Micriμm

Empowering Embedded Systems

μC/DHCPc

V1.85

User's Manual

www.Micrium.com

Disclaimer

Specifications written in this manual are believed to be accurate, but are not guaranteed to be entirely free of error. Specifications in this manual may be changed for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, Micrium assumes no responsibility for any errors or omissions and makes no warranties. Micrium specifically disclaims any implied warranty of fitness for a particular purpose.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies. Brand and product names are trademarks or registered trademarks of their respective holders.

Registration

Please register the software via email. This way we can make sure you will receive updates or notifications of updates as soon as they become available. For registration please provide the following information:

- Your full name and the name of your supervisor
- Your company name
- Your job title
- Your email address and telephone number
- Company name and address
- Your company's main phone number
- Your company's web site address
- Name and version of the product

Please send this information to: **licensing@micrium.com**

Contact address

Micrium
949 Crestview Circle
Weston, FL 33327-1848
U.S.A.
Phone : +1 954 217 2036
FAX : +1 954 217 2037
WEB : **www.micrium.com**
Email : **support@micrium.com**

Manual versions

If you find any errors in this document, please inform us and we will make the appropriate corrections for future releases.

Manual Version	Date	By	Description
V1.00	2004/06/08	JJL	First version.
V1.85	2006/05/11	SR	Code and documentation update.

Table Of Contents

Introduction

DHCP is a protocol designed to enable clients to get IP configuration from a centralized database. This protocol has slightly evolved over the years from the BOOTP protocol initially designed to enable diskless clients to boot from the network. The µC/DHCPc module implements part of the following RFC:

RFC #2131	**ftp://ftp.rfc-editor.org/in-notes/rfc2131.txt**
RFC #2132	**ftp://ftp.rfc-editor.org/in-notes/rfc2132.txt**

This document describes how to configure and use the µC/DHCPc module in a µC/TCP-IP and µC/OS-II environment. A CSB337 Cogent embedded computer and IAR Embedded Workbench is used to demonstrate the typical application of this module, but other embedded platforms and tool chains may be used as well.

Required modules version

The current version of the µC/DHCPc module has been developed and tested using version **1.84** of µC/TCP-IP and version **2.82** of µC/OS-II.

Chapter 2

Directories and Files

The code and documentation of the µC/DHCPc module are organized in a directory structure according to "AN-2002, µC/OS-II Directory Structure". Specifically, the files may be found in the following directories:

\Micrium\Software\uC-DHCPc

This is the main directory for µC/DHCPc.

\Micrium\Software\uC-DHCPc\Doc

This directory contains the µC/DHCPc documentation files, including this one.

\Micrium\Software\ uC-DHCPc\Cfg\Template

This directory contains a template of µC/DHCPc configuration.

\Micrium\Software\uC-DHCPc\Source

This directory contains the µC/DHCPc source code. This protocol is implemented in two OS independent files:

```
dhcp-c.c
dhcp-c.h
```

Note that the '-c' at the end of `dhcp` stands for client and thus contains 'client' side code.

\Micrium\Software\uC-DHCPc\OS\uCOS-II

This is where OS dependant code is located. This distribution of µC/DHCPc comes with the µC/OS-II version. Note that it would be possible to use this DHCP client with other operating systems by developing an appropriate `dhcp-c_os.c` implementation file.

\Micrium\Software\EvalBoards\Cogent\CSB337\IAR\BSP

This directory contains the Board Support Package (BSP) for the CSB337:

`bsp.c`
`bsp.h`

Board-dependant code for µC/OS-II and other board specific functions.

`CSB33x_lnk_ram.xcl`

IAR Embedded Workbench linker configuration file (memory map).

`net_bsp.c`
`net_bsp.h`

Board-dependant code for µC/TCP-IP NIC module.

`net_isr.c`

Board-dependant code for µC/TCP-IP NIC module ISR manager.

\Micrium\Software\EvalBoards\Cogent\CSB337\IAR\uC-Apps\Ex1

This directory is the directory that contains the source code for Example 1 running on a CSB337 card with the IAR tool chain:

`app.c`

Test code.

`app_cfg.h`

Example application configuration file.

