

---

# RT-THREAD WEBNET User Manual

---

**RT-THREAD** Documentation Center

Copyright ©2019 Shanghai Ruiside Electronic Technology Co., Ltd.



**WWW.RT-THREAD.ORG**

**Wednesday 7th November, 2018**

## Versions and Revisions

Date	Version	Author	Note
2013-05-05	v1.0.0	bernard	initial version
2018-08-06	v2.0.0	chenyong	new version update

Table of contents

Versions and Revisions	i
Table of contents	ii
1 Software Package Introduction	1
1.1 Package directory structure.	1
1.2 Software package features.	1
1.3 Software package performance testing.	3
1.4 Software package resource usage.	4
2 Sample Program	5
2.1 Preparation	5
2.1.1 Obtaining the software package.	5
2.1.2 Page file preparation.	6
2.2 Startup routine	6
2.2.1 AUTH Basic authentication routine.	8
2.2.2 CGI event handling routines.	8
2.2.3 ASP variable substitution routine.	9
2.2.4 SSI file nesting routines.	10
2.2.5 INDEX directory display routine.	10
2.2.6 ALIAS Alias access routine.	11
2.2.7 Upload file upload routine.	11
3 Working Principle	13
4 Usage Guide	15
4.1 Preparation	15
4.1.1 ENV configuration instructions.	15

4.2 Instructions for using the file system. . . . .	16
4.3 Workflow . . . . .	16
4.4 Usage. . . . .	20
4.5 Frequently Asked Questions . . . . .	26
<b>5 API Description</b>	<b>28</b>
5.1 Initialization function. . . . .	28
5.2 Set the listening socket port. . . . .	28
5.3 Get the listening socket port. . . . .	29
5.4 Set up the server root directory. . . . .	29
5.5 Get the server root directory. . . . .	29
5.6 Get the type of request link. . . . .	30
5.7 Add ASP variable processing method. . . . .	30
5.8 Add CGI event handling method. . . . .	30
5.9 Set the CGI event root directory. . . . .	31
5.10 Set basic authentication information. . . . .	31
5.11 Set up directory aliases. . . . .	31
5.12 Send HTTP request header. . . . .	32
5.13 Send HTTP response data. . . . .	32
5.14 Sending HTTP fixed-format response data . . . . .	33
5.15 Get the name of the uploaded file. . . . .	33
5.16 Get the type of uploaded file. . . . .	33
5.17 Get uploaded file parameters. . . . .	34
5.18 Get the file descriptor of the uploaded file. . . . .	34

# Chapter 1

## Software Package Introduction

The WebNet software package is independently developed by RT-Thread and is implemented as a Web server based on the HTTP protocol. It not only has the basic functions of HTTP Client communication, but also supports multiple module function extensions to meet the functional requirements of developers for embedded device servers.

### 1.1 Software package directory structure

The directory structure of the WebClient package is as follows:

name	illustrate
docs	Documentation
inc	Header file directory
src	Source file directory
module	Function module file directory
samples	Sample File Directory
LICENSE	License File
README.md	Software package instructions
SConscript	RT-Thread default build script

### 1.2 Software Package Features

The WebNet software package provides a wide variety of functional support. Most of the functions are provided in the form of modules. Each functional module provides a separate control mode and different functional modules do not interfere with each other. The modules and the main program cooperate to form a set of complete functions, high performance, high concurrency and A highly customizable web server framework.

- WebNet software package features:
- Support **HTTP 1.0/1.1**

The WebNet software package is used to run network server functions in embedded devices. Its functions are implemented based on the HTTP protocol and support HTTP 1.0 and HTTP 1.1 protocols to improve web server compatibility.

- Support **CGI** function

CGI (Common Gateway Interface) function is an external program specification when the server is running, which can be used to expand the functions of the server. CGI function can complete the interaction between the browser and the server. The CGI function in the WebNet software package allows users to customize and register **CGI** execution functions. The server determines the CGI type requested by the browser, executes the corresponding function, and gives corresponding feedback information.

- Support **ASP** variable replacement function

ASP (Active Server Pages) function, which implements the function of variable replacement in web pages, that is, when the `<% %>` tag appears in the page code accessed by the browser (supporting page files with the .asp suffix), it will be automatically replaced with a custom registered **ASP** execution function and give corresponding feedback information. This function can be used to easily modify the implementation method of each variable execution function independently, complete the modification of different parts of the entire page, and facilitate the division of labor and cooperation between front-end and back-end development.

- Support **AUTH** basic authentication function

In the process of using HTTP protocol for communication, HTTP protocol defines Basic Authentication process to allow HTTP server to authenticate the user identity of WEB browser. Basic authentication function provides a simple user authentication method for the server. Its authentication method is fast and accurate, and is suitable for systems and devices that have certain requirements for server security performance. The basic authentication function in WebNet software package is designed to be linked to the directory in the URL, and permissions can be divided according to the directory when used.

- Support **INDEX** directory file display function

After the WebNet server is started, enter the corresponding device IP address and directory name in the browser to access the directory and list the  
The information of all files in the directory is similar to a simple file server and can be used to query and download server files.

- Support **ALIAS** alias access function

ALIAS alias access function can set multiple aliases for folders, which can be used to simplify operations on long paths and facilitate user access to resources.

- Support **SSI** file embedding function

The SSI (Server Side Include) function is generally used to execute server programs on a page or insert text content into a web page. The SSI function in WebNet currently only supports inserting text content into a web page. When there is a tag in the page code (supporting page files with .html and .htm suffixes), it will be automatically parsed into the information in the corresponding file. Through the SSI file embedding function, the entire page processing can be modularized, so that the page can be modified by modifying the specified file.

- Support file upload function

Supports browsers to upload files to WebNet servers. You can customize file processing functions to store uploaded files in files.  
system or directly write to the target memory. This function can be used to achieve:

1. Upload resource files to the file system, such as pictures and web pages;

2. Upload the memory image to burn the memory in the system, such as SPI FLASH;
3. Upload the configuration file to the system to update the system settings;
4. Upload the firmware and write it directly to FLASH to realize the firmware update function.

