printf family config */
#ifndef __platform_printf
void __platform_printf(char* fmt, ...);
#endif
int __platform_sprintf(char* buff, char* fmt, ...);
int __platform_vsprintf(char* buff, char* fmt, va_list args);
int __platform_vsnprintf(char* buff,
                         size_t size,
```

(continues on next page)

(continued from previous page)

```
                        const char* fmt,
                        va_list args);

/* libc config */
void* __platform_malloc(size_t size);
void __platform_free(void* ptr);
void* __platform_memset(void* mem, int ch, size_t size);
void* __platform_memcpy(void* dir, const void* src, size_t size);

/* pika memory pool config */
void __platform_wait(void);
uint8_t __is_locked_pikaMemory(void);

/* support shell */
char __platform_getchar(void);

/* file API */
FILE* __platform_fopen(const char* filename, const char* modes);
int __platform_fclose(FILE* stream);
size_t __platform_fwrite(const void* ptr, size_t size, size_t n, FILE* stream);

/* error */
void __platform_error_handle(void);
```

Configuration items:

- Interrupt Protection - Provides an interrupt master switch to protect PikaPython memory safety
- libC - select the implementation of libC
- Memory management - replace malloc free memory management algorithm

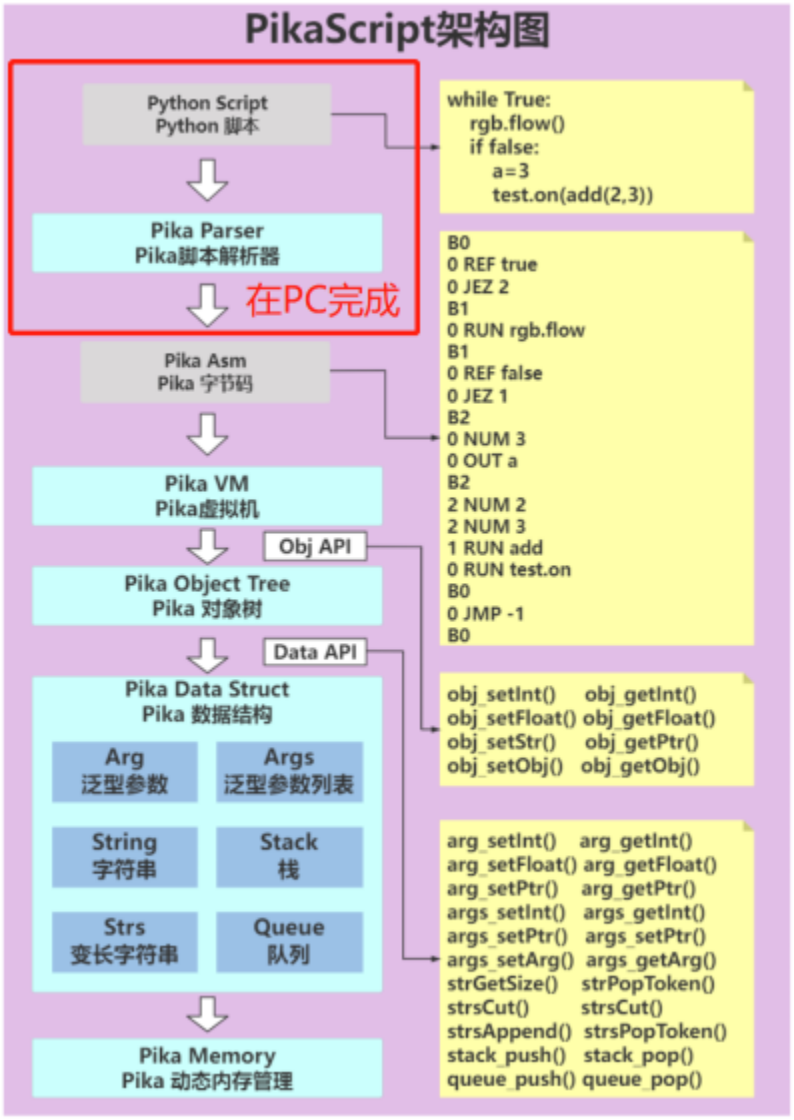
Sample code:

- https://github.com/pikastech/pikascript/blob/master/bsp/stm32g030c8/Booter/pika_config.c
- https://github.com/pikastech/pikascript/blob/master/package/pikaRTThread/pika_config.c

9.2 Run Bytecode Directly

The runtime architecture of PikaPython is shown below. By default, the process of parsing Python scripts into Pika bytecode is executed in the MCU, which allows the MCU to run Python scripts directly, including support for interactive running. Instead, in resource-constrained cases, the process of parsing Python scripts into bytecode can be done earlier at the PC, allowing the Python script to be executed directly instead of parsing it in the MCU, so that **the code to parse the Python script can be trimmed out**.

By avoiding the use of `obj_run()` to execute python scripts and running the bytecode directly, the compiler will automatically optimize the Python parsed code and reduce the code size footprint.



9.2.1 Converting Python to bytecode on PC.

The precompiler `rust-msc-latest-win10.exe` integrates a bytecode generator that compiles main.py and the `.py` files `imported` by main.py (including indirect `import`) to bytecode when precompiling, and the generated bytecode files are in the pikascript-api folder.

The `.py` file is generated as a `.py.o` bytecode file, e.g. `main.py` generates `pikascript-api/main.py.o`.

At the same time, all `.py.o` files are automatically packaged into a library file `pikascript-api/pikaModules.py.a`, which contains all bytecode files.

To facilitate the loading of the library files in mcu when compiling the firmware, the precompiler also automatically converts the library files to C byte array files `pikascript-api/__asset_pikaModules_py_a.c`.

```
/* __asset_pikaModules_py_a.c */
#include "PikaPlatform.h"
/* warning: auto generated file, please do not modify */
```

(continues on next page)

(continued from previous page)

```
PIKA_BYTECODE_ALIGN const unsigned char pikaModules_py_a[] = {
    0x7f, 0x70, 0x79, 0x61, 0x01, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6d, 0x61, 0x69, 0x6e,
...
```

9.2.2 Using library file

Library files can be imported using the `obj_linkLibrary()` API, refer to the automatically generated `pikaScriptInit()`

```
PikaObj *pikaScriptInit(void){
...
    __pikaMain = newRootObj("pikaMain", New_PikaMain);
    extern unsigned char pikaModules_py_a[];
    obj_linkLibrary(__pikaMain, pikaModules_py_a);
...
}
```

After importing a library file, you can `import` the modules contained in the library file directly inside the python script.

It is also possible to run a module directly as a script, e.g.

```
obj_runModule(__pikaMain, "main");
```

9.2.3 Run a single bytecode

Read data from a single bytecode file `.py.o` and then use the `pikaVM_runByteCode()` API to just run the single bytecode directly, see the usage of starting from bytecode in g030.

https://github.com/pikastech/pikascript/blob/master/bsp/stm32g030c8/Booter/main.c

Note

1. The byte code run by the `pikaVM_runByteCode()` API must be of type `const`, if the byte code is not `const`, you need to use `pikaVM_runByteCodeInconstant()`.
2. If you have already docked the file system, you can use the `pikaVM_runByteCodeFile()` API to run the `.py.o` file directly.
3. `pikaVM_runByteCodeInconstant()` and `pikaVM_runByteCodeFile()` require kernel version >=`v1.11.7`.

9.3 Event callback mechanism

9.3.1 Overview

The PikaPython kernel provides an event callback mechanism that supports triggering Python defined callback functions in C events/interrupts.

Note: requires kernel version no less than: v1.8.7

9.3.2 Headers