`Ex1.*`

IAR Embedded Workbench project files.

`includes.h`

Master include file used by the application.

`net_cfg.h`

µC/TCP-IP configuration file.

`os_cfg.h`

µC/OS-II configuration file.

Chapter 3

Using µC/DHCPc

3.01 µC/DHCPc test code

As mentioned in the previous section, the test code for this board is found in the following directory and will be briefly described:

\Micrium\Software\EvalBoards\Cogent\CSB337\IAR\uC-Apps\Ex1

The file `app.c` contains the application code for example 1, and was written to illustrate the capabilities of the µC/DHCPc module. That code simply initializes µC/OS-II, µC/TCP-IP and µC/DHCPc, and creates a few tasks and other kernel objects that will give the user information about the state of the system.

Note that some sections of the source code have been removed to help focus on the µC/DHCPc module use.

Listing 3-1

```
void  AppInit_TCPIP (void)
{
    NET_ERR      err;
    CPU_INT08U  *pmac


    APP_DEBUG_TRACE("Initialize TCP/IP stack...\n");

    pmac = (CPU_INT08U *)mon_getenv("ETHERADD");                                (1)
    NetASCII_Str_to_MAC(pmac, NetIF_MAC_Addr, &err);

    err = Net_Init();                                                           (2)
    if (err != NET_ERR_NONE) {
        APP_DEBUG_TRACE("Net_Init() failed: error #%d, line #%d.\n", err, __LINE__);
        while (DEF_YES) {
            ;
        }
    }

    APP_DEBUG_TRACE("    IP  = 0.0.0.0\n");
    APP_DEBUG_TRACE("    MAC = %s\n", pmac);

    NetIP_CfgAddrThisHost(NET_IP_ADDR_THIS_HOST, NET_IP_ADDR_NONE);             (3)
    NetIP_CfgAddrDfltGateway(NET_IP_ADDR_NONE);
}
```

```
void  AppInit_DHCPc (void)
{
    NET_ERR      err;
    CPU_INT08U  *opt;
    CPU_INT08U   buf[16];
    CPU_INT08U   param_req_list[] = {DHCP_OPT_HOST_NAME,                          (4)
                                     DHCP_OPT_DOMAIN_NAME,
                                     DHCP_OPT_DOMAIN_NAME_SERVER,
                                     DHCP_OPT_TIME_SERVER,
                                     DHCP_OPT_TIME_OFFSET};


    DHCPc_SetMacAddr(NetIF_MAC_Addr);                                            (5)
    DHCPc_SetClientID(0x01, NetIF_MAC_Addr, NET_IF_ADDR_SIZE);                   (6)
    DHCPc_SetVendorClassID("YourCompany", 11);                                   (7)
    DHCPc_SetParamRequestList(param_req_list,                                    (8)
                              sizeof(param_req_list) / sizeof(param_req_list[0]));

    APP_DEBUG_TRACE("Get DHCP lease... ");
    err = DHCPc_Start();                                                         (9)
    if (err != DHCPc_ERR_NONE) {                                                 (10)
        APP_DEBUG_TRACE("fail!\n");
        while (DEF_YES) {
            ;
        }
    }

    APP_DEBUG_TRACE("success!\n");

    DHCPc_CfgStack();                                                            (11)

    DHCPc_Print(DHCPc_GetHdr());                                                 (12)


    Mem_Copy (&AppIPAddr,                                                        (13)
              &(DHCPc_GetHdr()->yiaddr),
               sizeof(DHCPc_GetHdr()->yiaddr));
    AppIP_Addr = NET_UTIL_NET_TO_HOST_32(AppIP_Addr);

    opt = DHCPc_GetOpt(DHCP_OPT_SUBNET_MASK);                                    (14)
    if (opt != (void *)0) {
        Mem_Copy ((void *)&AppIPMsk,    (void *)(opt + 2), *(CPU_INT08U *)(opt + 1));
        AppIP_Mask = NET_UTIL_NET_TO_HOST_32(AppIP_Mask);
    }

    opt = DHCPc_GetOpt(DHCP_OPT_ROUTER);
    if (opt != (void *)0) {
        Mem_Copy ((void *)&AppIPGw, (void *)(opt + 2), *(CPU_INT08U *)(opt + 1));
        AppIP_DfltGateway = NET_UTIL_NET_TO_HOST_32(AppIP_DfltGateway);
    }

    opt = DHCPc_GetOpt(DHCP_OPT_DOMAIN_NAME_SERVER);
    if (opt != (void *)0) {
        Mem_Copy ((void *)&AppIPDnsSrv, (void *)(opt + 2), *(CPU_INT08U *)(opt + 1));
        AppIP_DNS_Srvr = NET_UTIL_NET_TO_HOST_32(AppIP_DNS_Srvr);
    }

    NetASCII_IP_to_Str(NET_UTIL_NET_TO_HOST_32(AppIPAddr),   buf, DEF_NO, &err);
    APP_DEBUG_TRACE("DHCP IP address : %s\n", buf);

    NetASCII_IP_to_Str(NET_UTIL_NET_TO_HOST_32(AppIPMsk),    buf, DEF_NO, &err);
    APP_DEBUG_TRACE("DHCP IP mask    : %s\n", buf);

    NetASCII_IP_to_Str(NET_UTIL_NET_TO_HOST_32(AppIPGw),     buf, DEF_NO, &err);
    APP_DEBUG_TRACE("DHCP IP gateway : %s\n", buf);

    NetASCII_IP_to_Str(NET_UTIL_NET_TO_HOST_32(AppIPDnsSrv), buf, DEF_NO, &err);
    APP_DEBUG_TRACE("DHCP IP dns     : %s\n", buf);
}
```