- Support pre-compression function

The pre-compression function is used to compress web resource files in advance (supports compressed files with the .gz suffix), reducing the demand for storage space and reducing the size of the files. Shorten file system reading time, reduce network traffic, and increase web page loading speed. Generally speaking, after using the pre-compression function, the WebNet server The storage space occupied is only 30%~40% of the original.

- Support cache function

The WebNet cache mechanism can determine whether the browser requested file has been modified, and thus decide whether to send the complete file content to the browser. WebNet package buffering mechanism is divided into three levels as follows:

1. Level 0: Disable the cache function, that is, the browser will read the complete file content from WebNet every time;
2. Level 1: Enable the cache function. WebNet reads the last modified time of the requested file. If it is the same as the local file, it returns the last modified time.  
Return 304 to inform the browser that the file has not been updated and will not be sent;
3. Level 2: Enable the cache function. Based on the original judgment of modification time, add support for cache file validity time and operation validity time  
The browser can access the file again.

- Support breakpoint resume function

WebNet server supports breakpoint resume function, that is, if the client fails to download the file, it only needs to provide The offset of the previous file download on the WebNet server. The server will send the file content to the client from the specified file offset, and resume the download from the breakpoint. This function can ensure that files are uploaded quickly and accurately, and improve server operation efficiency.

### 1.3 Software Package Performance Test

Test environment: AM1808, main frequency 400M, DDR2 memory

test environment	Page file on SD card, DM9000AEP external network card
Static page request	142 pages/s
CGI page request	429 pages/s

test environment	Page file on SD card, EMAC external network card
Static page request	190 pages/s
CGI page request	567 pages/s

## 1.4 Software Package Resource Usage

ROM: 16 Kbytes

RAM: 1.5 Kbytes + 1.1 Kbytes x number of connections

NOTE: The above data is for reference only. The example platform is Cortex-M3. Due to different instruction lengths and frameworks, the software package resource usage may vary.

When WebNet processes pages, some resources are consumed in the file system and modules.



## chapter 2

### Sample Program

The WebNet package provides a comprehensive sample page to demonstrate the various features of the package, including: AUTH, CGI, ASP, SSI, INDEX, ALIAS, Upload and other functions. This chapter mainly introduces the usage of each functional module example in the WebNet software package.

Sample Files	
Sample program path	illustrate
<code>samples/wn_sample.c</code>	Comprehensive sample code
<code>samples/wn_sample_upload.c</code>	Upload file sample code
<code>samples/index.html</code>	Comprehensive sample page
<code>samples/index.shtml</code>	SSI Functionality Example Page
<code>samples/version.asp</code>	ASP Function Sample Page

## 2.1 Preparation

### 2.1.1 Obtaining the Software Package

- menuconfig configuration to obtain software packages and sample code
- Open the ENV tool provided by RT-Thread and use **menuconfig** to configure the software package.
- Enable the WebNet package and configure the Enable **webnet** samples configuration as shown below:

```
RT-Thread online packages
IoT - internet of things --->
  [*] WebNet: A HTTP Server for RT-Thread
      (80) Server listen port                                ## Server listening socket port number
      (16) Maximum number of server connections ## The maximum number of connections supported by the server
      (/webnet) Server root directory ## Server root directory
          Select supported modules --->                                ## Functional modules enabled by default
              [ ] LOG: Enable output log support
              -* AUTH: Enable basic HTTP authentication support
```

```

-* CGI: Enanle Common Gateway Interface support
-* ASP: Enable Active Server Pages support
-* SSI: Enanle Server Side Includes support
-* INDEX: Enable list all the files in the directory support
-* ALIAS: Enanle alias support [
    ] DAV: Enanle Web-based Distributed Authoring and Versioning
    support
-* UPLOAD: Enable upload file support [
    ] GZIP: Enable compressed file support by GZIP
(0) CACHE: Configure cache level
[*] Enable webnet samples                                ## Start the test routine
Version (latest) --->

```

- Use the `pkgs --update` command to download the package
- Compile and download

### 2.1.2 Page file preparation

In the WebNet package example, you need to obtain local static pages, which requires the support of the file system (FAT file system, ROMFS file system). system, etc., only the device virtual file system that supports RT-Thread is needed).

Static pages need to be uploaded to the server root directory in the file system (the root directory used in the example is `/webnet`). After the device mounts the file system successfully, you need to perform the following operations in sequence:

1. Use the `mkdir webnet` command to create the WebNet package root directory `/webnet`, and use the `cd webnet` command to enter the directory;
2. Use the `mkdir admin` and `mkdir upload` commands to create `/webnet/admin` and `/webnet/upload` for AUTH Functional and Upload functional testing;
3. Upload the three files **index.html**, **index.shtml**, and **version.asp** in the WebNet software package/sample directory to the device `/webnet` directory (WebNet root directory) in sequence.  
(You can use the TFTP tool to upload files. For specific operation methods, refer to the TFTP instructions)

After successfully creating the directory and uploading the file, you can start the routine to test the WebNet software functions.

## 2.2 Startup routine

The parameters and environment configuration of this routine are as follows:

- Listening port number: 80
- Root directory address: `/webnet`
- File system: FAT file system

After the device is started and connected to the network successfully, enter the `webnet_test` command in the Shell command line to start the WebNet server.

The Shell command line displays the following log information, indicating that the WebNet server has been initialized successfully:

```
msh />webnet_test [/l/wn]
RT-Thread webnet package (V2.0.0) initialize success.
```

Then use the `ifconfig` command in the Shell command line to obtain the IP address of this device, which is **192.168.12.29**.

```
msh />ifconfig
network interface: w0 (Default)
MTU: 1500
MAC: 44 32 c4 75 e0 59
FLAGS: UP LINK_UP ETHARP BROADCAST IGMP
ip address: 192.168.12.29 gw address:
192.168.10.1
net mask : 255.255.0.0
DNS server #0: 192.168.10.1
DNS server #1: 223.5.5.5
```

Then enter the device IP address in the browser (here we use Google Chrome), and the `/index.html` file in the root directory of the device will be accessed by default. As shown in the figure below, the page file is displayed normally:



Figure 2.1: Routine main page

This page shows the basic functions of the WebNet software package and provides corresponding demonstration examples according to different functions.