```
#include "PikaObj.h"
```

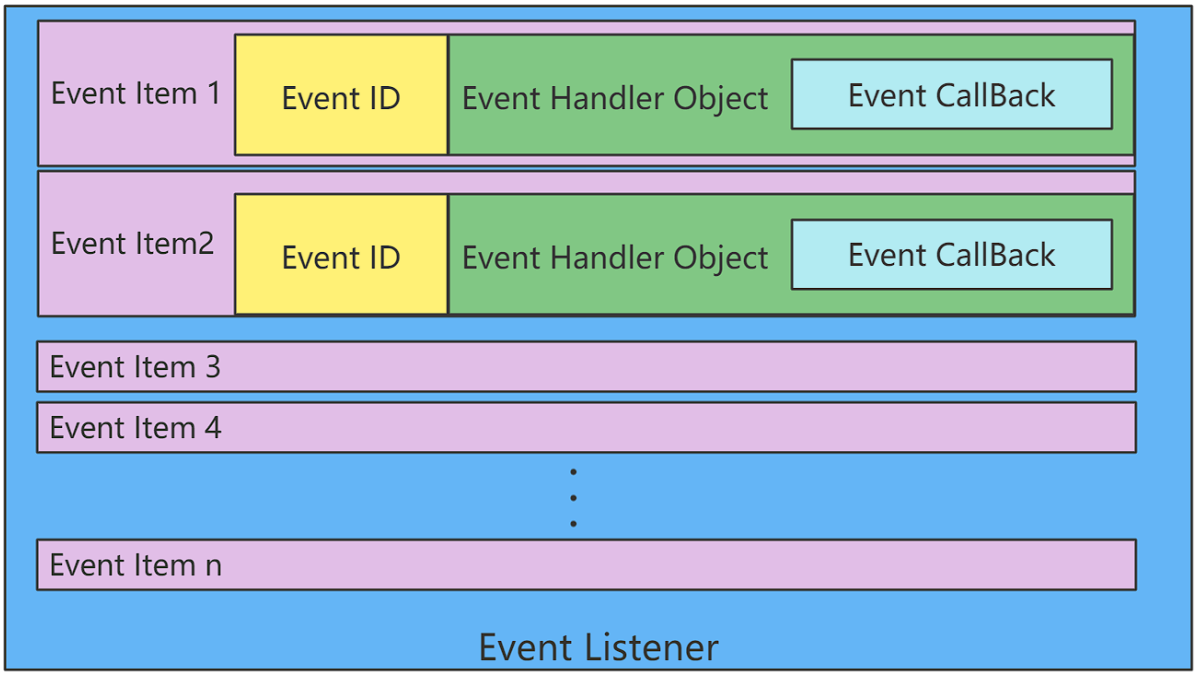
9.3.3 Data types

```
typedef PikaObj PikaEventListener;
```

The event callback mechanism relies on the `PikaEventListener` event listener, which records the ID of each registered event. When a signal is sent to the event listener, the event listener will call the corresponding Python callback function based on the event ID, and pass the semaphore.

9.3.4 The Event Model

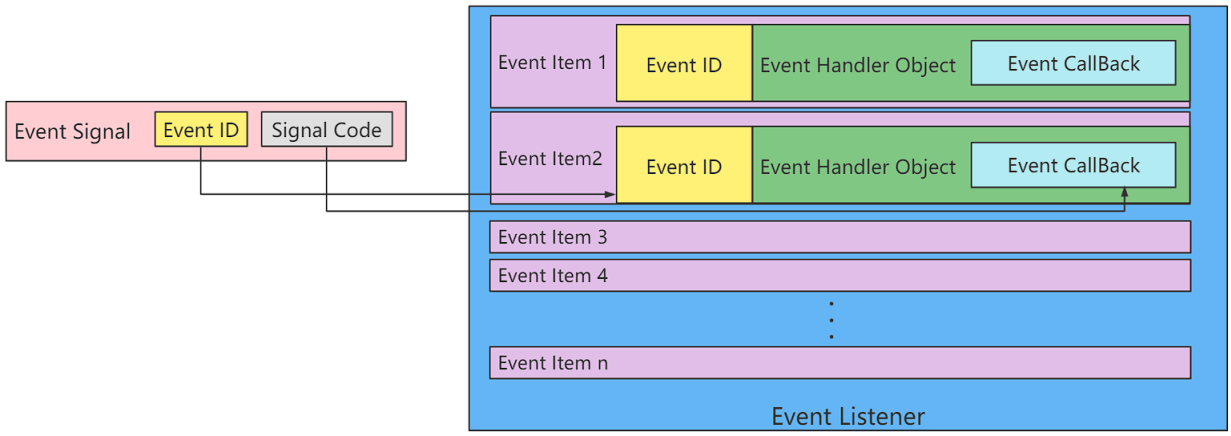
The core of the event model is the `PikaEventListener` event listener.



The `PikaEventListener` model is shown above. After registering an event to the event listener, an event item `Event Item` will be recorded inside the `PikaEventListener`, including.

- `Event ID` the unique ID of the event
- `Event Handler Object` event object, which records all the information about the event item
- `Event CallBack` event callback function (Python function)

When the `Event Signal` event signal arrives, the event listener will match the `Event ID` to find the corresponding event item, then pass the signal code `Event Code` to `Event CallBak` to trigger the callback function.



9.3.5 Event callback mechanism flow

1. Initialize the event listener
2. register callback functions in Python
3. Signal the event listener in C (usually in an interrupt or a callback in C)
4. the callback function registered in Python is executed

9.3.6 Support event callbacks via PikaStdDevice

Inheriting PikaStdDevice is the easiest way to support event callbacks, the `PikaStdDevice.BaseDev` device base class already supports the event registration method `addEventCallBack`.

```
class BaseDev:
    def addEventCallBack(self, eventCallback: any): ...

    # need override
    def platformGetEventId(self): ...
```

- The device classes in `PikaStdDevice` (e.g. GPIO) all inherit from `BaseDev`, so they all get the `addEventCallBack` method and can register callbacks.

/package/PikaStdDevice/PikaStdDevice.pyi

```
class GPIO(BaseDev):
...
```

After the platform driver inherits from `PikaStdDevice.GPIO`, it also gets the `addEventCallBack` method.

/package/TemplateDevice/TemplateDevice.pyi

```
# TemplateDevice.pyi
class GPIO(PikaStdDevice.GPIO):
    # overrid
...
    def platformGetEventId(self): ...
...
```

Just override the `platformGetEventId` platform method to be able to support registration callbacks.

For example.

/package/TemplateDevice/TemplateDevice_GPIO.c

```
const uint32_t GPIO_PA8_EVENT_ID = 0x08;
void TemplateDevice_GPIO_platformGetEventId(PikaObj* self) {
    char* pin = obj_getStr(self, "pin");
    if (strEqu(pin, "PA8")) {
        obj_setInt(self, "eventId", GPIO_PA8_EVENT_ID);
    }
}
```

9.3.7 Registering callback functions in Python

- Define a callback function `callBack1` that takes an input parameter `signal`, `signal` can receive the incoming signal number.

/examples/TemplateDevice/gpio_cb.py

```
import TemplateDevice

io1 = TemplateDevice.GPIO()
io1.setPin('PA8')
io1.setMode('in')
io1.enable()

EVENT_SIGAL_IO_RISING_EDGE = 0x01
EVENT_SIGAL_IO_FALLING_EDGE = 0x02

def callBack1(signal):
    if signal == EVENT_SIGAL_IO_RISING_EDGE:
        print('get rising edge!')
    elif signal == EVENT_SIGAL_IO_FALLING_EDGE:
        print('get falling edge!')

io1.addEventCallBack(callBack1)
```

9.3.8 Signal triggering

Send a signal to `PikaEventListener` when an event callback needs to be triggered.

Example: /port/linux/test/event-test.cpp

- Get the event listener provided by `PikaStdDevice` via `extern PikaEventListener* g_pika_device_event_listener`.
- Send `eventID` and `signal code` via `pks_eventLisener_sendSignal`.

```
extern PikaEventListener* g_pika_device_event_listener;
#define EVENT_SIGAL_IO_RISING_EDGE 0x01
#define EVENT_SIGAL_IO_FALLING_EDGE 0x02
#define GPIO_PA8_EVENT_ID 0x08
```

(continues on next page)

(continued from previous page)

```
TEST(event, gpio) {
    /* init */
    PikaObj* pikaMain = newRootObj("pikaMain", New_PikaMain);
    /* run */
    pikaVM_runFile(pikaMain, "... /... /examples/TemplateDevice/gpio_cb.py");
    /* simulate run in the call back */
    pks_eventLisener_sendSignal(g_pika_device_event_listener, GPIO_PA8_EVENT_ID,
                            EVENT_SIGAL_IO_RISING_EDGE);
    pks_eventLisener_sendSignal(g_pika_device_event_listener, GPIO_PA8_EVENT_ID,
                            EVENT_SIGAL_IO_FALLING_EDGE);
...
}
```

- Running results.

```
get rising edge!
get falling edg!
```

Waiting for the return value

Event callback functions can have return values, such as returning `signal` directly.

```
def callBack1(signal):
    return signal
io1.addEventCallBack(callBack1)
```

This function requires OS support, and the `__platform_thread_delay()` method needs to be overridden to be able to dispatch events to the main process while waiting for a return value. If a return value is required, the trigger event can use `pks_eventLisener_sendSignalAwaitResult` to get the return value of the callback function, which is an ``Arg*` type.