L3-1(1) Ask the monitor to obtain the MAC address. This address is absolutely mandatory for the DHCP process to be brought forward. You will have to determine how to obtain the MAC address of your own target, if it's not a CSB337.

L3-1(2) Initialization of the µC/TCP-IP stack.

L3-1(3) Configuration of the stack with generic values. DHCP **REQUIRES** that you use IP / Mask address 0.0.0.0 during lease negotiation.

L3-1(4) This is an example of a parameter request list. If you use µC/DNSc or µC/SNTPc, you may use this example to obtain DNS or SNTP configuration dynamically via DHCP.

L3-1(5) Provision of µC/DHCPc with the MAC address.

L3-1(6) Provision of µC/DHCPc with the Client ID. As an example, we use the MAC address as the Client ID. The Client ID can be any type of information.

L3-1(7) Provision of µC/DHCPc with the Vendor Class ID. As an example, we have set the value "YourCompany"; you can use your own organization name if needed.

L3-1(8) Provision of µC/DHCPc with the parameter request list. The µC/DHCPc module will request these options from the server.

L3-1(9) Start DHCP negotiation. If the process is successful, the DHCP client task is started, as well as its associated timers, and the service is fully functional.

L3-1(10) Check for successful negotiation. On failure, you can restart the process or bail out. You cannot assign yourself an IP address which is in the DHCP server scope for it might create IP address assignation conflicts.

L3-1(11) Configure µC/TCP-IP stack with the IP configuration received from server. IP address, mask and gateway are configured. Any other options are to be configured by the application.

L3-1(12) Print the last DHCP packet received (*ACK* if everything went fine) for debugging purposes.

L3-1(13) Obtain the value of the IP address attributed by the DHCP server.

L3-1(14) Obtain the value of the subnet mask, default gateway (router), and domain name server options. See section 3.03 for explanations on how to fetch options using the `DHCPc_GetOpt()` function.

3.03 µC/DHCPc module use

The µC/DHCPc module relies on the µC/TCP-IP stack to work. As you can see in the example file (see section 3.01), the configuration of the µC/TCP-IP stack is different when DHCP is used.

