MICROPYTHON User Manual

RT-THREAD Document Center

WWW.RT-THREAD.ORG

Thursday 21st February, 2019

Table of contents

Table of contents i

1 Introduction to the RT-Thread MicroPython Development Manual 1

1.1 Main Features . . . 1

1.2 Advantages of MicroPython. 1

1.3 Application areas of MicroPython. 1

1.3.1 Product prototype verification. 2

1.3.2 Hardware Testing . . . 2

1.3.3 Education. 2

1.3.4 DIY Maker 2

2 RT-Thread MicroPython Quick Start 3

2.1 Getting started with MicroPython. 3

2.1.1 Select the appropriate BSP platform. 3

2.1.2 Installation of MicroPython package. 3

2.1.3 Select a development environment. . 4

2.1.4 Running MicroPython . . . 4

2.2 Basic MicroPython functions. 5

2.2.1 Python Syntax and Built-in Functions . . . 5

2.2.1.1 Using the Python interactive command line. 5

2.2.1.2 Paste mode in interactive command line. 5

2.2.2 MicroPython built-in modules. 6

2.3 MicroPython Examples. 6

2.3.1 Flashing lights. 6

2.3.2 Button light. 7

3 RT-Thread MicroPython Basics **8**

3.1 Running Python Files . . . 8

3.2 Glossary. 8

3.2.1 board . 8

3.2.2 CPython. . . 8

3.2.3 GPIO . 8

3.2.4 interned string . 9

3.2.5 MCU . 9

3.2.6 micropython-lib . 9

3.2.7 stream . 9

3.2.8 upip . . 9

4 MicroPython Module **10**

4.1 Python Standard Library and Microlibrary . . . 10

4.2 RT-Thread MicroPython module. 10

4.2.1 System Module . . . 10

4.2.2 Hardware Module . . . 11

4.2.3 System Modules . . . 11

4.2.4 Tool Module . . . 11

4.2.5 Network Module . . . 11

4.3 **rtthread** – system related functions. 11

4.3.1 Functions. 11

4.3.1.1 rtthread.current_tid() . . . 11

4.3.1.2 rtthread.is_preempt_thread() . . 11

4.3.1.3 rtthread.stacks_analyze() . 12

4.3.2 Examples. 12

4.4 **utime** – Time-related functions. 12

4.4.1 Functions. 12

4.4.1.1 **utime.localtime([secs]) .** 12

4.4.1.2 **utime.mktime() .** 13

4.4.1.3 **utime.sleep(seconds) . . .** 13

4.4.1.4 **utime.sleep_ms(ms) . .** 13

4.4.1.5 **utime.sleep_us(us) . . .** 13

4.4.1.6 **utime.ticks_ms() .** . . . 13
4.4.1.7 **utime.ticks_us() .** . . . 13
4.4.1.8 **utime.ticks_cpu() .** . . . 13
4.4.1.9 **utime.ticks_add(ticks,** delta) . . . 14
4.4.1.10 **utime.ticks_diff(ticks1,** ticks2) . . . 14
4.4.1.11 **utime.time() . . .** . . . 14
4.4.2 Examples. . . . 15
4.5 **sys** – System-specific functions 15
4.5.1 Functions. . . . 15
4.5.1.1 sys.exit(retval=0) . . . 15
4.5.2 Constants. . . . 15
4.5.2.1 **sys.argv . .** . . . 15
4.5.2.2 **sys.byteorder . . .** . . . 15
4.5.2.3 **sys.implementation . . .** . . . 15
4.5.2.4 **sys.modules .** . . . 15
4.5.2.5 **sys.path . .** . . . 16
4.5.2.6 **sys.platform . . .** . . . 16
4.5.2.7 **sys.stderr .** . . . 16
4.5.2.8 **sys.stdin . .** . . . 16
4.5.2.9 **sys.stdout .** . . . 16
4.5.2.10 **sys.version .** . . . 16
4.5.2.11 **sys.version_info .** . . . 16
4.5.3 Examples. . . . 16
4.6 **math** – Mathematical functions 17
4.6.1 Constants. . . . 17
4.6.1.1 **math.e . . .** . . . 17
4.6.1.2 **math.pi.** . . . 17
4.6.2 Functions. . . . 17
4.6.2.1 **math.acos(x) . . .** . . . 17
4.6.2.2 **math.acosh(x) . .** . . . 17
4.6.2.3 **math.asin(x) . . .** . . . 17
4.6.2.4 **math.asinh(x) .** . . . 17
4.6.2.5 **math.atan(x) . . .** . . . 18

4.6.2.6 **math.atan2(y, x) .** 18
4.6.2.7 **math.atanh(x) . .** 18
4.6.2.8 **math.ceil(x) . . .** 18
4.6.2.9 **math.copysign(x, y) . .** 18
4.6.2.10 **math.cos(x) .** 18
4.6.2.11 **math.cosh(x) . . .** 18
4.6.2.12 **math.degrees(x) .** 18
4.6.2.13 **math.erf(x) .** 18
4.6.2.14 **math.erfc(x) . . .** 19
4.6.2.15 **math.exp(x) . . .** 19
4.6.2.16 **math.expm1(x) .** 19
4.6.2.17 **math.fabs(x) . . .** 19
4.6.2.18 **math.floor(x) . . .** 19
4.6.2.19 **math.fmod(x, y) .** 19
4.6.2.20 **math.frexp(x) . .** 20
4.6.2.21 **math.gamma(x) .** 20
4.6.2.22 **math.isfinite(x) .** 20
4.6.2.23 **math.isinf(x) . . .** 20
4.6.2.24 **math.isnan(x) . .** 20
4.6.2.25 **math.ldexp(x, exp) . . .** 20
4.6.2.26 **math.lgamma(x) .** 20
4.6.2.27 **math.log(x) .** 20
4.6.2.28 **math.log10(x) . .** 21
4.6.2.29 **math.log2(x) . . .** 21
4.6.2.30 **math.modf(x) . .** 21
4.6.2.31 **math.pow(x, y) .** 21
4.6.2.32 **math.radians(x) .** 21
4.6.2.33 **math.sin(x) .** 21
4.6.2.34 **math.sinh(x) . . .** 22
4.6.2.35 **math.sqrt(x) . . .** 22
4.6.2.36 **math.tan(x) . . .** 22
4.6.2.37 **math.tanh(x) . . .** 22
4.6.2.38 **math.trunc(x) . .** 22

4.7 **uio** – Input/Output Streams. 22

4.7.1 Functions. 22

4.7.1.1 **uio.open(name,** mode='r', **kwargs) Open a file, associated with the built-in function open(). All ports (used to access the file system) need to support the mode parameter, but support other Other ports have different parameters. 22

4.7.2 Classes 23

4.7.2.1 **class uio.FileIO(...) . . .** 23

4.7.2.2 **class uio.TextIOWrapper(...) .** 23

4.7.2.3 **class uio.StringIO([string]) . . .** 23

4.7.2.4 **class uio.BytesIO([string]) . . .** 23

4.7.2.5 **getvalue() .** 23

4.8 **ucollections** – Collection and container types 23

4.8.1 Classes 23

4.8.1.1 **ucollections.namedtuple(name,** fields) 23

4.8.1.2 **ucollections.OrderedDict(...) .** 24

4.9 **ustruct** – Packing and unpacking primitive data types. 24

4.9.1 Functions. 24

4.9.1.1 **ustruct.calcsize(fmt) . .** 24

4.9.1.2 **ustruct.pack(fmt,** v1, v2, ...) 25

4.9.1.3 **ustruct.unpack(fmt,** data) 25

4.9.1.4 **ustruct.pack_into(fmt,** buffer, offset, v1, v2, ...) 25

4.9.1.5 **ustruct.unpack_from(fmt,** data, offset=0) 25

4.10 **array** – An array of numeric data. 26

4.10.1 Constructors 26

4.10.1.1 **class** array.array(typecode[, iterable]) 26

4.10.2 Methods. 26

4.10.2.1 **array.append(val) .** 26

4.10.2.2 **array.extend(iterable) . .** 26

4.11 **gc** – Controlling garbage collection. 27

4.11.1 Functions. 27

4.11.1.1 **gc.enable() .** 27

4.11.1.2 **gc.disable() .** 27

4.11.1.3 **gc.collect() .** 27

4.11.1.4 **gc.mem_alloc() .** 27

4.11.1.5 **gc.mem_free() . .** 27

4.12 **machine** – Hardware-related functions. 27

4.12.1 Functions. 27

4.12.1.1 Reset related functions. 27

4.12.1.1.1 **machine.info() .** 27

4.12.1.1.2 **machine.rest() .** 27

4.12.1.1.3 **machine.reset_cause() .** 27

4.12.1.2 Interrupt related functions. 27

4.12.1.2.1 **machine.disable_irq() .** 27

4.12.1.2.2 **machine.enable_irq(state) . .** 28

4.12.1.3 Power consumption related functions. 28

4.12.1.3.1 **machine.freq() .** 28

4.12.1.3.2 **machine.idle() .** 28

4.12.1.3.3 **machine.sleep() .** 28

4.12.1.3.4 **machine.deepsleep() .** 28

4.12.2 Constants. 28

4.12.2.1 **machine.IDLE . .** 28

4.12.2.2 **machine.SLEEP .** 28

4.12.2.3 **machine.DEEPSLEEP .** 28

4.12.2.4 **machine.PWRON_RESET .** 28

4.12.2.5 **machine.HARD_RESET . .** 28

4.12.2.6 **machine.WDT_RESET . . .** 28

4.12.2.7 **machine.DEEPSLEEP_RESET . .** 28

4.12.2.8 **machine.SOFT_RESET .** 28

4.12.2.9 **machine.WLAN_WAKE . . .** 28

4.12.2.10 **machine.PIN_WAKE .** 28

4.12.2.11 **machine.RTC_WAKE .** 28

4.12.3 Classes 29

4.12.3.1 class Pin - Controls I/O pins. 29

4.12.3.2 class I2C - I2C protocol. 29

4.12.3.3 class SPI — SPI protocol. 29

4.12.3.4 class UART - Serial port. 29

4.12.4 Examples. 29

4.13 machine.Pin 30

4.13.1 Constructors 30

4.13.1.1 **class machine.Pin(id,** mode = -1, pull = -1ývalue) 30

4.13.2 Methods. 30

4.13.2.1 **Pin.init(mode=** -1, pull= -1, *, value, drive, alt) Reinitialize the pin according to the input parameters.

Initialize the pins. Only those parameters that are specified will be set, and the status of the other pins will remain unchanged.

For detailed parameters, please refer to the above constructor. 30

4.13.2.2 **Pin.value([x]) . . .** 30

4.13.2.3 **Pin.name() .** 30

4.13.3 Constants. 31

4.13.3.1 Select pin mode: 31

4.13.3.1.1 **Pin.IN . .** 31

4.13.3.1.2 **Pin.OUT .** 31

4.13.3.1.3 **Pin.OPEN_DRAIN .** 31

4.13.3.2 Select pull-up/pull-down mode: 31

4.13.3.2.1 **Pin.PULL_UP .** 31

4.13.3.2.2 **Pin.PULL_DOWN . .** 31

4.13.3.2.3 **None . . .** 31

4.13.4 Examples. 31

4.14 machine.I2C 31

4.14.1 Constructors 31

4.14.1.1 **class machine.I2C(id=** -1, scl, sda, freq=400000) 32

4.14.2 Methods. 32

4.14.2.1 **I2C.init(scl,** sda, freq=400000) 32

4.14.2.2 **I2C.deinit() .** 32

4.14.2.3 **I2C.scan() .** 32

4.14.3 I2C basic methods. 32

4.14.3.1 **I2C.start() .** 32

4.14.3.2 **I2C.stop() .** 32

4.14.3.3 **I2C.readinto(buf,** nack=True) 32

4.14.3.4 **I2C.write(buf) . .** 33

4.14.4 I2C Standard Bus Operation. 33

4.14.4.1 **I2C.readfrom(addr,** nbytes, stop=True) . . . 33

4.14.4.2 **I2C.readfrom_into(addr,** buf, stop=True) . . 33

4.14.4.3 **I2C.writeto(addr,** buf, stop=True) . . . 33

4.14.5 Memory Operations . . . 33

4.14.5.1 **I2C.readfrom_mem(addr,** memaddr, nbytes, *, addrsize=8) ÿ addr

Read n bytes starting from the specified address memaddr in the slave device.

The length of the address specified by the number. Returns a bytes object storing the data read. . . . 33

4.14.5.2 **I2C.readfrom_mem_into(addr,** memaddr, buf, *, addrsize=8) ÿ

addr specifies the address memaddr in the slave device to read data into buf.

The number of bytes retrieved is the length of buf. This method has no return value. . . . 33

4.14.5.3 **I2C.writeto_mem(addr,** memaddr, buf, *, addrsize=8)

The data is written to the memaddr address of the slave specified by addr. This method does not return

Return value. . . . 33

4.14.6 Examples. . . . 33

4.14.6.1 Software Simulation of I2C . . . 33

4.14.6.2 Hardware I2C . . . 34

4.15 machine.SPI . . . 34

4.15.1 Constructors . . . 34

4.15.1.1 **class machine.SPI(id,** ...) . . . 34

4.15.2 Methods. . . . 35

4.15.2.1 SPI.init(baudrate=1000000, *, polarity=0, phase=0, bits=8, first-

bit=SPI.MSB, sck=None, mosi=None, miso=None) . . 35

4.15.2.2 **SPI.deinit() .** . . . 35

4.15.2.3 SPI.read(nbytes, write=0x00) . . 35

4.15.2.4 **SPI.readinto(buf,** write=0x00) . . . 35

4.15.2.5 **SPI.write(buf) . .** . . . 35

4.15.2.6 **SPI.write_readinto(write_buf,** read_buf) . . 35

4.15.3 Constants. . . . 36

4.15.3.1 **SPI.MASTER . .** . . . 36

4.15.3.2 **SPI.MSB .** . . . 36

4.15.3.3 **SPI.LSB . .** . . . 36

4.15.4 Examples. . . . 36

4.15.4.1 Software Simulation SPI . . . 36

4.15.4.2 Hardware SPI 36
4.16 machine.UART 37
4.16.1 Constructors 37
4.16.1.1 **class machine.UART(id,** ...) 37
4.16.2 Methods. 37
4.16.2.1 UART.init(baudrate = 9600, bits=8, parity=None, stop=1) 37
4.16.2.2 **UART.deinit() . .** 37
4.16.2.3 UART.read([nbytes]) 37
4.16.2.4 **UART.readinto(buf[,** nbytes]) 37
4.16.2.5 **UART.readline() .** 37
4.16.2.6 **UART.write(buf) .** 38
4.16.3 Examples. 38
4.17 **uos** – Basic “operating system” services 38
4.17.1 Functions. 38
4.17.1.1 **uos.chdir(path) . .** 38
4.17.1.2 **uos.getcwd() . . .** 38
4.17.1.3 **uos.listdir([dir]) . .** 38
4.17.1.4 **uos.mkdir(path) .** 38
4.17.1.5 **uos.remove(path) .** 38
4.17.1.6 **uos.rmdir(path) .** 39
4.17.1.7 uos.rename(old_path, new_path) 39
4.17.1.8 **uos.stat(path) . . .** 39
4.17.1.9 **uos.sync() .** 39
4.17.2 Examples. 39
4.18 **uselect** – Wait for stream events. 39
4.18.1 Constants. 40
4.18.1.1 **select.POLLIN** — Read available data. 40
4.18.1.2 **select.POLLOUT** — Write more data 40
4.18.1.3 **select.POLLERR** - An error occurred. 40
4.18.1.4 **select.POLLHUP** — End-of-stream/connection termination detection 40
4.18.2 Functions. 40
4.18.2.1 **select.select(rlist,** wlist, xlist[, timeout]) 40
4.18.3 Poll class. 40