```
Arg* res_123 = pks_eventLisener_sendSignalAwaitResult(
        g_pika_device_event_listener, GPIO_PA8_EVENT_ID, 123);
int res = arg_getInt(res_123);
```

Note: requires kernel version >= `v1.11.7`

9.3.9 Advanced: Custom event registration functions

- In addition to event callbacks supported by `PikaStdDevice`, you can also customize event registration functions, which is an advanced part.
- Custom event registration requires a better understanding of PikaPython's C-module mechanism and object mechanism.
- Define a Python interface to a C module that receives incoming event callback functions.

For example.

/package/PikaStdDevice/PikaStdDevice.pyi

```
class BaseDev:
    def addEventCallBack(self, eventCallback: any): ...
```

The type annotation for the event callback function is any.

- Registering events in the C module implementation

Example: /package/PikaStdDevice/PikaStdDevice_BaseDev.c

```
PikaEventListener* g_pika_device_event_listener;

void PikaStdDevice_BaseDev_addEventCallBack(PikaObj* self, Arg* eventCallBack) {
    obj_setArg(self, "eventCallBack", eventCallBack);
    /* init event_listener for the first time */
    if (NULL == g_pika_device_event_listener) {
        pks_eventLisener_init(&g_pika_device_event_listener);
    }
    if (PIKA_RES_OK != obj_runNativeMethod(self, "platformGetEventId", NULL)) {
        obj_setErrorCode(self, 1);
        __platform_printf("Error: Method %s no found.\r\n",
                          "platformGetEventId");
    }
    uint32_t eventId = obj_getInt(self, "eventId");
    pks_eventLicener_registEvent(g_pika_device_event_listener, eventId, self);
}
```

- Create a global `PikaEventListener`: `g_pika_device_event_listener`.
- Pass `self` as `event handler object` and `evnetCallBack` into `self`.
- Get `evnetID`.
  - This example gets the `eventID` by calling the `platformGetEventId()` platform function, which requires `BaseDev` inheritance, then rewrites `platformGetEventId()` and sets `self. eventId` in the overridden `platformGetEventId()`.
  - For example: /package/TemplateDevice/TemplateDevice_GPIO.c
- Call `pks_eventLicener_registEvent` to register `eventId` and `self` into the event listener.

9.4 Compact Memory Pools

9.4.1 Overview

PikaPython has a built-in compact memory pool for small resource chips, which is not enabled by default.

Compact memory pooling can reduce memory fragmentation from the usual 20-30% to less than 5%.

Note] Compact memory pooling can slow down the operation speed.

9.4.2 Enabling method

Note that the kernel version must be at least v1.9.0.

Enable user configuration

Refer to the *configuration document*

Add configuration items

```
/* pika_config.h */
#define PIKA_POOL_ENABLE 1
#define PIKA_POOL_SIZE 0x1900
```

Where `PIKA_POOL_ENABLE` means open compact memory pool, `PIKA_POOL_SIZE` means the size of the memory pool, the memory pool pre-apply memory from heap, please make sure the heap can apply to that size.

Refer to bsp/stm32g030c8/Booter/pika_config.h

Memory pool initialization

Initialize the memory pool before `pikaScriptInit()` or `newRootObj()`.

```
mem_pool_init();
```

Reference: bsp/stm32g030c8/Booter/main.c

9.4.3 Freeing the memory pool

If the memory pool needs to be freed, call

```
mem_pool_deinit();
```

9.5 Interrupting a running script

Calling `pks_vm_exit()` forces the interruption of a running script (also in a dead loop), which can be placed in the interrupt function.

> [Note]
>
> Requires kernel version not lower than `v1.11.0`.
>
> After interrupting a running script, only the VM is exited, the root object can still be used and will not be freed, if you need to free memory, you should execute `obj_deinit()` on the root object.

CHAPTER

TEN

CONTRIBUTE

10.1 How to contribute

10.1.1 We sincerely appreciate your contributions and welcome to submit code through GitHub, Gitee's fork and Pull Request process

You can contribute to PikaPython:

- Contribute to python examples
- *Contribute to BSP for new platform*
- *Contribute to module*
- Contribute to standard library
- *Contribute to kernel*

If you are new to the open source community, it is recommended to refer to Open Source Guide guide to learn how to participate in the code contribution of the open source community.

10.1.2 It is not only by contributing code to participate in community contributions

you can also:

- use PikaPython in company products, personal projects or competitions;
- present your work, ask or answer questions in the PikaPython Forum;
- Submit an issue and report bugs to PikaPython at GitHub or Gitee;
- Participate in community activities and B station live broadcast;
- Purchase Development Board - Pika Pie officially supported by PikaPython ;

- Invite the author to drink the ice;

10.2 How to contribute to PikaPython BSP

10.2.1 Steps to make BSP:

Make pikascript template project

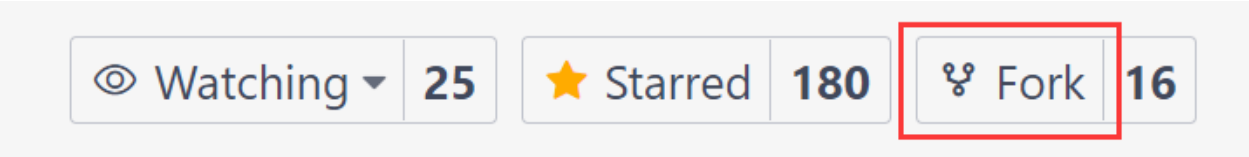
- The BSP of pikascript is very simple, it is a pikascript template project that can be compiled independently.
- This project only needs to be able to run pikascript **basically**.
- You can refer to the **New Platform Porting Guide** to ensure that `print('hello PikaPython!')` in main.py can be run normally.

Clean up project

- Clean up **compiled products**, leaving only project files and source code. (The compilation products include intermediate files .o .d , binary products .bin, .hex , executable files .exe , etc.).
- Clean up the **auto-pull** and **auto-generate** codes in the pikascript folder. Only the main.py, requsetment.txt, and pikaPackage.exe files can be kept in the pikascript folder.
- Clean up unused source code and libraries, and control the size of the project to within 50MB. If the size of the project is still larger than 50MB after cleaning, you can create a new special warehouse to place the BSP, and only place a README.md containing a link to the special warehouse in pikscript/bsp.

Submit file

- Enter the pikascript code repository, either gitee or github, fork a pikascript repository, and then clone the forked repository locally.



- Create a new folder in the [repository after fork]/bsp directory, then copy it into the template project, use the git command to add files, and push it to the pikascript repository after **fork**.

```
cd pikascript/bsp
git add *
git commit -m 'add bsp'
git push
```

- (Optional) Update BSP information in pikascript/README.md and pikascript/README_zh.md.
- Open Pull Request and wait for merge.

10.3 How to contribute to modules

10.3.1 Help improve existing modules

Pull the latest module

- When adding new content to an existing module, make sure you have pulled the latest module.
- The way to pull the latest module is to use the latest version in requestment.txt.

E.g:

```
STM32G0==latest
```

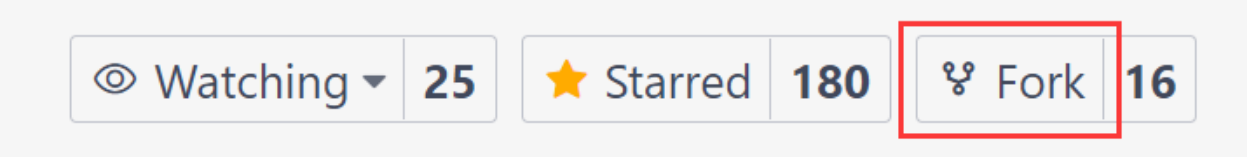
- **Delete the modules that need to be developed in reqeustment.txt**, to prevent misoperation (such as pulling the module again) causing the module being developed to be overwritten.

Modify the module and test

- Add new Python interface for modules –> [module].pyi
- Or provide a better implementation –> pikascript-lib/[module]/*.c
- (Optional) Update module information in pikascript/README.md and pikascript/README_zh.md.

Submit the module's files

- fork a pikascript repository, then clone it locally.



- Copy [module].pyi to pikascript-lib/[module] folder.
- Copy the entire modified pikascript-lib/[module] folder into the forked pikascript/package folder.
- git add adds files, and git commit commits once.
- git log View the commit id after the commit, fill in the new version name in pikascript/packages.toml after fork, and copy the current commit id.

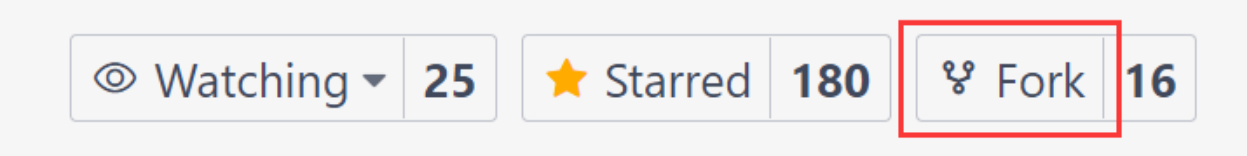
E.g:

```
[[packages]]
name = "STM32G0"
releases = [
  "v1.0.2 0052a28582ac8a85cc48e1d676d9a3be5cb1b93f",
  "<new version name> <current commit id>",
]
```

- git commit -a commits again, adding modifications to packages.toml.
- git push to your forked repository.
- Submit a pull request.

10.3.2 Commit the new module

- Create a new [module].pyi file and pikascript-lib/[module] folder.
- Develop and test new modules.
- (Optional) Update module information in pikascript/README.md and pikascript/README_zh.md.
- Submit the module's files
  - Fork a pikascript repository, then clone it locally.



- Copy [module].pyi to pikascript-lib/[module] folder.
- Copy the entire pikascript-lib/[module] folder to the forked pikascript/package folder.
- git add adds files, and git commit commits once.
- git log View the submitted commit id, add a new module to pikascript/packages.toml after fork, and fill in the module name, version name and current commit id.

E.g:

```
[[packages]]
name = "<new module name>"
releases = [
  "<new version name> <current commit id>",
]
```

- git commit -a commits again, adding modifications to packages.toml.
- git push to your forked repository.
- Submit a pull request.

10.4 How to contribute to the standard library

10.4.1 What are PikaPython standard libraries?

PikaPython standard libraries are a set of cross-platform libraries for common tools such as string, time, etc.

Some of these libraries provide APIs consistent with or similar to CPython, and some provide common tools for MCU development.

10.4.2 PikaPython standard library development environment construction

The PikaPython standard library is cross-platform, so it can't use the proprietary resources of the platform (e.g. stm32), to ensure this, the standard library is developed on linux platform.

PikaPython deploys GoogleTest unit testing framework on linux platform to provide test cases for these standard libraries, GoogleTest can be run on the developer's local machine and also automatically in the cloud (based on Github Actions).

Build Docker container

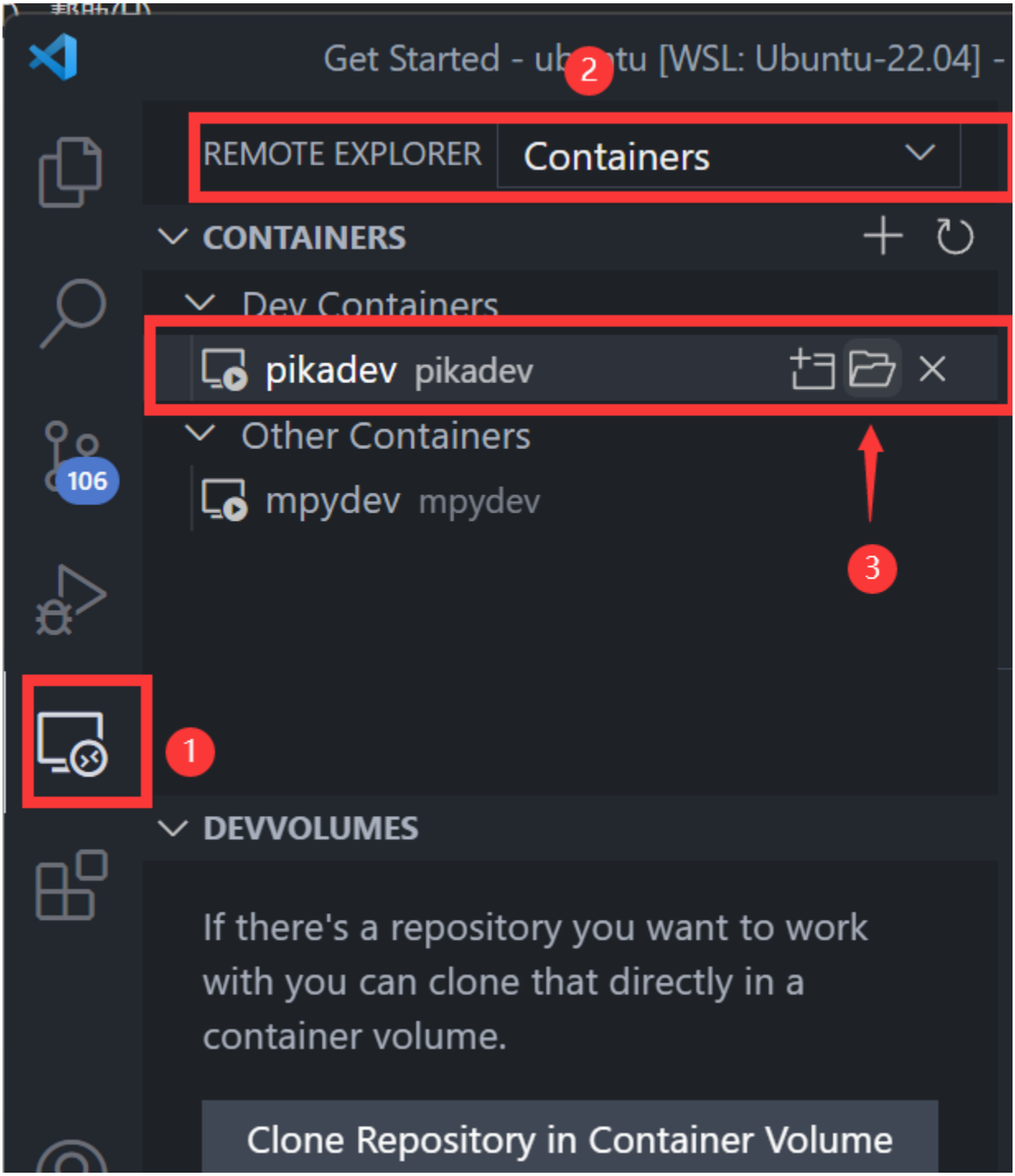
get start -> get start with docker

10.4.3 Use VSCODE to connect to the container for development

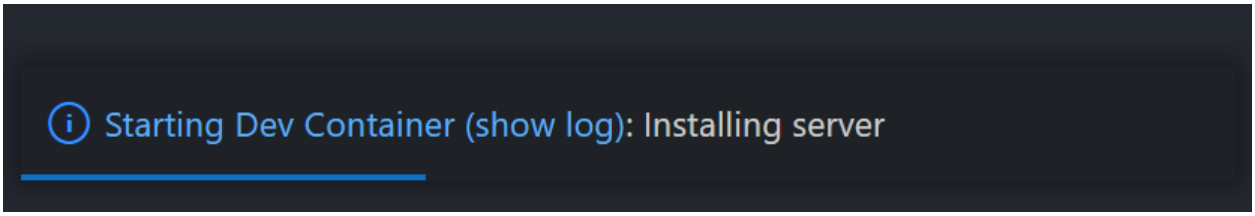
Start

VSCODE provides tools to connect to containers for development, and the development experience is as good as if you were outside the container.

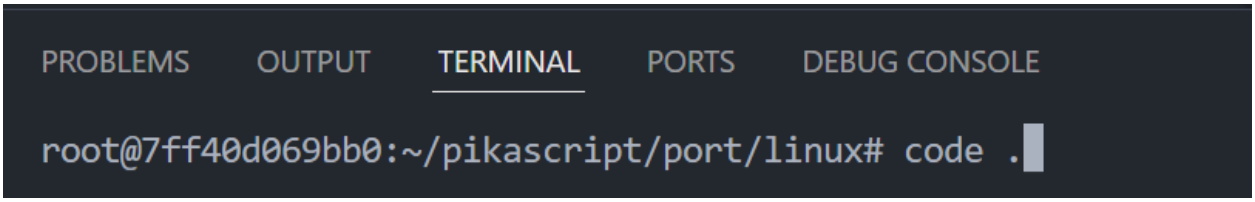
Select Remote, Containers, pikadev in the VSCODE sidebar, then click Open Directory to connect to Docker inside VSCODE.



The first time you open it, you need to wait for some plugins to be installed automatically, then you can open it again and start it directly.



cd to ~/pikascript/port/linux, then type `code .` to switch the working path to pikascript/port/linux



Compile and run

- Initialize

