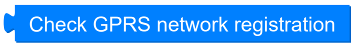
IoT Base Catm

API



```
catmiot.get_gprs_network_registration()
```

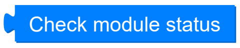
- Check whether the device is registered in the GPRS network.

```
catmiot.get_network_registration()
```

- Check whether the device is registered in a regular network.

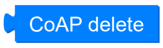
```
catmiot.get_single_quality()
```

- Check the current signal quality of the device.



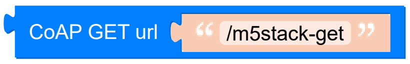
```
catmiot.check_status()
```

- Check the status of the module to see if it is functioning properly.



```
catmiot.delete_coap()
```

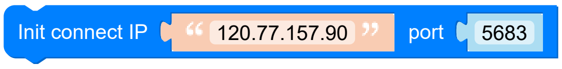
- Send a DELETE request via the CoAP protocol to delete resources on the server.



```
catmiot.coap_request('/m5stack-get')
```

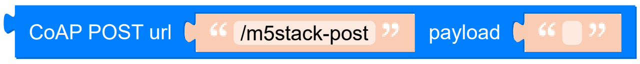
- Send a GET request via the CoAP protocol to retrieve resources from the specified URL.

    - /m5stack-get: The specified URL path from which to retrieve resources.



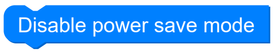
```
catmiot.coap_to_connect('120.77.157.90', 5683)
```

- Initialize a connection to the specified IP address and port for CoAP protocol communication.
    - IP: 120.77.157.90
    - Port: 5683



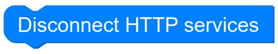
```
catmiot.coap_request('/m5stack-post', 2, '')
```

- Send a POST request via the CoAP protocol to the specified URL.
    - URL: /m5stack-post
    - Payload: Data payload to be sent.



```
catmiot.power_save_mode(0, 0, 0, 0, False)
```

- Disable the device's power-saving mode to ensure it maintains high-performance operation.

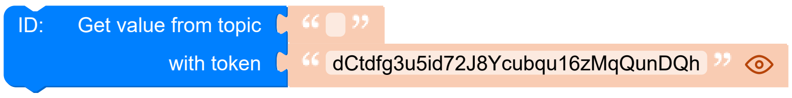


```
catmiot.disconnect_server()
```

- Disconnect the device from the current HTTP service.

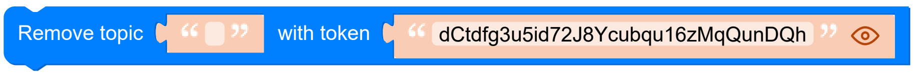
```
catmiot.enable_PDP_context()
```

- Enable the PDP context, typically used for GPRS or LTE network communication, ensuring that the data transmission context is established.



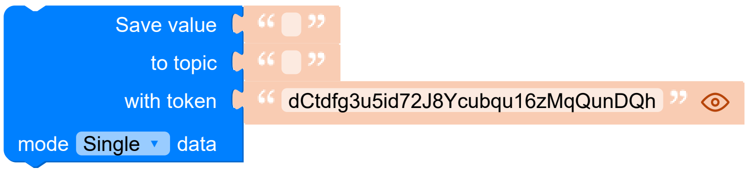
```
catmiot.get_ezdata(ezdata_get_kslNzcb, 'GCJ3Ic5h2eXnzV3rT3bBXvrncCaJnART', '')
```

- Retrieve data from the specified topic.
  - Token: dCtdfg3u5id72J8YCubqu16zMqQunDQh



```
catmiot.remove_ezdata('GCJ3Ic5h2eXnzV3rT3bBXvrncCaJnART', '')
```

- Delete the specified topic from the remote server.
  - Token: dCtdfg3u5id72J8YCubqu16zMqQunDQh

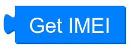


```
catmiot.set_ezdata('GCJ3Ic5h2eXnzV3rT3bBXvrncCaJnART', '', '', 0)
```

- Save data to the specified topic and authenticate with the specified token.
  - Token: dCtdfg3u5id72J8YCubqu16zMqQunDQh
  - Mode: Single (indicating single data save mode).
  - Data: The data to be saved.

```
catmiot.get_CCID()
```

- Get the current SIM card's CCID (Integrated Circuit Card Identifier), which is the unique identifier for the SIM card.