The following routines are introduced in sequence:

- AUTH basic authentication routine •

CGI event handling routine • ASP

variable substitution routine • SSI

file nesting routine • INDEX

directory display routine • ALIAS alias

access routine • Upload file upload

routine

### 2.2.1 AUTH Basic Authentication Routine

In the homepage of the example program (/index.html), click the basic authentication function test under the AUTH Test module: the username and password are admin:admin . The basic authentication dialog box pops up, enter the username **admin** and the password **admin**. After successfully entering the username and password, you will enter the /admin directory under the root directory, as shown in the following figure:

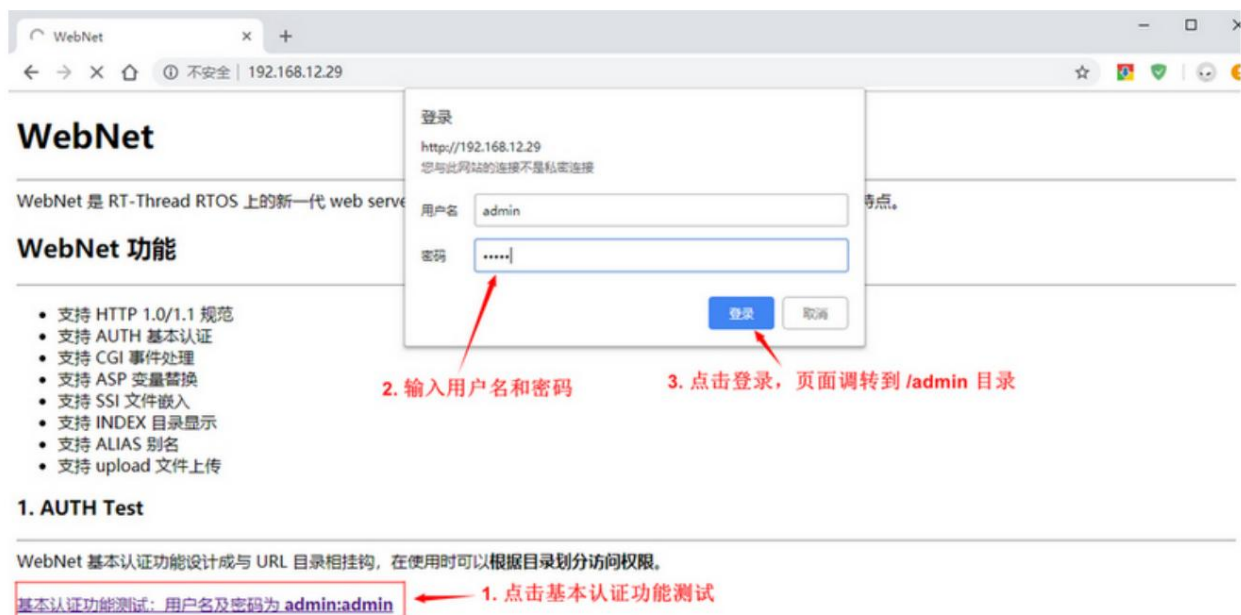


Figure 2.2: Basic authentication routines

### 2.2.2 CGI event handling routines

This example provides two CGI examples: the **hello world** example and the **calc** example, which are used to demonstrate the basic functions of CGI event processing.

- Hello world example

The hello world example demonstrates the function of displaying text content on a page. Click the > [hello world](#) button under the CGI Test module on the example homepage to jump to a new page to display log information. The new page displays hello world, indicating that the CGI event is successfully processed. Then click the [Go back](#) to root button on the new page to return to the example main page, as shown in the following figure:

Figure 2.3: *hello world*

Routines

- **calc routines**

The calc routine uses CGI functions to display a simple addition calculator function on the page. Click on the CGI Test module on the routine homepage.

> **Calc** button, it will jump to a new page, enter two numbers and click the **Calculate** button, the page will show the result of adding the two numbers.

Click the **Go back to root** button to return to the main page of the routine, as shown below:

Figure 2.4: *calc*

Routines

### 2.2.3 ASP variable replacement routine

The ASP example demonstrates the page variable replacement function. Click **ASP Function Test** under the **ASP Test** module on the example homepage: [Access version](#)

.asp file button will jump to the version.asp page in the root directory, which displays the version number of the current RT-Thread system.

Click the **Go back to root** button on the new page to return to the main page of the routine, as shown below:



Figure 2.5: ASP Routines

## 2.2.4 SSI file nesting routine

The SSI example demonstrates the ability to nest a page within another page. Click [SSI Function Test](#) in the SSI Test module on the example homepage.

Try: Access the [/version.shtml](#) page button, it will jump to the [index.shtml](#) page in the root directory, which displays the main page nested in the page [index.html](#) content. Click the [Go back to root](#) button on the new page to return to the main page of the example.



Figure 2.6: SSI Routines

## 2.2.5 INDEX directory display routine

The INDEX routine demonstrates the page file listing functionality.

First, you need to upload a file to the [/admin](#) directory of the root directory. In the example, the [admin.txt](#) file has been uploaded to this directory.

Then click INDEX function test in the INDEX Test module on the routine homepage : Access the [/admin directory](#) button and you will be redirected to the root directory. [/admin](#) directory, and lists all file names and file lengths in the directory, as shown in the following figure:



Figure 2.7: INDEX Routines

2.2.6 ALIAS Alias Access Routines

The ALIAS routine demonstrates the use of a directory alias to access the directory. In this routine code, the /test directory has been set to an alias of /admin, click ALIAS Function Test in the ALIAS Test module on the routine homepage : Accessing the /test directory will jump to the /admin directory button, We actually access the /test directory, but will jump to the /admin directory and list all the file information in the directory, as shown in the following figure:



Figure 2.8: ALIAS Routines

2.2.7 Upload file upload routine

The Upload routine implements the function of uploading files to a fixed directory on the WebNet server. On the routine homepage, under the Upload File Test module, click Click the Select File button, select the file to be uploaded (this example uses the upload.txt file), and click Upload to upload the file to the root.