```
sh pull-core.sh # Update kernel source code
```

- Pre-compile and configure CMake

```
sh init.sh
```

- Compile

```
sh only_make.sh
```

- test

```
sh gtest.sh # run google test
sh ci_benchmark.sh # run benchmark
sh valgrind.sh # run valgrind
```

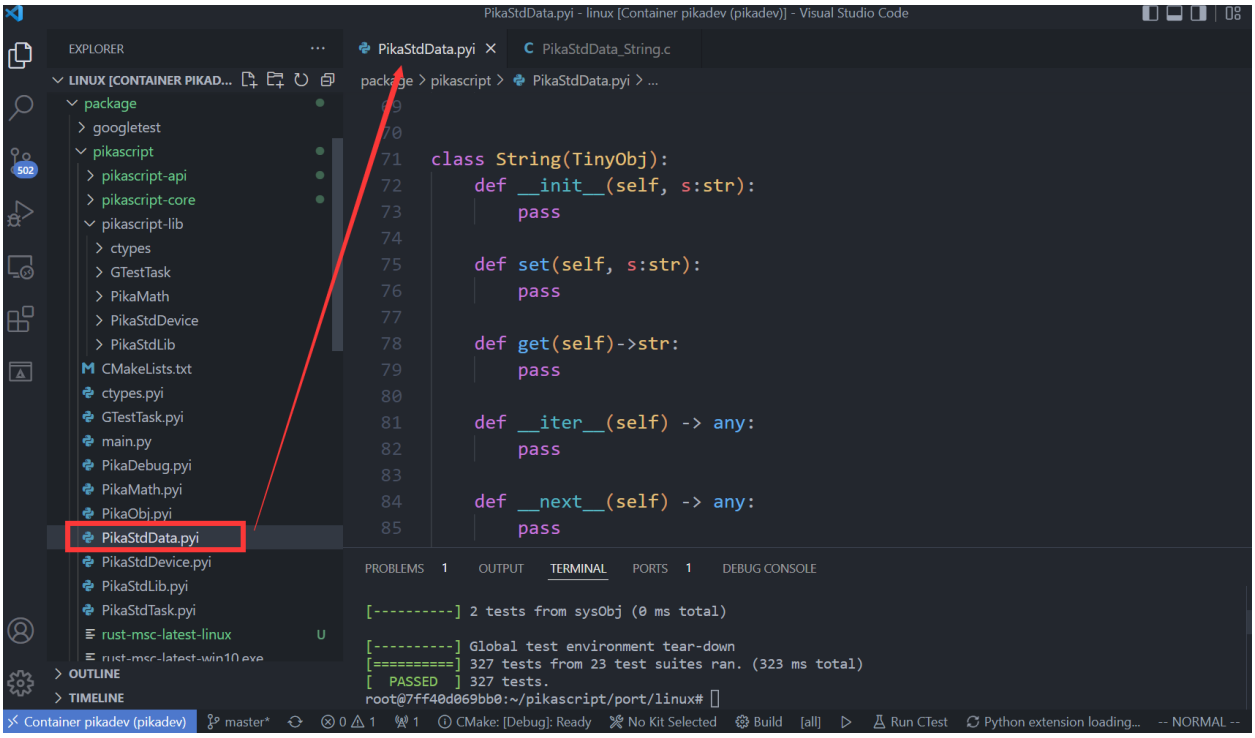
- run

```
sh run.sh # Start REPL
```

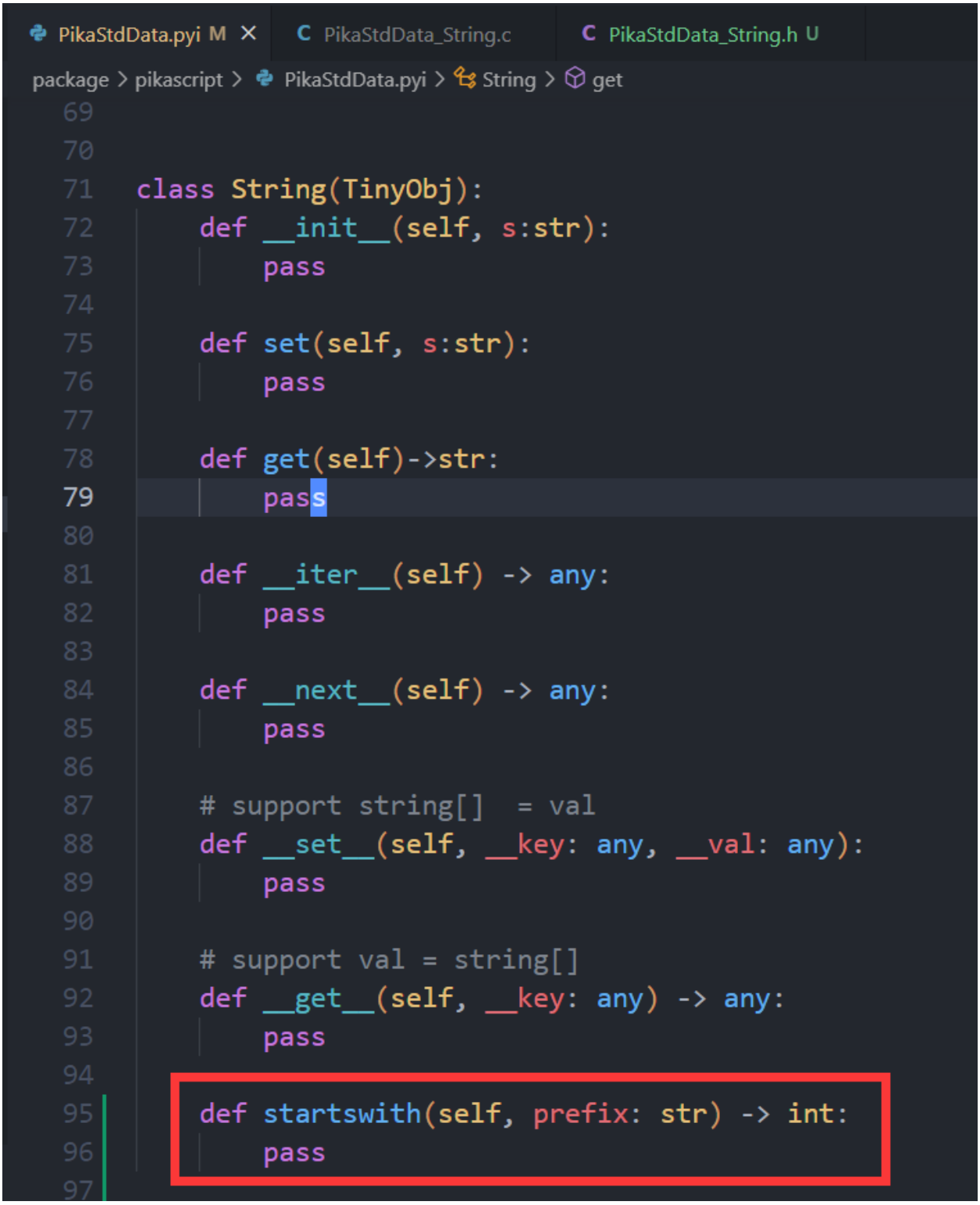
Development

The pyi declaration files for the standard library are in the package/pikascript directory. The standard library includes PikaStdLib.pyi, PikaStdData.pyi, PikaDebug.pyi, PikaStdTask.pyi, etc.

The implementation files are in the PikaStdLib folder.



Then you can add classes, or functions to the standard library, for example, add a `startswith()` method to the `PikaStdData.String` class by first adding a declaration for the `startswith()` method under the `String` class in PikaStdData.pyi.



Then run.

```
sh init.sh
```

to pre-compile and reconfigure CMake.

Then open `PikaStdData_String.h` and you will find the c function declaration for the automatically generated startswith method.

Next, implement this function in PikaStdData_String.c.

PikaStdData.pyi M | C PikaStdData_String.c | C PikaStdData_String.h U

package > pikascript > pikascript-lib > PikaStdLib > C PikaStdData_String.c > ...