Configuring µC/TCP-IP to be used with µC/DHCPc

- The stack must be initially configured with generic IP, mask and gateway;
- The µC/DHCPc module must be configured (see section 3.04);
- The DHCP negotiation process is started (call `DHCPc_Start()`);
- On success, the `DHCPc_CfgStack()` is called by the application to have the IP, mask and gateway set with the values obtained from the DHCP server. The usage of `DHCPc_CfgStack()` is optional. The user may choose to configure the stack directly in the application instead of calling this function for there might be other parameters needed to be set.
- The application may use the `DHCPc_GetHdr()` and the `DHCPc_GetOpt()` functions to obtain more information about the lease obtained. The `DHCPc_GetHdr()` return the DHCP *ACK* packet header and the `DHCPc_GetOpt()` returns a pointer to the option entry in the DHCP *ACK* packet. If the option is not found, the `NULL` value is returned.

DHCP header and option entries format
The DHCP header and option entries format are defined in `dhcp-c.h`, according to RFC 2131. For convenience sake, the 'DHCP options' section of the `dhcp-c.h` file defines all DHCP options and associated length.

DHCP lease negotiation
Here is what happens when the `DHCPc_Start()` function is called.

- The DHCP client task is started.
- A UDP socket is opened and bound to IP address 0.0.0.0, at the port specified by `DHCPc_CFG_IP_PORT_CLIENT`. This enables reception of packet from any host on port 68 (DHCP client).
- A DHCP *DISCOVER* packet is built using your MAC address, Client ID and Vendor Class ID and is broadcasted on the LAN, at the port specified by `DHCPc_CFG_IP_PORT_SERVER`.
- The first server to answer with a DHCP *OFFER* packet is selected by the µC/DHCPc module.
- If nothing or a bad reply is received, a new DHCP *DISCOVER* packet is sent after 4, 8, 16, 32 and 64 seconds. After a number or tries equal to `DHCPc_CFG_MAX_REQ_LEASE_RETRY`, the `DHCP_Start()` function returns with an error.
- A DHCP *REQUEST* packet is built using your MAC address, Client ID, Vendor Class ID, and Transaction ID, to select the *OFFER* previously received and this packet is broadcasted on the LAN at the port specified by `DHCPc_CFG_IP_PORT_SERVER.`
- Only the selected server responds with a DHCP *ACK* packet.

- If nothing or a bad reply is received, a new DHCP *DISCOVER* packet is sent after 4, 8, 16, 32 and 64 seconds. After a number or tries equal to `DHCPc_CFG_MAX_REQ_LEASE_RETRY`, the `DHCP_Start()` function returns with an error.

- The lease is acquired, and its associated timers are set up. The function then returns without error.

µC/DHCPc module internal

Figure 3.1 above illustrates the internal operation this implementation of the DHCP protocol. It mostly concentrates on the DHCP client task created when DHCPc_Start() is called. This figure hence explains the transitions in the task's state machine.