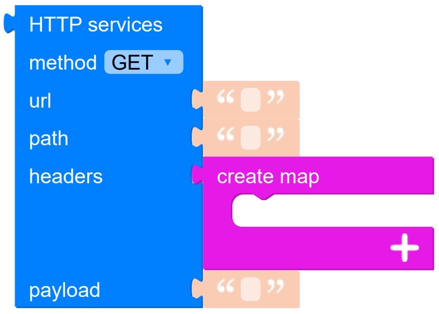


```
catmiot.get_IMEI()
```

- Get the device's IMEI (International Mobile Equipment Identity), which is the unique identifier for the device.

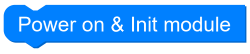
```
catmiot.gprs_service(1)
```

- Set or check the GPRS service status. Here it is set to ACTIVE, meaning GPRS service is enabled.



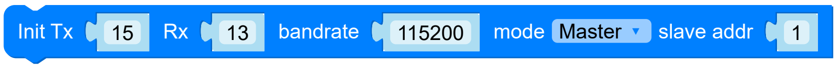
```
catmiot.http_service(1, '', '', {}, '')
```

- Send an HTTP GET request to fetch data from a remote server.
  - Method: GET (HTTP request method)
  - URL: The requested address.
  - Headers: Create a Map containing HTTP request headers.
  - Payload: For POST requests, data can be passed through this parameter.



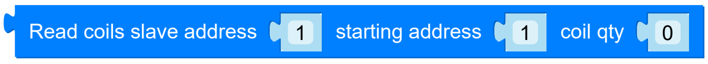
```
catmiot.init_modem(True)
```

- Start and initialize the module, preparing it for subsequent communication and data interaction.



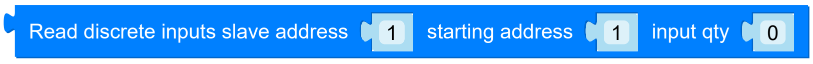
```
catmiot.modbus_init(15, 13, 115200, 1, 1)
```

- Initialize the UART communication interface, set the TX (transmit) pin to 15, RX (receive) pin to 13, baud rate to 115200, mode to master, and slave address to 1.



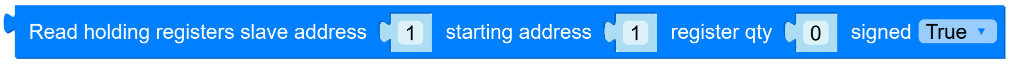
```
modbus.read_coils(1, 1, 0)
```

- Read the coil status from slave address 1, starting at address 1, reading 0 coils.


Read discrete inputs slave address 1 starting address 1 input qty 0

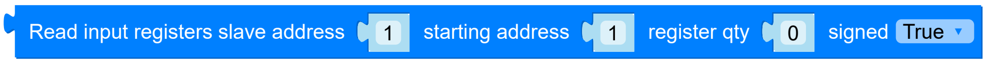
```
modbus.read_discrete_inputs(1, 1, 0)
```

- Read the discrete input status from slave address 1, starting at address 1, reading 0 inputs.


Read holding registers slave address 1 starting address 1 register qty 0 signed True

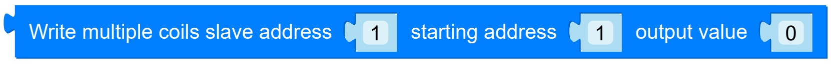
```
modbus.read_holding_registers(1, 1, 0, True)
```

- Read data from the holding registers of slave address 1, starting at address 1, reading 0 registers. The data is treated as signed values.


Read input registers slave address 1 starting address 1 register qty 0 signed True

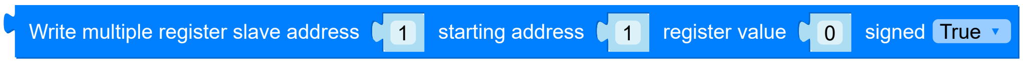
```
modbus.read_input_registers(1, 1, 0, True)
```

- Read data from the input registers of slave address 1, starting at address 1, reading 0 registers. The data is treated as signed values.


Write multiple coils slave address 1 starting address 1 output value 0

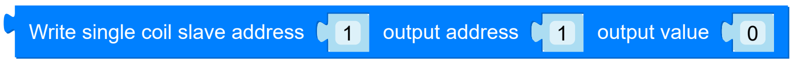
```
modbus.write_multiple_coils(1, 1, 0)
```

- Write data to multiple coils of slave address 1, starting at address 1, with the output value set to 0.