4.18.3.1 **select.poll() .** 40

4.18.3.2 **poll.register(obj[,** eventmask]) 41

4.18.3.3 **poll.unregister(obj) . . .** 41

4.18.3.4 **poll.modify(obj,** eventmask) 41

4.18.3.5 poll.poll([timeout]) 41

4.19 **uctypes** – Access binary data in a structured manner. 42

4.19.1 Constants. 42

4.19.2 Constructors 42

4.19.2.1 class uctypes.struct(addr, descriptor, type) 42

4.19.3 Methods. 42

4.19.3.1 **uctypes.sizeof(struct) . .** 42

4.19.3.2 **uctypes.addressof(obj) .** 43

4.19.3.3 **uctypes.bytes_at(addr,** size) 43

4.19.3.4 **uctypes.bytearray_at(addr,** size) 43

4.20 **uerrno** – System error code module. 43

4.20.1 Examples. 43

4.21 _thread – Multithreading support. 44

4.21.1 Examples. 44

4.22 **cmath** – Mathematical functions on complex numbers. 44

4.22.1 Functions. 44

4.22.1.1 **cmath.cos(z) . . .** 44

4.22.1.2 **cmath.exp(z) . . .** 44

4.22.1.3 **cmath.log(z) . . .** 44

4.22.1.4 **cmath.log10(z) . .** 44

4.22.1.5 **cmath.phase(z) . .** 44

4.22.1.6 **cmath.polar(z) . .** 45

4.22.1.7 **cmath.rect(r,** phi) 45

4.22.1.8 **cmath.sin(z) . . .** 45

4.22.1.9 **cmath.sqrt(z) . . .** 45

4.22.2 Constants. 45

4.22.2.1 **cmath.e . .** 45

4.22.2.2 **cmath.pi . .** 45

4.23 **ubinascii** – Binary/ASCII conversion. 45

4.23.1 Functions. 45

4.23.1.1 **ubinascii.hexlify(data[,** sep]) 45

4.23.1.2 **ubinascii.unhexlify(data) .** 46

4.23.1.3 **ubinascii.a2b_base64(data) . .** 46

4.23.1.4 **ubinascii.b2a_base64(data) . .** 46

4.24 **uhashlib** – Hash algorithms. 46

4.24.1 Algorithm Function 46

4.24.1.1 **SHA256 . .** 46

4.24.1.2 **SHA1 . . .** 46

4.24.1.3 **MD5 .** 46

4.24.2 Functions. 47

4.24.2.1 **class uhashlib.sha256([data]) .** 47

4.24.2.2 **class uhashlib.sha1([data]) . . .** 47

4.24.2.3 **class uhashlib.md5([data]) . . .** 47

4.24.2.4 **hash.update(data) .** 47

4.24.2.5 **hash.digest() . . .** 47

4.24.2.6 **hash.hexdigest() .** 47

4.25 **uheapq** – Heap sort algorithm. 47

4.25.1 Functions. 47

4.25.1.1 **uheapq.heappush(heap,** item) 47

4.25.1.2 **uheapq.heappop(heap) .** 47

4.25.1.3 **uheapq.heapify(x) .** 47

4.26 **ujson** – JSON encoding and decoding. 48

4.26.1 Functions. 48

4.26.1.1 **ujson.dumps(obj) .** 48

4.26.1.2 **ujson.loads(str) . .** 48

4.27 **ure** – Regular Expressions 48

4.27.1 Matching Character Sets. 48

4.27.1.1 Matches any character. 48

4.27.1.2 Matches a character set, supporting a single character and a range. 48

4.27.1.3 Supports multiple matching metacharacters. 49

4.27.2 Functions. 49

4.27.2.1 **ure.compile(regex) .** 49

4.27.2.2 **ure.match(regex,** string) 49

4.27.2.3 **ure.search(regex,** string) 49

4.27.2.4 **ure.DEBUG . . .** 49

4.27.3 Regular Expression Objects: 49

4.27.3.1 **regex.match(string) . . .** 49

4.27.3.2 **regex.search(string) . . .** 49

4.27.3.3 **regex.split(string,** max_split=-1) 49

4.27.4 Matching Objects: 49

4.27.4.1 **match.group([index]) . .** 49

4.28 **uzlib** – zlib decompression. 50

4.28.1 Functions. 50

4.28.1.1 **uzlib.decompress(data) .** 50

4.29 **urandom** - Random number generation module. 50

4.29.1 Functions. 50

4.29.1.1 **urandom.choice(obj) . .** 50

4.29.1.2 **urandom.getrandbits(size) . . .** 50

4.29.1.3 **urandom.randint(start,** end) 51

4.29.1.4 **urandom.random() . . .** 51

4.29.1.5 **urandom.randrange(start,** end, step) 51

4.29.1.6 **urandom.seed(sed) .** 52

4.29.1.7 **urandom.uniform(start,** end) 53

4.30 **usocket** – The socket module. 53

4.30.1 Constants. 53

4.30.1.1 Address Clusters 53

4.30.1.2 Socket Types. 53

4.30.1.3 IP protocol number. 53

4.30.1.4 Socket Option Levels. 53

4.30.2 Functions. 54

4.30.2.1 socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP) 54

4.30.2.2 **socket.getaddrinfo(host,** port) 54

4.30.2.3 **socket.close() . . .** 54

4.30.2.4 **socket.bind(address) . . .** 54

4.30.2.5 **socket.listen([backlog]) .** 54

4.30.2.6 **socket.accept() . .** 54

4.30.2.7 **socket.connect(address) .** 54

4.30.2.8 **socket.send(bytes) .** 55

4.30.2.9 **socket.recv(bufsize) . . .** 55

4.30.2.10 **socket.sendto(bytes,** address) 55

4.30.2.11 **socket.recvfrom(bufsize) .** 55

4.30.2.12 **socket.setsockopt(level,** optname, value) 55

4.30.2.13 **socket.settimeout(value) .** 56

4.30.2.14 **socket.setblocking(flag) .** 56

4.30.2.15 **socket.read([size]) .** 56

4.30.2.16 **socket.readinto(buf[,** nbytes]) 56

4.30.2.17 **socket.readline() .** 56

4.30.2.18 **socket.write(buf) .** 56

4.30.3 Examples. 56

4.30.3.1 TCP Server example 56

4.30.3.2 TCP Client example 57

5 RT-Thread MicroPython Package Management **58**

5.1 1. Using the micropython-lib source code 58

5.1.1 1.1 Clone/download the source code of micropython-lib from GitHub to your local computer. 58

5.1.2 1.2 Using extension packages. 58

5.2 2. Using the upip package manager 59

6 RT-Thread MicroPython Network Programming Guide **62**

6.1 Preliminary knowledge 62

6.2 HttpClient 62

6.2.1 Obtain and install the urequests module. 62

6.2.2 Use of the urequests module. 63

6.3 HttpServer 63

6.3.1 Obtain and install the MicroWebSrv module. 63

6.3.2 Use of MicroWebSrv module. 64

6.3.3 Modification of server functions. 67

6.4 MQTT 68

6.4.1 Obtain and install the umqtt.simple module. 68

6.4.2 Use of the umqtt.simple module. 68

6.4.2.1 MQTT subscription function. 68

6.4.2.2 MQTT publishing function. 69

6.5 OneNET. 70

6.5.1 Preparation 70

6.5.2 Product Creation 70

6.5.2.1 User Registration. 70

6.5.2.2 Product Creation. 70

6.5.3 Hardware Access 71

6.5.3.1 Device registration and access. 71

6.5.3.2 The cloud platform sends commands to the device. 72

6.5.3.3 The device uploads data to the cloud platform. 73

6.5.3.4 Add a standalone application. 75

6.5.4 Code Explanation 75

6.5.4.1 Appendix Example Code. 76

Chapter 1

Introduction to the RT-Thread MicroPython Development Manual

1.1 Main Features

- MicroPython is a streamlined and efficient implementation of the Python 3 programming language that includes a subset of the Python standard library.
  and is optimized to run on microcontrollers and constrained environments.
- RT-Thread MicroPython can run on any embedded platform equipped with RT-Thread operating system and with certain resources
  superior.
- MicroPython can run on a development board with certain resources, giving you a low-level Python operating system that can be used to control
  Make various electronic systems.
- MicroPython is rich in advanced features, such as interactive prompts, arbitrary precision integers, closure functions, list comprehensions, generators,
  Exception handling, etc.
- MicroPython aims to be as compatible as possible with regular Python, allowing developers to easily transfer code from the desktop to microcontrollers or embedded systems. Program portability is strong,
  as there is no need to consider the underlying drivers, so program porting becomes easy and simple.

1.2 Advantages of MicroPython

- Python is an easy-to-use scripting language with powerful functions and elegant and simple syntax. Programming with MicroPython
  It can lower the threshold for embedded development and allow more people to experience the fun of embedded systems.
- Use MicroPython to access and control the underlying hardware, making it easy to control the hardware without having to understand the underlying registers, datasheets, manufacturer's library functions,
  etc. • Peripherals and common functions have

corresponding modules, reducing development difficulty and making development and porting easy and fast.

1.3 Application Areas of MicroPython

- MicroPython fully implements the core functions of Python3 on embedded systems, which can be used in various stages of product development.
  Bring convenience to those who use it.
- Through the libraries and functions provided by MicroPython, developers can quickly control LEDs, LCDs, servos, various sensors, SD,
  UART, I2C, etc., to achieve various functions without having to study the use of the underlying hardware modules and read the register manual.

This not only reduces development difficulty but also reduces repetitive development work, speeding up development and improving development efficiency. Functions that previously required highly skilled embedded engineers to spend days or even weeks to complete can now be achieved by ordinary embedded developers in just a few hours.

- With the continuous development of semiconductor technology, the functions of chips, internal memory capacity and resources are increasing, and the cost is decreasing. There will be more and more application areas where MicroPython will be used for development and design.

1.3.1 Product Prototype Verification

- As we all know, prototyping is a crucial step in developing new products. This step requires quickly designing a rough model of the product and validating business processes or technical points. Compared to traditional development methods, using MicroPython is very useful for prototyping, making the prototyping process easier and faster.

- When developing IoT functions, MicroPython's network functionality is also a strength, allowing you to leverage the many readily available MicroPython network modules, saving development time. These functions, if implemented using C/C++, would take several times longer.

1.3.2 Hardware Testing

Embedded product development typically involves both hardware and software development. Hardware engineers aren't always skilled in software development, so software engineers are often required to participate in testing new hardware. This can lead to software engineers spending significant time helping hardware engineers identify design or soldering issues. With MicroPython, simply burning the MicroPython firmware to the new hardware to be tested allows them to check for soldering, wiring, and other issues using simple Python commands. This allows hardware engineers to handle the entire process without having to rely on others.

1.3.3 Education

- MicroPython is easy to use and convenient, making it a great introduction to programming. Students and hobbyists can quickly develop interesting projects with MicroPython, learning programming concepts and improving their hands-on skills in the process.
- Here are some MicroPython educational projects:
  - Start your programming journey with TurnipBit
  - MicroBit Creative Programming

1.3.4 DIY

- MicroPython does not require complex settings, special software environments, or additional hardware. You can program with any text editor. Most hardware functions can be driven with a single command, allowing for rapid development without having to understand the underlying hardware.

  This makes MicroPython very suitable for makers to develop some creative projects. • Here are some DIY projects made with MicroPython:

  - **WIFI** clock showing temperature and humidity
  - OpenMV Smart Camera **–Mobile** phone remote control car **–**
  - **Build** an MQTT server

Chapter **2**

RT-Thread MicroPython Quick Start

2.1 Getting Started with **MicroPython**

2.1.1 Selecting a suitable **BSP** platform

The maximum resource usage of the RT-Thread MicroPython mini version does not exceed:

- 1ÿROM : 190KB
- 2ÿRAM : 8KB

Many common development boards can run MicroPython, such as stm32f10x , stm32f40x , stm32f429-apollo, imxrt1052 -evk , iot-camera , etc.

2.1.2 Installation of MicroPython Package

- The MicroPython package can be downloaded online using the env tool. Before downloading the MicroPython package, it is recommended to use

  The pkgs --upgrade command updates the software package list and selects the latest version when configuring the version , as shown in the figure:

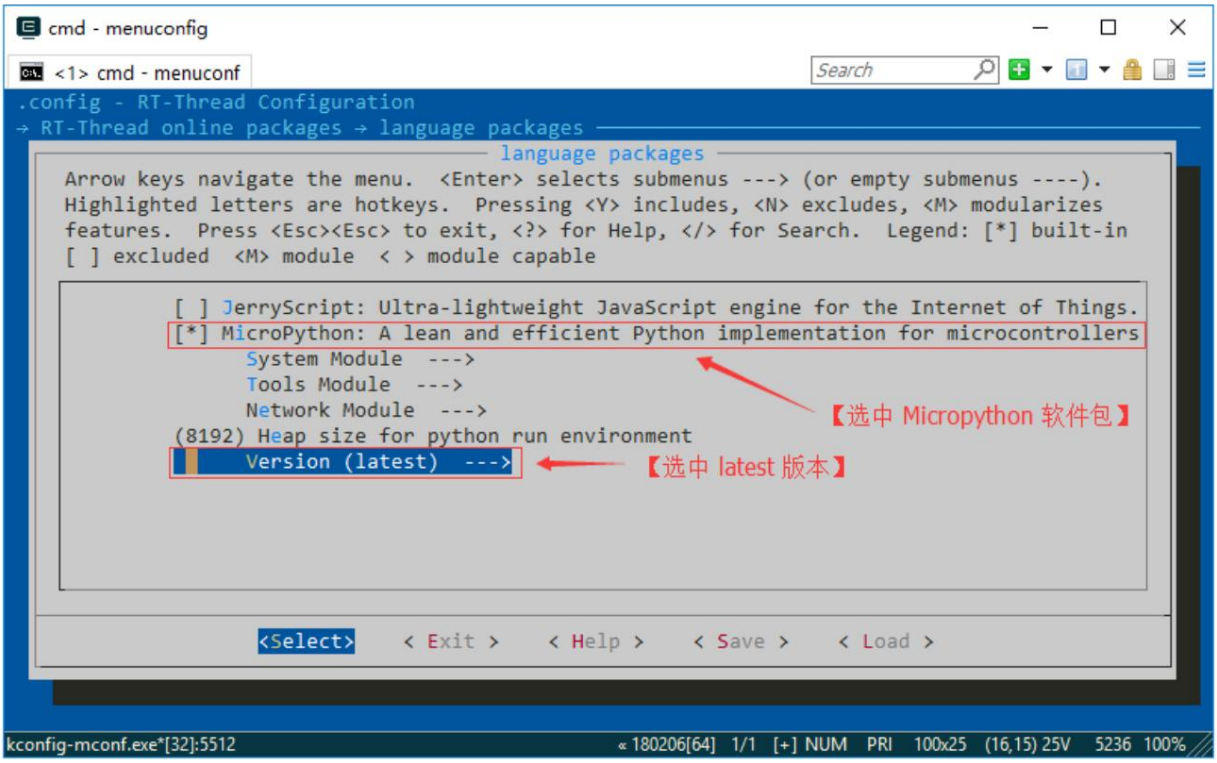

Figure **2.1:** *elect_micropytho*

- For instructions on how to use env to download MicroPython packages, please refer to: RT-Thread env Tool User Manual

2.1.3 Selecting a Development Environment

- Currently MicroPython supports three development environments: MDK / IAR / GCC. Select the appropriate development environment and use the env tool.

  After opening the MicroPython software package, you need to regenerate the project, compile, and download it.

2.1.4 Running **MicroPython**

- Type python in the Finsh/MSH command line to enter the MicroPython interactive command line REPL (Read-Evaluate-

  Print-Loop), you can see the following interface in the terminal:

```
 \ | /
- RT -     Thread Operating System
 / | \     3.0.3 build Mar 22 2018
 2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
using iccarm, version: 8020001
build time: Mar 22 2018 18:50:15
msh />[PHY] wait autonegotiation complete...
sdcard init fail or timeout: -2!
[PHY] wait autonegotiation complete...

msh />python

MicroPython v1.9.3-477-g7b0a020-dirty on 2018-03-21; Universal python platform with RT-Thread
Type "help()" for more information.
>>>
```

Figure **2.2:** *elect_micropytho*

Use Ctrl-D or enter quit() and exit() to exit REPL and return to RT-Thread Finsh/MSH.

2.2 Basic Functions of MicroPython

2.2.1 Python Syntax and Built-in Functions

2.2.1.1 Using the Python interactive command line

- MicroPython is a streamlined and efficient implementation of the Python 3 programming language, with the same syntax as Python 3 and rich Built-in functions, you can run Python code using the MicroPython interactive command line:

Figure **2.3:** *elect_micropytho*

2.2.1.2 Paste Mode in Interactive Command Line

- MicroPython has a special paste mode compared to the general Python interactive environment, which allows you to paste multiple lines of Python at a time. code.
- At the command prompt, press Ctrl-E . A prompt will appear: paste mode; Ctrl-C to cancel, Ctrl -D to finish . After pasting the code you want to run, press Ctrl-D to exit paste mode. The entered code will also be automatically executed.
- If you want to cancel a program while it is running, you can use Ctrl-C.

Enter the code:

```
for i in range(1,10):
    print(i)
```

The execution effect is as follows:

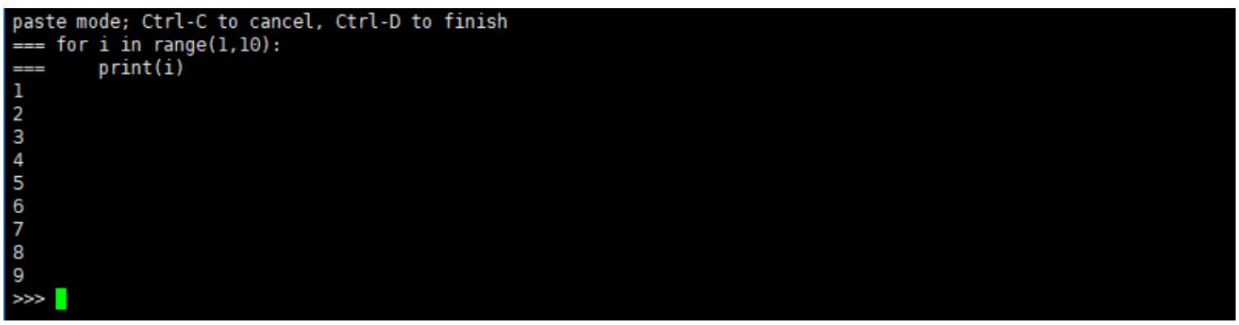

Figure **2.4:** *elect_micropytho*

2.2.2 MicroPython built-in modules

- MicroPython provides a rich set of built-in modules to complete related program functions. At the same time, RT-Thread also provides rtthread

  The module is used to return information related to system

operation. • Taking the rtthread and time modules as an example, the calling method is as follows:

```
>>> import rtthread
GC: total: 7936, used: 704, free: 7232
 No. of 1-blocks: 11, 2-blocks: 6, max blk sz: 8, max free sz: 123
>>> rtthread.is_preempt_thread()
True
>>> rtthread.current_tid()
536935332
>>> rtthread.stacks_analyze()
thread       pri  status      sp     stack size max used left tick  error
------------ ---  ------- ---------- ----------  ------  ---------- ---
tshell        20  ready   0x000001e8 0x00001000    36%   0x00000002 000
ntp_sync      26  suspend 0x0000007c 0x00000600    29%   0x00000001 000
phy           30  suspend 0x00000090 0x00000200    46%   0x00000002 000
tcpip         10  suspend 0x000000c4 0x00000800    16%   0x00000003 000
etx           12  suspend 0x00000098 0x00000400    14%   0x00000010 000
erx           12  suspend 0x000000a0 0x00000400    43%   0x00000008 000
mmcsd_detect  22  suspend 0x000000a0 0x00000400    47%   0x00000012 000
tidle         31  ready   0x0000005c 0x00000100    35%   0x00000015 000
main          10  suspend 0x0000008c 0x00000800    24%   0x00000009 000
>>> import time
>>> time.sleep(1)
>>> time.sleep_
sleep_ms        sleep_us
>>> time.sleep_ms(500)
>>> time.
__class__       __name__        sleep           sleep_ms
sleep_us        ticks_add       ticks_cpu       ticks_diff
ticks_ms        ticks_us        time
>>> time.sleep_us(10)
>>>
```

Figure **2.5:** *elect_micropytho*

Tip: The default MicroPython package downloaded is the mini version, which is the smallest version of MicroPython released by RT-Thread. If you want to use more MicroPython modules, you can enable more module options in the menuconfig configuration item.

2.3 MicroPython Examples

MicroPython allows you to control the hardware resources of the development board in a very simple way. The following two examples illustrate this:

The following example runs on the i.MX RT1050 development board. Before running, you need to enable the RT-Thread Pin device function.

2.3.1 Flashing Light

- i.MX RT1050 development board: pin 52 LED D18, shared with phy reset pin

```
import time
from machine import Pin

LED = Pin(("LED1", 52), Pin.OUT_PP)                    #Set Pin 52 device to output mode
while True:
        LED.value(1)
```

```
        time.sleep_ms(500)
        LED.value(0)
        time.sleep_ms(500)
```

Modify the pin number for your own development board and input the above script using the paste mode described in Section 3.1.2. You can see the LED light on. Flashes at the specified frequency. Press Ctrl-C to cancel the currently running program.

2.3.2 Button Light

- i.MX RT1050 development board: pin 125 For SW8

```
from machine import Pin

led = Pin(("LED1", 52), Pin.OUT_PP) key =
Pin(("KEY", 125), Pin.IN, Pin.PULL_UP) #Set Pin 125 to pull-up input mode
while True:
        if key.value():
                led.value(0)
        else:
                led.value(1)
```

Modify the pin number for your own development board, use paste mode to input the above script, and you can control the brightness of the LED light by pressing the KEY key. Destroy.

Chapter 3

RT-Thread MicroPython Basics

3.1 Run Python file

Running Python files on MicroPython has the following requirements:

• The system uses the rt-thread file system. • The msh function is enabled.