```c
    return res;
}

Arg* PikaStdData_String___get__(PikaObj* self) {
    int key_i = obj_getInt(self, "__key");
    char* str = obj_getStr(self, "str");
    uint16_t len = strGetSize(str);
    char char_buff[] = " ";
    if (key_i < len) {
        char_buff[0] = str[key_i];
        return arg_setStr(NULL, "", (char*)char_buff);
    } else {
        return arg_setNull(NULL);
    }
}

int PikaStdData_String_startswith(PikaObj *self, char* prefix){
    char* str = obj_getStr(self, "str");
    if(strIsStartWith(str, prefix)){
        /* true */
        return 1;
    }
    /* false */
    return 0;
}

void PikaStdData_String___set__(PikaObj* self) {}
```

Testing

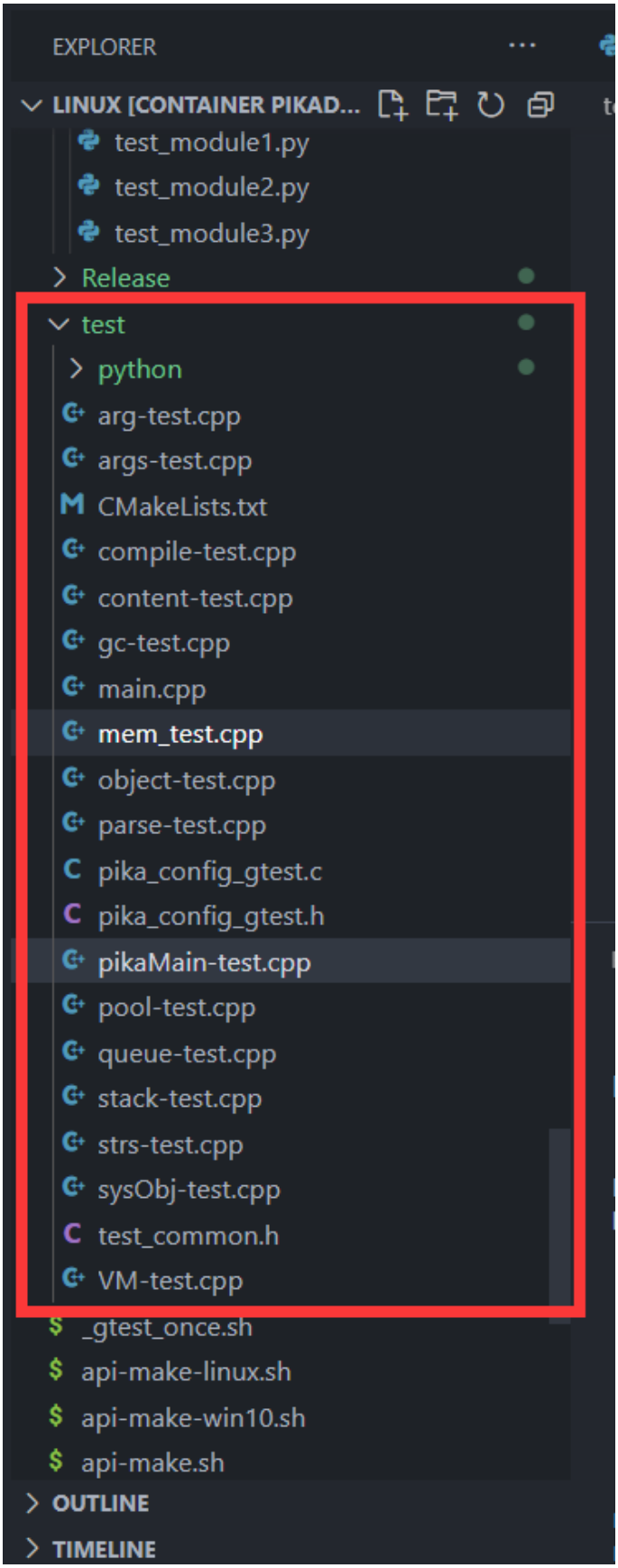
Then you can run GoogleTest to see if it breaks the original code.

```
sh gtest.sh
```

```
[----------] 2 tests from sysObj
[ RUN      ] sysObj.print
hello world
[       OK ] sysObj.print (0 ms)
[ RUN      ] sysObj.noMethod
BEGIN
NameError: name 'printttt' is not defined
   STR hello world              (#1)
 -> RUN printttt                (#13)
[       OK ] sysObj.noMethod (0 ms)
[----------] 2 tests from sysObj (0 ms total)

[----------] Global test environment tear-down
[==========] 330 tests from 23 test suites ran. (302 ms total)
[  PASSED  ] 330 tests.
root@7ff40d069bb0:~/pikascript/port/linux#
```

If the tests all pass, you can write the code for the functional tests.

The test code is in the test directory.


EXPLORER
LINUX [CONTAINER PIKAD...
test_module1.py
test_module2.py
test_module3.py
Release
test
python
arg-test.cpp
args-test.cpp
CMakeLists.txt
compile-test.cpp
content-test.cpp
gc-test.cpp
main.cpp
mem_test.cpp
object-test.cpp
parse-test.cpp
pika_config_gtest.c
pika_config_gtest.h
pikaMain-test.cpp
pool-test.cpp
queue-test.cpp
stack-test.cpp
strs-test.cpp
sysObj-test.cpp
test_common.h
VM-test.cpp
_gtest_once.sh
api-make-linux.sh
api-make-win10.sh
api-make.sh
OUTLINE
TIMELINE

The tests for the standard library can be placed under pikaMain-test.cpp.

The contents of a test case are as follows: first, declare a test case with the TEST macro, then fill in the name of the test group, and the name of the test case, the name of the test group can be the same as the other test cases in the current file, the test name needs to be different from the other test cases.

```
TEST(<test group>, <test name>){

    /* do something */

    /* assert */

    /* deinit */
}
```

The measurement example is divided into three main parts.

- Running
- Judgment
- Analysis

  Here is a typical test case, we copy this test case and change the name of the test case.

```
TEST(pikaMain, a_signed) {
    /* init */
    pikaMemInfo.heapUsedMax = 0;
    PikaObj* pikaMain = newRootObj("pikaMain", New_PikaMain);
    /* run */
    obj_run(pikaMain, "a = -1\n");
    /* collect */
    int a = obj_getInt(pikaMain, "a");

    /* assert */
    EXPECT_EQ(-1, a);

    /* deinit */
    obj_deinit(pikaMain);
    EXPECT_EQ(pikaMemNow(), 0);
}
```

We modify the `obj_run()` part, run a python script, and then take the result and use the EXPECT_EQ macro to determine the result.

```
TEST(pikaMain, string_startswith) {
    /* init */
    pikaMemInfo.heapUsedMax = 0;
    PikaObj* pikaMain = newRootObj("pikaMain", New_PikaMain);
    /* run */
    obj_run(pikaMain,
    "a = PikaStdData.String('test')\n"
    "res1 = a.starswith('te')\n"
    "res2 = a.startswith('st')\n"
    );
    /* collect */
```

(continues on next page)

(continued from previous page)

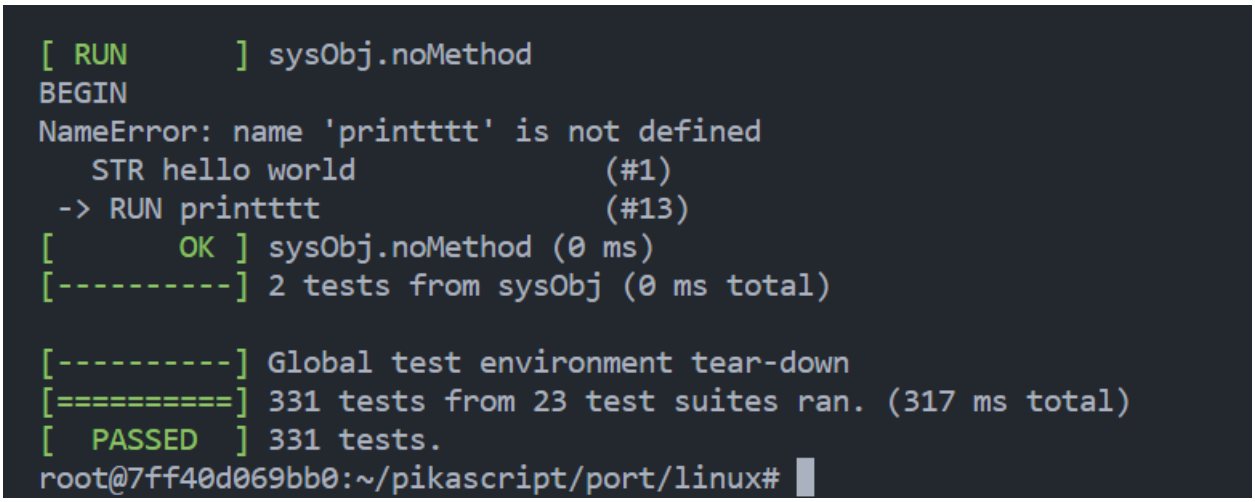
```
    int res1 = obj_getInt(pikaMain, "res1");
    int res2 = obj_getInt(pikaMain, "res2");

    /* assert */
    EXPECT_EQ(res1, 1);
    EXPECT_EQ(res2, 0);

    /* deinit */
    obj_deinit(pikaMain);
    EXPECT_EQ(pikaMemNow(), 0);
}
```

The EXPECT_EQ macro is provided by GoogleTest to determine if two values are equal, if not, GoogleTest will throw an error, you can check GoogleTest's documentation to learn more.

Then we run GoogleTest again

```
sh gtest.sh
```

As you can see, the number of test cases is 331, one more than the previous 330, and they all pass, which means the test is successful.



Commit

Once the test passes, you can commit the changes.

Before committing the changes, you need to fork the PikaPython repository, Gitee and Github are both available.

The first time you commit, you need to change your commit information, including your username, email, and the repository address after fork.