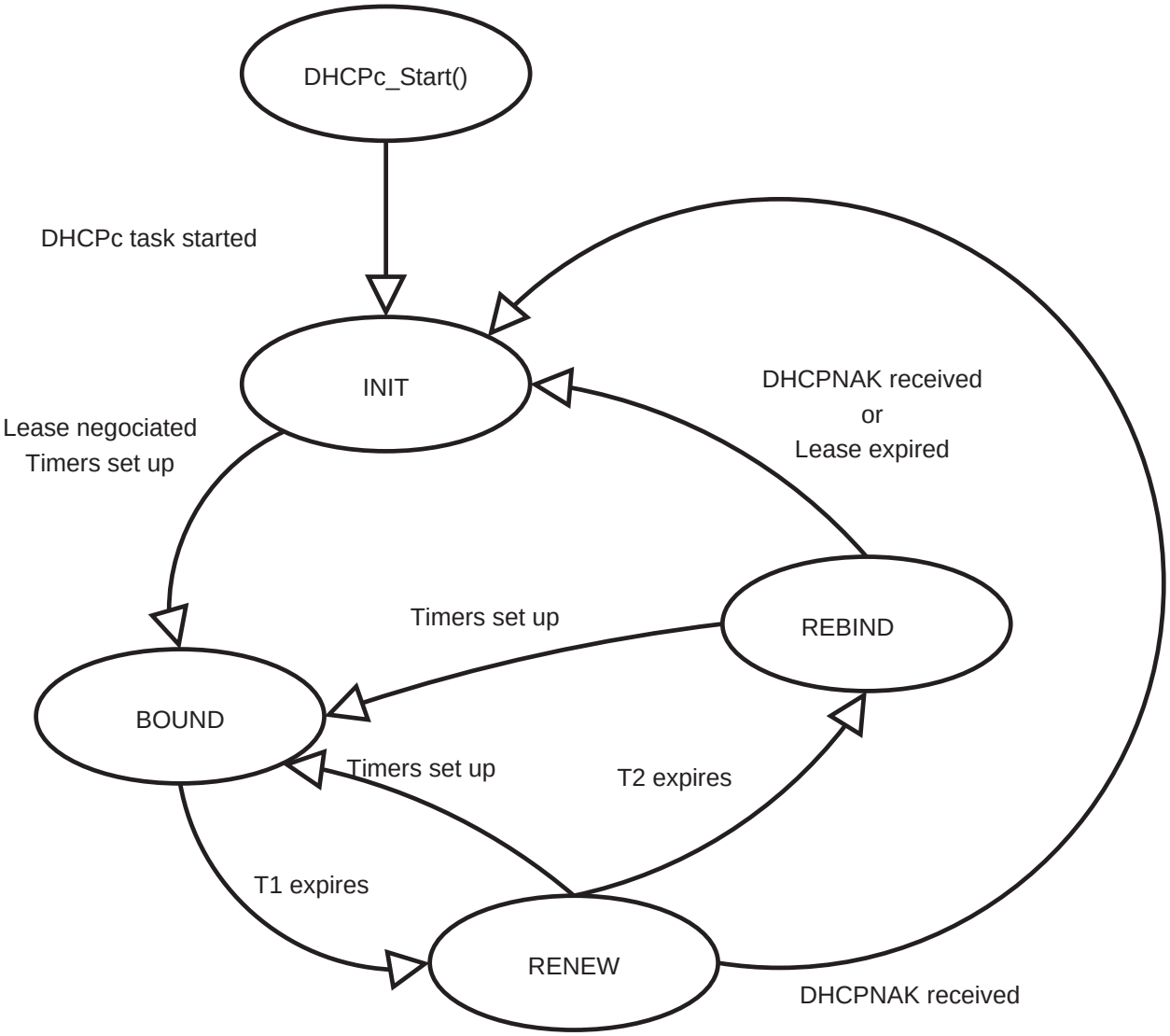

Figure 3-1, µC/DHCPc internal

DHCP broadcast and unicast messages

Some implementations of DHCP servers send DHCP *OFFER* and DHCP *ACK* messages in unicast packets addressed to the newly assigned IP address. The µC/TCP-IP stack does not support this behavior since it can only accept packets addressed to the current IP address or 0.0.0.0, but not to the newly assigned IP address.

When µC/DHCPc is used with the µC/TCP-IP stack, the DHCPc_CFG_BROADCAST_BIT configuration parameter must be set to DEF_ENABLED to make the module work properly. Refer to section 3.04 for more information on configuration variables.

3.04 Interface with RTOS

µC/DHCPc requires the presence of a Real Time Operating System (RTOS). As mentioned in chapter 2, the sample application uses µC/OS-II. It is possible to port µC/DHCPc so it is used with other OS, simply by providing a appropriate implementation of dhcp-c_os.c.

The main task of the OS related source code is to provide accurate timers for the DHCP client task. Indeed, it is possible that the server gives a time limited lease to the client, and the later is responsible for the lease renewal. The OS code also has to provide some sort of synchronization mechanism. Finally, it has to be able to take track of time (at the second), for the client must take into account the duration of the negotiation in the total lease time.

The µC/OS-II implementation relies on a semaphore to synchronize the DHCP client task in the initialization process. This semaphore makes sure that task does not begin its work as long as the lease has not been acquired. A series of three timers are used for T1, T2 and the lease time (see RFC #2131 for the significance of those times), and the services of a message queue are brought forward to let the DHCP client task know a timer has expired.

To sum up, the RTOS used must at the very least provide timers facility, as well as some synchronization mechanism (semaphore, flags, message queue, …). Please refer to the µC/OS-II source code (dhcp-c_os.c) if you are interested in porting the µC/DHCPc module to another RTOS.

