RT-THREAD AT Component Application Notes **-** Client

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Table of contents

This application note introduces the basics of RT-Thread AT components and AT client Usage method to help developers better use RT-Thread AT components.

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

With the gradual popularization of AT commands, more and more embedded products use AT commands. AT commands serve as the protocol interface between the main chip and the communication module. The hardware interface is generally a serial port, so that the main control device can complete a variety of operations through simple commands and hardware design.

Although AT commands have been standardized to a certain extent, the AT commands supported by different chips are not completely unified, which directly increases the complexity of user use. There is no unified processing method for sending and receiving AT commands and parsing data. And when using AT devices to connect to the network, only simple device connection and data transmission and reception functions can be completed through commands. It is difficult to adapt to the upper-layer network application interface, which is not conducive to the development of product equipment.

In order to facilitate users to use AT commands and simply adapt to different AT modules, RT-Thread provides AT components for AT device connection and data communication. The implementation of AT components includes two parts: client and server. For embedded devices, in most cases, devices use AT components as clients to connect to server devices, so this article will focus on the main functions, transplantation methods and implementation principles of the client in the AT component, and introduce the implementation of the standard BSD Socket API based on the client, and use AT commands to complete complex network communications.

1.2 Structure of this paper

This application note will introduce the RT-Thread AT component from the following aspects:

• AT Component Introduction • AT Client Configuration and Usage • AT Socket Configuration and Usage

2 Problem Statement

This application note will introduce the RT-Thread AT component around the following issues.

• What is an AT command? What is the main function of the AT component? • What is an AT Client and how does it interact with the AT Server? • How to use AT commands to implement the standard BSD Socket API and support multiple network software packages and functions?

To solve these problems, we need to understand the basic principles and functional usage of RT-Thread AT components. The configuration, migration and function usage of AT Client are introduced step by step, so that users can quickly get started with AT Client functions.

3. Problem Solving

3.1 Introduction to AT commands

AT command is a method used for device connection and data communication **between AT** client **and AT** server . Its basic structure is shown in the figure below:

Figure **1:** *AT* Order

3.1.1. Basic concepts of AT commands

- Generally, an AT command consists of three parts: prefix, body and terminator. The prefix consists of the characters AT; the body consists of the command, parameters and possible data; the terminator is generally <CR><LF> (\r\n).
- The implementation of AT function requires the joint efforts of AT Server and AT Client;
- AT Server is mainly used to receive commands sent by AT Client, determine the received commands and parameter formats, and send them Corresponding response data, or actively sending data;
- AT Client is mainly used to send commands, wait for AT Server response, and respond to AT Server response data or main The data sent automatically is analyzed and processed to obtain relevant information.
- AT Client and AT Server support multiple data communication methods (UART, SPI, etc.), the most common The serial UART communication method is used.
- The data types received by AT Client are divided into two types: response data and URC data.
  - Response data: the response status and information received by the AT Server after the AT Client sends a command;
  - **URC** data: the data that the AT Server actively sends to the AT Client, which usually appears in some special situations, such as WIFI connection disconnection, TCP receiving data, etc. These situations often require users to take corresponding actions.

3.1.2. **AT** Component Introduction

The AT component is an implementation of the AT Server and AT Client based on the RT-Thread system . The component completes the entire AT command data interaction process, including sending AT commands, judging command formats and parameters, responding to commands, receiving response data, parsing response data, and processing URC data.

Through the AT component, the device can be used as an AT Client to connect to other devices using the serial port to complete data transmission, reception and analysis. It can be used as an AT Server to allow other devices or even PC devices to connect to complete data response. The CLI mode can also be started in the local shell to enable the device to support both AT Server and AT Client functions. It is mostly used for device development and debugging.

3.2 AT Client Function

This article will give an AT group based on the Zhengdian Atom STM32F4 Explorer Development Board and Espressif ESP8266 Development Board. The configuration, migration and use of AT Client functions in the software.

The following figure shows the baseboard of the two development boards used in this article. Developers can use the ESP8266 development board or module. If the STM32F4 Explorer development board is missing, other development boards with additional serial ports can be used instead. It is necessary to ensure that the development board can run the RT-Thread system normally and the serial port is working properly:

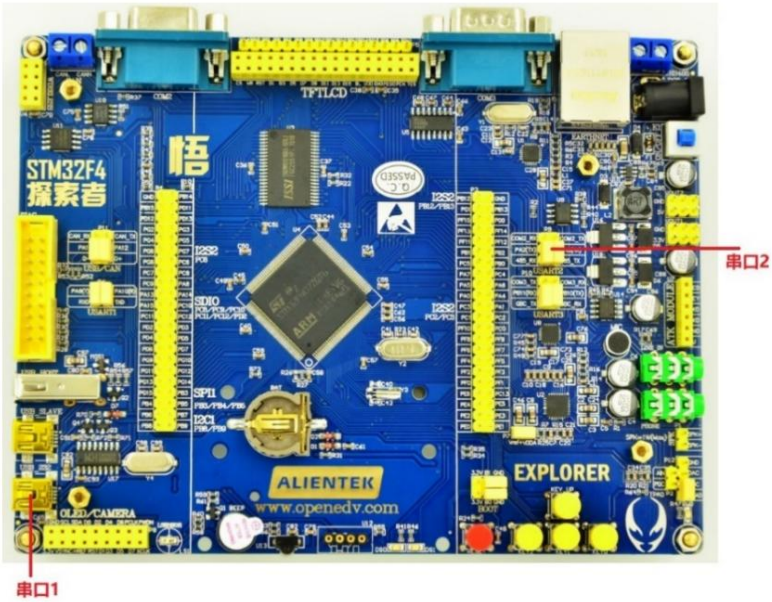

Figure 2: *STM32F4* Baseboard

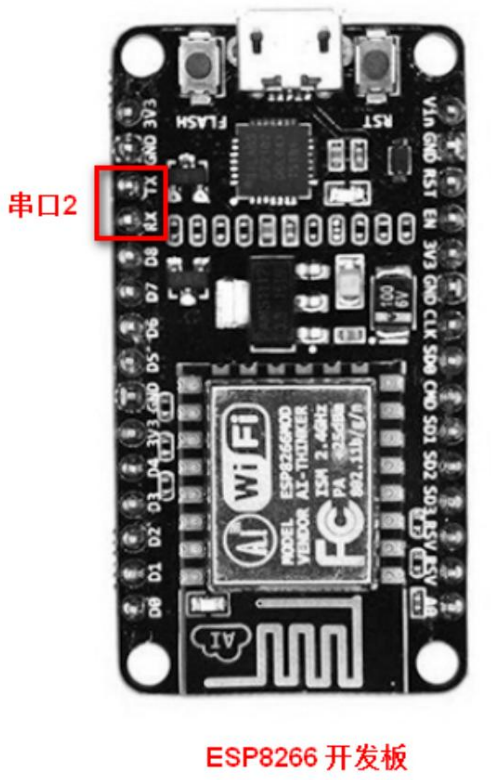

Figure **3:** *ESP8266* Baseboard

In the AT component, AT Client mainly completes the sending of AT commands and the receiving and parsing of response data.

Use the serial port 2 of the Zhengdian Atom STM32F4 Explorer development board as the AT Client to connect to the serial port 2 of the ESP8266 development board.

The serial port 2 of the ESP8266 development board is used as the AT Server to complete the functions of sending, receiving and parsing AT Client data.

Specific introduction to configuration, transplantation and usage is given.

3.2.1. AT Client Configuration

1. Download RT-Thread source code
2. Download the env tool
3. Open the env tool, enter the rt-thread\bsp\stm32f4xx-HAL directory, and enter menu-

config Enter the configuration interface to configure the project.

- Configure serial port support: Check the Using UART1 and Using UART2 options, and select the chip model as STM32F407ZG, the external clock source is 8MHz.

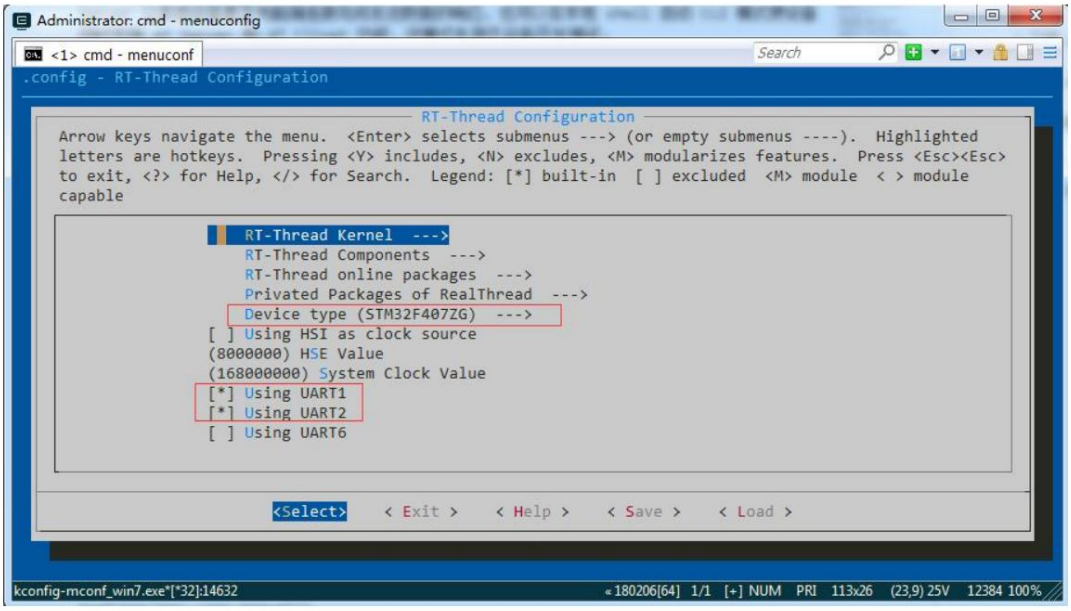

Figure 4: Configuring serial port support

• Configure the shell device: RT-Thread Kernel —> Kernel Device Object —> Modify the device

The name for console is uart1, and the default device of the shell is configured as serial port 1.

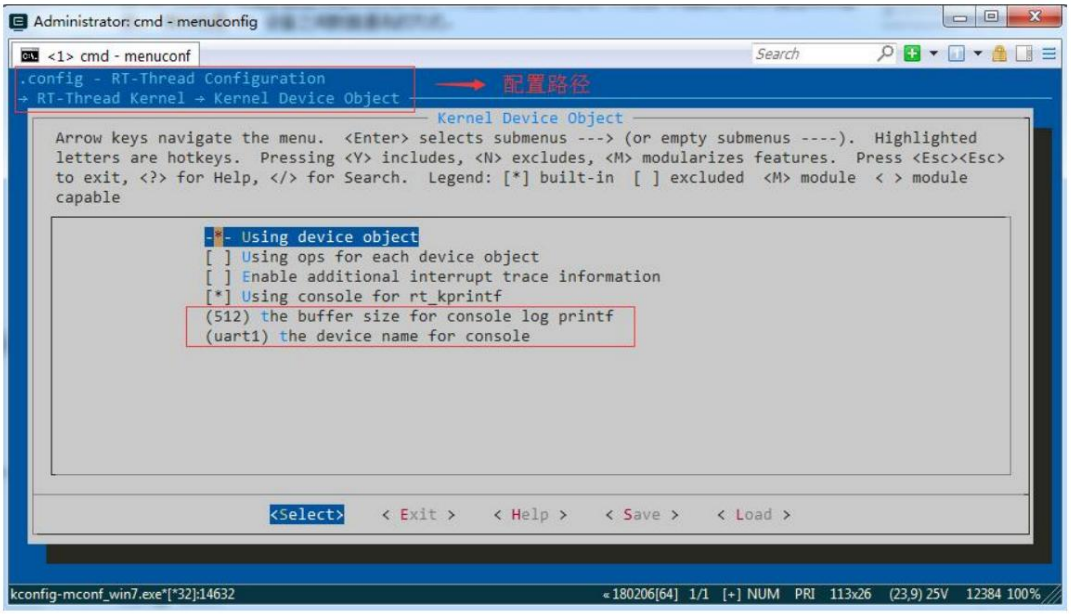

Figure 5: Configuration shell equipment

• Enable AT Client function: RT-Thread Components —> Network —> AT commands –>

Enable AT DEBUG and AT Client support. Currently, AT Client supports multiple connections.