If the above two points are met, you can use the python command in the msh command line plus the *.py file name to execute a Python file.

3.2 Glossary

3.2.1 board

Development board, usually this term is used to refer to a development board with a specific MCU as the core. It can also be used to refer to a port MicroPython to a specific development board can also mean a port without a development board, such as a Unix port.

3.2.2 CPython

CPython is an implementation of the Python programming language, the most well-known and used. However, it is just one of many implementations (including Jython, IronPython, PyPy, MicroPython , and others). Since there is no official Python language specification, only CPython documentation, it is difficult to draw a clear line between the Python language itself and implementations like CPython . This also leaves more freedom for other implementations. For example, MicroPython does many things differently from CPython while still being a popular choice.

An implementation of the Python language.

3.2.3 GPIO

General Purpose Input/Output. The simplest way to control electronic signals. Through GPIO, users can configure hardware signal pins as input or output. And set or get its digital signal value (logical '0' or '1'). MicroPython uses the abstract class machine.Pin to access GPIO.

3.2.4 interned string

A string referenced by its unique identifier rather than its address. Thus, interned strings can be quickly compared by their identifier rather than by their content . The disadvantage of interned strings is that insertion takes time (proportional to the number of existing interned strings, meaning it takes longer over time), and the space used to insert interned strings is not reclaimable. When an interned string is required by the application (as a keyword argument) or beneficial to the system (by reducing lookup time), it is automatically generated by the MicroPython compiler and runtime environment. Due to the above disadvantages, most string and input/output operations do not generate interned strings.

3.2.5 MCU

Microcontrollers, also known as single-chip computers, usually have fewer resources than full-fledged computer systems, but are smaller, cheaper, and use less power. Low. MicroPython is designed to be small enough and optimized to run on a common modern microcontroller.

3.2.6 micropython-lib

MicroPython is usually a single executable/binary file with very few built-in modules. It does not have an extensive standard library that can be compared to CPython . Unlike CPython , MicroPython has a related but independent project micropython-lib, which Provides implementations of many modules from the CPython standard library. However, most of these modules require a POSIX- like environment and can only work on Unix ports of MicroPython. The installation method is also different from CPython and requires manual copying or installation using upip . Since the RT-Thread operating system provides good support for the POSIX standard, micropython-lib Many modules can run on RT-Thread MicroPython .

3.2.7 stream

Also known as a file-like object, a stream object provides sequential read and write access to underlying data. The stream object implements a corresponding interface, including methods such as read(), write(), readinto(), seek( ), flush(), and close() . Streams are an important concept in MicroPython, and many input/output objects implement the stream interface so they can be used consistently in different contexts. For more information about MicroPython streams, see uio.

3.2.8 upip

Literally meaning "miniature pip ," upip is a MicroPython package manager inspired by CPython , but much smaller and with fewer features. It can run on Unix ports of MicroPython . Because the RT-Thread operating system provides excellent support for the POSIX standard, upip can also run on RT-Thread MicroPython . Using upip , you can download MicroPython extension modules online and automatically download their dependent modules, greatly facilitating the expansion of MicroPython functionality.

Chapter 4

MicroPython modules

4.1 Python Standard Library and Microlibrary

After Python's standard library is "miniaturized", it becomes the MicroPython standard library, also known as MicroPython modules. They only provide the core functions of the module and are used to replace the Python standard library. Some modules use the name of the Python standard library, but with the prefix "u", such as ujson instead of json. In other words, the MicroPython standard library (micro library) only implements a part of the module functions. By naming these libraries differently, users can write a Python-level module to expand the functionality of the micro library to facilitate compatibility with the CPython standard library (this work is micropython-lib projects in progress).

On some embedded platforms, Python-level wrapper libraries can be added to achieve naming compatibility with CPython. Modules using the MicroPython standard library can use either their u-name or non-u-name. Modules using non-u-names can be overwritten by modules of the same name in the library path folder.

For example, when importing json, it will first search for a json.py file or a json directory in the library path folder to load. If not found, it will load the built-in ujson module.

4.2 RT-Thread MicroPython Module

4.2.1 System Module

• rtthread – RT-Thread system-related functions • utime – time-related functions • sys – system-specific functions • math – mathematical functions • uio – input/output streams • ucollections – collection and container types • ustruct – packing and unpacking primitive data types • array – arrays of numeric data • gc – control garbage collection

4.2.2 Hardware Module

- machine – hardware-related functionality
- machine.Pin
- machine.I2C
- machine.SPI
- machine.UART

4.2.3 System Module

- uos – basic “operating system” services •

select – wait for stream

events • uctypes – access binary data in a structured

manner • uerrno – system error code

module • _thread – multithreading support

4.2.4 Tool Module

- cmath – Mathematical functions for

complex numbers • ubinascii – Binary/ASCII

conversion • uhashlib – Hash

algorithms • uheapq – Heap sort

algorithm • ujson – JSON encoding and

decoding • ure – Regular

expressions • uzlib – Zlib

decompression • urandom – Random number generation module

4.2.5 Network Module

- usocket – socket module

4.3 rtthread – system-related functions

4.3.1 Function

4.3.1.1 rtthread.current_tid()

Returns the id of the current thread.

4.3.1.2 rtthread.is_preempt_thread()

Returns whether the thread is preemptible.

4.3.1.3 rtthread.stacks_analyze()

Returns the current system thread and stack usage information.

4.3.2 Examples

```
>>> import rtthread
>>>
>>> rtthread.is_preempt_thread() preemptible          # determine if code is running in a
    thread
True
>>> rtthread.current_tid()                            # current thread id
268464956
>>> rtthread.stacks_analyze() pri status              # show thread information
thread                                  sp        stack size max used left tick error
------------------------------------------------
elog_async 31 suspend 0x000000a8 0x00000400 20 ready          26% 0x00000003 000
tshell           0x00000260 0x00001000 31 ready 0x00000070    39% 0x00000003 000
tidle            0x00000100                                   51% 0x0000000f 000
SysMonitor 30 suspend 0x000000a4 0x00000200 4 suspend         32% 0x00000005 000
timer             0x00000080 0x00000200                       25% 0x00000009 000
>>>
```

4.4 utime – Time-related functions

Initial time: Unix uses the POSIX system standard, starting at 1970-01-01 00:00:00 UTC . Embedded programs start at 2000-01-01 Starts at 00:00:00 UTC .

Keeping the actual calendar date/time: A real-time clock (RTC) is required. In the underlying system (including some RTOS), the RTC is already included Among them, setting the time is done through OS/RTOS instead of MicroPython, and querying the date/time also needs to be done through the system API. The system clock of the board depends on the machine.RTC() object. Set the time through the machine.RTC().datetime(tuple) function and Maintaining the surface:

- Backup battery (may be an option, expansion board, etc.).
- Use Network Time Protocol (requires user setup).
- Manually set each time the power is turned on (most of them only need to be set during a hard reset, and a few need to be set during each reset).

If the actual time is not maintained by the system/MicroPython RTC, the following function results may not be the same as expected.

4.4.1 Function

4.4.1.1 utime.localtime([secs])

Converts the seconds of the original time to a tuple: (year, month, day, hour, minute, second, weekday, yearday) . If secs is empty or None, then Use the current time.

• year includes the century (e.g., 2014). • month ranges from 1-12 • day ranges from 1-31 • hour ranges from 0-23 • minute ranges from 0-59 • second ranges from 0-59 • weekday ranges from 0-6, corresponding to Monday through Sunday • yearday ranges from 1-366

4.4.1.2 utime.mktime()

The inverse function of time, which takes a tuple of 8 arguments and returns an integer representing the number of seconds since January 1, 2000.

4.4.1.3 utime.sleep(seconds)

Sleep for a specified time (in seconds). Seconds can be a floating point number. Note that some versions of MicroPython do not support floating point numbers. You can use the sleep_ms() and sleep_us() functions.

4.4.1.4 utime.sleep_ms(ms)

The delay is specified in milliseconds. The parameter cannot be less than 0.

4.4.1.5 utime.sleep_us(us)

The delay is specified in microseconds. The parameter cannot be less than 0.

4.4.1.6 utime.ticks_ms()

Returns a continuously incrementing millisecond counter that resets after some (unspecified) value. The count value itself has no special meaning and is only suitable for use with ticks_diff(). Note: Performing standard mathematical operations (+, -) or relational operators (<, >, >=) directly on these values will produce invalid results. Performing mathematical operations and then passing the result as an argument to ticks_diff() or ticks_add() will also produce invalid results.

4.4.1.7 utime.ticks_us()

Similar to ticks_ms() above , but returns microseconds.

4.4.1.8 utime.ticks_cpu()

Similar to ticks_ms() and ticks_us() , but with higher precision (using the CPU clock). Not every port implements this function.

4.4.1.9 utime.ticks_add(ticks, delta)

Given a positive or negative number as a tick offset delta, this function allows you to calculate a tick value delta ticks before or after the given tick value, using the modulo arithmetic definition of the tick value . The ticks argument must be the result of a function called ticks_ms(), ticks_us(), or ticks_cpu() . However, delta can be any integer or numeric expression. The ticks_add function is useful for calculating deadlines for events/tasks. (Note: ticksdiff() must be used to handle deadlines.)

Code example:

```
## Find the beat value 100ms ago
print(utime.ticks_add(utime.ticks_ms(), -100))

## Calculate the deadline for the operation and then test it
deadline = utime.ticks_add(utime.ticks_ms(), 200) while
utime.ticks_diff(deadline, utime.ticks_ms()) > 0:
        do_a_little_of_something()

## Find the maximum value of the transplant beat value
print(utime.ticks_add(0, -1))
```

4.4.1.10 utime.ticks_diff(ticks1, ticks2)

Calculates the time between two calls to ticksms(), ticks_us(), or ticks_cpu(). Because the counts of these functions may wrap around, they cannot be directly subtracted; the ticks_diff() function must be used instead. The "old" time must be before the "new" time, otherwise the result is undefined. This function should not be used to calculate very long times (because the ticks*() functions wrap around, and the period is usually not very long). It is usually called from a polling event with a timeout:

Code example:

```
## Wait for the GPIO pin to be valid, but wait for up to 500 microseconds
start = time.ticks_us() while
pin.value() == 0: if
        time.ticks_diff(time.ticks_us(), start) > 500:
                raise TimeoutError
```

4.4.1.11 utime.time()

Returns the number of seconds since the last time (an integer), assuming the RTC has been set as described above. If the RTC is not set, the function returns the number of seconds since the reference point (after power-up or reset, for boards without a battery-backed RTC). If you are developing portable MicroPython applications, you should not rely on this function to provide better than second precision. If high precision is required, use the ticks_ms() and ticks_us() functions. If calendar time is required, localtime() with no arguments is a better choice.

Tip: "Differences from CPython" In CPython , this function returns the number of seconds since the Unix epoch (1970-01-01 00:00 UTC) as a floating-point number, typically with millisecond precision. In MicroPython, only the Unix version uses the same epoch, and if floating-point precision is enabled, sub-second precision is returned. Embedded hardware often lacks the ability to represent long-term access with floating-point numbers and sub-second precision, so the return value is an integer. Some embedded system hardware does not support battery-powered RTCs , so the returned seconds are calculated from the last power-up, relative to a specific time, or hardware-specific time (such as reset).

4.4.2 Examples

```
>>> import utime >>>
utime.sleep(1) >>>                  # sleep for 1 second # sleep
utime.sleep_ms(500) >>>             for 500 milliseconds # sleep for 10
utime.sleep_us(10) >>> start =      microseconds
utime.ticks_ms() # get value of millisecond counter >>> delta = utime.ticks_diff(utime.ticks_ms(),
start) # compute time difference
>>> delta
6928
>>> print(utime.ticks_add(utime.ticks_ms(), -100))
1140718
>>> print(utime.ticks_add(0, -1))
1073741823
```

For more information, see utime .

4.5 sys – system-specific functions

4.5.1 Functions

4.5.1.1 sys.exit(retval=0)

Terminates the current program with the given exit code. The function throws a SystemExit exception. #### **sys.print_exception(exc,** file=sys.stdout) Print exceptions and tracebacks to a file-like object file (or sys.stdout by default).

4.5.2 Constants

4.5.2.1 sys.argv

A mutable list of arguments that the current program was started with.

4.5.2.2 sys.byteorder

System byte order ("little" or "big").

4.5.2.3 sys.implementation

Information about the current Python implementation. For MicroPython, this has the following attributes: - name - "micropython" - version - Tuple (major, minor, minor), such as (1, 9, 3)

4.5.2.4 sys.modules

A dictionary of loaded modules. In some ports, this may not include built-in modules.

4.5.2.5 sys.path

A list of addresses to search for imported modules.

4.5.2.6 sys.platform

Returns information about the current platform.

4.5.2.7 sys.stderr

The standard error stream.

4.5.2.8 sys.stdin

The standard input stream.

4.5.2.9 sys.stdout

The standard output stream.

4.5.2.10 sys.version

A Python language version that conforms to the given string.

4.5.2.11 sys.version_info

The Python language version used in this implementation is represented as a tuple.

4.5.3 Examples

```
>>> import sys
>>> sys.version
'3.4.0'
>>> sys.version_info (3, 4,
0)
>>> sys.path ['',
'/libs/mpy/']
>>> sys.__name__
'sys'
>>> sys.platform
'rt-thread'
>>> sys.byteorder
'little'
```

For more information, please refer to sys .

4.6 math – mathematical functions

4.6.1 Constants

4.6.1.1 math.e

The base of the natural logarithm.

Example:

```
>>>import math
>>>print(math.e)
2.718282
```

4.6.1.2 math.pi

The ratio of a circle's circumference to its diameter.

Example:

```
>>> print(math.pi)
3.141593
```

4.6.2 Functions

4.6.2.1 math.acos(x)

Calculates the inverse trigonometric function of cos(x) given a value in radians.

4.6.2.2 math.acosh(x)

Returns the inverse hyperbolic cosine of x .

4.6.2.3 math.asin(x)

Pass in a radian value and calculate the inverse trigonometric function of sin(x). Example:

```
>>> x = math.asin(0.5) >>> print(x)

0.5235988
```

4.6.2.4 math.asinh(x)

Returns the inverse hyperbolic sine of x.

4.6.2.5 math.atan(x)

Returns the inverse tangent of x .

4.6.2.6 math.atan2(y, x)

Return the principal value of the inverse tangent of y/x.

4.6.2.7 math.atanh(x)

Return the inverse hyperbolic tangent of x.

4.6.2.8 math.ceil(x)

Round up. Example:

```
>>> x = math.ceil(5.6454) >>> print(x)

6
```

4.6.2.9 math.copysign(x, y)

Return x with the sign of y.

4.6.2.10 math.cos(x)

Calculate the cosine by passing in a radian value. Example: Calculate cos60°

```
>>> math.cos(math.radians(60))
0.5
```

4.6.2.11 math.cosh(x)

Return the hyperbolic cosine of x.

4.6.2.12 math.degrees(x)

Convert radians to degrees. Example:

```
>>> x = math.degrees(1.047198) >>> print(x)

60.00002
```

4.6.2.13 math.erf(x)

Return the error function of x.

4.6.2.14 math.erfc(x)

Return the complementary error function of x.

4.6.2.15 math.exp(x)

Calculates e raised to the power of x. Example:

```
>>> x = math.exp(2) >>>
print(x)
7.389056
```

4.6.2.16 math.expm1(x)

Computes math.exp(x) - 1.

4.6.2.17 math.fabs(x)

Calculates the absolute value. Example:

```
>>> x = math.fabs(-5) >>> print(x)

5.0
>>> y = math.fabs(5.0) >>> print(y)

5.0
```

4.6.2.18 math.floor(x)

Round down. Example:

```
>>> x = math.floor(2.99) >>> print(x)

2
>>> y = math.floor(-2.34) >>> print(y)

-3
```

4.6.2.19 math.fmod(x, y)

Take the modulus of x divided by y. Example:

```
>>> x = math.fmod(4, 5) >>> print(x)

4.0
```

4.6.2.20 math.frexp(x)

Decomposes a floating-point number into its mantissa and exponent. The returned value is the tuple (m, e) such that x == m * 2**e exactly. If x == 0 then the function returns (0.0, 0), otherwise the relation 0.5 <= abs(m) < 1 holds.

4.6.2.21 math.gamma(x)

Returns the gamma function. Example:

```
>>> x = math.gamma(5.21) >>>
print(x)
33.08715
```

4.6.2.22 math.isfinite(x)

Return True if x is finite.

4.6.2.23 math.isinf(x)

Return True if x is infinite.

4.6.2.24 math.isnan(x)

Return True if x is not-a-number

4.6.2.25 math.ldexp(x, exp)

Return x * (2**exp).

4.6.2.26 math.lgamma(x)

Returns the natural logarithm of the gamma function. Example:

```
>>> x = math.lgamma(5.21) >>>
print(x)
3.499145
```

4.6.2.27 math.log(x)

Calculates the base-e logarithm of x. Example:

```
>>> x = math.log(10) >>>
print(x)
2.302585
```

4.6.2.28 math.log10(x)

Calculates the base 10 logarithm of x. Example:

```
>>> x = math.log10(10) >>>
print(x)
1.0
```

4.6.2.29 math.log2(x)

Calculates the base 2 logarithm of x. Example:

```
>>> x = math.log2(8) >>>
print(x)
3.0
```

4.6.2.30 math.modf(x)

Return a tuple of two floats, being the fractional and integral parts of x. Both return values have the same sign as x.

4.6.2.31 math.pow(x, y)

Calculates x raised to the power of y. Example:

```
>>> x = math.pow(2, 3) >>>
print(x)
8.0
```

4.6.2.32 math.radians(x)

Convert degrees to radians. Example:

```
>>> x = math.radians(60) >>>
print(x)
1.047198
```

4.6.2.33 math.sin(x)

Calculate the sine of an angle in radians. Example: Calculate sin90°

```
>>> math.sin(math.radians(90))
1.0
```

4.6.2.34 math.sinh(x)

Return the hyperbolic sine of x.

4.6.2.35 math.sqrt(x)

Calculates the square root. Example:

```
>>> x = math.sqrt(9) >>>
print(x)
3.0
```

4.6.2.36 math.tan(x)

Calculate the tangent by passing in a radian value. Example: Calculate tan60°

```
>>> math.tan(math.radians(60))
1.732051
```

4.6.2.37 math.tanh(x)

Return the hyperbolic tangent of x.

4.6.2.38 math.trunc(x)

Round to the nearest integer. Example:

```
>>> x = math.trunc(5.12) >>> print(x)

5
>>> y = math.trunc(-6.8) >>> print(y)

-6
```

For more information, please refer to math .

4.7 uio – Input/Output Streams

4.7.1 Functions

4.7.1.1 uio.open(name, mode='r', **kwargs) opens a file, associated with the built-in function **open() .** All ports

The port (used to access the file system) needs to support the mode parameter, but different ports support other parameters.

4.7.2 Class

4.7.2.1 class uio.FileIO(…)

This file type opens the file in binary mode, equivalent to using open(name, "rb"). This instance should not be used directly.