3.05 µC/DHCPc module configuration

The µC/DHCPc module has to be configured according to your specific needs. A template configuration file (dhcp-c_cfg.h) is included in the module package (see Chapter 2, Directories and Files), and this configuration should be copied into your app_cfg.h file. Here is the list of the values and description of each of the configuration variable. However, keep in mind that future releases of this module might include more configuration options.

```
#define  DHCPc_OS_CFG_TASK_NAME                "DHCP (Client)"
```

Name of the µC/DHCPc task, mostly used for debugging purposes. With µC/OS-II aware debuggers, you can recognize easily this task among the others in the system.

```
#define  DHCPc_OS_CFG_TASK_PRIO                 13
```

Value of the µC/DHCPc task priority. The value assigned depends of the software architecture of your system, and the importance of this task response time relative to the other tasks. This configuration constant should be placed in the product's APP_CFG.H (if one exist for the project) in order to group all task related configuration in one place.

```
#define  DHCPc_OS_CFG_TASK_STK_SIZE             256
```

Value of the µC/DHCPc task stack size. This default value SHOULD be enough for most environments, but you should check this on your system for reliability or performance purpose. This configuration

constant should be placed in the product's APP_CFG.H (if one exist for the project) in order to group all task related configuration in one place.

```
#define  DHCPc_CFG_IP_PORT_SERVER                    67
#define  DHCPc_CFG_IP_PORT_CLIENT                    68
```

Define respectively the TCP port µC/DHCPc will send requests to and the one used to receive replies. Those default values are the ones specified in RFC 2131.

```
#define  DHCPc_CFG_MAX_REQ_LEASE_RETRY                5
```

Maximum number of attempts to negotiate lease with the DHCP server.

```
#define  DHCPc_CFG_BROADCAST_BIT                  DEF_ENABLED
```

Whether or not to instruct the server to use broadcast when sending replies. This MUST be set to DEF_ENABLED when used with µC/TCP-IP. See section 3.03 for more details.

Before starting the IP lease negociation (DHCPc_Start()), some information must be provided to the µC/DHCPc module. Refer to listing 3-1 to see how to pass this information to the module.

MAC address (mandatory)

The DHCP protocol requires the MAC address of the client to index the configuration information in the server database. Note that recent DHCP servers can use the Client ID instead of the MAC address to index database but even in this case, the MAC address is still mandatory.

Client ID (optional)

In an environment with multiple physical layers supporting the IP network, it may be inconsistent to mix MAC addresses with other address spaces to index server database. For this reason, recent DHCP servers can use the Client ID instead of hardware address. Client ID must be unique in the subnet. It can represent a MAC address (type = 1), other hardware addresses, or anything else (type = 0).

Vendor Class ID (optional)

The vendor class ID can be used to instruct the server to send special configuration or options to clients sold by a single vendor. Some vendor units may need configuration which is not defined in the DHCP standard, using option #43 (vendor-specific information), or options with values greater than 128 (site-specific options).

Parameter Request List (optional)

A list of options requested to the DHCP server. By default, µC/DHCPc requests the subnet mask and gateway. You can request any other option you need in your application. When the server replies, you can get the option value(s) using the DHCPc_GetOpt() function. Note that the DHCP server may not have a value for all options. In that case, the DHCPc_GetOpt() function returns a NULL pointer.

3.06 µC/DHCPc module limitations

The µC/DHCPc module does not perform a final check on the allocated address received from the server. RFC 2131 suggests to issue an ARP request for the given address to make sure it is not in use on the network. Since this is not a mandatory behavior, this implementation does not perform this action.

3.07 µC/DHCPc memory requirements

Using the µC/DHCPc module would typically consume 592 bytes on the **calling task's stack**. You should therefore make sure this MINIMUM amount of memory is available.

As for the µC/DHCPc task itself, refer to section 3.04 (`DHCPc_OS_CFG_TASK_STK_SIZE`) for stack requierements.