```
git config --global user.name < your user name >
git config --global user.email < your email >
git config remote.origin.url < your forked git repo url >
```

Run

```
sh push-core.sh
```

Commit the modified code to ~/pikascript/package/PikaStdLib.

Then run

```
git commit -a
```

Enter the commit information, and if you are not familiar with vim, learn the basics of using vim yourself.

```
support starswith() for PikaStdData.String()
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is up to date with 'origin/master'.
#
# Changes to be committed:
#       modified:   ../../package/PikaStdLib/PikaStdData.pyi
#       modified:   ../../package/PikaStdLib/PikaStdData_String.c
#       modified:   package/pikascript/PikaStdData.pyi
```

Next you can commit

```
git push
```

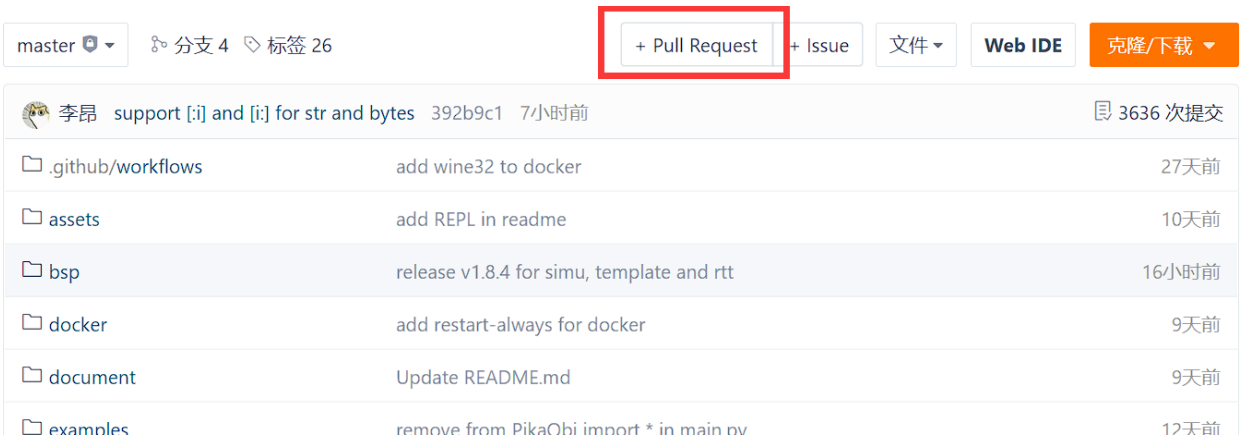
If there is a conflict, you can first

```
git pull --rebase
```

and then `git push`. For more information on how to use git, see the git manual.

```
root@7ff40d069bb0:~/pikascript/port/linux# git push
To https://gitee.com/lyon1998/pikascript
 ! [rejected]          master -> master (fetch first)
error: failed to push some refs to 'https://gitee.com/lyon1998/pikascript'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
root@7ff40d069bb0:~/pikascript/port/linux#
```

Then launch a Pull Request in gitee / github



10.5 How to contribute to the kernel

10.5.1 Development Conventions

Note: For items listed as Avoid-in-principle, if they are really required, each use-case need to be discussed separately.

Exceptions

- PLOOC - The Protected Low-overhead Object Oriented Programming
- __instruction_def.h - simplify the management of VM instructions in coding.

10.5.2 Kernel development environment

Option 1 Development under docker (recommended)

get start -> get start with docker

Option 2 pico real machine development

Prepare a copy of the Raspberry Pi pico development board, then clone the complete repository and use the bsp/pico-dev project in the repository.

10.5.3 Object-Oriented Programming with ANSI-C

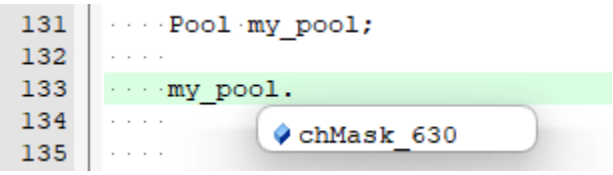
Overview

PikaPython employs the popular **Object-Oriented Programming with ANSI-C**, a.ka. **OOPC** methodology in the design and uses an open-source OOPC template, i.e. PLOOC in the kernel. In addition to the normal structure based class definition, PLOOC introduced a so-called masked-structures. With this trick, members of a class can not only be marked as **private**, **protected** and **public**, but also actually protected as ***private/protected*** as other native OO languages do, such as C++, C# etc.

For example, in the `dataMemory.h`, it defines a class `Pool`:

```
#if defined(__DATA_MEMORY_CLASS_IMPLEMENT__)
#define __PLOOC_CLASS_IMPLEMENT__
#elif defined(__DATA_MEMORY_CLASS_INHERIT__)
#define __PLOOC_CLASS_INHERIT__
#endif

#include "__pika_ooc.h"

...

struct Pool{
    private_member(
        BitMap bitmap;
        uint8_t* mem;
        uint8_t aline;
        uint32_t size;
        uint32_t first_free_block;
        uint32_t purl_free_block_start;
    )
};
```

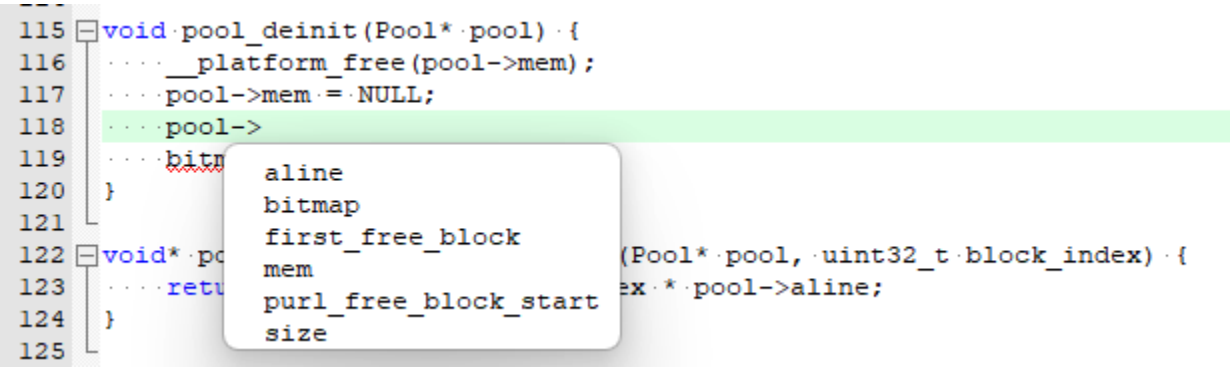
Here, all members are embraced with `private_member()`, that means outside the class scope, people cannot see/access those private members, as shown below:



While, in the `dataMemory.h`, a macro `__DATA_MEMORY_CLASS_IMPLEMENT__` is added before any includings:

```
#define __DATA_MEMORY_CLASS_IMPLEMENT__
#include "dataMemory.h"
#include "PikaPlatform.h"
...
```

hence, inside those method functions of the class `Pool`, we can see/access all members listed as private:



This is because macro `__DATA_MEMORY_CLASS_IMPLEMENT__` marks the whole `dataMemory.c` as it is inside the `Pool` class scope.

Visibility Control

PLOOC is a tool to force a visibility control in the c programming. There are plenty of ways to remove those visibility control in different scales, as shown in the **Table 3-1**:

Table 3-1 Summary of visibility controls in PLOOC

NOTE: Please use these Tokens carefully and following the OO design principles.

Rules of using PLOOC inside PikaPython

- We only use PLOOC inside kernel in principle
- Contributors are **NOT** forced to use PLOOC even contribute to the kernel.
  - Unless otherwise state, we assume that you agree that the maintainer are authorized to modify your code for adding PLOOC.

CHAPTER

ELEVEN

COLUMN TUTORIAL

11.1 STM32F429 PikaPython Practice Notes

Author: Once_day

- PikaScript(1)hello world

11.2 MM32 PikaPython Practical development

Author xld0932

- [MM32 ecology] Based on pikascript on the mm32 platform, the python development environment
- [MM32 ecology] Serial port download Python script, run snake

CHAPTER

TWELVE

SELECTED TECHNICAL ARTICLES

12.1 Issue 1

12.1.1 [MM32] Python

12.1.2 [MM32] PikaScriptMM32Python

12.1.3 HC32F460 Upython19264–PikaScript

12.1.4 [Hacker News] Python TypeScript

12.1.5 [YouTube] PikaPython Build and Test in STM32 Clone

12.1.6 [CNX] PikaPythonSTM32MCUPython

12.1.7 [YouTube] Python in STM32? w806? Not so fast... - PikaPython Review

12.1.8 [reddit] pikascript: An ultra-lightweight Python engine that can run with 4KB of RAM and 32KB of Flash (such as STM32G030C8 and STM32F103C8), and is very easy to deploy and expand.

12.1.9 [Hacker News] Pikascript: An ultra-lightweight Python engine that can run in 4Kb of RAM

12.1.10 [OpenNet] PikaPython 1.8, Python

12.1.11 [CNX] PikaPython – A lightweight Python implementation that runs on STM32 and other low-end MCUs

12.1.12 [whycan] IARUpy

CHAPTER

THIRTEEN

BUSINESS COOPERATION

13.1 General

1. The PikaPython open source project abides by the **MIT Open Source License**.

13.2 Source code usage

1. The use of the source code follows the **MIT** agreement, **no additional authorization is required**.
2. When using the PikaPython source code, **must not have any behavior or intention beyond the MIT open source agreement**.

13.3 Custom Development Services

1. Customized development services include: development board adaptation, driver development, desktop software development, server software development, circuit board design, product design, etc.
2. Custom development services **Negotiated fees based on labor volume**, and:
    1. Must sign **labor contract**.
    2. **Requirements document** must be provided
    3. The purchaser must be a **valid legal person**.
    4. The deposit must be no less than **50%**.
    5. **Documentation, maintenance, technical support** service fees are negotiable and must be signed** independent contracts, not included in custom development services. **
3. **The source code, documents, design drawings and other copyrights that have been disclosed in the PikaPython code repository still belong to the PikaPython project team**.
4. The **copyright of the custom development part belongs to the service purchaser**, and the **purchaser completely decides whether to open source, and how to use it**.

13.4 product marketing

PikaPython-based products (development boards, modules, kits, etc.) can be promoted on the PikaPython project homepage, manual, website, etc. Charges are charged according to the promotion time/form, and new products can apply for a period of free promotion.

13.5 Training Services

1. The training service is negotiated and charged according to the working time, work content and expected effect, and:
    1. **Must sign a labor contract**.
    2. Party A must be a **valid legal person**.
    3. The deposit must be no less than **30%**.
    4. After the training **service period ends**, **technical support will no longer be provided**.

CHAPTER

FOURTEEN

DEVELOPMENT MEETING

14.1 PikaPython kernel advanced

Video link

14.1.1 outline

1. Overview of kernel development

Second, the construction of the kernel development environment

1. Test Driven Development
2. Kernel distribution and upstream and downstream

14.1.2 Kernel development overview

Kernel development environment: linux

Kernel deployment environment: mcu (ARM, Risc-V, Others)

Reasons to choose linux

Kernel requirements: cross-platform capability, stability

Only cross-platform, cross-platform Only fully tested can it be stable

Development requirements: mainstream platform, convenient debugging, complete testing tools

Use mainstream platforms, mainstream technologies, and build only one wheel at a time

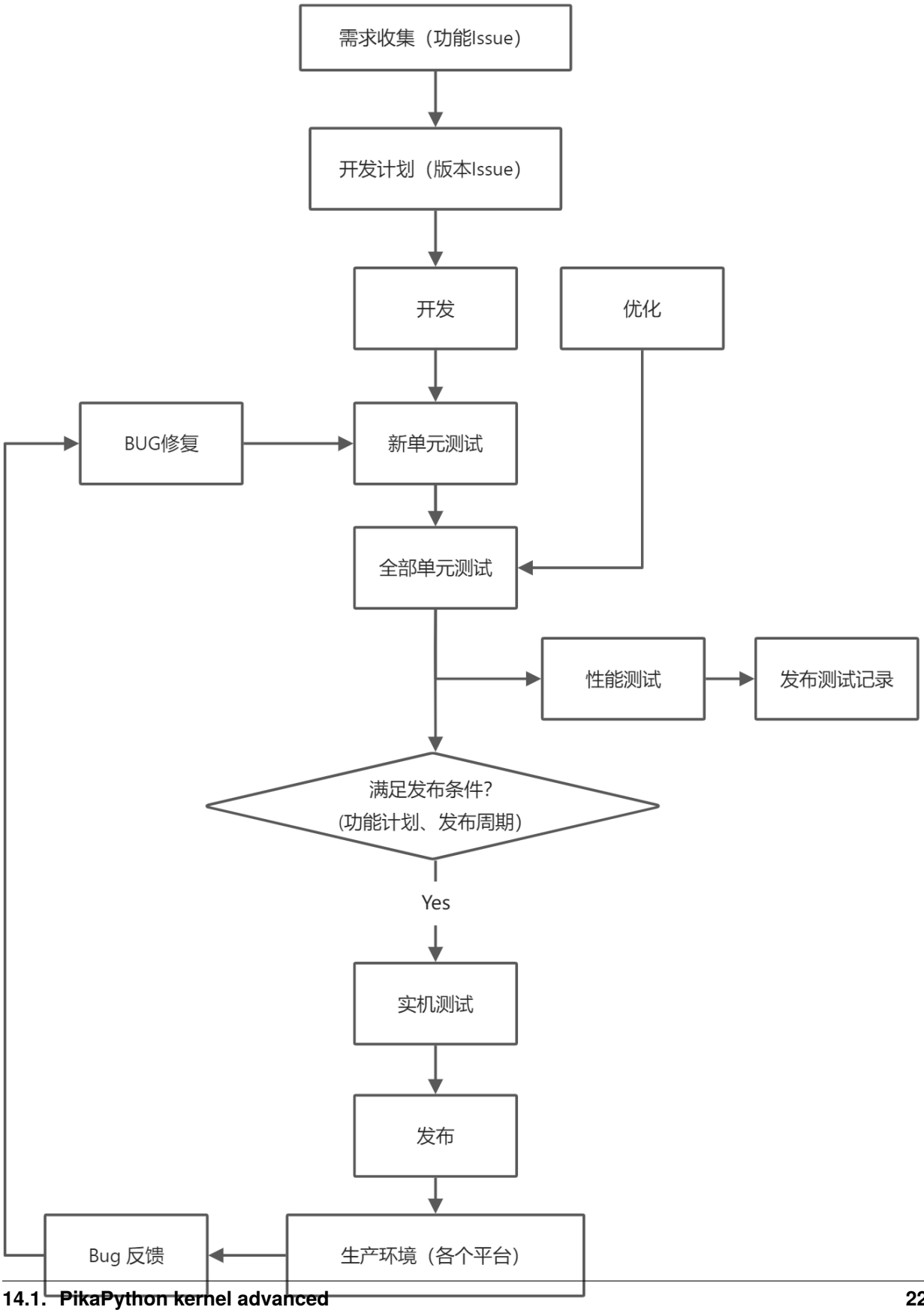
Team requirements: Avoid relying on hardware, unified development environment

Reduce the difficulty of joining new members and solve the obstacle of physical distance (the express fee is very expensive) Reduce the cost of trial and error (what should I do if the hardware test board is burned) Simplify the construction of the development environment (why can't your software be used on my computer)

Project requirements: easy to deploy automation facilities, CI-CD, easy software distribution

Automate all steps that can be automated to reduce maintenance costs

Kernel development steps



95% of the workload before the real machine test

14.1.3 Kernel environment construction

14.1.4 Test Driven Development

Implement functionality -> write unit tests

14.1.5 Kernel distribution and upstream and downstream