The /upload directory under the directory is as shown below:



Figure 2.9: upload files

After the file is uploaded successfully, return to the routine homepage and click the Browse Upload File Directory button to access the /upload file and view the file you just uploaded.

The file information just uploaded.



Figure 2.10: View Uploaded Files



## Chapter 3

### working principle

The WebNet software package is mainly used to implement HTTP servers on embedded devices. The main working principle of the software package is based on the HTTP protocol.

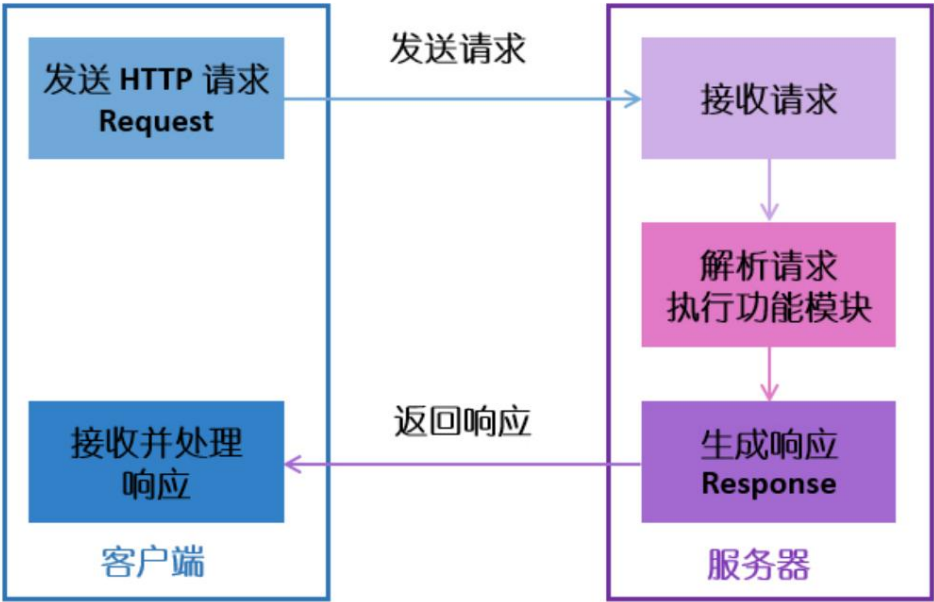


Figure 3.1: WebNet How the software package works

The HTTP protocol defines how the client requests data from the server and how the server transmits data to the client. The HTTP protocol adopts a request/response model. The client sends a request message to the server, which contains the request method, URL, protocol version, request header, and request data. The server parses the request header information, executes the corresponding function module, and sends response data to the client. The content of the response data includes the protocol version, success or error code, server information, response header, and response data.

In the actual use of the HTTP protocol, the following process is generally followed:

1. Client connects to server

Usually, a TCP connection is established through a TCP three-way handshake. The default connection port number in WebNet is 80.

## RT-Thread WebNet User Manual

### 2. The client sends an HTTP request

Through the TCP socket, the client sends a request message to WebNet. A request message consists of four parts: request line, request header, blank line and request data.

### 3. The server receives the request and parses the data information

After receiving the client's request, the server starts parsing the request information, locates the resource file requested on the server, and then sends the response data to the client, which is read by the client. A response data consists of four parts: status line, response header, blank line and response data.

### 4. The client and server are disconnected

If the connection mode between the client and the server is normal mode, the server actively closes the TCP connection, and the client passively closes the connection and releases the TCP connection. If the connection mode is keepalive mode, the connection is maintained for a period of time, during which data can continue to be received.

## Chapter 4

### user's guidance

This section mainly introduces the basic usage process of the WebNet software package, and explains the structures and important APIs that are often involved in the usage process.

Line brief description.

## 4.1 Preparation

### 4.1.1 ENV Configuration Instructions

First, you need to download the WebNet software package and add it to the project. In the BSP directory, use the menuconfig command to open the ENV configuration interface and select the WebNet software package in [RT-Thread online packages](#) › [IoT - internet of things](#) . The specific path is as follows:

```
RT-Thread online packages
  IoT - internet of things --->
    [*] WebNet: A HTTP Server for RT-Thread
      (80) Server listen port
      (16) Maximum number of server connections (/webnet)
      Server root directory
        Select supported modules --->
          [*] LOG: Enable output log support
          [*] AUTH: Enable basic HTTP authentication support
          [*] CGI: Enable Common Gateway Interface support
          [*] ASP: Enable Active Server Pages support
          [*] SSI: Enable Server Side Includes support
          [*] INDEX: Enable list all the files in the directory support
          [*] ALIAS: Enable alias support
          [*] DAV: Enable Web-based Distributed Authoring and Versioning
                support
          [*] UPLOAD: Enable upload file support
          [*] GZIP: Enable compressed file support by GZIP
          (2) CACHE: Configure cache level
              (1800) Cache-Control time in seconds
        [*] Enable webnet samples
      Version (latest) --->
```

**Server listen** port: configure the server listening port number;

**Maximum number of server** connections: Configure the maximum number of connections supported by the server;

**Server root** directory: configure the server root directory path;

**Select supported** modules: Select the function modules supported by the server. By default, all function modules are supported.

- **LOG**: configure to enable WebNet software package log function support; • **AUTH**: configure to enable Basic Authentication function support; • **CGI**: configure to enable CGI event processing function support; • **ASP**: configure to enable ASP variable replacement function support; • **SSI**: configure to enable file embedding function support; • **INDEX**: configure to enable display of current page file list function support; • **ALIAS**: configure to enable alias access function support; • **DAV**: configure to enable Web-based distributed authoring and version control support; • **Upload**: configure to enable file upload function support; • **GZIP**: configure to enable server compressed file access support; • **CHCHE**: configure to enable cache function support (optional levels 0, 1, 2); • **Cache-Control** time: configure cache function validity period, the cache function level needs to be 2;

**Enable webnet samples** : configure and add server sample files;

**Version**: Configure the software package version.

Here we enable all functions of the WebNet software package by default, and select the **latest** version. Select the appropriate configuration items and

After the version is updated, use the `pkgs --update` command to download the package and update the user configuration.