Chapter 4

µC/DHCPc API Reference

This chapter provides a reference to the µC/DHCPc API. Each of the user-accessible services is presented in alphabetical order. The following information is provided for each of those services:

- A brief description
- The function prototype
- The filename of the source code
- A description of the arguments passed to the function
- A description of the returned value(s)
- Specific notes and warnings on using the service

DHCPc_CfgStack()

```
NET_ERR DHCPc_CfgStack(void);
```

File	Called from
dhcp-c.c	Application

DHCPc_CfgStack()configures µC/TCP-IP with the IP address, mask and gateway/router with values taken from the last DHCP *OFFER* packet.

Arguments

None.

Returned Values

Error message:

DHCPc_ERR_NONE	No error.
	- RETURNED BY DHCPc_CfgStackVal():-
DHCPc_ERR_NONE	No error.
DHCPc_ERR_CFG_STACK_FAILED	Error configuring stack.

Notes/Warnings

None.

Example

```
void Task (void *p_arg)
{
    /* DHCP already started */

    DHCPc_CfgStack();
}
```

DHCPc_GetHdr()

```
DHCP_HDR DHCPc_GetHdr(void);
```

File	Called from
dhcp-c.c	Application

`DHCPc_GetHdr()`gets the header of the last DHCP packet received.

Arguments

None.

Returned Values

Header of the DHCP packet.

Notes/Warnings

None.

Example

```
void Task (void *p_arg)
{
    DHCP_HDR  *hdr;


    /* DHCP already started */

    hdr = DHCPc_GetHdr();
}
```

DHCPc_GetOpt()

```
CPU_INT08U DHCPc_GetOpt(CPU_INT08U opt_val);
```

File	Called from
dhcp-c.c	Application

DHCPc_GetOpt()searches through the option list for the specified value and returns a pointer to that option.

Arguments

opt_val Option value to search.

Returned Values

Pointer to the option (the first octet following the option's number passed in argument). For more information, please refer to section 'DHCP OPTIONS' in dhcp-c.h.

Notes/Warnings

None.

Example

```
void Task (void *p_arg)
{
    CPU_INT08U  *opt
    NET_IP_ADDR  AppIPDnsSrv;



    /* DHCP already started */

    opt = DHCPc_GetOpt(DHCP_OPT_DOMAIN_NAME_SERVER);
    if (opt != (void *)0) {
        Mem_Copy((void *)&AppIPDnsSrv, (void *)(opt + 2), *(CPU_INT08U *)(opt + 1);
    }
}
```

DHCPc_SetClientID()

```
void DHCPc_SetClientID(CPU_INT08U  client_id_type,
                       CPU_INT08U *client_id,
                       CPU_INT08U  client_id_len);
```

File	Called from
dhcp-c.c	Application

`DHCPc_SetClientID()` sets the DHCP client ID.

Arguments

client_id_type	Client ID type.
client_id	Client ID.
client_id_len	Length of `client_id`.

Returned Values

None.

Notes/Warnings

None.

Example

```
void Task (void *p_arg)
{
    /* Setting client ID to value of MAC address */
    DHCPc_SetClientID(0x01, NetIF_MAC_Addr, NET_IF_ADDR_SIZE);
}
```

DHCPc_SetMacAddr()

```
void DHCPc_SetMacAddr(CPU_INT08U *mac_addr);
```

File	Called from
dhcp-c.c	Application

`DHCPc_SetMacAddr()` sets the MAC address.

Arguments

mac_addr MAC address to set.

Returned Values

None.

Notes/Warnings

The MAC address is mandatory for the DHCP protocol to work.

Example

```
void Task (void *p_arg)
{
    DHCPc_SetMacAddr(NetIF_MAC_Addr);
}
```

DHCPc_SetParamRequestList()

```
void DHCPc_SetParamRequestList(CPU_INT08U *param_request_list,
                               CPU_INT08U  param_request_list_len);
```

File	Called from
dhcp-c.c	Application

`DHCPc_SetParamRequestList()` sets the DHCP parameter request list.

Arguments

param_request_list	Parameter request list.
param_request_list_len	Length of `param_request_list`.

Returned Values

None.

Notes/Warnings

None.