4.7.2.2 class uio.TextIOWrapper(…)

This type opens the file in text mode, equivalent to using open(name, "rt"). This instance should not be used directly.

4.7.2.3 class uio.StringIO([string])

4.7.2.4 class uio.BytesIO([string])

In-memory file objects. StringIO is used for text-mode I/O (opening a file with "t"), and BytesIO is used for binary I/O (opening a file with "b"). The initial contents of the file object can be specified using a string argument (plain strings for stringIO, bytes objects for bytesIO). All file methods, such as read(), write(), seek(), flush(), and close() , can be used on these objects, including the following methods:

4.7.2.5 getvalue()

Get the buffer contents.

For more information, please refer to uio .

4.8 ucollections – Collection and container types

4.8.1 Class

4.8.1.1 ucollections.namedtuple(name, fields)

This is a factory function that creates a new namedtuple type with a specified field name and set. A namedtuple is a tuple that allows subclasses to access its fields not only by numeric index, but also with attribute access using symbolic field name syntax. fields is a sequence of strings specifying the field names. For compatibility, implementations may also use space-delimited strings to name the fields (but this is less efficient).

Code example:

```
from ucollections import namedtuple

MyTuple = namedtuple("MyTuple", ("id", "name")) t1 = MyTuple(1, "foo") t2 =
MyTuple(2, "bar") print(t1.name)
assert t2.name == t2[1]
ucollections.OrderedDict(...)
```

4.8.1.2 ucollections.OrderedDict(…)

A subclass of the dictionary type that remembers and preserves the order in which keys/values were appended. When an ordered dictionary is iterated over, the keys/values are returned in the order in which they were added:

```
from ucollections import OrderedDict

## To make benefit of ordered keys, OrderedDict should be initialized ## from sequence of (key, value)
pairs. d = OrderedDict([("z", 1), ("a", 2)])

## More items can be added as usual
d["w"] = 5 d["b"]
= 3 for k, v in
d.items():
    print(k, v)
```

Output:

z 1 a 2 w 5 b 3

For more information, please refer to ucollections .

4.9 ustruct – Packing and Unpacking Primitive Data Types

• Supported size/byte prefixes: @, <, >, !. • Supported

format codes: b, B, h, H, i, I, l, L, q, Q, s, P, f, d (the last two are required to support floating point numbers).

4.9.1 Functions

4.9.1.1 ustruct.calcsize(fmt)

Returns the number of bytes required to store data of the type fmt.

```
fmt: data type b —
    byte type
    B — Unsigned byte h —
    Short integer
    H — unsigned short
    integer i — integer
    I — unsigned integer l
    — integer
    L — unsigned integer
    q — long integer
    Q — unsigned long integer f
    — floating point d
    — double-precision floating point
    P — Unsigned
```

Example:

```
>>> print(struct.calcsize("i"))
4
>>> print(struct.calcsize("B"))
1
```

4.9.1.2 ustruct.pack(fmt, v1, v2, …)

Packs arguments v1, v2, … according to the format string fmt. The return value is a bytes object with the packed arguments.

```
fmt: Same as above
```

Example:

```
>>> struct.pack("ii", 3, 2)
b'\x03\x00\x00\x00\x02\x00\x00\x00'
```

4.9.1.3 ustruct.unpack(fmt, data)

Unpacks data from fmt. The return value is a tuple of the unpacked arguments.

```
data: Byte object to be decompressed
```

Example:

```
>>> buf = struct.pack("bb", 1, 2) >>> print(buf)

b'\x01\x02'
>>> print(struct.unpack("bb", buf)) (1, 2)
```

4.9.1.4 ustruct.pack_into(fmt, buffer, offset, v1, v2, ...)

Compresses arguments v1, v2, … into buffer buffer according to format string fmt, starting at offset. When offset is negative, it counts from the end of buffer.

4.9.1.5 ustruct.unpack_from(fmt, data, offset=0)

Use fmt as the rule to start unpacking data from the offset position of data. If offset is a negative number, it is calculated from the end of the buffer. The return value is the unpacked argument tuple.

```
>>> buf = struct.pack("bb", 1, 2) >>>
print(struct.unpack("bb", buf)) (1, 2) >>>

print(struct.unpack_from("b", buf, 1)) (2,)
```

For more information, please refer to ustruct .

4.10 array – Array of numeric data

4.10.1 Constructor

4.10.1.1 class array.array(typecode[, iterable])

Creates an array with elements of the given type. The initial contents of the array are provided by iterable, or an empty array if none is provided.

```
typecode: array type iterable:
array initial content
```

Example:

```
>>> import array >>>
a = array.array('i', [2, 4, 1, 5]) >>> b = array.array('f') >>>
print(a) array('i', [2, 4, 1, 5]) >>>
print(b) array('f')
```

4.10.2 Methods

4.10.2.1 array.append(val)

Appends a new element to the end of the array.

Example:

```
>>> a = array.array('f', [3, 6]) >>> print(a)
array('f', [3.0,
6.0]) >>> a.append(7.0) >>>
print(a) array('f', [3.0,
6.0, 7.0])
```

4.10.2.2 array.extend(iterable)

Append a new array to the end of the array. Note that the data type of the appended array must be consistent with that of the original array.

Example:

```
>>> a = array.array('i', [1, 2, 3]) >>> b = array.array('i',
[4, 5]) >>> a.extend(b) >>> print(a) array('i', [1,
2, 3, 4, 5])
```

For more information, please refer to array .

4.11 gc – Controlling Garbage Collection

4.11.1 Functions

4.11.1.1 gc.enable()

Allows automatic reclamation of memory fragments.

4.11.1.2 gc.disable()

Automatic recycling is prohibited, but memory fragments can be manually recycled through the collect() function.

4.11.1.3 gc.collect()

Run a garbage collection.

4.11.1.4 gc.mem_alloc()

Returns the amount of allocated memory.

4.11.1.5 gc.mem_free()

Returns the amount of memory remaining.

For more information, please refer to gc .

4.12 machine – hardware-related functions

4.12.1 Functions

4.12.1.1 Reset related functions

4.12.1.1.1 machine.info() displays information about the system and memory usage.

4.12.1.1.2 machine.rest() restarts the device, similar to pressing the reset button.

4.12.1.1.3 machine.reset_cause() gets the reset reason and see the constants of possible return values.

4.12.1.2 Interrupt related functions

4.12.1.2.1 machine.disable_irq() disables interrupt requests. Returns the previous IRQ state, which should be considered an unknown value. This return value should be passed to the enable_irq function before the disable_irq function is called to reset the interrupt to its initial state.

4.12.1.2.2 machine.enable_irq(state) re-enables interrupt requests. The state parameter should be the value returned from the most recent call that disabled the function.

4.12.1.3 Power consumption related functions

4.12.1.3.1 machine.freq() returns the CPU operating frequency.

4.12.1.3.2 machine.idle() blocks the clock signal to the CPU to reduce power consumption in shorter or longer cycles. When an interrupt occurs, the peripherals will continue to work.

4.12.1.3.3 machine.sleep() stops the CPU and disables all peripherals except WLAN . The system will resume from the point where the sleep request was made. To ensure that wakeup will occur, the interrupt source should be configured first.

4.12.1.3.4 machine.deepsleep() stops the CPU and all peripherals (including the network interface). Execution resumes from the main function, as if it had been reset. The cause of the reset can be determined by checking the machine.DEEPSLEEP parameter. To ensure that a wakeup occurs, an interrupt source should be configured first, such as a pin transition or an RTC timeout.

4.12.2 Constants

4.12.2.1 machine.IDLE

4.12.2.2 machine.SLEEP

4.12.2.3 machine.DEEPSLEEP

The wakeup value of the IRQ .

4.12.2.4 machine.PWRON_RESET

4.12.2.5 machine.HARD_RESET

4.12.2.6 machine.WDT_RESET

4.12.2.7 machine.DEEPSLEEP_RESET

4.12.2.8 machine.SOFT_RESET

The reason for the reset.

4.12.2.9 machine.WLAN_WAKE

4.12.2.10 machine.PIN_WAKE

4.12.2.11 machine.RTC_WAKE

The reason for awakening.

4.12.3 Class

4.12.3.1 class Pin - Control I/O pins

4.12.3.2 class I2C - I2C protocol

4.12.3.3 class SPI - SPI protocol

4.12.3.4 class UART - Serial Port

4.12.4 Examples

```
>>> import machine
>>>
>>> machine.info()                    # show information about the board
------------------
RT-Thread
------------------
total memory: 131048
used memory : 4920
maximum allocated memory: 5836
thread          at status       sp      stack size max used left tick error
--------------------------------------------------------------------------------
elog_async 31 suspend 0x000000a8 0x00000400 20 ready      26% 0x00000003 000
tshell          0x0000019c 0x00001000 31 ready 0x0000006c 39% 0x00000006 000
tidle           0x00000100                                50% 0x0000000b 000
SysMonitor 30 suspend 0x000000a8 0x00000200 4 suspend     32% 0x00000005 000
timer           0x0000007c 0x00000200                     24% 0x00000009 000
------------------
qstr:
    n_pool=0
    n_qstr=0
    n_str_data_bytes=0
    n_total_bytes=0
------------------
GC:
    16064 total
    464 : 15600
    1=14 2=6 m=3
>>> machine.enable_irq() >>>          # enable interrupt
machine.disable_irq() dangerous       # disable interrupt, WARNING: this operation is

>>> machine.reset()                   # hard reset, like push RESET button
```

For more information, please refer to machine .

4.13 machine.Pin

Pin objects are used to control input/output pins (also known as GPIO). Pin objects are typically associated with a physical pin and can drive output voltages and read input voltages. The Pin class includes methods for setting the pin's mode (input/output) and for getting and setting the digital logic (0 or 1).

A Pin object is constructed from an identifier that explicitly specifies a specific input or output. The form of the identifier and the mapping to the physical pin are specific to a port. The identifier can be an integer, a string, or a tuple with the port and pin numbers. In RT-Thread MicroPython, a pin identifier is a tuple consisting of a port name and a pin number, such as ("X1", 33) in Pin(("X1", 33), Pin.OUT_PP) .

4.13.1 Constructor

The constructor of the Pin object in RT-Thread MicroPython is as follows:

4.13.1.1 class machine.Pin(id, mode = -1, pull = -1ÿvalue)

- **id :** It is composed of the user-defined pin name and the Pin device pin number, such as ("X1", 33), "X1" is the user-defined pin name, 33 is the pin number of the RT-Thread Pin device driver in this port.
- **mode :** specifies the pin mode, which can be one of the following:
  - **Pin.IN :** Input mode **– Pin.OUT :** Output mode **– Pin.OPEN_DRAIN :** Open drain mode
- **pull :** If the specified pin is connected to a pull-up or pull-down resistor, it can be configured into the following states:
  - **None :** No pull-up or pull-down resistor.
  - **Pin.PULL_UP :** Enable the pull-up resistor.
  - **Pin.PULL_DOWN :** Enable the pull-down resistor.
- **value :** The value of value is only valid for output mode and open-drain output mode, and is used to set the initial output value.

4.13.2 Methods

4.13.2.1 Pin.init(mode= -1, pull= -1, *, value, drive, alt) Reinitializes the pins based on the input parameters. Only the specified parameters are set; the states of the remaining pins remain unchanged. For detailed parameters, refer to the constructor above.

4.13.2.2 Pin.value([x])

If no parameter is given, x is . This method gets the value of a pin. If the given parameter x is 0 or 1, then the value of the pin is set. either logic 0 or logic 1.

4.13.2.3 Pin.name()

Returns the user-defined pin name of the pin object when it is constructed.

4.13.3 Constants

The following constants are used to configure the Pin object.

4.13.3.1 Select pin mode:

4.13.3.1.1 Pin.IN

4.13.3.1.2 Pin.OUT

4.13.3.1.3 Pin.OPEN_DRAIN

4.13.3.2 Select pull-up/pull-down mode:

4.13.3.2.1 Pin.PULL_UP

4.13.3.2.2 Pin.PULL_DOWN

4.13.3.2.3 None Use the value None to indicate no pull-up or pull-down.

4.13.4 Examples

```
>>> from machine import Pin
>>>
>>> p_out = Pin(("X1", 33), Pin.OUT_PP) >>>
p_out.value(1) # set io high >>> p_out.value(0)
                                        # set io low
>>>
>>> p_in = Pin(("X2", 32), Pin.IN, Pin.PULL_UP) >>> p_in.value()
                                        # get value, 0 or 1
```

For more information, please refer to machine.Pin .

4.14 machine.I2C

- I2C is a two-wire protocol for communication between devices. At the physical layer, it consists of two wires: SCL and SDA , which are clock and data.
  Wire.
- I2C objects are created to a specific bus. They can be initialized when they are created or later. • Printing an I2C object will
  print out information about the configuration.

4.14.1 Constructor

The constructor of the I2C object in RT-Thread MicroPython is as follows:

4.14.1.1 class machine.I2C(id= -1, scl, sda, freq=400000)

Constructs and returns a new I2C object using the following parameters:

- **id :** Identifies a specific I2C peripheral. If id = -1, software emulation is selected for I2C implementation. In this case, any pins can be used to emulate the I2C bus. Therefore, you must specify scl and sda during initialization . For software I2C initialization, refer to the software I2C example. For hardware I2C initialization, refer to the hardware I2C example.
- **scl :** should be a Pin object, specified as a Pin object for scl .
- **sda :** should be a Pin object, specified as a Pin object for sda .
- **freq :** should be the maximum frequency set for scl .

4.14.2 Methods

4.14.2.1 I2C.init(scl, sda, freq=400000)

Initialize the I2C bus. For parameter description, please refer to the parameters in the constructor.

4.14.2.2 I2C.deinit()

Close the I2C bus.

4.14.2.3 I2C.scan()

Scans all I2C addresses between 0x08 and 0x77 and returns a list of addresses that respond.
When it reaches its address, it will respond by pulling SDA low.

4.14.3 I2C Basic Methods

The following method implements basic I2C bus operations and can be combined into any I2C communication operation.
If you have controls, you can use them, otherwise you can use the standard usage methods described later.

4.14.3.1 I2C.start()

Generates a start signal on the bus. (When SCL is high, SDA transitions to low)

4.14.3.2 I2C.stop()

Generates a stop signal on the bus. (When SCL is high, SDA transitions to high)

4.14.3.3 I2C.readinto(buf, nack=True)

Read bytes from the bus and store them into buf . The number of bytes read is the length of buf . After receiving all but the last byte, an ACK is sent on the bus. After receiving the last byte, a NACK is sent if the data is correct, otherwise an ACK is sent .

4.14.3.4 I2C.write(buf)

Connect the data in buf to the bus, check if an ACK is received after each byte, and stop transmitting the remaining bytes if a NACK is received. This function returns the number of ACKs received .

4.14.4 I2C Standard Bus Operation

The following methods implement standard I2C master read and write operations to a given slave device.

4.14.4.1 I2C.readfrom(addr, nbytes, stop=True)

Read n bytes from the slave device specified by addr . If stop = True, a stop signal is generated at the end of the transfer. The function returns a bytes object storing the read data.

4.14.4.2 I2C.readfrom_into(addr, buf, stop=True)

Read data from the slave device specified by addr and store it in buf . The number of bytes read will be the length of buf . If stop = True, Then a stop signal is generated at the end of the transfer. This method has no return value.

4.14.4.3 I2C.writeto(addr, buf, stop=True)

Writes the data in buf to the slave device specified by addr . If a NACK is received during the write process , the remaining bytes will not be sent. If stop = True, a stop signal will be generated at the end of the transfer, even if a NACK is received . This function returns the number of ACKs received .

4.14.5 Memory Operations

Some I2C devices act as a memory device that can be read from and written to. In this case, there are two addresses associated with I2C , the slave The following methods are convenience functions for communicating with these devices.

4.14.5.1 I2C.readfrom_mem(addr, memaddr, nbytes, *, addrsize=8) Reads **n** bytes from the slave device specified by **addr ,** starting at address **memaddr .** The **addrsize** parameter specifies the length of the address. Returns a bytes object storing the read data.

4.14.5.2 I2C.readfrom_mem_into(addr, memaddr, buf, *, addrsize=8) reads data from the slave device specified by **addr** at address **memaddr** into **buf .** The number of bytes read is the length of **buf .** This method has no return value.

4.14.5.3 I2C.writeto_mem(addr, memaddr, buf, *, addrsize=8) writes the data in **buf** to the **memaddr** address of the slave device specified by **addr .** This method has no return value.

4.14.6 Examples

4.14.6.1 Software Simulation of I2C

```
>>> from machine import Pin, I2C
>>> clk = Pin(("clk", 43), Pin.OUT_OD) # Select the 43 pin device as the clock
>>> sda = Pin(("sda", 44), Pin.OUT_OD) # Select the 44 pin device as the data line
>>> i2c = I2C(-1, clk, sda, freq=100000) # create I2C peripheral at frequency of 100
    kHz
>>> i2c.scan()                          # scan for slaves, returning a list of 7-bit
    addresses
[81]                                    # Decimal representation
>>> i2c.writeto(0x51, b'123')           # write 3 bytes to slave with 7-bit address 42
3
>>> i2c.readfrom(0x51, 4)               # read 4 bytes from slave with 7-bit address
    42
with '\xf8\xc0\xc0\xc0'
>>> i2c.readfrom_mem(0x51, 0x02, 1) # read 1 bytes from memory of slave 0x51(7-bit
    ),
b'\x12'                                 # starting at memory-address 8 in the slave
>>> i2c.writeto_mem(0x51, 2, b'\x10') # write 1 byte to memory of slave 42,
                                        # starting at address 2 in the slave
```

4.14.6.2 Hardware I2C

You need to enable the I2C device driver first . To find the device, you can enter the list_device command in msh .
If you pass in 0, the system will search for a device named i2c0 , and use it to build an I2C object:

```
>>> from machine import Pin, I2C
>>> i2c = I2C(0) >>>          # create I2C peripheral at frequency of 100kHz
i2c.scan()                    # scan for slaves, returning a list of 7-bit
    addresses
[81]                          # Decimal representation
```

For more information, see machine.I2C .

4.15 machine.SPI

- SPI is a synchronous serial protocol driven by the host. At the physical layer, the bus has three lines: SCK, MOSI, and MISO. Multiple devices can share
  Sharing the same bus, each device is selected by a separate signal SS , also known as the chip select signal.
- The host selects a device to communicate with via the chip select signal. The management of the SS signal should be the responsibility of the user code. (via machine.Pin)

4.15.1 Constructor

The constructor of the SPI object in RT-Thread MicroPython is as follows:

4.15.1.1 class machine.SPI(id, ...)

Constructs an SPI object on the given bus , id depends on the specific port.

If you want to use software SPI refer . If the pin is used to simulate the SPI bus, the first parameter of the initialization needs to be set to -1 .
to the software SPI example.

When using hardware SPI, you only need to enter the SPI device number during initialization , such as '50' represents the 0th device on the SPI5 bus.
For the initialization method, please refer to the hardware SPI example.