Manually initialize AT Client.

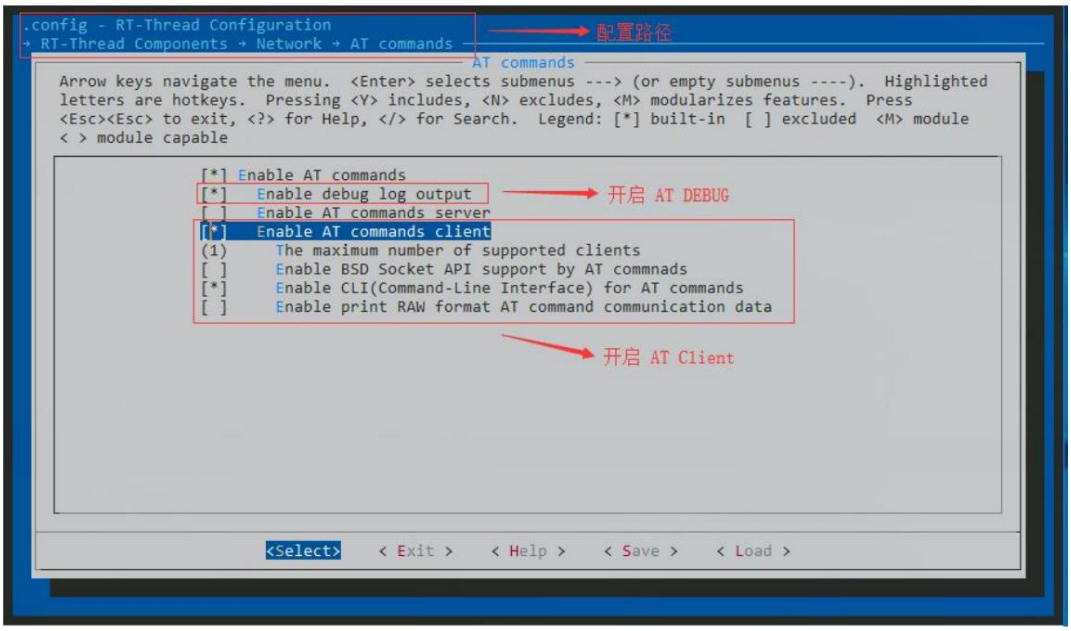

Figure **6:** *AT Client* Configuration

AT Client configuration options are described as follows:

• Enable debug log output: Enable debug log output; • Enable AT commands client: Enable AT client; • The muxinum number of supported clients: The maximum number of clients supported at the same time.

To use a single client connection, set the value to 1.

• Enable BSD Socket API support by AT commands: Enable BSD Socket API support. This example is not used and can be disabled. • Enable CLI (Command-Line Interface) for AT commands: Enable AT command line interface.

Mode.

• Enable print RAW format AT command communication data: Enables sending and receiving data.

Print function.

4. After the configuration is completed, save and exit the configuration options, and enter the command scons –target=mdk5 to generate the keil project;

3.2.2. **AT Client** Porting

The migration of AT Client mainly processes **URC** data (data sent actively by the server) to implement the function of executing corresponding operation functions when obtaining different URC data. If you do not consider **AT Client URC** data processing, you can ignore the migration part.

For URC data, the AT component has provided a complete URC data judgment and processing method. The following is the AT Client URC data processing flow. Developers can define and modify the URC processing of the corresponding device to complete the transplantation of the AT Client.

Note: "OK" and "ERROR" are normal command response result judgment strings and should not be used as URC data.

Set to the URC list.

```
/* URC data related structure definition*/
struct at_urc
{
    const char *cmd_prefix; const                    //URC data prefix
    char *cmd_suffix; void (*func)                   //URC data suffix
    (const char *data, rt_size_t size);              //URC Data Execution Letter
                number
};

static void urc_func(const char *data, rt_size_t size)
{
    /* Customize URC data processing method*/
    LOG_D("URC data : %.*s", size, data);
}

static struct at_urc urc_table[] = {
    {"ready", "\r\n", urc_func},
    {"WIFI CONNECTED", "\r\n", urc_func},
    {"WIFI DISCONNECT", "\r\n", urc_func},
};

int at_client_obj_init(void)
{
    /* Initialize AT Client */
    at_client_init("uart2", 512);

    /* Add multiple URC structures to the URC list. When receiving a structure that matches both the URC prefix and suffix
        Data, execute URC function*/
    at_set_urc_table(urc_table, sizeof(urc_table) / sizeof(urc_table[0]))
        ;

    return RT_EOK;
}
```

3.2.3. AT Client example added

Download AT Client The sample code is added to the opened Keil project, as shown below:

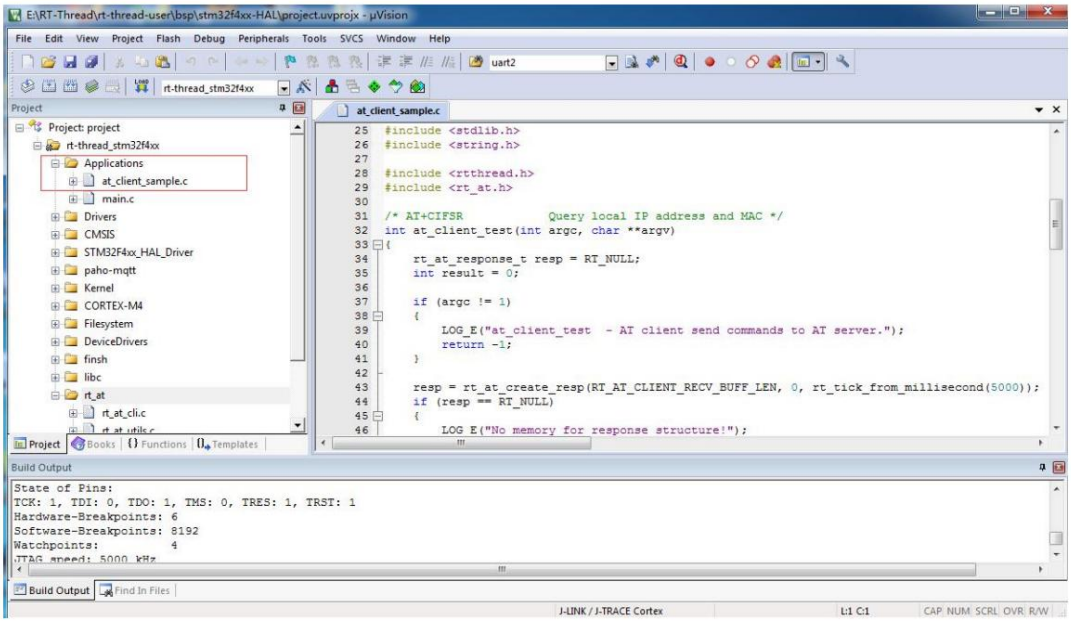

Figure **7:** *AT Client* Example Addition

After adding the example, you can compile and download the program to the development board, then open the serial port tool on the PC. Here we use xshell tool, select the correct serial port (configure the serial port parameters to 115200-8-1-N, no flow control), and then press reset. You can see the RT-Thread system startup log on the terminal connected to serial port 1.

After the system is initialized successfully, execute the at_client_init uart2 command in the shell , where uart2 is turned on. Then you can see the initialization log of AT Client, which shows that AT Client

The function configuration is started successfully, as shown in the following figure:

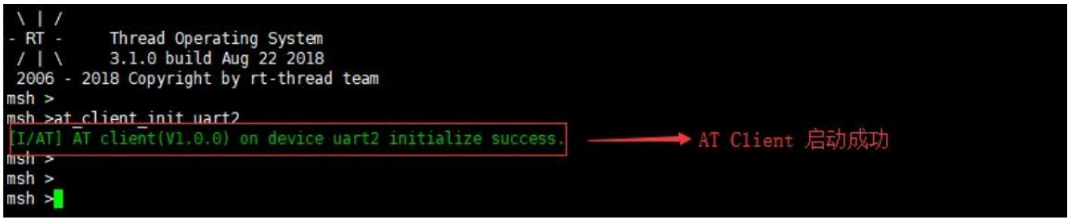

Figure **8:** *AT Client* Successful startup

3.2.4. AT Client Usage

1.AT Client mode

In this mode, the serial port 2 of the STM32F4 Explorer development board acts as the AT Client, and the ESP8266 development board acts as For AT Server, enter the data interaction mode and enter the at_client_test command in the local shell .

The command is used to send AT commands to the server, and receive and parse the server response data, as shown in the following figure:

Figure **9:** *AT Client* Running the Example

2.AT Client CLI Mode

The AT Client CLI function can forward the data entered by the local shell to the AT Server serial port device connected to the device. The data received by the AT Client serial port is displayed in real time on the local shell.

Command to enter AT Client CLI mode to send and receive data. Through AT Client CLI mode, users can

Conveniently complete the connection and debugging with AT Server, greatly improving development efficiency.

The following figure demonstrates the use and exit of the AT Client CLI function:

Figure **10:** *AT Client CLI* Running the Example

3.2.5. AT Client Usage Process

The AT Client sample code used in this article demonstrates the entire usage process of AT Client. The sample code completes The STM32F4 device sends AT commands and receives and parses the response data of the ESP8266 device.