## 4.2 File System Usage Instructions

WebNet software package requires file system support (FAT file system, ROMFS file system, etc., support RT-Thread

Device virtual file system), used for storage of static pages accessed in the WebNet software package, storage of uploaded and downloaded files, and other functions.

## 4.3 Workflow

Before using the WebNet software package, you can first understand the basic workflow of the WebNet software package, as shown below:

- Initialize the WebNet software package and start listening for connection requests;
- Receive connection request and create connection session;
- Receive HTTP request data and parse request information;
- Determine the requested functional module and execute the corresponding function;
- Return HTTP response data;

- Close the connection session.

The WebNet software package can access the pages saved on the device side through the browser, and upload and download files on the device side. The WebNet software package uses the request and response process based on the HTTP protocol. It analyzes the type and parameters of the HTTP request to make judgments, execute the corresponding functional modules, and correctly return the HTTP response data to complete the entire process.

The following uses the browser accessing the main page in the root directory of the WebNet server as an example to introduce the basic workflow of WebNet:

1. Initialize the **WebNet** software package

```
int webnet_init(void);
```

The WebNet software package needs to be initialized before use. The webnet thread is created in the initialization function. This thread is used to initialize the enabled functional modules, complete the creation of the server listening socket, and use the listening socket to wait for client connections and data interaction. The following figure shows the main operations of the thread function:

```
/* WebNet server thread processing function*/
static void webnet_thread(void *parameter) {

    ....
    /* Create a listening socket */
    listenfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    ...
    /* Initialize the enabled function
    module*/ webnet_module_handle_event(RT_NULL, WEBNET_EVENT_INIT);

    /* Waiting for connection request, waiting for receiving data*/
    for (;;) {

        ...
        sock_fd = select(maxfdp1, &tempfds, &tempwrtfds, 0, 0); if (sock_fd == 0)

            continue;

        if (FD_ISSET(listenfd, &tempfds)) {

            /* Handle new connection request */
        }

        /* Process received or sent data */
        webnet_sessions_handle_fds(&tempfds, &writeset);
    }
    ...
}
```

2. Receive connection request and create connection session

After the WebNet initialization thread is successfully created, when a new connection request is generated, a connection session structure will be created. The structure definition is as follows:

```

struct webnet_session {

    struct webnet_session *next;                                //Session structure list

    int socket;
    struct sockaddr_in cliaddr; struct
    webnet_request* request;                                // Session request related information

    rt_uint16_t buffer_length; rt_uint16_t
    buffer_offset; rt_uint8_t
    buffer[WEBNET_SESSION_BUFSZ];                            //Session buffer data, used to receive the request number

    rt_uint32_t session_phase; rt_uint32_t                    // Current session status
    session_event_mask; const struct
    webnet_session_ops* session_ops; // Session event execution functions read, write,
    close, etc.
    rt_uint32_t user_data;
};

```

The `webnet_session` structure is used to store some information about the currently established connection session, and can be used to connect to the entire process of the current session. Before performing HTTP data interaction, you need to create and initialize this structure. The creation of a new session has been completed in the **webnet** thread, as shown below:

```

struct webnet_session* accept_session; accept_session
= webnet_session_create(listenfd); if (accept_session == RT_NULL) {

    /* Creation failed, close the connection */
}

```

3. Receive **HTTP** request data, parse request information After successfully creating the session structure, when the connection session receives the HTTP request, it will process the received HTTP request and parse the request type, header information and additional parameters in sequence. The general process of parsing request information is as follows:

```

/* This function is used to parse the request mode, header information and parameters
of the current session connection*/ static void _webnet_session_handle_read(struct webnet_session*
session) {
    /* Read the current session HTTP connection session data*/
    ....

    if (session->buffer_offset) {

        /* Parse HTTP request mode (GET, POST, etc.) */ if
        (session->session_phase == WEB_PHASE_METHOD) {

            webnet_request_parse_method(...);
        }
    }
}

```

```

/* Parse HTTP request header information*/
if (session->session_phase == WEB_PHASE_HEADER) {

    webnet_request_parse_header(...);
}

/* Parse the request parameters in the HTTP URL*/
if (session->session_phase == WEB_PHASE_QUERY) {

    webnet_request_parse_post(...);
}
}
}

```

#### 4. Determine the requested function module and execute the corresponding function

By parsing the request mode and header information, the basic information of the current connection session request is obtained, and then the type of functional module used is determined and the corresponding module is executed. The general process of determination is as follows:

```

/* This function is used in WebNet to judge and execute the current session request, such as logging function and CGI event processing function
wait*/
static int _webnet_module_system_uri_physical(struct webnet_session* session, int
event)
{
    /* If the LOG function module is enabled, use the LOG log output function*/
#ifdef WEBNET_USING_LOG
    webnet_module_log(session, event);
#endif

    /* If the ALIAS function module is enabled, determine whether the current request is an alias request*/
#ifdef WEBNET_USING_ALIAS
    result = webnet_module_alias(session, event); if (result ==
WEBNET_MODULE_FINISHED) return result;
#endif

    /* If the AUTH function module is enabled, determine whether the current request requires basic authentication operations*/
#ifdef WEBNET_USING_AUTH
    result = webnet_module_auth(session, event); if (result ==
WEBNET_MODULE_FINISHED) return result;
#endif

    /* If the CGI function module is enabled, determine whether the current request needs to perform CGI operations*/
#ifdef WEBNET_USING_CGI
    result = webnet_module_cgi(session, event); if (result ==
WEBNET_MODULE_FINISHED) return result;
#endif
    ...
}

```

5. Return **HTTP** response data

After successfully judging the function module type and correctly executing the corresponding function, the WebNet server will respond to the request of the current session connection. For example, after the CGI function is executed, the `webnet_session_printf` or `webnet_session_write` function can be used in the CGI execution function to send the response data to the client.

```
/* This function is the CGI function execution function. When the browser accesses the current CGI event, this function is executed to return the response header information
and data*/
static void cgi_hello_handler(struct webnet_session* session) {

    /* Splice the page data that needs to be sent*/
    ....

    /* Send response header information */
    webnet_session_set_header(session, mime_get_type(".html"), 200, "Ok", strlen(
        status));

    /* Send response data */
    webnet_session_write(session, (const rt_uint8_t*)status, rt_strlen(status));
}
```

## 6. Close the connection session

After the current session connection request is parsed successfully, the function module is executed, and the response data is sent, the current connection session will be closed, the session structure will be released, and the entire HTTP data interaction process will be completed, so that the web page files provided by the device can be accessed on the browser, or the operations of uploading and downloading files on the server can be completed.