If no additional parameters are given, the SPI object will be created but not initialized. If additional parameters are given, the bus will be
For initialization, please refer to the SPI.init method below for initialization parameters .

4.15.2 Methods

4.15.2.1 SPI.init(baudrate=1000000, *, polarity=0, phase=0, bits=8, firstbit=SPI.MSB, sck=None, mosi=None, miso=None)

Initialize the SPI bus with the given parameters:

- **baudrate :** SCK clock frequency. •
  polarity : Polarity can be 0 or 1, and is the level at which the clock is idle. • **phase :** Phase
  can be 0 or 1, and data is collected on the first or second clock edge, respectively. • **bits :** The length of data
  transmitted each time, generally 8 bits. • **firstbit :** Whether the
  transmitted data starts from the high bit or the low bit, can be SPI.MSB or SPI.LSB. • **sck :** Machine.Pin object for sck. •
  mosi : **Machine.Pin** object for mosi . • miso : Machine.Pin
  object for miso .

4.15.2.2 SPI.deinit()

Close the SPI bus.

4.15.2.3 SPI.read(nbytes, write=0x00)

Reads n bytes while continuously writing the given single byte. Returns a bytes object containing the read data.

4.15.2.4 SPI.readinto(buf, write=0x00)

Reads n bytes into buf while continually writing the single byte given by write . This method returns the number of bytes read.

4.15.2.5 SPI.write(buf)

Writes the bytes contained in buf . Returns None.

4.15.2.6 SPI.write_readinto(write_buf, read_buf)

When reading data into readbuf , write data from writebuf . The buffers can be the same or different, but the two buffers
Must have the same length. Returns None.

4.15.3 Constants

4.15.3.1 SPI.MASTER

Used to initialize the SPI bus as a master.

4.15.3.2 SPI.MSB

Set to transmit data starting from the high bit.

4.15.3.3 SPI.LSB

Set to transmit data starting from the low bit.

4.15.4 Examples

4.15.4.1 Software Simulation SPI

```
>>> from machine import Pin, SPI >>> clk =
Pin(("clk", 43), Pin.OUT_PP) >>> mosi = Pin(("mosi",
44), Pin.OUT_PP) >>> miso = Pin(("miso", 45), Pin.IN)
>>> spi = SPI(-1, 500000, polarity = 0, phase = 0,
bits = 8, firstbit = 0, sck = clk
      .   one = one, eyes = eyes)
>>> print(spi)
SoftSPI(baudrate=500000, polarity=0, phase=0, sck=clk, mosi=mosi, miso=miso) >>> spi.write("hello rt-
thread!") >>> spi.read(10)

b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

4.15.4.2 Hardware SPI

You need to enable the SPI device driver first . To find the device, you can enter the list_device command in msh .
If 50 is passed in , the system will search for a device named spi50 , and use this device to construct an SPI object:

```
>>> from machine import SPI >>> spi
= SPI(50) >>> print(spi)

SPI(device port : spi50) >>>
spi.write(b'\x9f') >>> spi.read(5)

b'\xff\xff\xff\xff\xff'
>>> buf = bytearray(1) >>>
spi.write_readinto(b"\x9f",buf)
>>> buf
bytearray(b'\xef') >>>
spi.init(100000,0,0,8,1)                    # Resetting SPI parameter
```

For more information, see machine.SPI .

4.16 machine.UART

UART implements the standard uart/usart duplex serial communication protocol. At the physical layer, it consists of two data lines: RX and TX. A signal unit is a character, which can be 8 or 9 bits wide.

4.16.1 Constructor

The constructor of the UART object in RT-Thread MicroPython is as follows:

4.16.1.1 class machine.UART(id, ...)

Constructs a UART object on the given bus . The id depends on the specific port. The initialization parameters can be found in the UART.init method below. Law.

When using hardware UART, you only need to enter the UART device number during initialization . For example, entering 1 means uart1 device. Initialization method Please refer to the examples.

4.16.2 Methods

4.16.2.1 UART.init(baudrate = 9600, bits=8, parity=None, stop=1)

- **baudrate :** SCK clock frequency. • **bits :** The length of each data transmission. • **parity :** The parity check method. • **stop :** The length of the stop bit.

4.16.2.2 UART.deinit()

Close the serial bus.

4.16.2.3 UART.read([nbytes])

Read characters. If n bytes are specified, then at most n bytes will be read. Otherwise, as much data as possible will be read. Return value: A bytes object containing the data read. If a timeout occurs, None will be returned.

4.16.2.4 UART.readinto(buf[, nbytes])

Read characters into buf . If n bytes are specified, then at most n bytes will be read. Otherwise, as much data as possible will be read. The length of the data read does not exceed the length of buf . Return value: the number of bytes read and stored in buf . If the timeout occurs, None is returned.

4.16.2.5 UART.readline()

Read a line of data, ending with a newline character. Return value: the number of lines read, or None if timeout occurs.

4.16.2.6 UART.write(buf)

Write the data in buf to the bus. Return value: the number of bytes written, or None if timeout occurs.

4.16.3 Examples

If 1 is passed as the first parameter of the constructor, the system will search for a device named uart1 and use this device to construct a UART object:

```
from machine import UART uart =
UART(1, 9600) uart.init(9600,                                   # init with given baudrate
bits=8, parity=None, stop=1) # init with given parameters uart.read(10) uart.read() uart.readline()
uart.readinto(buf)          # read 10 characters, returns a bytes object
# read and store            # read all available characters
into the given buffer       # read a line
uart.write('abc') # write the 3 characters
```

For more information, please refer to machine.UART .

4.17 uos – Basic “operating system” services

4.17.1 Functions

4.17.1.1 uos.chdir(path)

Change the current directory.

4.17.1.2 uos.getcwd()

Get the current directory.

4.17.1.3 uos.listdir([dir])

Without argument, lists the current directory; otherwise, lists the given directory.

4.17.1.4 uos.mkdir(path)

Create a directory.

4.17.1.5 uos.remove(path)

Delete the file.

4.17.1.6 uos.rmdir(path)

Delete a directory.

4.17.1.7 uos.rename(old_path, new_path)

Rename a file or folder.

4.17.1.8 uos.stat(path)

Get the status of a file or directory.

4.17.1.9 uos.sync()

Synchronize all file systems.

4.17.2 Examples

```
>>> import uos
>>> uos.                                    # Tab
__name__          uname          dir              getcwd
list              mkdir          remove           rmdir
stat              unlink         mount            umount
>>> uos.mkdir("rtthread")
>>> uos.getcwd()
'/'
>>> uos.chdir("rtthread")
>>> uos.getcwd()
'/rtthread'
>>> uos.listdir()
['web_root', 'rtthread', '11']
>>> uos.rmdir("11")
>>> uos.listdir()
['web_root', 'rtthread']
>>>
```

For more information, please refer to uos .

4.18 uselect – Waiting for stream events

4.18.1 Constants

4.18.1.1 select.POLLIN — Read Available Data

4.18.1.2 select.POLLOUT — Write more data

4.18.1.3 select.POLLERR — An Error Occurred

4.18.1.4 select.POLLHUP — End-of-stream/connection termination detection

4.18.2 Functions

4.18.2.1 select.select(rlist, wlist, xlist[, timeout])

Monitors when an object is readable or writable. If the monitored object's state changes, returns a result (blocking the thread). This function is for compatibility purposes and is not very efficient. The poll function is recommended instead.

```
rlist: Array of file descriptors waiting to be read ready
wlist: Array of file descriptors waiting to be write ready
xlist: Array of file descriptors waiting
for exceptions timeout: Waiting time (unit: seconds)
```

Example:

```
def selectTest(): global
    s rs, ws,
    es = select.select([s,], [], []) #The program will wait here
    until the object s is readable
    print(rs)
    for i in rs:
        if i == s:
            print("s can read now")
            data,addr=s.recvfrom(1024)
            print('received:',data,'from',addr)
```

4.18.3 Poll Class

4.18.3.1 select.poll()

Create a poll instance.

Example:

```
>>>poller = select.poll()
>>>print(poller) <poll>
```

4.18.3.2 poll.register(obj[, eventmask])

Register an object for monitoring and set the monitoring flag of the monitored object.

```
obj: monitored object flag:
monitored flag select.POLLIN
        — readable select.POLLHUP —
        hung up select.POLLERR — error

        select.POLLOUT — writable
```

4.18.3.3 poll.unregister(obj)

Unregisters a monitored object.

```
obj: registered object
```

Example:

```
>>>READ_ONLY = select.POLLIN | select.POLLHUP | select.POLLERR >>>READ_WRITE =
select.POLLOUT | READ_ONLY >>>poller.register(s, READ_WRITE)
>>>poller.unregister(s)
```

4.18.3.4 poll.modify(obj, eventmask)

Modify the monitoring flag of a registered object.

```
obj: registered monitored object flag:
modified monitoring flag
```

Example:

```
>>>READ_ONLY = select.POLLIN | select.POLLHUP | select.POLLERR >>>READ_WRITE =
select.POLLOUT | READ_ONLY >>>poller.register(s, READ_WRITE)
>>>poller.modify(s, READ_ONLY)
```

4.18.3.5 poll.poll([timeout])

Waits until at least one of the registered objects is ready. Returns a tuple (obj, event, ...), where the event element specifies an event that occurred on the stream and is a combination of the select.POLL* constants described above. Depending on the platform and version, there may be additional elements in the tuple, so do not assume that the tuple size is 2. If the timeout is reached, returns an empty list.

For more information, please refer to uselect .

4.19 uctypes – Accessing binary data in a structured manner

4.19.1 Constants

• uctypes.LITTLE_ENDIAN — Little endian packed structure. •

uctypes.BIG_ENDIAN — Big endian packed structure type. • NATIVE

— MRICOPython native storage type

4.19.2 Constructors

4.19.2.1 class uctypes.struct(addr, descriptor, type)

Converts a structure or union packed in memory in C format to a dictionary and returns the dictionary.

```
addr: address to start
conversion descriptor: conversion
descriptor format : "field_name": offset|uctypes.UINT32
offset: offset, unit:
byte, VOID, UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, BFUINT8
    ÿBFINT8ÿBFUINT16ÿBFINT16ÿBFUINT32ÿBFINT32ÿBF_POSÿBF_LENÿFLOAT32ÿFLOAT64
    ÿPTRÿARRAY
type: c structure or union storage type, default is local storage type
```

Example:

```
>>> a = b"0123"
>>> s = uctypes.struct(uctypes.addressof(a), {"a": uctypes.UINT8 | 0, "b": uctypes.
    UINT16 | 1}, uctypes.LITTLE_ENDIAN)
>>> print(s)
<struct STRUCT 3ffc7360>
>>> print(s.a)
48
>>> s.a = 49
>>> print(a)
b'1123'
```

4.19.3 Methods

4.19.3.1 uctypes.sizeof(struct)

Returns the size of the data in bytes. The parameter can be a class or a data object (or collection). Example:

```
>>> a = b"0123"
>>>b = uctypes.struct(uctypes.addressof(a), {"a": uctypes.UINT8 | 0, "b": uctypes.
    UINT16 | 1}, uctypes.LITTLE_ENDIAN)
>>> b.a
48
```

```
>>> print(uctypes.sizeof(b))
3
```

4.19.3.2 uctypes.addressof(obj)

Returns the object address. The parameter must be bytes or bytearray. Example:

```
>>> a = b"0123"
>>> print(uctypes.addressof(a))
1073504048
```

4.19.3.3 uctypes.bytes_at(addr, size)

Capture the memory data starting from addr and ending at size address offsets as a bytearray object and return it. Example:

```
>>> a = b"0123"
>>>print( uctypes.bytes_at(uctypes.addressof(a), 4))
b'0123'
```

4.19.3.4 uctypes.bytearray_at(addr, size)

Captures the memory of the given size and address as a bytearray object. Unlike the bytes_at() function, it can be written again.

Access the parameters of a given address. Example:

```
>>> a = b"0123"
>>> print(uctypes.bytearray_at(uctypes.addressof(a), 2)) bytearray(b'01')
```

For more information, please refer to uctypes .

4.20 uerrno – System Error Code Module

4.20.1 Examples

```
try:
    uos.mkdir("my_dir")
except OSError as exc:
    if exc.args[0] == errno.EEXIST:
        print("Directory already exists")
uerrno.errorcode
Dictionary mapping numeric error codes to strings with symbolic error code (see
    above):

>>> print(uerrno.errorcode[uerrno.EEXIST])
EXIST
```

For more information, please refer to uerrno .

4.21 _thread – Multithreading support

4.21.1 Examples

```
import _thread
import time def
testThread():
    while True:
        print("Hello from thread") time.sleep(2)


_thread.start_new_thread(testThread, ())
while True:
    pass
```

Output results (start a new thread and print characters every two seconds):

Hello from thread Hello from thread Hello from thread Hello from thread Hello from thread

For more information, please refer to _thread .

4.22 cmath – Mathematical functions on complex numbers

4.22.1 Functions

4.22.1.1 cmath.cos(z)

Returns the cosine of z.

4.22.1.2 cmath.exp(z)

Returns the exponential of z.

4.22.1.3 cmath.log(z)

Returns the logarithm of z.

4.22.1.4 cmath.log10(z)

Returns the common logarithm of z.

4.22.1.5 cmath.phase(z)

Returns the phase of z, in the range (-pi, +pi], expressed in radians.

4.22.1.6 cmath.polar(z)

Returns the polar coordinate of z.

4.22.1.7 cmath.rect(r, phi)

Returns a complex number with modulus r and phase phi.

4.22.1.8 cmath.sin(z)

Returns the sine of z.

4.22.1.9 cmath.sqrt(z)

Returns the square root of z.

4.22.2 Constants

4.22.2.1 cmath.e

The exponential of the natural logarithm.

4.22.2.2 cmath.pi

Pi.

For more information, please refer to cmath .

4.23 ubinascii – Binary/ ASCII conversion

4.23.1 Functions

4.23.1.1 ubinascii.hexlify(data[, sep])

Converts a string to its hexadecimal representation.

Example:

```
>>> ubinascii.hexlify('hello RT-Thread')
b'68656c6c6f2052542d546872656164'
>>> ubinascii.hexlify('summer')
b'73756d6d6572'
```

If the second parameter sep is specified, it will be used to separate the two hexadecimal numbers.

Example:

```
If the second parameter sep is specified, it will be used to separate two

hexadecimal numbers. Example: >>> ubinascii.hexlify('hello RT-Thread'," ")
b'68 65 6c 6c 6f 20 52 54 2d 54 68 72 65 61 64'
>>> ubinascii.hexlify('hello RT-Thread',",")
b'68,65,6c,6c,6f,20,52,54,2d,54,68,72,65,61,64'
```

4.23.1.2 ubinascii.unhexlify(data)

Convert a hexadecimal string to a binary string. This is the opposite of hexlify.

Example:

```
>>> ubinascii.unhexlify('73756d6d6572')
b'summer'
```

4.23.1.3 ubinascii.a2b_base64(data)

Converts Base64-encoded data to binary representation. Returns a byte string.

4.23.1.4 ubinascii.b2a_base64(data)

Encodes binary data in base64 format and returns a string.

For more information, please refer to ubinascii .

4.24 uhashlib – hashing algorithms

4.24.1 Algorithm Function

4.24.1.1 SHA256

A modern hashing algorithm (SHA2 family) suitable for cryptographic security purposes is included in the MicroPython kernel and is recommended for all development boards unless there are specific code size constraints.

4.24.1.2 SHA1

The previous generation algorithm is not recommended for new applications, but the SHA1 algorithm is part of the Internet standard and existing applications. Therefore, development boards with convenient network connections will provide this function.

4.24.1.3 MD5

A legacy algorithm that is considered unsafe to use as a password. Only certain development boards will use it for compatibility with older applications. Provide this algorithm.

4.24.2 Functions

4.24.2.1 class uhashlib.sha256([data])

Create a SHA256 hash object and provide data as the value.

4.24.2.2 class uhashlib.sha1([data])

Create a SHA1 hash object and provide data as the value.

4.24.2.3 class uhashlib.md5([data])

Create an MD5 hash object and provide data as the value.

4.24.2.4 hash.update(data)

Put more binary data into the hash table.

4.24.2.5 hash.digest()

Returns a bytes object containing all the data in the hash. After calling this method, no more data can be fed into the hash.

4.24.2.6 hash.hexdigest()

This method is not implemented. Use ubinascii.hexlify(hash.digest()) to achieve a similar effect.

For more information, please refer to uhashlib .

4.25 uheapq – Heap Sort Algorithm

4.25.1 Functions

4.25.1.1 uheapq.heappush(heap, item)

Push the object onto the heap.

4.25.1.2 uheapq.heappop(heap)

Pops the first element from the heap and returns it. If the heap is empty, an IndexError is raised.

4.25.1.3 uheapq.heapify(x)

Convert the list x into a heap.

For more information, please refer to uheapq .

4.26 ujson – JSON Encoding and Decoding

4.26.1 Functions

4.26.1.1 ujson.dumps(obj)

Convert the dict type to str.

obj: the object to be converted

Example:

```
>>> obj = {1:2, 3:4, "a":6} >>>
print(type(obj), obj) #Original dict type <class 'dict'> {3: 4, 1:
2, 'a': 6} >>> jsObj = json.dumps(obj) #Convert dict
type to str >>> print(type(jsObj), jsObj) <class 'str'> {3: 4, 1: 2, "a": 6}
```

4.26.1.2 ujson.loads(str)

Parse a JSON string and return an object. If the string is not in the correct format, a ValueError exception will be raised. Example:

```
>>> obj = {1:2, 3:4, "a":6} >>> jsDumps
= json.dumps(obj) >>> jsLoads =
json.loads(jsDumps) >>> print(type(obj), obj)
<class 'dict'> {3: 4, 1: 2, 'a': 6} >>>
print(type(jsDumps), jsDumps) <class 'str'> {3: 4, 1:
2, "a": 6} >>> print(type(jsLoads), jsLoads) <class
'dict'> {'a': 6, 1: 2, 3: 4}
```

For more information, please refer to ujson .

4.27 ure – Regular Expressions

4.27.1 Matching Character Sets

4.27.1.1 Matching Any Character

'.'

4.27.1.2 Matching character sets, supporting single characters and ranges

'[]'

4.27.1.3 Support for multiple matching metacharacters

'^' '$' '?' '*' '+' '??' '*?' '+?' '{m,n}'

4.27.2 Functions

4.27.2.1 ure.compile(regex)

Compiles a regular expression, returning a regex object.

4.27.2.2 ure.match(regex, string)

When matching regex with string, the match always starts from the beginning of the string.

4.27.2.3 ure.search(regex, string)

Searches for regex in string. Unlike match, it searches for the first match of the regular expression in string (the result may be 0).

4.27.2.4 ure.DEBUG

Flag value to display debugging information for the expression.

4.27.3 Regular Expression Objects:

Compile a regular expression and create an instance using ure.compile() .

4.27.3.1 regex.match(string)

4.27.3.2 regex.search(string)

4.27.3.3 regex.split(string, max_split=-1)

4.27.4 Matching Objects:

Match objects are the return value of the match() and search() methods.

4.27.4.1 match.group([index])

Only numeric groups are supported.

For more information, please refer to .

4.28 uzlib – zlib decompression

4.28.1 Functions

4.28.1.1 uzlib.decompress(data)

Returns the decompressed bytes data.

For more information, please refer to uzlib .

4.29 urandom - Random number generation module

4.29.1 Functions

4.29.1.1 urandom.choice(obj)

Randomly generate the number of elements in the object obj.