The developer can modify the sample code and run it according to the platform they are using, mainly modifying the command name and parsing method. The following introduces the specific use process of AT Client through the sample code:

```
#include <stdlib.h>
#include <string.h>
#include <rtthread.h>
#include <at.h>

/* AT+CIFSR                    Query local IP address and MAC */
int at_client_test(int argc, char **argv) {

    at_response_t resp = RT_NULL;
    int result = 0;

    if (argc != 1) {

        LOG_E("at_client_test - AT client send commands to AT server."); return -1;

    }

    /* Create a response structure and set the maximum supported response data length to 256 bytes
      (The maximum response length is customized by the user according to actual needs), unlimited number of response data rows, timeout
        5 seconds*/
    resp = at_create_resp(256, 0, rt_tick_from_millisecond(5000)); if (resp == RT_NULL) {


        LOG_E("No memory for response structure!");
        return -2;
    }

    /* Disable echo function */
    at_exec_cmd(resp, "ATE0");

    /* AT Client sends the command to query IP address and receives the response from
    AT Server*/ /* The response data and information are
    stored in the resp structure*/ result = at_exec_cmd(resp,
    "AT+CIFSR"); if (result !=
    RT_EOK) {
        LOG_E("AT client send commands failed or return response error!")
            ;
        goto __exit;
    }

    /* Loop through the received response data by line*/
    {
```

```
    const char *line_buffer = RT_NULL;

    LOG_D("Response buffer");
    for(rt_size_t line_num = 1; line_num <= resp->line_counts; line_num++)

    {
        if((line_buffer = at_resp_get_line(resp, line_num)) !=
            RT_NULL)
        {
            LOG_D("line %d buffer : %s", line_num, line_buffer);
        }
        else
        {
            LOG_E("Parse line buffer error!");
        }
    }

} /* Parse the data according to the custom expression (sscanf parsing method) to get the corresponding data*/
{
    char resp_arg[AT_CMD_MAX_LEN] = { 0 }; /* Custom
    data parsing expression, used to parse the string information between two double quotes*/
    const char *      resp_expr = "%*[^\"]\"%[^\"]\"";

    LOG_D(" Parse arguments"); /* Parse
    the first line of data in the response data and get the corresponding IP address*/
    if (at_resp_parse_line_args(resp, 1, resp_expr, resp_arg) == 1) {

        LOG_D("Station IP: %s", resp_arg);
        memset(resp_arg, 0x00, AT_CMD_MAX_LEN);
    }
    else
    {
        LOG_E("Parse error, current line buff : %s", at_resp_get_line
            (resp, 4));
    }

    /* Parse the second line of data in the response data to get the corresponding MAC address*/
    if (at_resp_parse_line_args(resp, 2, resp_expr, resp_arg) == 1) {

        LOG_D("Station MAC : %s", resp_arg);
    }
    else
    {
        LOG_E("Parse error, current line buff : %s", at_resp_get_line
            (resp, 5)); goto
        __exit;
```

```
            }

} __exit:
        if(resp) {

            /* Delete resp structure */
            at_delete_resp(resp);
        }

        return result;

} /* Set the maximum length of data received at one time supported by the current AT client*/
#define AT_CLIENT_RECV_BUFF_LEN int                    512
at_client_test_init(int argc, char **argv) {

        if (argc != 2) {

            rt_kprintf("at_client_init <dev_name> -- AT client initialize.\
                n");
            return -RT_ERROR;
        }

        at_client_init(argv[1], AT_CLIENT_RECV_BUFF_LEN);

        return RT_EOK;

} #ifdef FINSH_USING_MSH
#include <finsh.h>
/* Add AT Client test command to shell */
MSH_CMD_EXPORT(at_client_test, AT client send cmd and get response); /* Add AT Client
initialization command to shell */
MSH_CMD_EXPORT_ALIAS(at_client_test_init, at_client_init, initialize AT
    client);
#endif
```

- The entire example is a single-client example and you can use the single-client mode API directly.
- The usage process of AT Client is as follows: at_create_resp() creates a response structure—> at_exec_cmd() sends a command and receives a response —> at_resp_get_line()/at_resp_parse_line_args() prints or parses the response data—> at_delete_resp() deletes the response structure.
- The at_exec_cmd() function completes the sending of the incoming AT command and the receiving of the response data. The response data is sent in lines. The data is stored in the structure in the form of rows, which makes it easier to print or parse the data row by row.
- When printing or parsing data, different response data for different commands have different data parsing methods, and you need to customize the data parsing expression. This requires the developer to know the specific response structure of the sent command in advance.

Please refer to the AT command manual for details.

3.3 AT Socket Function (Advanced)

In order to facilitate developers to use AT components for network-related operations and reduce the RT-Thread system's dependence on separate protocol stack network connections, the RT-Thread system launched the AT Socket function based on the AT component and SAL component.

The AT Socket function is based on the AT Client function. Its main function is to use AT commands to connect devices to the network and perform data communication. The network interface provided by the AT Socket function supports the standard BSD Socket API and realizes interface unification through the interface abstraction of the SAL component.

The AT Socket function allows the device to connect to the network directly using the serial port without the need for network connection, which simplifies the hardware and software design of device development and facilitates developer development. In addition, unlike the traditional software network protocol stack, the operation of the AT Socket network function is mainly completed on the AT Server device connected to the serial port. Depending on the different AT Server devices, 5-6 sockets can be supported at the same time, which greatly reduces the MCU resource usage on the AT Client device, improves the MCU work efficiency, and ensures the quality of data communication and the reasonable allocation of hardware resources. The AT Socket function currently occupies the least resource volume of about 20K ROM and 3K RAM (supporting 5 sockets).

The AT Socket function needs to complete the porting and adaptation process for different AT devices. Currently, it has completed the adaptation of multiple devices, including: ESP8266, M26, MC20, etc. The adaptation method is through the at_device software package The following mainly introduces the configuration and use of AT Socket function through ESP8266 device.

3.3.1. AT Socket Configuration

The use of AT Socket function depends on the following components:

- **AT** component: AT Socket function is based on the implementation of AT Client function;
- **SAL** component: SAL component is mainly an abstraction of AT Socket interface, implementing standard BSD Socket API;
- **at_device** software package: AT Socket porting and sample files for different devices, provided in the form of software package out;

The following mainly introduces the whole process of configuring the AT Socket function in env:

1. Open the env tool, enter the rt-thread\bsp\stm32f4xx-HAL directory, and enter menu-config Enter the configuration interface to configure the project.

2. Open AT Socket: RT-Thread Components —> Network —> AT commands —> Enable BSD Socket API support by AT commnads Enable AT Socket support.