## 4.4 Usage

For the various functional modules of the WebNet server, some functional modules require setting corresponding configuration parameters before use. The module needs to cooperate with the page code to realize the function. Next, we will introduce how to use different functional modules of the WebNet server.

- **LOG** log display function

After it is turned on, it can display basic information of the session request, such as the IP address and port number of the connection setting, HTTP request type, access Address and other information. It is recommended to turn it on when debugging the code.

- **AUTH** basic authentication function

Basic Authentication can divide access rights by directory. Before initializing the WebNet server, you need to call the `webnet_auth_set` function to set the directory's username and password (the format is username: password). When accessing the directory in the browser, you need to enter the correct username and password to access the directory. The relevant function definitions are as follows:

```
/* Set basic authentication information for the directory */
void webnet_auth_set(const char* path, const char* username_password);
```



The sample code of AUTH basic authentication function is as follows:

```
void webnet_test(void) {

    /* Set the username and password of the /admin directory to admin */
    webnet_auth_set("/admin", "admin:admin"); webnet_init();

}
```

After setting the basic authentication function for the /admin directory, when you access the /admin directory in a browser, you need to enter the set username and password to access it, as shown in the figure:

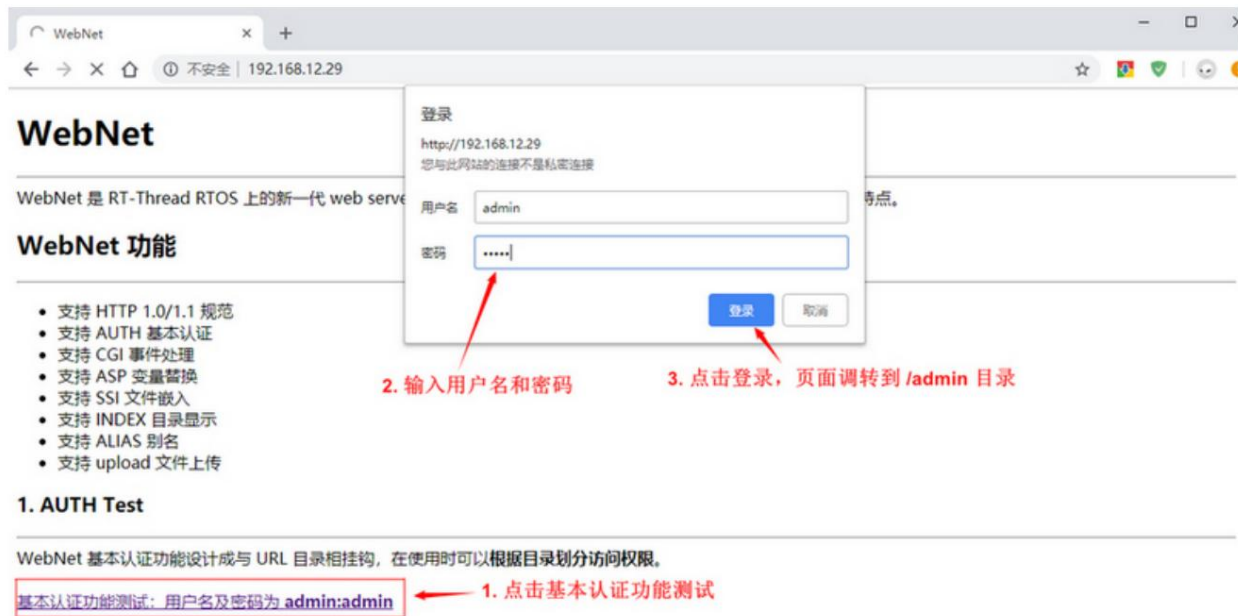


Figure 4.1: Basic authentication function

#### • CGI functionality

The CGI function can customize the execution function of an event. When the browser sends a request for the event, the WebNet server can perform the corresponding operation. You need to call the `webnet_cgi_register` function to register the CGI execution function before initializing the WebNet server. The related function definitions are as follows:

```
/* Set the CGI event root directory */
void webnet_cgi_set_root(const char* root); /* Set CGI event
execution function*/ void
webnet_cgi_register(const char* name, void (*handler)(struct webnet_session*
session));

/* Send header information to the client of the WebNet connection, used in the CGI event
execution function*/ void webnet_session_set_header(struct webnet_session *session, const char* mimetype,
int code, const char* status, int length);
/* Send fixed format data to the client connected to WebNet, used in CGI event execution function*/
void webnet_session_printf(struct webnet_session *session, const char* fmt, ...); /* Send data to the client of the WebNet
connection, used in the CGI event execution function*/
```

```
int webnet_session_write(struct webnet_session *session, const rt_uint8_t* data,
    rt_size_t size);
```

The sample code for using the CGI function is as follows:

```
static void cgi_hello_handler(struct webnet_session* session) {

    const char* hello = "Hello World\n";
    webnet_session_set_header(session, mime_get_type(".html"), 200, "Ok", strlen(
        status));
    webnet_session_write(session, hello, rt_strlen(hello)); webnet_session_printf(session,
        "%s", hello);

} void webnet_test(void) {

    /* Set the CGI event execution function*/
    webnet_cgi_register("hello", cgi_hello_handler);
    webnet_init();
}
```

The corresponding page code is as follows. Clicking the **hello world** button on the browser will send the corresponding CGI request to the server.

```
<html>
<body>
<hr>
<h3> CGI Test </h3>
WebNet CGI function allows users to execute specified functions, CGI test:
<br/><br/>

<br/>
</body>
</html>
```

#### • ASP variable replacement function

The ASP variable replacement function can match the variables contained in the `<% %>` tags in the web page code and replace them with the execution functions registered in the code.