Write multiple register slave address 1 starting address 1 register value 0 signed True

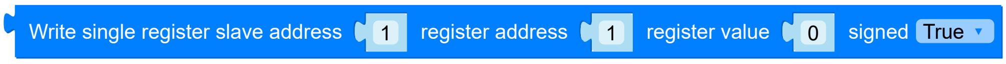
```
modbus.write_multiple_registers(1, 1, 0, True)
```

- Write data to multiple registers of slave address 1, starting at address 1, with the register value set to 0. The data is treated as signed values.


Write single coil slave address 1 output address 1 output value 0

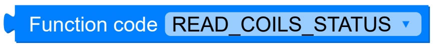
```
modbus.write_single_coil(1, 1, 0)
```

- Write a value to a single coil of slave address 1, with the output address set to 1 and the value written as 0.


Write single register slave address 1 register address 1 register value 0 signed True

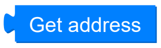
```
modbus.write_single_register(1, 1, 0, True)
```

- Write data to a single register of slave address 1, with the register address set to 1 and the value written as 0. The data is treated as signed values.



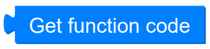
```
1~6,15,16
```

- Set the MODBUS function code to READ_COILS_STATUS, used to read the status of coils.



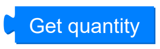
```
modbus.find_address
```

- Get the slave address currently in use for MODBUS communication.



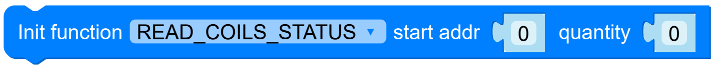
```
modbus.find_function
```

- Get the function code currently being executed in MODBUS operation.



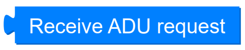
```
modbus.find_quantity
```

- Get the quantity of values being read or written in the current MODBUS request.



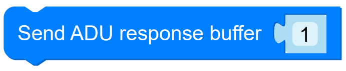
```
modbus.function_init(1, 0, 0)
```

- Initialize the MODBUS slave function code as READ_COILS_STATUS to read the status of coils starting from the specified address and quantity.



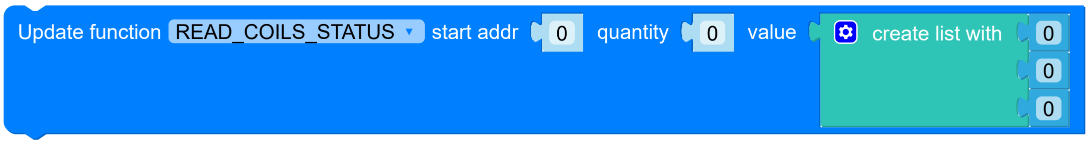
```
modbus.receive_req_create_pdu()
```

- Receive an ADU (Application Data Unit) request, indicating that data has been received from the master device.



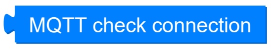
```
modbus.create_slave_response(1)
```

- Send the ADU response data buffer, responding to the master's request.


Update function READ_COILS_STATUS start addr 0 quantity 0 value create list with 0 0 0

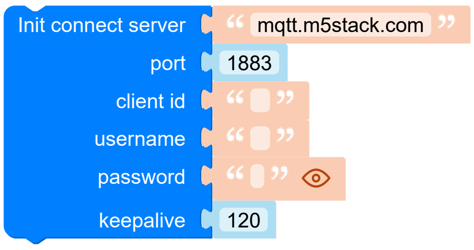
```
modbus.update_process(1, 0, 0, [0, 0, 0])
```

- Update the READ_COILS_STATUS function, read the status of the specified number of coils from the starting address, and return the corresponding data list.


MQTT check connection

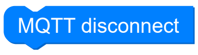
```
catmiot.mqtt_ischeck_connect()
```

- Check the connection status of the MQTT server.


Init connect server "mqtt.m5stack.com"
port 1883
client id ""
username ""
password ""
keepalive 120

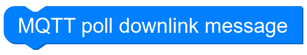
```
catmiot.mqtt_to_connect('mqtt.m5stack.com', 1883, '', '', '', 120)
```

- Initialize an MQTT server connection, setting the server address, port, client ID, username, password, and keep-alive time.


MQTT disconnect

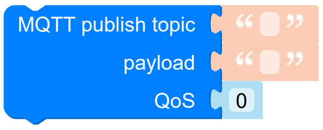
```
catmiot.mqtt_disconnect()
```

- Disconnect from the MQTT server.


MQTT poll downlink message

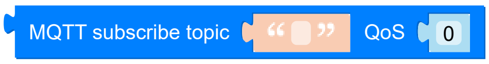
```
catmiot.mqtt_poll()
```

- Poll for downlink messages to check if the MQTT server has any new messages sent to the device.