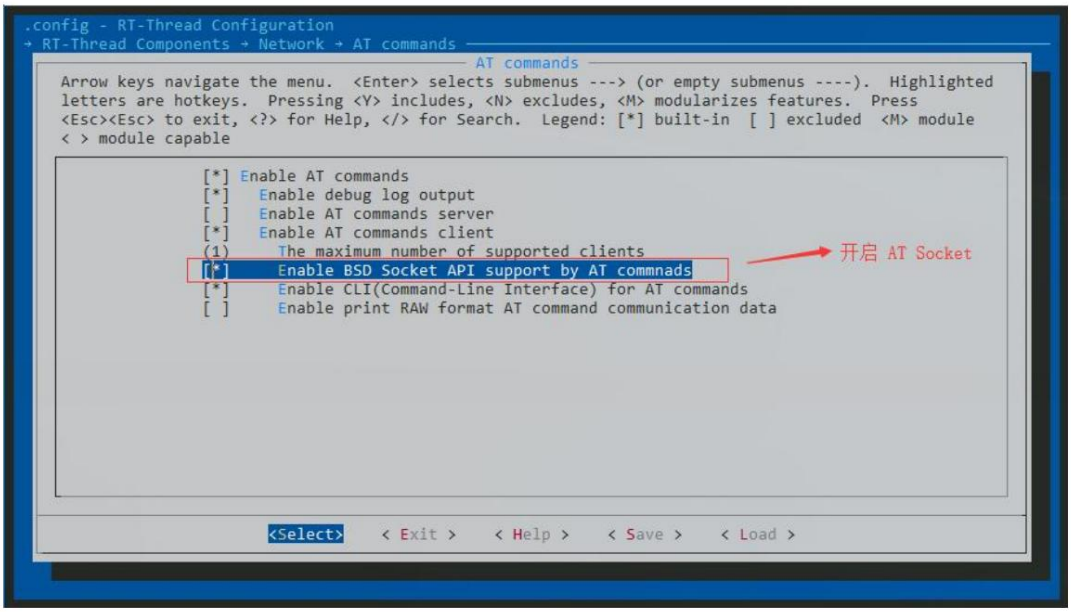

Figure **11:** Configuration Enable *AT Socket*

3. After the AT Socket function is configured successfully, you need to enable the at_device software package. The at_device software package needs to be configured

Set the AT module model and AT Client device name to ensure correct operation: RT-Thread online packages

—> IoT - internet of things —> Enable AT DEVICE package support and configure AT Socket device

modules is ESP8266 device, configure AT Client device name and maximum supported receiving data length, configure wifi

ssid and wifi password are used for device networking, and the version uses the latest laster version.

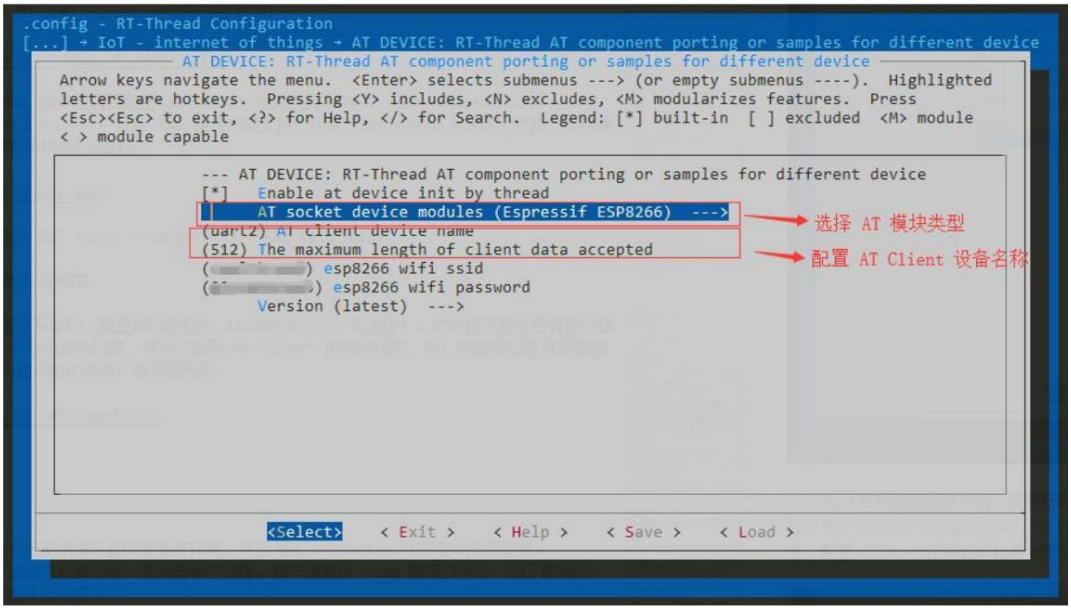

Figure **12:** Configuration *at device* Software Packages

4. Then you need to enable SAL component support. In the SAL component, you need to configure AT Socket function support: RT-

Thread Components —> Network —> Socket abstraction layer —> Enable SAL component function support

Support, enable SAL_USING_POSIX support, and support the use of file system interface functions such as read/write, poll/select, etc.

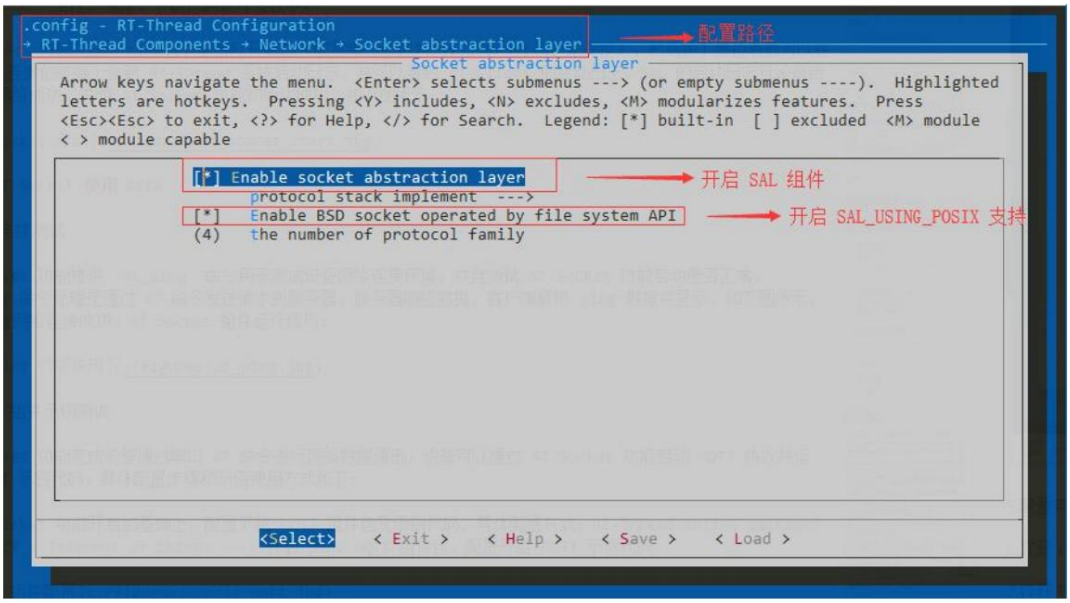

Figure **13:** Open *SAL* support

5. After the configuration is completed, save and exit the configuration options, and enter the command scons –target=mdk5 to generate the keil project.