Therefore, before initializing WebNet, you need to call `webnet_esp_add_var` to set the ASP variable replacement function. The related function definitions are as follows:

```
/* Set ASP variable execution function*/
void webnet_esp_add_var(const char* name, void (*handler)(struct webnet_session*
    session));
```

The ASP function sample code is as follows:

```
static void asp_var_version(struct webnet_session* session) {

    webnet_session_printf(session, "RT-Thread: %d.%d.%d\n", RT_VERSION,
        RT_SUBVERSION, RT_REVISION);
```

```

}

void webnet_test(void) {

    /* Set ASP variable to execute function*/
    webnet_asp_add_var("version", asp_var_version); webnet_init();

}

```

The corresponding page code is as follows (the file name is version.asp). Access the page code to replace ASP to display the latest version of RT-Thread:

```

<html>
<head>
<title> ASP Test </title>
</head>
<body>
<% version %>                                /* ASP variable replaced with RT_Thread version number display*/
</body>
</html>

```

- **SSI** file nesting function

WebNet supports embedding text files into web pages. If there is a text file or tag in the page, it will be replaced with the corresponding file content. SSI Functional pages usually end with .shtml or .stm . The following is the sample page code (the file name is index.shtml):

```

<html>
<head>
<title> SSI Test </title>
</head>
<body>
<h1> SSI Test</h1>
<font size="+2">The index.html page embedded in the following page</font>
<hr>
<!--#include virtual="/index.html" --> /* This page embeds the content of the index.html file*/
</body> </
html>

```

- **INDEX** directory file display function

After the WebNet server is initialized successfully, directly enter the set IP address and the directory to be accessed in the browser to list the current directory. Record all the files, as shown in the figure below, access the server/admin directory, and list all the files in the directory:



Figure 4.2: Directory display function

- **ALIAS** alias access function

ALIAS Alias access function can set up alias access for a folder. You need to set up the folder before initializing the WebNet server.

Alias, as shown in the following code, calls the `webnet_alias_set` function to set the alias of the `/test` directory to `/admin`. When the browser accesses `/test` Will jump to the `/admin` directory:

```
void webnet_test(void)
{
    /* Set the alias of the /test directory to /admin */
    webnet_alias_set("/test", "/admin");
    webnet_init();
}
```



Figure 4.3: Alias access function

- **Upload** file upload function

Upload file upload function is used to upload local files to the specified directory of WebNet server. Before uploading files, you need to create and implement Now upload the file structure as shown below:

```
struct webnet_module_upload_entry
{
    const char* url; /* Directory name for file uploads */

    int (*upload_open) (struct webnet_session* session); int (*upload_close) /* open a file */
    (struct webnet_session* session); int (*upload_write)(struct webnet_session* /* Close the file */
    session, const void* data, rt_size_t
    length); /* Write data to file */

    int (*upload_done) (struct webnet_session* session); /* Download completed */
};
```

This structure defines the directory file for uploading files and the event callback functions that need to be used, such as: opening files, closing files, writing data to Documents, etc.

Users need to implement the callback function according to the actual situation. The general operations in each callback function are as follows:

- **upload\_open**: create and open a file in the specified directory by parsing the file name; • **upload\_close**: close the file; • **upload\_write**: write data to the opened file; • **upload\_done**: after the file is uploaded successfully, return the response data to the browser.

In the process of implementing the callback function, the function definition that may be used to obtain the current upload file session information is as follows:

```
/* Get the name of the currently uploaded file*/
const char* webnet_upload_get_filename(struct webnet_session* session); /* Get the type of the
current uploaded file*/ const char*
webnet_upload_get_content_type(struct webnet_session* session); /* Get the file descriptor of the current
uploaded file after it is opened*/
const void* webnet_upload_get_userdata(struct webnet_session* session);
```

For specific implementation methods, please refer to the implementation methods of each function in the software package/samples/wn\_sample\_upload.c.

Finally, before WebNet is initialized, you need to call the `webnet_upload_add` function to set the uploaded file information, as shown in the following code:

```
static int upload_open (struct webnet_session* session) {

    /* Open or create a new file */

} static int upload_close (struct webnet_session* session) {

    /* Close the file */

} static int upload_write (struct webnet_session* session) {

    /* Write data to file */

} static int upload_done (struct webnet_session* session) {

    /* Download completed, return response data*/
}
const struct webnet_module_upload_entry upload_entry_upload = {

    "/upload",
    upload_open,
    upload_close,
    upload_write,
    upload_done
};

void webnet_test(void) {
```

```

/* Register file upload execution function */
webnet_upload_add(&upload_entry_upload); webnet_init();

}

```

The code for uploading files on the corresponding page is as follows:

```

<html>
<body>
  <h3>Upload File Test</h3> The file
  upload module can be used to upload files to a specified directory, here uploaded to the /upload directory under the root directory.
  <br/><br/>
  <form name="upload" method="POST" enctype="multipart/form-data" action="/upload">
    <input type="file" name="file1" > <input
      type="submit" name="submit" value=" Upload ">
  </form>
  <br/>

  <br/><br/>
</body>
</html>

```

#### • Pre-compression function

The WebNet server pre-compression function requires the page resource files to be compressed in advance on the server side to generate a compressed file with the .gz suffix. Take the index.html page in the root directory as an example. When the browser accesses the page file, if there is a compressed file named **index.html.gz**, the compressed file content will be sent to the browser, and the browser will automatically parse and display the original page file.

When using the pre-compression function, you need to upload the compressed **index.html.gz** file to the file system first, as shown in the following figure:



Figure 4.4: Pre-compression function

## 4.5 Frequently Asked Questions

1. The browser does not display page information when accessing the device IP address

• Reason: The root directory address is set incorrectly;

• Solution: Make sure that the root directory address you set is consistent with the directory address created on the device file system, and that there is a page file under the root directory.

Piece.

**2. The device has an `out of pbuf` error**

- Cause: Insufficient device memory;
- Solution: WebNet software package functions such as uploading files require additional resource space. It is recommended to run it on a device with sufficient resource space.  
line, or use it on qemu.

## Chapter 5

### API Description

For the convenience of users, commonly used APIs are listed here and related instructions are given.