```
obj: a list of numbers
```

Example:

```
>>> print(random.choice("DFRobot"))
R
>>> print(random.choice("DFRobot"))
D
>>> print(random.choice([0, 2, 4, 3]))
3
>>> print(random.choice([0, 2, 4, 3]))
3
>>> print(random.choice([0, 2, 4, 3]))
2
```

4.29.1.2 urandom.getrandbits(size)

Randomly generate a positive integer in the range of 0 to size binary digits. For example:

• If size = 4, then it is a random positive integer from 0 to 0b1111. • If size = 8, then it is a random positive integer from 0 to 0b11111111.

```
size: bit size
```

Example:

```
>>> print( random.getrandbits(1)) #1 binary bit, ranging from 0 to 1 (decimal: 0 to 1)
1
>>> print(random.getrandbits(1))
0
>>> print(random.getrandbits(8)) #8 binary bits, ranging from 0000 0000 to 1111 11111 (decimal:
    0~255ÿ
224
>>> print(random.getrandbits(8))
155
```

4.29.1.3 urandom.randint(start, end)

Randomly generate an integer between start and end.

```
start: specifies the starting value in the range, included in the range
end: specifies the ending value in the range, included in the range
```

Example:

```
>>> import random >>>
print(random.randint(1, 4))
4
>>> print(random.randint(1, 4))
2
```

4.29.1.4 urandom.random()

Generates a random floating point number between 0 and 1. Example:

```
>>> print(random.random())
0.7111824
>>> print(random.random())
0.3168149
```

4.29.1.5 urandom.randrange(start, end, step)

Randomly generate positive integers in the range from start to end and increment by step. For example, in randrange(0, 8, 2), the randomly generated The number is any one of 0, 2, 4, and 6.

```
start: specifies the starting value in the range, included in the range end:
specifies the ending value in the range, included in the range step:
increments the base number
```

Example:

```
>>> print(random.randrange(2, 8, 2))
4
>>> print(random.randrange(2, 8, 2))
6
>>> print(random.randrange(2, 8, 2))
2
```

4.29.1.6 urandom.seed(sed)

Specifies the random number seed, typically used with other random number generation functions. Note: MicroPython's random numbers are actually a stable sequence of results generated by a stable algorithm, not a random sequence. sed is the first value calculated by this algorithm. Therefore, as long as sed is the same, all subsequent "random" results and sequences will be exactly the same.

```
sed: Random number seed
```

Example:

```
import random

for j in range(0, 2): random.seed(13) #Specify
    random number seed for i in range(0, 10): #Generate a random
    sequence in the range of 0 to 10
        print(random.randint(1, 10)) print("end")
```

Run results:

```
5
2
3
2
3
4
2
5
8
2
end
5
2
3
2
3
4
2
5
8
2
```

```
end
```

From the above you can see that generating two random number lists is the same. You can also generate more random number lists to see the results.

4.29.1.7 urandom.uniform(start, end)

Randomly generate floating point numbers between start and end.

```
start: specifies the starting value within the range, included in the range
stop: specifies the ending value within the range, included in the range
```

Example:

```
>>> print(random.uniform(2, 4))
2.021441
>>> print(random.uniform(2, 4))
3.998012
```

For more information, see urandom .

4.30 usocket – The socket module

4.30.1 Constants

4.30.1.1 Address Cluster

• socket.AF_INET =2 — TCP/IP – IPv4 • socket.AF_INET6 =10 — TCP/IP – IPv6

4.30.1.2 Socket Types

• socket.SOCK_STREAM =1 — TCP stream • socket.SOCK_DGRAM =2 — UDP datagram • socket.SOCK_RAW =3 — Raw socket • socket.SO_REUSEADDR =4 — Socket can be reused

4.30.1.3 IP Protocol Number

• socket.IPPROTO_TCP =16
• socket.IPPROTO_UDP =17

4.30.1.4 Socket Option Levels

• socket.SOL_SOCKET =4095

4.30.2 Functions

4.30.2.1 socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)

Creates a new socket with the specified address, type, and protocol number.

4.30.2.2 socket.getaddrinfo(host, port)

Convert the host domain name (host) and port (port) to a sequence of 5-tuples used to create a socket. The structure of the tuple list is as follows:

```
(family, type, proto, canonname, sockaddr)
```

Example:

```
>>> info = socket.getaddrinfo("rt-thread.org", 10000) >>> print(info) [(2, 1, 0, '',
('118.31.15.152', 10000))]
```

4.30.2.3 socket.close()

Close the socket. Once closed, all socket functions will be disabled. The remote end will not receive any data (after clearing the queue data).

Although the socket will be automatically closed during garbage collection, it is still recommended to use close() to close it when necessary.

4.30.2.4 socket.bind(address)

Bind a socket to an address. The socket must not already be bound.

4.30.2.5 socket.listen([backlog])

Listening socket, enabling the server to receive connections.

```
backlog: The maximum number of sockets to accept, at least 0. If not specified, a reasonable value is assumed by default.
```

4.30.2.6 socket.accept()

Receive connection request. Note: This function can only be called after binding address and port number and listening, and returns conn and address.

```
conn: A new socket object that can be used to send and receive
messages address: The client address connected to the server
```

4.30.2.7 socket.connect(address)

Connect to the server.

```
address: a tuple or list of server addresses and port numbers
```

4.30.2.8 socket.send(bytes)

Send data and return the number of bytes successfully sent. The number of bytes returned may be less than the length of the data sent.

```
bytes: bytes type data
```

4.30.2.9 socket.recv(bufsize)

Receive data and return the received data object.

```
bufsize: Specifies the maximum amount of data received at one time
```

Example:

```
data = conn.recv(1024)
```

4.30.2.10 socket.sendto(bytes, address)

Send data, the destination is determined by address, commonly used in UDP communication, and returns the size of the data sent.

```
bytes: bytes type data
address: tuple of target address and port number
```

Example:

```
data = sendto("hello RT-Thread", ("192.168.10.110", 100))
```

4.30.2.11 socket.recvfrom(bufsize)

Receive data, commonly used in UDP communication, and return the received data object and the address of the object.

```
bufsize: Specifies the maximum amount of data received at one time
```

Example:

```
data,addr=fd.recvfrom(1024)
```

4.30.2.12 socket.setsockopt(level, optname, value)

Set up the socket according to the option value.

```
level: socket option level
optname: socket option value:
can be an integer or a bytes class object representing a buffer.
```

Example:

```
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

4.30.2.13 socket.settimeout(value)

Set the timeout period in seconds. Example:

```
s.settimeout(2)
```

4.30.2.14 socket.setblocking(flag)

Set blocking or non-blocking mode: If flag is false, set non-blocking mode.

4.30.2.15 socket.read([size])

Read up to size bytes from the socket. Return a bytes object. If size is not given, it reads all data available from the socket until EOF; as such the method will not return until the socket is closed. This function tries to read as much data as requested (no “short reads”). This may be not possible with non-blocking socket though, and then less data will be returned.

4.30.2.16 socket.readinto(buf[, nbytes])

Read bytes into the buf. If nbytes is specified then read at most that many bytes. Otherwise, read at most len(buf) bytes. Just as read(), this method follows “no short reads” policy. Return value: number of bytes read and stored into buf.

4.30.2.17 socket.readline()

Receives a line of data, ends with a newline character, and returns the object that receives the data.

4.30.2.18 socket.write(buf)

Write byte type data to the socket and return the size of the data written successfully.

4.30.3 Examples

4.30.3.1 TCP Server example

```
>>> import usocket >>> s =
usocket.socket(usocket.AF_INET,usocket.SOCK_STREAM) # Create STREAM TCP
    socket
>>> s.bind(('192.168.12.32', 6001)) >>> s.listen(5) >>>
s.setblocking(True)
>>> sock,addr=s.accept() >>>
sock.recv(10) b'rt-thread\r' >>>
s.close()
```

4.30.3.2 TCP Client example

```
>>> import usocket >>> s =
usocket.socket(usocket.AF_INET,usocket.SOCK_STREAM) >>> s.connect(("192.168.10.110",6000))
>>> s.send("micropython")

11
>>> s.close()
```

connect to a web site example:

```
s = socket.socket()
s.connect(socket.getaddrinfo('www.micropython.org', 80)[0][-1])
```

For more information, please refer to usocket .

Chapter 5

RT-Thread MicroPython package management

5.1 1. Using the micropython-lib source code

5.1.1 1.1 Clone/download micropython-lib from GitHub The source code is locally

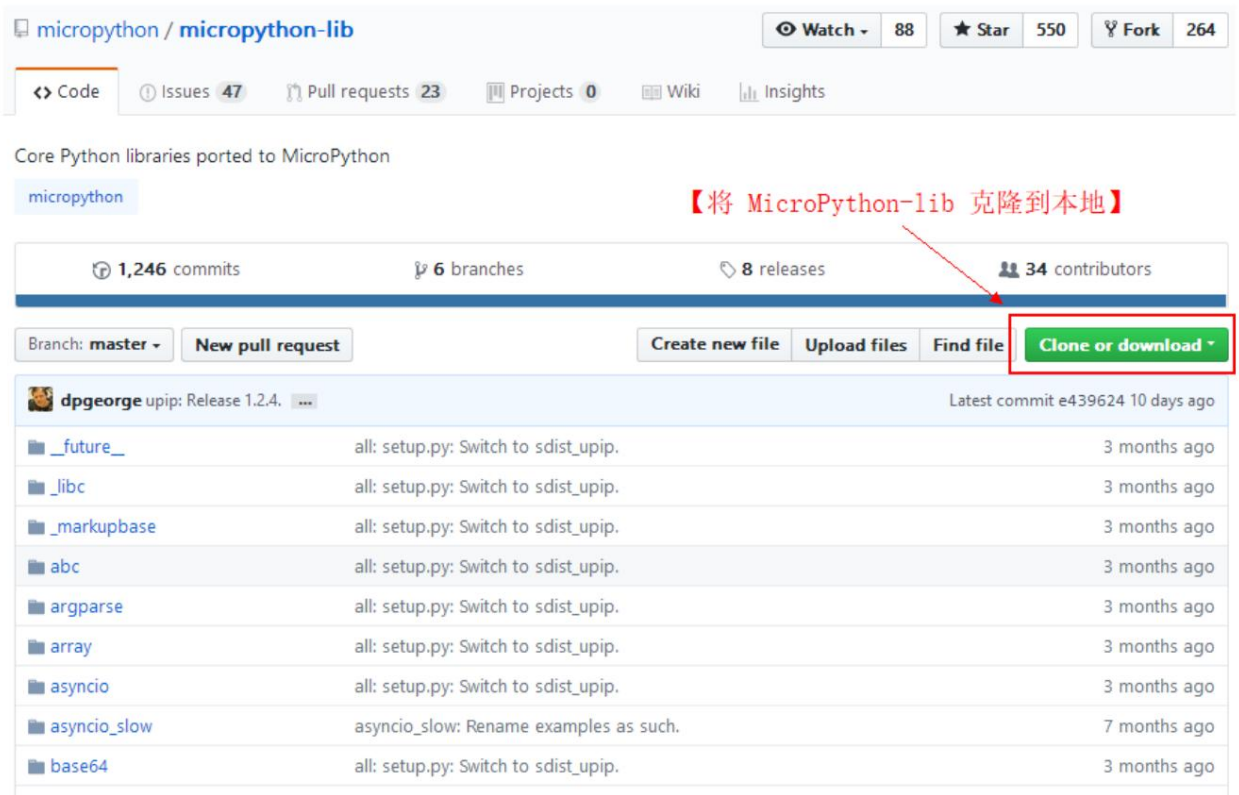

Figure **5.1:** *1525330162579*

5.1.2 1.2 Using Extension Packs

If you use the abc extension package, you can copy the contents of the abc folder except metadata.txt and setup.py (different modules have different contents. If you want to avoid errors, you can just copy all the contents of the abc folder) to the /libs/mpy directory of the SD card (file system):

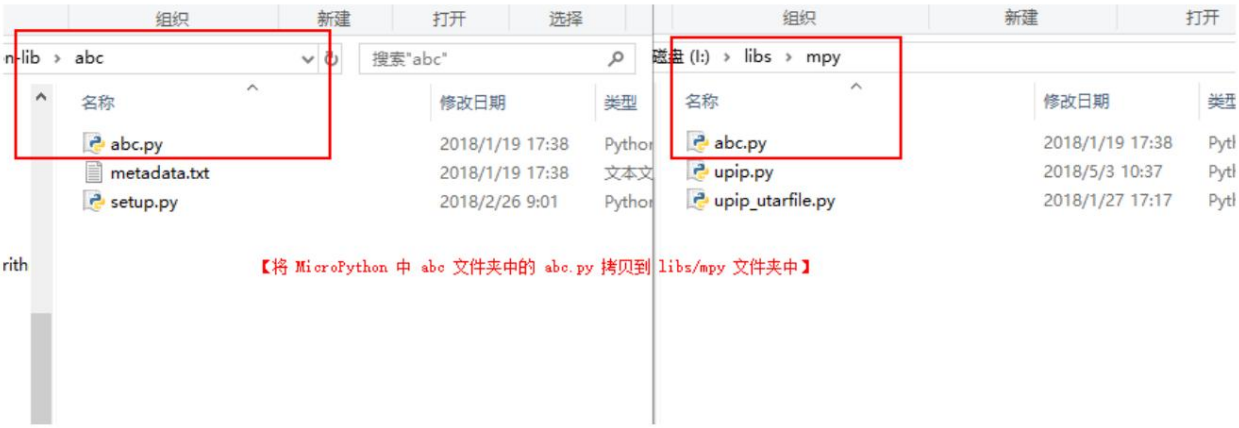

Figure **5.2:** *1525333595133*

Now you can import the abc module in the MicroPython interactive command line.

```
MicroPython v1.9.3-477-g7b0a020-dirty on 2018-03-21; Universal python platform with RT-Thread
Type "help()" for more information.
>>> import abc
>>> abc.
__class__          __file__           __name__           abstractmethod
>>> abc.abstractmethod("Hello RT-Thread!")
'Hello RT-Thread!'
>>>
```

Figure **5.3:** *1525340341541*

!!! note "Note" When the installed module depends on the functions of other modules, you need to install the dependent modules first. At this time, you need to make clear Clearly understand the dependent modules and then install the dependent modules one by one.

5.2 2. Using the upip package manager

upip is the MicroPython package manager. Because the RT-Thread operating system provides excellent support for the POSIX standard, upip runs well on RT-Thread MicroPython. Using upip , you can download MicroPython extension modules online and automatically download their dependent modules, greatly facilitating the expansion of MicroPython functionality.

The upip package manager takes up a lot of system resources. It is recommended to use upip on a development board with more system resources. Follow the steps below to configure:

- Copy the required contents in the upip folder in micropython-lib to the /libs/mpy directory on the SD card (file system):

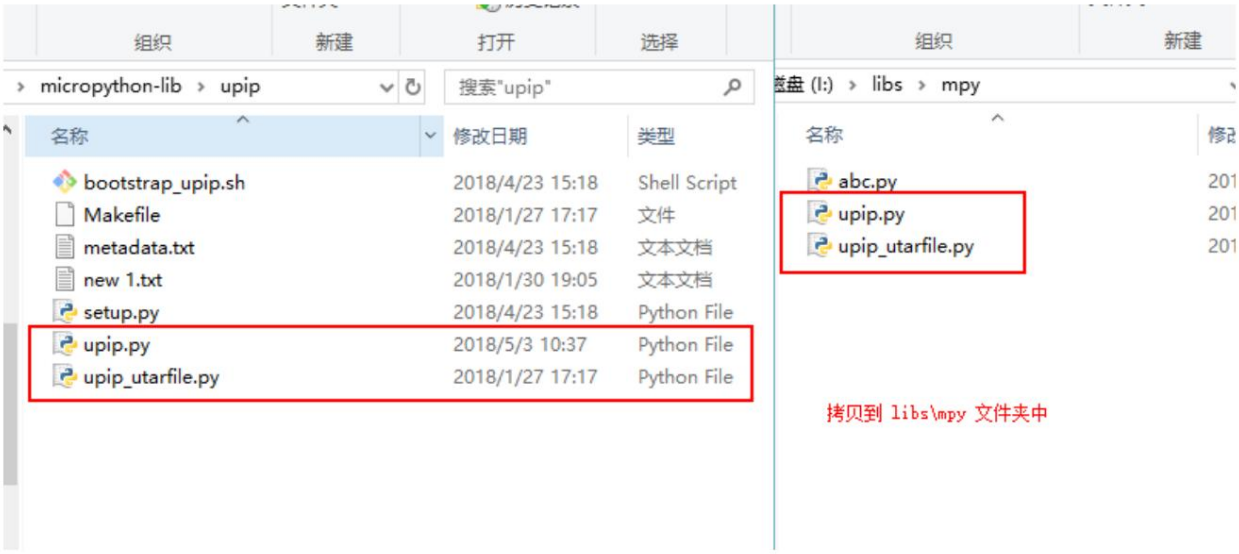

Figure **5.4:** *1525336695038*

- Enable the ussl module in the MicroPython network module in env.

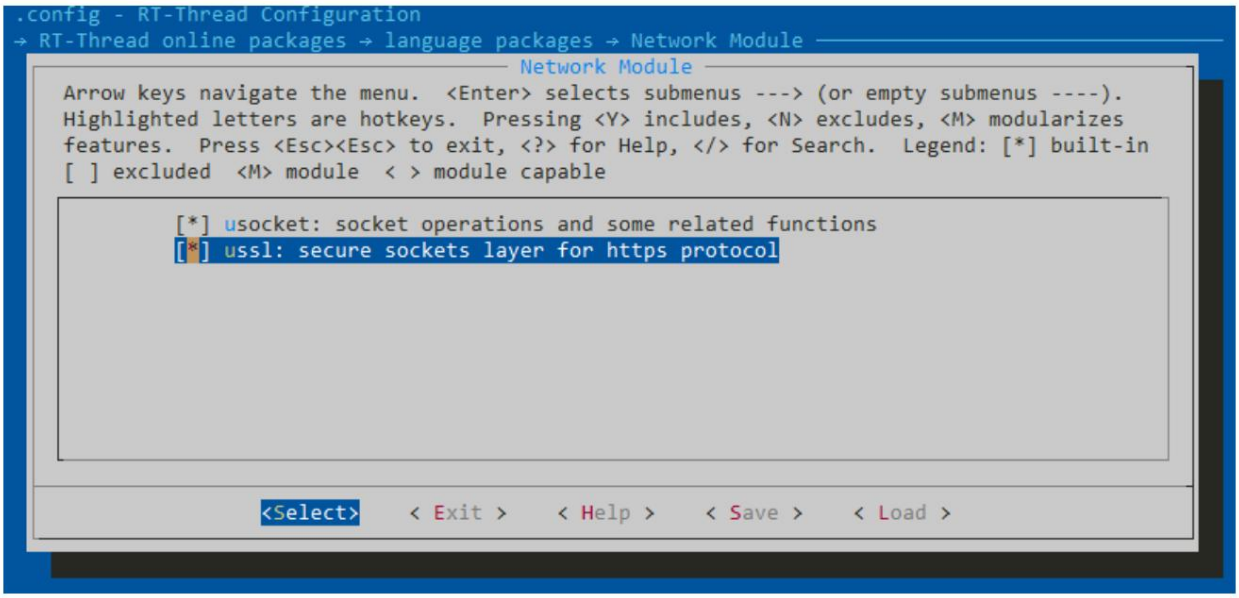

Figure **5.5:** *1525337170291*

- Modify the Maxium fragment length in bytes parameter of the mbedtls package in the security class to 5120.