6. Open the Keil project, compile and download the code to the development board.

7. Open the serial port tool xshell on the PC and configure the serial port (configure the serial port parameters to 115200-8-1-N, no stream

Then press reset to see the RT-Thread system startup log on the terminal connected to serial port 1, and you can

If you see the startup log of AT Client and SAL and the device automatically connects to the network successfully, it means that AT Socket

The function is initialized successfully, as shown in the figure below.

Figure **14:** *AT Socket* start up

3.3.2. AT Socket Usage

1. Network connection test

The AT Socket function provides the at_ping command to test the device network connection environment and test the AT Socket function.

Can it start normally? The principle of at_ping command is to send a request to the server through AT command, and the server responds with data.

The client parses the ping data and displays it. As shown in the following figure, the command device network connection is successful and the AT Socket component runs successfully:

```
msh />
msh />
msh />
msh />at_ping baidu.com
32 bytes from baidu.com icmp_seq=1 time=62 ms
32 bytes from baidu.com icmp_seq=2 time=36 ms
32 bytes from baidu.com icmp_seq=3 time=59 ms
32 bytes from baidu.com icmp_seq=4 time=33 ms
msh />
msh />
msh />
```

Figure **15:** *at_ping* Function Usage

2.MQTT component example test

AT Socket function completes the network data communication of the device through the serial port AT command.

The socket function starts the MQTT protocol and runs the MQTT sample code. The specific configuration steps and sample usage are as follows:

- On the basis of enabling the AT Socket function, configure and download the MQTT component package and sample code. The specific configuration method is: RT-Thread online packages —> IOT - internet of things —> Enable the paho MQTT component package and enable the MQTT sample code.

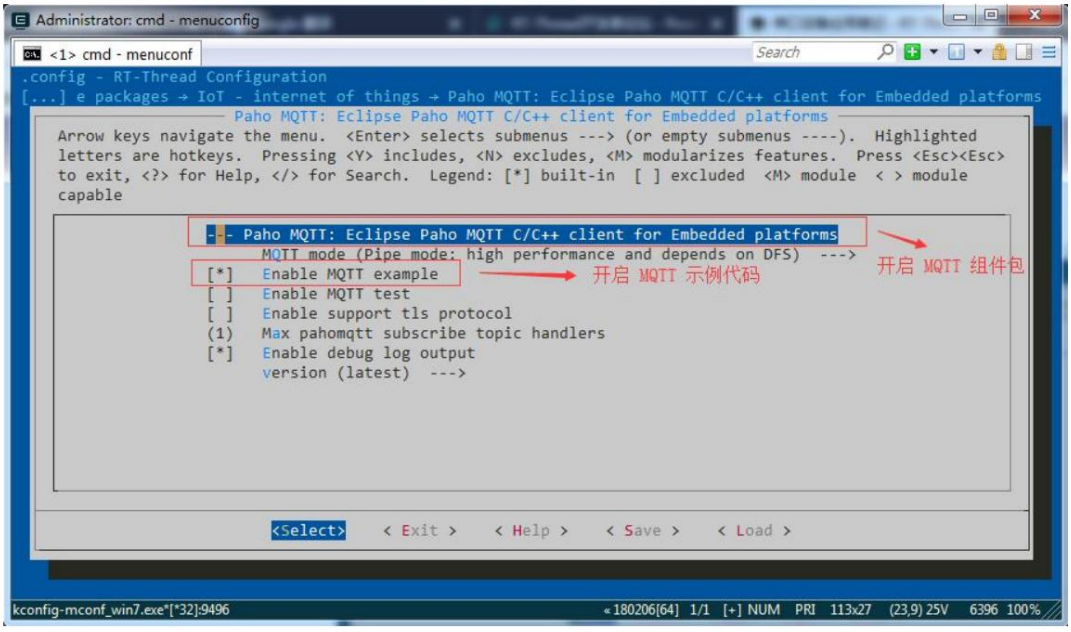

Figure **16:** *MQTT* Component Configuration

- After configuration is completed, save and exit the configuration options, regenerate the project with scons, and compile and download the code to the development board.
- Open the serial port tool, the system starts successfully, enter the mq_start command to start the MQTT protocol, and after the startup is complete, enter the mq_pub mqtt_test_data command to send data to a fixed MQTT Topic.

  The MQTT server will immediately send the same data to the Topic, and the MQTT example test is completed, as shown in the following figure:

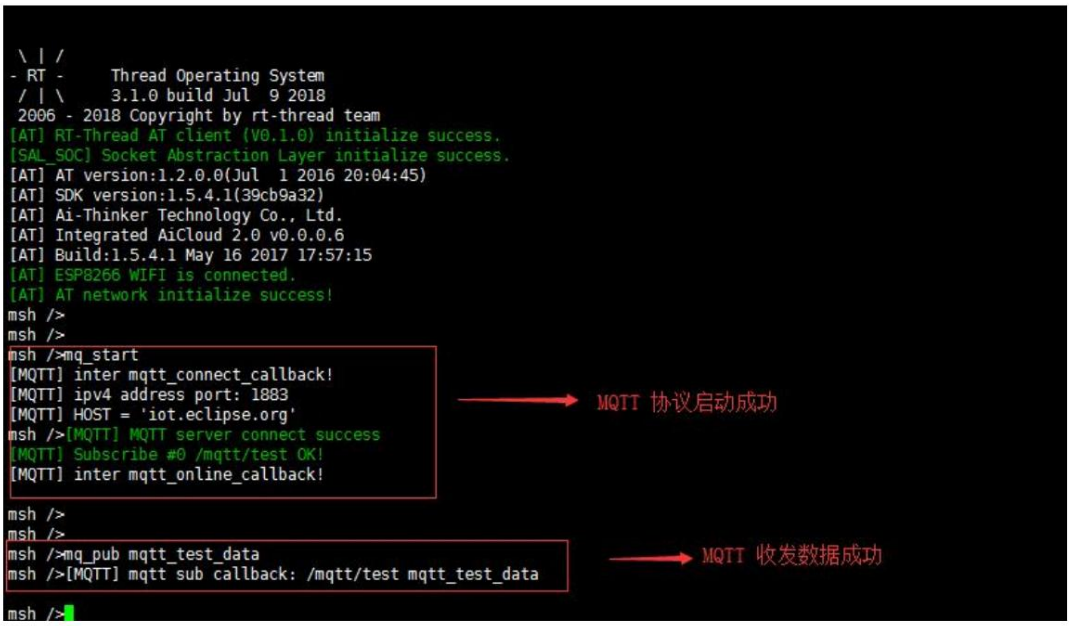

Figure **17:** *MQTT* Example Run

The above shows an example of using the AT Socket function to run the MQTT network on the STM32F4 device without connecting to the network, and implements the AT Socket network data sending and receiving function. Currently, the AT Socket function only supports the device as a network client to connect to the server, which is also in line with the characteristics of embedded devices that are mostly used as client devices. AT Socket currently supports a variety of network-related software packages and functions, as shown below:

• MQTT package • webclient package • mbedtls package • onenet package • NTP time query function • iperf network test function • at_ping network test function

For more information about AT components and AT Socket functions, please refer to the **AT** component introduction chapter in the RT-Thread Programming Manual .

4 Frequently Asked Questions

1. When the real-time printing function of AT command sent and received data is enabled, what should I do if the log on the shell displays errors?

   • Increase the baud rate of the shell corresponding serial port device to 921600, increase the serial port printing speed, and prevent the printing of large amounts of data. Printing display error.

2. When the AT Socket function is enabled, the compiler prompts " The AT socket device is not selected, please What about selecting it through the env menuconfig ?

• This error occurs because after the AT Socket function is enabled, the at device software package is enabled by default to configure the corresponding device.
Device model, enter the at device software package, configure the device as ESP8266 device, configure WIFI information, and restart scons generates the project, compiles and downloads it.

3. What should I do if the ESP8266 device automatically disconnects from the Wi-Fi after a while after connecting to the network?

• This error is usually caused by a power supply problem on the esp8266 device. You can use a multimeter to check the current voltage of the device.
If there is insufficient power supply problem, you can add additional power supply to the esp8266 vin interface.

5 References

5.1 All relevant **APIs** in this article

5.1.1. **API** List

API	Location
at_client_init()	at_client.c
at_create_resp()	at_client.c
at_delete_resp()	at_client.c
at_exec_cmd()	at_client.c
at_resp_get_line()	at_client.c
at_resp_parse_line_args()	at_client.c
at_set_urc_table()	at_client.c
at_client_port_init()	at_client.c

5.1.2. Detailed explanation of core **API**

5.1.3. **AT Client** Initialization

```
int at_client_init(const char *dev_name, rt_size_t recv_bufsz);
```

AT Client initialization function, which belongs to the application layer function, needs to be used when using the AT Client function or using the AT CLI

The **at_client_init function** completes the initialization of the AT Client device, the threads used by the AT Client, Initialize resources such as semaphores and mutexes.

parameter	describe
dev_name	AT client uses device name
recv_bufsz	AT client supports the maximum receiving data length
return	describe
>=0	success
-1	fail
-5	Failed, out of memory

5.1.4. Create a response structure

```
at_response_t at_create_resp(rt_size_t buf_size);
```

This function is used to create a structure object with a specified maximum return data length, which is used to receive and parse the command response later.

Response data.

parameter	describe
buf_size	Maximum supported return data length
return	describe
!= NULL	Success, returns a pointer to the response structure
= NULL	fail

5.1.5. Deleting the response structure

```
void at_delete_resp(at_response_t resp);
```

This function is used to delete the created response structure object, and usually appears in pairs with the **at_create_resp** creation function.

parameter	describe
resp	Pointer to the response structure to be deleted
return	describe
none	none

5.1.6. Sending commands and receiving responses

```
rt_err_t at_exec_cmd(at_response_t resp, rt_size_t resp_line, const char
    *cmd_expr, ...);
```

This function is used by AT Client to send commands to AT Server and wait for responses. It is an important function in AT Client.

To command the sending function.

parameter	describe
resp	Response structure pointer
resp_line	The number of rows of data to be returned. 0 means all rows are returned. >0 means return a fixed number of rows
cmd_expr	Send command expression
…	Send command data
return	describe
>=0	success
-1	fail
-2	Failed, timeout in receiving response

5.1.7. Parse the response data of the specified line number

```
int at_resp_parse_line_args(at_response_t resp, rt_size_t resp_line,
    const char *resp_expr, ...);
```

This function is used to obtain a row of data with a specified row number in the AT Server response data and parse the parameters in the row of data.

number.

parameter	describe
resp	Response structure pointer
resp_line	The row number of the data to be parsed
resp_expr	Custom parameter parsing expressions
…	Parsed parameters
return	describe
>0	Success, returns the number of parameters parsed successfully

parameter	describe
=0	Failed, no matching number of parameters to parse expression
-1	Failed, parameter parsing error

5.1.8. URC data list initialization

```
void at_set_urc_table(const struct at_urc *table, rt_size_t size);
```

This function is used to initialize the URC data list customized by the developer and is mainly used in the AT Client transplantation function.

parameter	describe
table	URC data structure array pointer
size	Number of URC data
return	describe
none	none