Example

```
void Task (void *p_arg)
{
    CPU_INT08U   ParamRequestList[] = { DHCP_OPT_HOST_NAME,
                                        DHCP_OPT_DOMAIN_NAME,
                                        DHCP_OPT_DOMAIN_NAME_SERVER,
                                        DHCP_OPT_TIME_SERVER,
                                        DHCP_OPT_TIME_OFFSET
                                      };

    DHCPc_SetParamRequestList(ParamRequestList,
                              (sizeof(ParamRequestList) /
                               sizeof(ParamRequestList[0])));
}
```

DHCPc_SetVendorClassID()

```
void DHCPc_SetVendorClassID(CPU_INT08U *vendor_class_id,
                            CPU_INT08U  vendor_class_id_len);
```

File	Called from
dhcp-c.c	Application

`DHCPc_SetVendorClassID()` sets the DHCP vendor class ID.

Arguments

vendor_class_id Vendor class ID.
vendor_class_id_len Length of `vendor_class_id`.

Returned Values

None.

Notes/Warnings

None.

Example

```
void Task (void *p_arg)
{
    DHCPc_SetVendorClassID("Micrium", 7);
}
```

DHCPc_Start()

```
NET_ERR DHCPc_Start(void);
```

File	Called from
dhcp-c.c	Application

`DHCPc_Start()` initializes the DHCPc task and starts DHCP lease negotiation (INIT state).

Arguments

None.

Returned Values

Error message:

DHCPc_ERR_NONE	No error.
DHCPc_ERR_TASK_INIT	Error initializing.
DHCPc_ERR_MAC_ADDR_NOT_SET	MAC address not set properly.
	- RETURNED BY DHCPc_INIT() : -
DHCPc_ERR_NONE	No error.
DHCPc_ERR_TASK_INIT	Error initializing.
DHCPc_ERR_SOCK_FAIL	Error opening / binding socket.
DHCPc_ERR_TX_FAIL	Error transmitting.
DHCPc_ERR_GIVING_UP	Maximum retry reached, giving up.
DHCPc_ERR_TMR	Error setting timers.

Notes/Warnings

None.

Example

```
void Task (void *p_arg)
{
    NET_ERR      err;
    CPU_INT08U   ParamRequestList[] = { DHCP_OPT_HOST_NAME,
                                        DHCP_OPT_DOMAIN_NAME,
                                        DHCP_OPT_DOMAIN_NAME_SERVER,
                                        DHCP_OPT_TIME_SERVER,
                                        DHCP_OPT_TIME_OFFSET
                                      };

    DHCPc_SetMacAddr(NetIF_MAC_Addr);
    DHCPc_SetClientID(0x01, NetIF_MAC_Addr, NET_IF_ADDR_SIZE);
    DHCPc_SetVendorClassID("Micrium", 7);
    DHCPc_SetParamRequestList(ParamRequestList,
                              (sizeof(ParamRequestList) /
                               sizeof(ParamRequestList[0])));

    err = DHCPc_Start();
    if (err != DHCPc_ERR_NONE) {
        APP_DEBUG_TRACE("DHCPc_Start fail!\n\r");
        return;
    } else {
        APP_DEBUG_TRACE("DHCPc_Start success!\n\r");
    }
}
```

Appendix A

µC/DHCPc Licensing Policy

You need to obtain an 'Object Code Distribution License' to embed µC/DHCPc in a product that is sold with the intent to make a profit. Each 'different' product (i.e. your product) requires its own license, but the license allows you to distribute an unlimited number of units for the life of your product. Please indicate the processor type(s) (i.e. ARM7, ARM9, MCF5272, MicroBlaze, Nios II, PPC, etc.) that you intend to use.

For licensing details, contact us at:

Micrium
949 Crestview Circle
Weston, FL 33327-1848
U.S.A.

Phone : +1 954 217 2036
FAX : +1 954 217 2037

WEB : **www.micrium.com**
Email : **licensing@micrium.com**

Appendix B

References

µC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
CMP Books, 2002
ISBN 1-57820-103-9

Embedded Systems Building Blocks
Jean J. Labrosse
R&D Technical Books, 2000
ISBN 0-87930-604-1