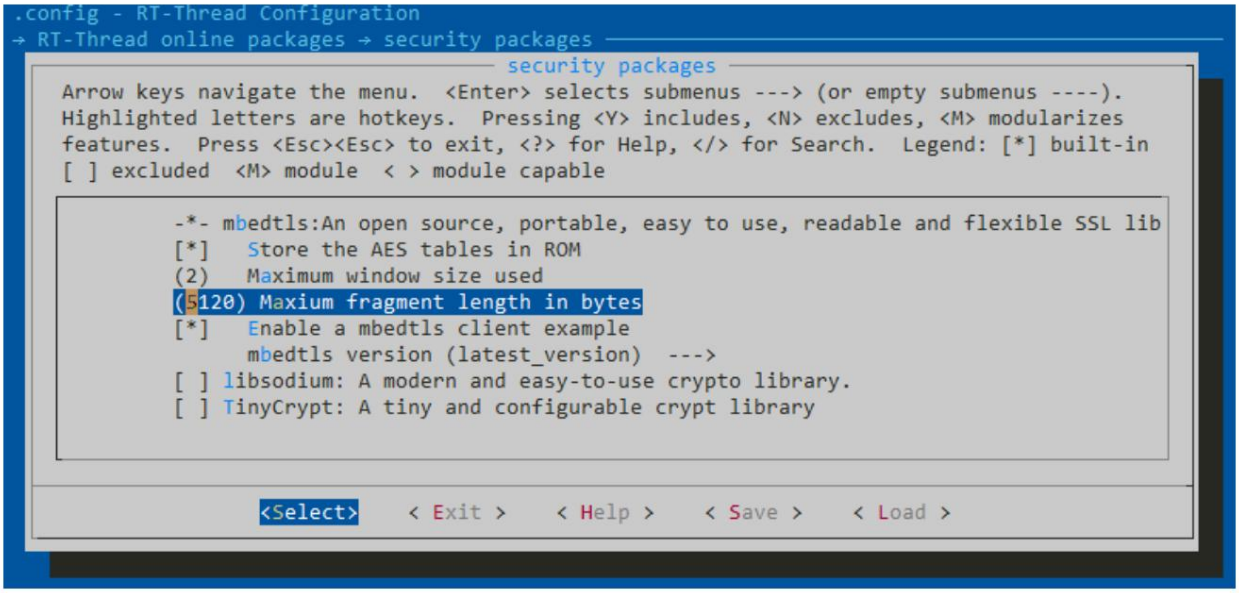

Figure **5.6:** *1525337315573*

• Update the software package and regenerate the project, recompile and download the firmware to the development board and then run it. At this point, the module is successfully downloaded to /libs/mpy folder and can be used directly.

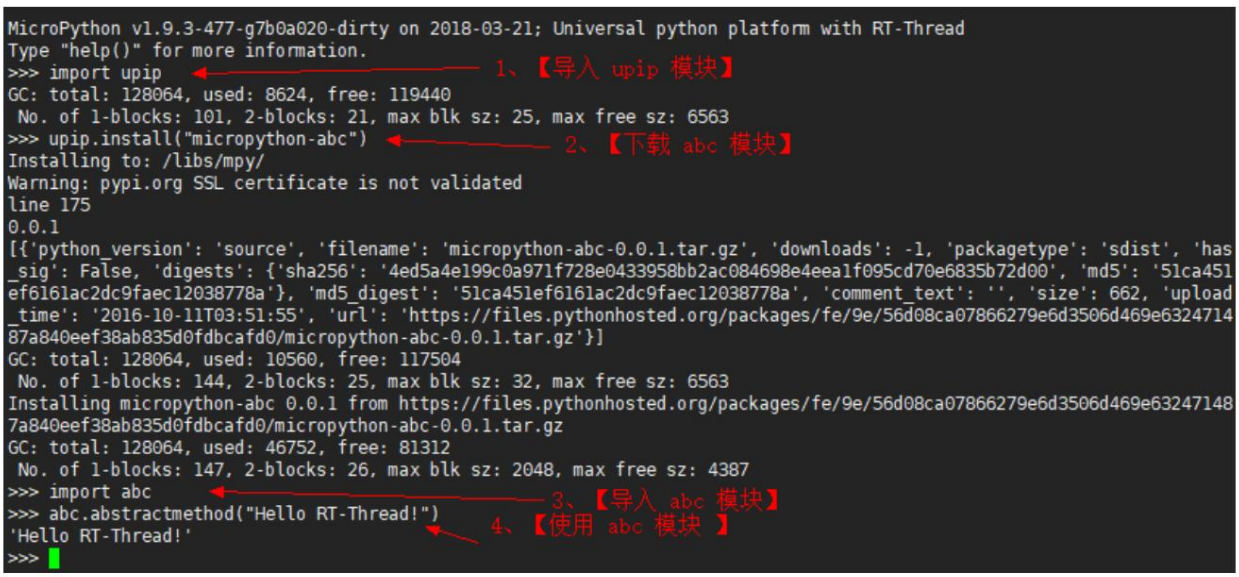

Figure **5.7:** *1525340694594*

In addition to the ones provided in micropython-lib, more MicroPython modules can be searched on pypi Search in.

!!! note "Note" When downloading a specific module, you need to add the micropython- prefix before the module . If you want to install the json module, use Use the upip.install("micropython-json") command.

Chapter 6

RT-Thread MicroPython Network Programming Guide

6.1 Preliminary Knowledge

- Before reading this network programming guide, you need to first understand the network module chapter of the MicroPython module and understand basic network connections. How to use the module.
- If you want to use more complex network functions, you need to open all modules in the tool module in menuconfig, and you must open the usocket module in the network module.

6.2 HttpClient

6.2.1 Obtain and install the urequests module

There are two ways to obtain this module. For detailed operations, please refer to the Package Management section:

- Method 1: Use the upip package management tool to download, here use the upip.install("micropython-urequests") command, upip The tool will automatically download and install the urequests module. The download process is shown in the figure:

```
>>> import upip
GC: total: 128064, used: 8480, free: 119584
 No. of 1-blocks: 101, 2-blocks: 21, max blk sz: 25, max free sz: 6580
>>> upip.install("micropython-urequests")
Installing to: /libs/mpy/
Warning: pypi.org SSL certificate is not validated
GC: total: 128064, used: 10576, free: 117488
 No. of 1-blocks: 143, 2-blocks: 24, max blk sz: 32, max free sz: 5998
Installing micropython-urequests 0.6 from https://files.pythonhosted.org/packages/c9/0c/3dd3d54ea3bd70ae3173209d
ee5c4aa75edbfb5f91abc79168ed367d9b1e/micropython-urequests-0.6.tar.gz
GC: total: 128064, used: 46912, free: 81152
 No. of 1-blocks: 146, 2-blocks: 24, max blk sz: 2048, max free sz: 3950
```

Figure **6.1:** *1525690379859*

- Method 2: Copy from MicroPython-lib to the /libs/mpy directory of the development board's file system.

Next the urequests module can be imported and used.

6.2.2 Use of the urequests module

The following example program uses the get command to grab the homepage information of http://www.baidu.com/ and format the output:

```
try:
    import urequests as requests
except ImportError: import
    requests

r = requests.get("http://www.baidu.com/") print(r)

print(r.content)
print(r.text) r.close()
```

6.3 HttpServer

6.3.1 Obtain and install the **MicroWebSrv** module

- First, clone the relevant files from https://github.com/jczic/MicroWebSrv.git to your local computer.
- Copy the www folder to the root directory of the file system (here the SD card is used as the root directory of the development board's file system).

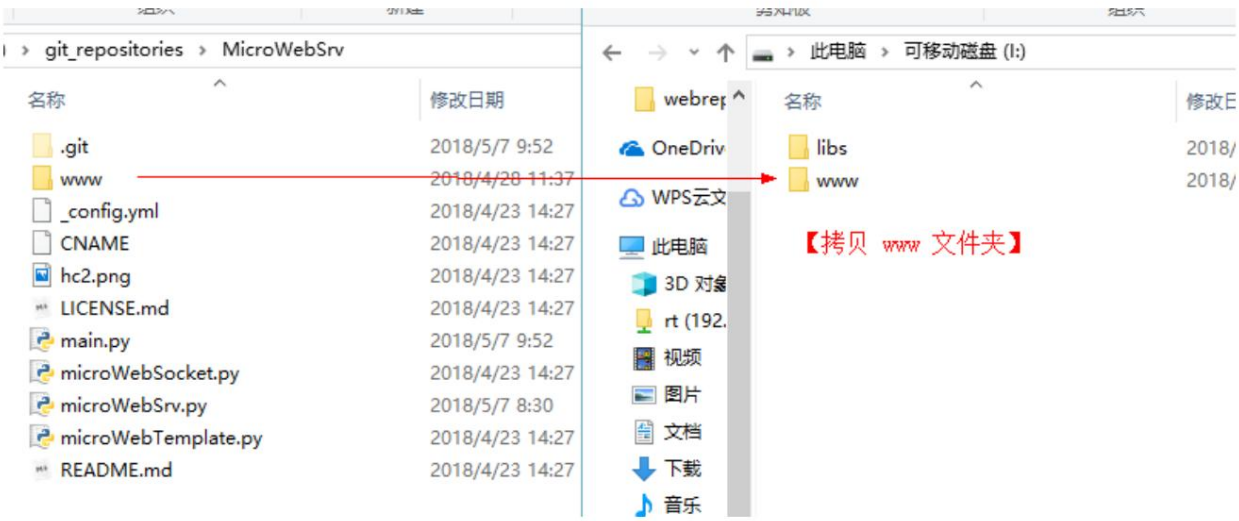

Figure **6.2:** *1525674983856*

- Copy the other files into the /libs/mpy/ folder.

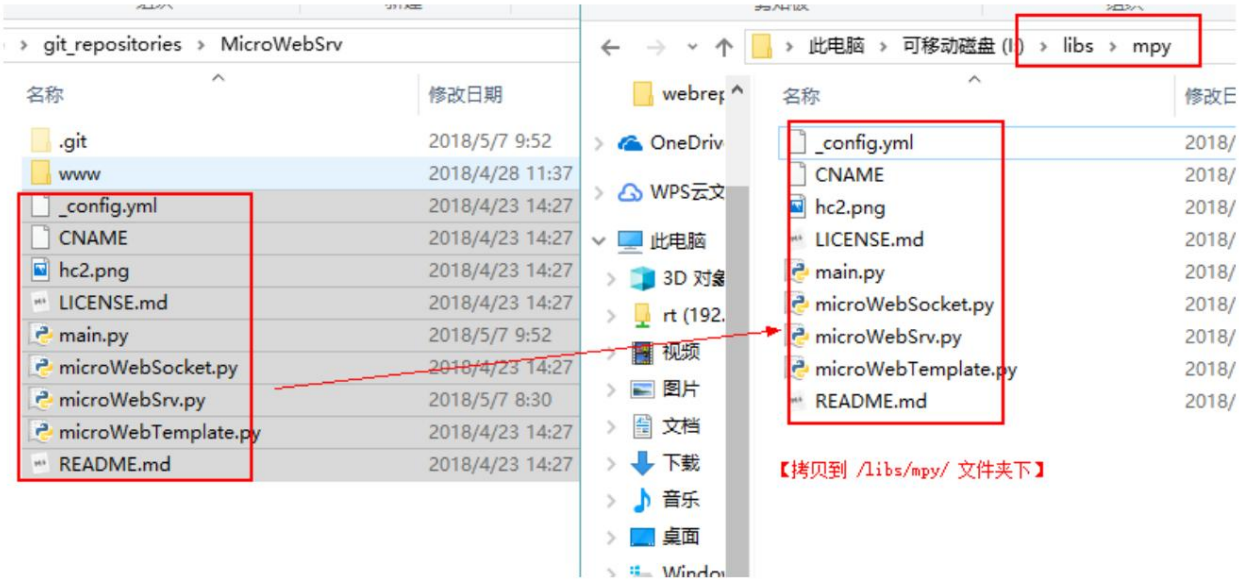

Figure **6.3:** *1525675205931*

- The MicroWebSrv module is now installed and can be imported directly using the import command in the MicroPython interactive command line
  -

6.3.2 Use of MicroWebSrv module

- In MSH, use the ifconfig command to view the IP address of the development board.
- Enter the python command to enter the MicroPython interactive command line.
- Use the import main command to start the web server.

```
msh />ifconfig
network interface: e0 (Default)
MTU: 1500
MAC: 00 04 9f 05 43 72
FLAGS: UP LINK_UP ETHARP IGMP
ip address: 192.168.12.23
gw address: 192.168.10.1
net mask   : 255.255.0.0
dns server #0: 192.168.10.1
dns server #1: 0.0.0.0
msh />python

MicroPython v1.9.3-477-g7b0a020-dirty on 2018-03-21; Universal python platform with RT-Thread
Type "help()" for more information.
>>> import main
```

Figure **6.4:** *1525659036361*

- Open the browser, enter the IP address of the development board in the address bar and press Enter to see the web page.

Figure **6.5:** *1525659139123*

• Enter the URL ip/test and fill in the example using the form.

Figure **6.6:** *1525659204069*

The following code completes the acquisition function of this table:

```
@MicroWebSrv.route('/test')
def _httpHandlerTestGet(httpClient, httpResponse) :
    content = """\
    <!DOCTYPE html>
    <html lang=en>
        <head>
            <meta charset="UTF-8" />
            <title>TEST GET</title>
        </head>
        <body>
            <h1>TEST GET</h1>
            Client IP address = %s
            <br />
            <form action="/test" method="post" accept-charset="ISO-8859-1">
                First name: <input type="text" name="firstname"><br />
                Last name: <input type="text" name="lastname"><br />
                <input type="submit" value="Submit">
            </form>
        </body>
    </html>
    """ % httpClient.GetIPAddr()
    httpResponse.WriteResponseOk( headers         = None,
                                  contentType     = "text/html",
                                  contentCharset  = "UTF-8",
                                  content         = content )
```

Figure **6.7:** *1525770427295*

• Click Submit and the server will return the information you filled in.

Figure **6.8:** *1525659232565*

The following code completes the data push function:

```
@MicroWebSrv.route('/test', 'POST')
def _httpHandlerTestPost(httpClient, httpResponse) :
    formData  = httpClient.ReadRequestPostedFormData()
    firstname = formData["firstname"]
    lastname  = formData["lastname"]
    content   = """\
    <!DOCTYPE html>
    <html lang=en>
        <head>
            <meta charset="UTF-8" />
            <title>TEST POST</title>
        </head>
        <body>
            <h1>TEST POST</h1>
            Firstname = %s<br />
            Lastname = %s<br />
        </body>
    </html>
    """ % ( MicroWebSrv.HTMLEscape(firstname),
            MicroWebSrv.HTMLEscape(lastname) )
    httpResponse.WriteResponseOk( headers         = None,
                                  contentType     = "text/html",
                                  contentCharset  = "UTF-8",
                                  content         = content )
```

Figure **6.9:** *1525770467078*

6.3.3 Modification of server functions

- If you want to use the server to implement the functions you need, you can modify the main.py file, import more modules, and use Python language to add more functionality.
- In the example of displaying the accelerometer and magnetometer on the web page, the following code completes the function of returning these data. You can refer to You can modify main.py by following the WebServer example to achieve the functions you want.

```
@MicroWebSrv.route('/sysdata')
def _httpHandlerTestGet(httpClient, httpResponse) :
    ax, ay, az = sensor.accelerometer
    mx, my, mz = sensor.magnetometer
    cpu_usage = machine.get_cpu_usage()
    cpu_value = cpu_usage[0] + cpu_usage[1] * 0.1
    ip = httpClient.GetIPAddr();

    content ={
        'status'      : 200,
        'body'        :{
        'status'      : 1,
        'result'      :{
            "versions":"3.0.3",
            "getTime":"1497594033",
            "cpuUtilization":cpu_value,
            "presentUtilization":1,
            "ipAddress": ip,
            "key":1,
            "Acceleration": {"accel_x" : ax, "accel_y" : ay, "accel_z" : az},
            "Magnetometer": { "mag_x"  : mx, "mag_y"   : my, "mag_z" :  mz}
        }
        }
            }
    headers = {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'POST,GET',
        'Access-Control-Allow-Headers': 'Content-Type, Authorization, Content-Length, X-Requested-With'
    }
    json_content = json.dumps(content)
    json_headers = json.dumps(headers)
    httpResponse.WriteResponseOk( headers         = headers,
                                  contentType     = "text/json",
                                  contentCharset  = "UTF-8",
                                  content         = json_content)
```

Figure **6.10:** *1525770559437*

6.4 MQTT

6.4.1 Obtain and install the **umqtt.simple** module

You can also use the two methods in package management to obtain it. If you use upip to install it, you can use upip.install("micropython-umqtt.simple") as shown in the figure:

```
>>> upip.install("micropython-umqtt.simple")
Installing to: /libs/mpy/
Warning: pypi.org SSL certificate is not validated
GC: total: 128064, used: 10592, free: 117472
 No. of 1-blocks: 143, 2-blocks: 24, max blk sz: 32, max free sz: 5790
Installing micropython-umqtt.simple 1.3.4 from https://files.pythonhosted.org/packages/bd/cf/697e3418b2f44222b3e
848078b1e33ee76aedca9b6c2430ca1b1aec1ce1d/micropython-umqtt.simple-1.3.4.tar.gz
GC: total: 128064, used: 46928, free: 81136
 No. of 1-blocks: 146, 2-blocks: 24, max blk sz: 2048, max free sz: 3742
```

Figure **6.11:** *1525690229174*

6.4.2 Using the umqtt.simple module

6.4.2.1 **MQTT** subscription function

- Use iot.eclipse.org as a test server

```
import time
```

```
from umqtt.simple import MQTTClient

# Publish test messages e.g. with: # mosquitto_pub
-t foo_topic -m hello

# Received messages from subscriptions will be delivered to this callback def sub_cb(topic, msg):
print((topic, msg))


def main(server="iot.eclipse.org"): # Test server is iot.eclipse.org
        c = MQTTClient("RT-Thread", server)
        c.set_callback(sub_cb)
        c.connect()
        c.subscribe(b"foo_topic")                    # Subscribe to the foo_topic topic
        while True:
                if True:
                        # Blocking wait for message
                        c.wait_msg()
                else:
                        # Non-blocking wait for message
                        c.check_msg()
                        # Then need to sleep to avoid 100% CPU usage (in a real # app other useful
                        actions would be performed instead) time.sleep(1)


        c.disconnect()

if __name__ == "__main__":
        main()
```

- Use the python command to execute the above code file, it will connect to the MQTT server and receive the data we publish from another client.