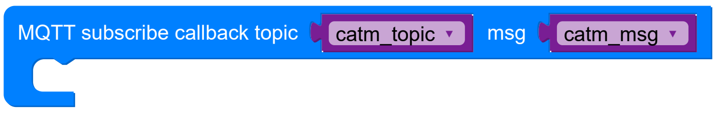
```
catmiot.mqtt_publish('', '', 0)
```

- Publish a message to the specified MQTT topic, including the message payload and QoS (Quality of Service) setting.



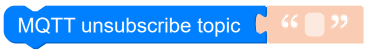
```
catmiot.mqtt

_subscribe('', iotbasecatm_mqtt_cb, 0)
```

- Subscribe to the specified MQTT topic and set the QoS (Quality of Service) level.



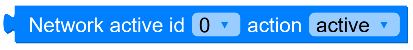
```
def iotbasecatm_mqtt_cb(catm_mq_topic, catm_mq_payload):
  global ezdata_value1, catm_topic, catm_msg
  catm_topic = catm_mq_topic
  catm_msg = catm_mq_payload
  pass
```

- Set the callback function for when an MQTT topic subscription receives a message, handling the received topic and message.



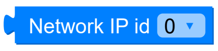
```
catmiot.mqtt_unsubscribe('')
```

- Unsubscribe from the specified MQTT topic.



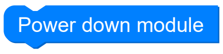
```
catmiot.network_active(0, 1)
```

- Activate the specified network ID and set its action to active.



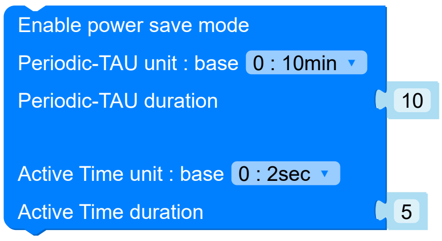
```
catmiot.get_network_ip(0)
```

- Retrieve the IP address of the specified network.


Power down module

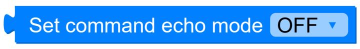
```
catmiot.poweroff()
```

- Power off the module.


Enable power save mode
Periodic-TAU unit : base 0 : 10min
Periodic-TAU duration 10
Active Time unit : base 0 : 2sec
Active Time duration 5

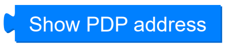
```
catmiot.power_save_mode(0, 10, 0, 5)
```

- Enable power-saving mode and set the periodic TAU (time interval) and active time period.
  - Periodic-TAU: Set in units of 10 minutes, 10 means the wake-up interval is 10 x 10 minutes, or 100 minutes.
  - Active Time: Set in units of 2 seconds, 5 means the device will remain active for 10 seconds after waking up.


Set command echo mode OFF

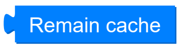
```
catmiot.set_command_echo_mode(0)
```

- Set the command echo mode to OFF, i.e., disable command echoing.


Show PDP address

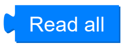
```
catmiot.show_PDP_address()
```

- Display the PDP (Packet Data Protocol) address, typically used for checking the network connection status of the device.


Remain cache

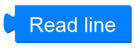
```
modbus._mdbus_uart.any()
```

- Check if there is any remaining data in the UART buffer.



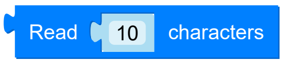
```
modbus._mdbus_uart.read()
```

- Read all available data from the UART.



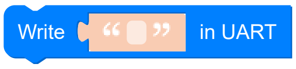
```
modbus._mdbus_uart.readline()
```

- Read one line of data from the UART until encountering a newline character (typically \n).



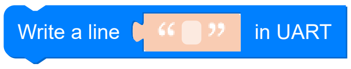
```
modbus._mdbus_uart.read(10)
```

- Read a specified number of characters from the UART, here set to read 10 characters.



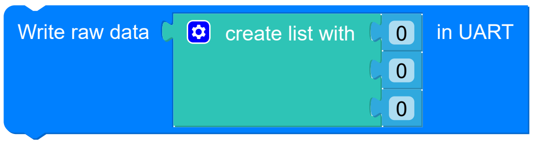
```
modbus._mdbus_uart.write('')
```

- Write the specified data to the UART.



```
modbus._mdbus_uart.write(''+"\r\n")
```

- Write one line of data to the UART, usually accompanied by a newline character (\n).



```
modbus._mdbus_uart.write(bytes([0, 0, 0]))
```

- Write a raw data list to the UART.
  - Here, a list is created containing three values 0, 0, 0, indicating that these values will be sent via UART.