#### 5.1 Initialization Function

```
int webnet_init(void);
```

Used to initialize the WebNet server, including creating threads to monitor client connection events, initializing enabled function modules, etc.

parameter	describe
none	none
return	describe
= 0	Initialization successful
< 0	initialization failed

#### 5.2 Set the listening socket port

```
void webnet_set_port(int port);
```

It is used to set the current WebNet server listening port number. The default listening port number of the WebNet server is 80, which is also the HTTP protocol.

When using the default port number to access the URL address, you can directly access it without entering the port number. When using a non-default port number, you need to specify the port number in the URL address, such as: `http://host:8080/index.html` . This function can only be used for the initialization of the **WebNet** server.

Before initialization.

parameter	describe
port	Set the listening socket port
return	describe



parameter	describe
none	none

### 5.3 Get the listening socket port

```
int webnet_get_port(void);
```

Used to obtain the current WebNet server listening socket port number.

parameter	describe
none	none
return	describe
$\geq 0$	Listening socket port number

### 5.4 Set the server root directory

```
void webnet_set_root(const char* webroot_path);
```

Used to set the current WebNet server root directory path. The default root directory of the WebNet server is `/webnet`.

The paths used or accessed in the function are based on the root directory path. When the browser accesses `http://host/index.html`, the file system

`/webnet/index.html` is returned to the browser.

parameter	describe
webroot_path	Set the root directory address
return	describe
none	none

### 5.5 Get the server root directory

```
const char* webnet_get_root(void);
```

Used to obtain the root directory address of the current WebNet server.

parameter	describe
none	none
return	describe

parameter	describe
!= NULL	Root directory address

5.6 Get the type of request link

```
const char* mime_get_type(const char* url);
```

Used to obtain the type of the currently requested URL link, such as web page, image, text, etc.

parameter	describe
url	Request link address
return	describe
!= NULL	Type of link requested

5.7 Add ASP variable processing method

```
void webnet_asp_add_var(const char* name, void (handler)(struct webnet_session session));
```

This function is used to add an ASP variable processing method. When the added `name` variable name appears in the ASP file, the corresponding `handle` operation.

parameter	describe
name	ASP variable name
void (handler)(struct webnet_session session)	ASP variable processing method
return	describe
none	none

5.8 Add CGI event processing method

```
void webnet_cgi_register(const char* name, void (handler)(struct webnet_session session));
```

This function is used to register a CGI event handler. When the browser requests a URL with `name`, the corresponding `handle` operation.

parameter	describe
name	CGI Event Name

parameter	describe
void (handler)(struct webnet_session session)	CGI event handling
return	describe
none	none

5.9 Setting the CGI event root directory

```
void webnet_cgi_set_root(const char* root);
```

The default CGI event root directory of the WebNet server is /cgi-bin. When the browser requests the address <http://host/cgi-bin/test> When the CGI event processing function corresponding to the `test` name is executed.

This function is used to set a new CGI event root directory. The previous CGI root directory will no longer work after it is successfully set.

parameter	describe
root	CGI Event Root Directory
return	describe
none	none

5.10 Set basic authentication information

```
void webnet_auth_set(const char* path, const char* username_password);
```

Used to set basic authentication information for directory access, including user name and password.

parameter	describe
path	Directory where basic authentication information needs to be set
username_password	The username and password to be set are in the format of username:password
return	describe
none	none

5.11 Setting Directory Aliases

```
void webnet_alias_set(char* old_path, char* new_path);
```

Used to set the alias of a directory. After the setting is successful, the directory alias can be used to access the directory.

parameter	describe
old_path	Directory that needs to be aliased
new_path	The directory alias set is generally a directory that exists on the server
return	describe
none	none

## 5.12 Send HTTP request header

```
void webnet_session_set_header(struct webnet_session* session, const char* mimetype, int
code, const char* title, int length);
```

Used to splice and send header information to the connected client, generally used in ASP variable processing functions and CGI event processing functions.

parameter	describe
session	The current server connected session
mimetype	The response file type (Content-Type) that needs to be sent, which can be obtained using the <a href="#">mime_get_type</a> function
code	The response status code sent is normally 200
title	The response status type sent, normally it is OK
length	The length of the response file to be sent (Content-Length)
return	describe
none	none

## 5.13 Send HTTP response data

```
int webnet_session_write(struct webnet_session* session, const rt_uint8_t* data, rt_size_t
size);
```

Used to send response data to the client, generally used in ASP variable processing functions and CGI event processing functions.

parameter	describe
session	The current server connected session
data	Pointer to the data being sent
size	The length of the data sent
return	describe
none	none

## 5.14 Send HTTP fixed format response data

```
void webnet_session_printf(struct webnet_session* session, const char* fmt, ...);
```

Used to send fixed-format response data to the client, generally used in ASP variable processing functions and CGI event processing functions.

parameter	describe
session	The current server connected session
fmt	Custom expression for input data
...	Input parameters
return	describe
none	none

## 5.15 Get the name of the uploaded file

```
const char* webnet_upload_get_filename(struct webnet_session* session);
```

Get the name of the currently uploaded file, used to open or create a file.

parameter	describe
session	The current server connected session
return	describe
!= NULL	The name of the currently uploaded file

## 5.16 Get the type of uploaded file

```
const char* webnet_upload_get_content_type(struct webnet_session* session);
```

Get the type of the currently uploaded file.

parameter	describe
session	The current server connected session
return	describe
!= NULL	The type of the currently uploaded file

## 5.17 Get uploaded file parameters

```
const char* webnet_upload_get_nameentry(struct webnet_session* session, const char*
name);
```

Gets the registered upload file delimiter (HTTP request boundary parameter).

parameter	describe
session	The current server connected session
name	Directory path for uploaded files
return	describe
!= NULL	The type of the currently uploaded file

## 5.18 Get the file descriptor of the uploaded file

```
const void* webnet_upload_get_userdata(struct webnet_session* session);
```

Get the file descriptor generated after the current uploaded file is opened, which is used to read and write data to the file.

parameter	describe
session	The current server connected session
return	describe
!= NULL	File descriptor for the uploaded file