  Content with foo_topic as the topic

```
msh /libs/mpy>python example_sub.py

(b'foo_topic', b'Hello RT-Thread !!!')
(b'foo_topic', b'Today is a sunnyday !!!')
```

Figure **6.12:** *1525665942426*

6.4.2.2 **MQTT** publishing function

- After executing the following code, the information with the topic foo_topic will be published to the MQTT server

```
from umqtt.simple import MQTTClient

# Test reception e.g. with: #
mosquitto_sub -t foo_topic
```

```
def main(server="iot.eclipse.org"):
        c = MQTTClient("SummerGift", server) c.connect()

        c.publish(b"foo_topic", b"Hello RT-Thread !!!") c.disconnect()


if __name__ == "__main__":
        main()
```

6.5 OneNET

6.5.1 Preparation

• First, you need to install the urequests module and the umqtt.simple module. For installation instructions, refer to the HttpClient and MQTT chapters. • The sample code for this chapter is in the appendix of the last section. You can copy it to the main.py file in MSH after adding the necessary registration information.

Use the python command to execute.

6.5.2 Product Creation

• If you want to connect the development board to the OneNET cloud platform, you must first create a product. Creation is divided into two steps. The first step is to register a user account, and the second step is to create a product based on a specific protocol.

6.5.2.1 User Registration

• In order to use the powerful functions of OneNET Device Cloud, you must first register a developer account on OneNET to create a dedicated "Development Center.

6.5.2.2 Product Creation

• Next, you need to create a product on the OneNET platform. Please note that when you select the device access method and device access protocol, since this example uses the MQTT protocol, you need to select the public protocol in the device access method and the device access protocol.

MQTTÿ

Figure **6.13:** *1525764833130*

6.5.3 Hardware Access

This chapter will introduce how to connect the device to the OneNET cloud platform. It will also demonstrate how the cloud platform sends commands to the device and the device sends commands to the cloud.

Example of the platform returning the number of times a command was sent.

6.5.3.1 Device Registration and Access

- After successfully creating the device, record the obtained product ID for later use in pushing data.

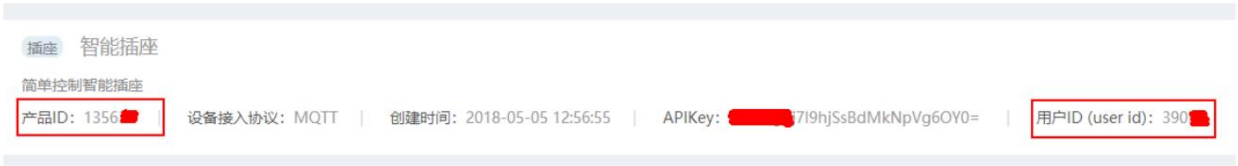

Figure **6.14:** *1525765493792*

- Record the device's official environment registration code for use in registering new devices.

Figure **6.15:** *1525765209683*

• Next, open main.py in the example program and modify sn to the device's unique identifier and product_id to the 6-digit product ID obtained above.

regKey is the official environment registration code recorded above.

```
def main():
    sn = 'RT_Thread_Test_Product'          #1、填入设备唯一标识符
    title = 'Device' + sn
    product_id = '[redacted]5636'          #2、填入创建设备时获得的产品 ID
    regKey = '[redacted]3uy7LZsqblMe'      #3、填入正式环境注册码
    url = 'http://api.heclouds.com/register_de?register_code=' + regKey
    reg = register.Register(url=url, title=title, sn=sn)      #根据上面的信息注册设备，如果已经注册不再重复注册
    if reg.regist()==0:
        mq = mqtt.mqtt(client_id=reg.device_id, username=product_id, password=reg.key) #开启 MQTT 服务
        mq.connect()
```

Figure **6.16:** *1525766961043*

• Run main.py on the development board and you can see the device we registered on OneNET.

```
msh /libs/mpy>python main.py

Connected to 183.230.40.39, subscribed to topic_sub topic.
```

Figure **6.17:** *1525767092149*

• The device named DeviceRT_Thread_Test_Product has been registered and is online.

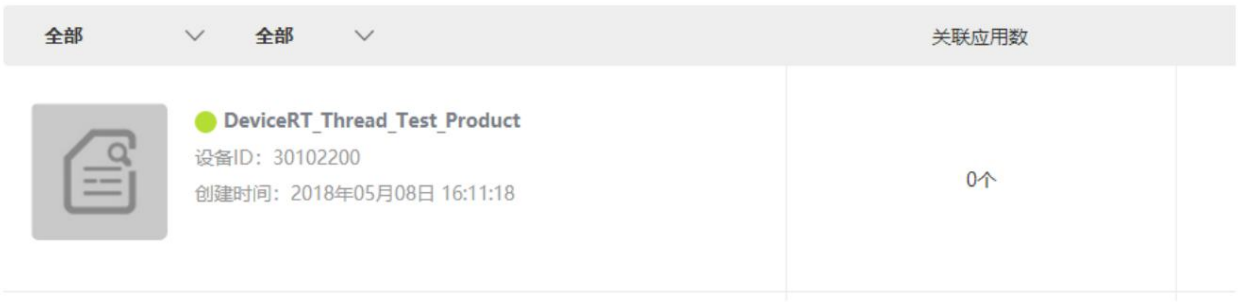

Figure **6.18:** *1525767167244*

6.5.3.2 The cloud platform sends commands to the device

• You can use the Send Command function to send several sets of commands to the development board.

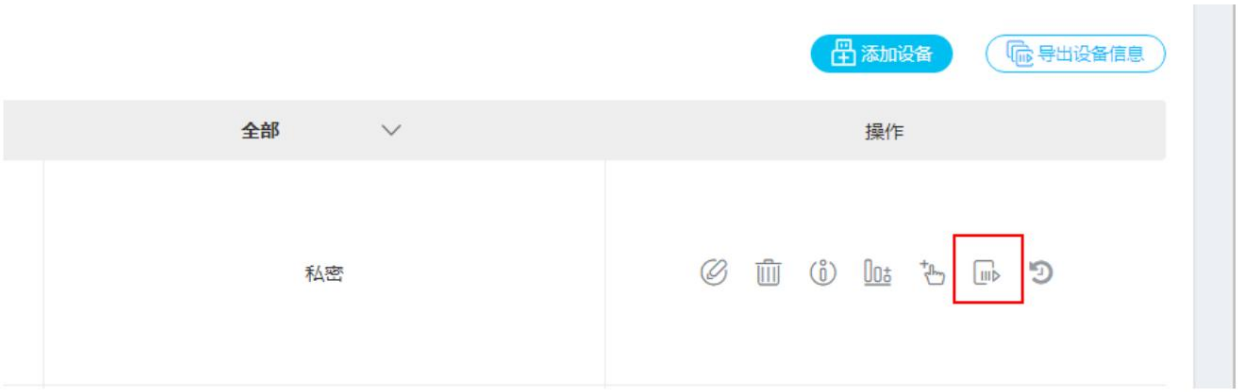

Figure **6.19:** *1525767264155*

Figure **6.20:** *1525767369050*

• You can see the data sent by the cloud platform on the device side, and the device side will also upload the data of the number of times the command is sent

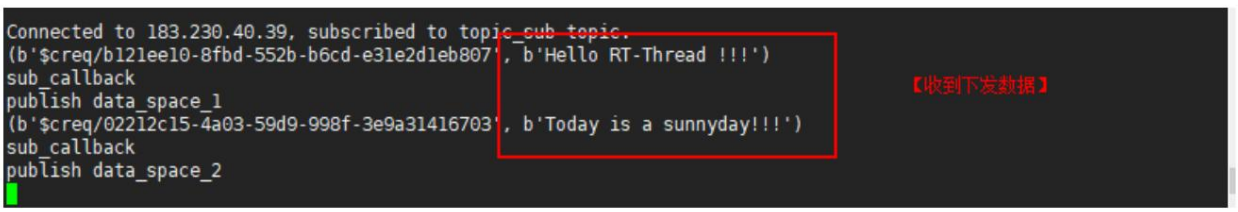

Figure **6.21:** *1525767520609*

6.5.3.3 Devices upload data to the cloud platform

• Click on the data stream management function to view the data uploaded by the device

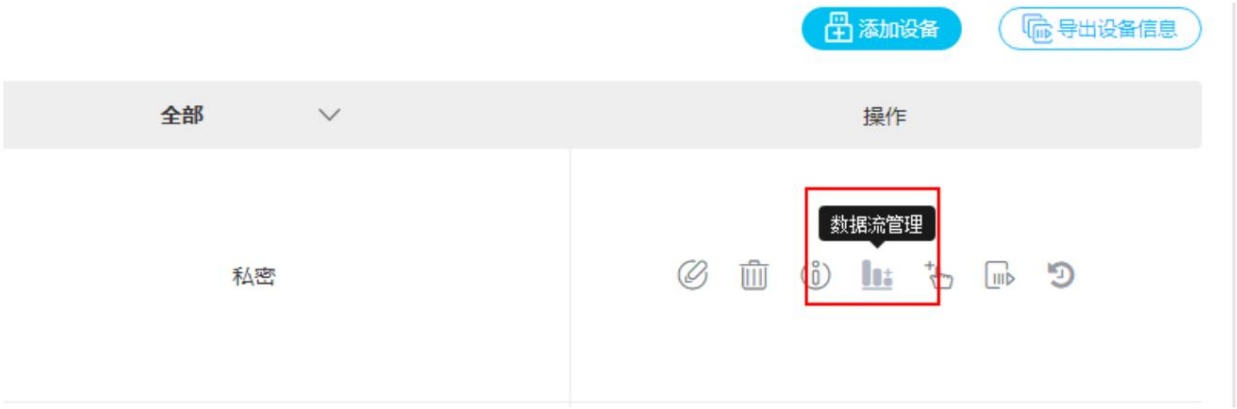

Figure **6.22:** *1525767621708*

• You can see the number of times the command sent by the device is uploaded in the data flow management switch

Figure **6.23:** *1525767740215*

• The switch data here is modified in the pubData function of mqtt.py, and can be used to pass the content of the value object to the cloud platform.

Upload different data to the platform.

```
def pubData(self, t):
    value = {'datastreams':[{"id":"switch","datapoints":[{"value": self.cmd_times }]}, {"id":"humi
    jdata = json.dumps(value)
    jlen = len(jdata)
    bdata = bytearray(jlen+3)
    bdata[0] = 1 # publish data in type of json
    bdata[1] = int(jlen / 256) # data lenght
    bdata[2] = jlen % 256      # data lenght
    bdata[3:jlen+4] = jdata.encode('ascii') # json data
    print('publish data', str(self.pid + 1))
    try:
        self.mqttClient.publish('$dp', bdata)  # $dp 为特殊系统 topic，可以通过这个 topic 给系统推
        self.cmd_times += 1
```

Figure **6.24:** *1525774755758*

• Now the device and OneNET cloud platform are connected.

6.5.3.4 Adding a Standalone Application

- For ease of use, you can also add independent applications to the device, the effect is as follows:

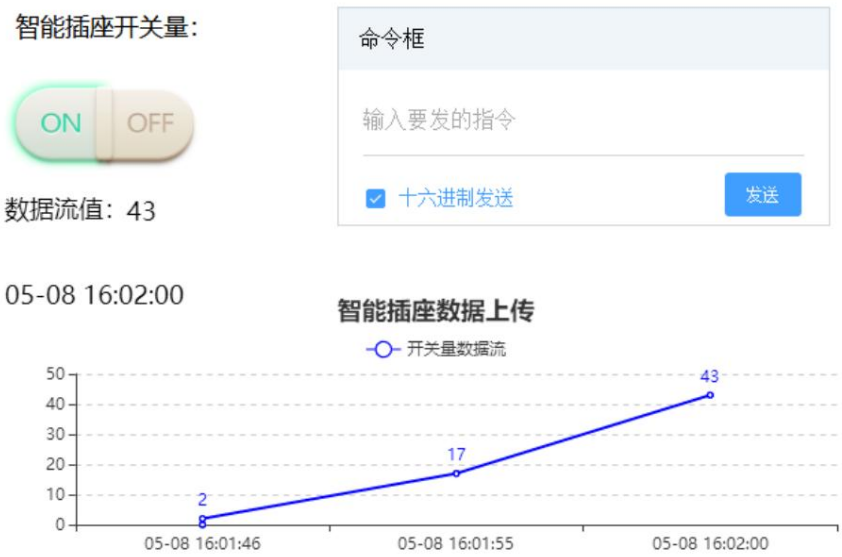

Figure **6.25:** *1525768143233*

6.5.4 Code Explanation

- Modify the data sent to the server by modifying the value object. Here, it is sent to a special system topic $dp

```
def pubData(self, t):
    value = {'datastreams':[{"id":"switch","datapoints":[{"value": self.cmd_times }]}, {"id":"humi","datapoints":[{"value":0}]}]}
    jdata = json.dumps(value)
    jlen = len(jdata)
    bdata = bytearray(jlen+3)
    bdata[0] = 1 # publish data in type of json
    bdata[1] = int(jlen / 256) # data lenght
    bdata[2] = jlen % 256      # data lenght
    bdata[3:jlen+4] = jdata.encode('ascii') # json data
    print('publish data', str(self.pid + 1))
    try:
        self.mqttClient.publish('$dp', bdata)  # $dp 为特殊系统 topic，可以通过这个 topic 给系统推送信息,但是不能订阅这个 topic
        self.cmd_times += 1
        self.failed_count = 0
    except Exception as ex:
        self.failed_count += 1
        print('publish failed:', ex.message())
        if self.failed_count >= 3:
            print('publish failed three times, esp resetting...')
            reset()
```

Figure **6.26:** *1525774792209*

- The trigger condition for sending data to the server is receiving a command from the server. In this way, there is no guarantee that data will always be sent to the server, so the MQTT connection may be disconnected after a period of no data exchange.

```
def connect(self):
    self.mqttClient.set_callback(self.sub_callback)
    self.mqttClient.connect()
    self.mqttClient.subscribe(self.topic)
    print("Connected to %s, subscribed to %s topic." % (self.server, self.topic))
    try:
        while 1:
            #self.mqttClient.wait_msg()
            self.mqttClient.check_msg()
            self.pubData('x')
```

Figure **6.27:** *1525768433046*

6.5.4.1 Appendix Example Code

```
import urequests as requests from
umqtt.simple import MQTTClient import ujson as
json import time


class Register(): def
    __init__(self, url='', title='', sn='', mac=''):
        self.url = url
        self.title = title
        self.sn = sn
        self.mac = mac
        self.sock = None
        self.tjson = {}
        self.error = 0
        self.key =
        self.device_id =

    def regist(self): assert
        self.url is not None, "Url is not set"
        _, _,     host, path = self.url.split('/', 3)
        if host == '':
            return
        device = {"mac":self.mac} if self.sn ==                    '' else {"sn":self.sn}
        if self.title != '':
            device['title'] = self.title jdata =
        json.dumps(device)

        resp = requests.post(self.url, data=jdata) if resp: self.tjson =
        resp.json()
            if self.tjson['errno'] == 0:

                self.key = self.tjson['data']['key'] self.device_id =
                self.tjson['data']['device_id']
            return 0
        else:
```

```
                return -1

class OneNetMqtt:
    failed_count = 0

    def __init__(self, client_id='', username='', password=''):
        self.server = "183.230.40.39"
        self.client_id = client_id
        self.username = username
        self.password = password self.topic
        = "topic_sub" self.mqttClient =                    # Fill in the test topic
        MQTTClient(self.client_id, self.server,6002,self.username,
            self.password)
        self.cmd_times = 0                                 # publish count

    def pubData(self, t): value =
        {'datastreams':[{"id":"switch","datapoints":[{"value": self.
            cmd_times }]}]} jdata
        = json.dumps(value) jlen = len(jdata)
        bdata = bytearray(jlen+3)
        bdata[0] = 1 bdata[1] = int(jlen / 256)
        bdata[2] = jlen %                                  # publish data in type of json # data lenght #
        256 bdata[3:jlen+4] = jdata.encode('ascii')        data lenght
        # json data print('publish data',
        str(self.cmd_times + 1)) try: self.mqttClient.publish('$dp', bdata) # $dp is a topic,
        and the source code is a source code.


                This topic pushes information to the system, but cannot be subscribed to this topic
            self.cmd_times += 1
            self.failed_count = 0
        except Exception as ex:
            self.failed_count += 1 print('publish
            failed:', ex.message()) if self.failed_count >= 3:

                print('publish failed three times, esp resetting...') reset()


    def sub_callback(self, topic, msg): print((topic,msg))
        cmd =
        msg.decode('ascii').split(" ") print('sub_callback')


    def connect(self):
        self.mqttClient.set_callback(self.sub_callback) self.mqttClient.connect()
        self.mqttClient.subscribe(self.topic)
        print("Connected to %s, subscribed to %s topic." %
        (self.server, self.topic)
            )
```

```
            try:
                while True:
                    self.mqttClient.check_msg()
                    print("pubdata")
                    self.pubData('x')

            finally:
                self.mqttClient.disconnect() print('MQTT
                closed')

def main():
    sn = 'RT_Thread_Test_Product'                    #1. Fill in the device unique identifier
    title = 'Device' + sn
    product_id = 'XXXXX' regKey                      #2. Enter the product ID obtained when creating the device. #3.
    = 'XXXXXXXX' url = 'http://                      Enter the official environment registration code.
    api.heclouds.com/register_de?register_code=' + regKey

    reg = Register(url=url, title=title, sn=sn) Once registered, no    #Register the device according to the above information, if
        more registration is required
    if reg.regist()==0:
        MQTT = OneNetMqtt(client_id=reg.device_id, username=product_id, password=reg .key) #Enable MQTT service

        MQTT.connect()
    else:
        print('Error: No Client ID!')

if __name__ == "__main__":
    main()
```

Chapter **7**

MicroPython Development Resources

- RT-Thread MicroPython Development Manual
- RT-Thread MicroPython source code
- RT-Thread MicroPython Forum
- MicroPython official website
- Official online documentation
- MicroPython online demo
- MicroPython source code
- MicroPython official forum
- MicroPython Chinese Community