



# CMX-MicroNet Evaluation Version for the Freescale MC9S12NE64

## Getting Started Guide

**TRADEMARKS:**

MC9S12NE64™ is a trademark of Freescale Semiconductor, Inc.  
 Codewarrior™ is a trademark of DevTech  
 CMX™ is a trademark of CMX Systems, Inc.

The version of the DevTech tools used for CMX-MicroNet™ was:  
 Codewarrior Development Studio for Freescale HC12 v4.6

CMX-MicroNet Evaluation Version for the Freescale MC9S12NE64 ..... 1  
 Getting Started Guide ..... 1  
     Installation..... 1  
     CMX-MicroNet Evaluation Version limitations ..... 2  
     Getting Started ..... 3  
     Mnconfig.h..... 3  
     The tcp\_app application..... 6  
     The web1 application..... 6  
     The web2 application..... 7  
     Using the included DevTech projects ..... 8  
     Debugging..... 8

### ***Installation***

The CMX-MicroNet software is distributed in the form of a setup.exe file.  
 If you received your software via email in a .zip file, the zip file is password-protected only to make it more likely to make it through corporate email firewalls. The zip file password is “cmx”.

Run setup.exe.  
 The default installation directory, c:\MICRONET, may be changed to another location.

**Caution:** We recommend installing the software in a root-level directory. The reason for this is that you may experience problems linking if the software is installed in too “deep” a directory. Some of the tools have a finite limit on the length of a directory/path specification, and if this limit is exceeded, then you will experience problems. This kind of issue has become much more common in recent years, with the advent of long file names.

What is installed:

Folders:

- Manual – We highly recommend reading the manual, mn309man.pdf.
- Demo\_apps – This directory contains the source files for the projects.
- Demo\_apps\web1\_pages – Has the web pages used by the web1 application.
- Mw\_hc12\demos\_ne64\netlibe\_ne64 – contains the evaluation version of the netlibe library.
- Mw\_hc12\demos\_ne64\tcp\_app – Holds the TCP/IP echo client/server application.
- Mw\_hc12\demos\_ne64\web1 – Holds a simple web server application.
- Mw\_hc12\demos\_ne64\web2 – Holds a more complex web server application.
- Mw\_hc12\demos\_ne64\web2\web2\_pages – Has the web pages used by the web2 application.
- Netlib – This directory normally hold the core CMX-MicroNet library. For the evaluation version only the CMX-MicroNet header files are here.
- PCtestprograms – These are programs, to run on a PC. They are used so that the CMX-MicroNet MC9S12NE64 software has something to talk to. See the README.TXT file.
- Util – These are various utilities used in building the software. See the util.txt file.

Files:

- Tcp\_app.c – example program for testing CMX-MicroNet TCP/IP echo client and server.
- Web1.c – example program showing a simple HTTP server.
- Web2.c – example program showing an HTTP server with server-side-includes, a POST function and a JAVA applet.
- Callback.c – user callback routines, described in the manual, for CMX-MicroNet.
- Callback2.c – user callback routines modified for the web2 application.
- install.log – Produced during the install, if you have an installation problem while running setup.exe, please email CMX tech support this file.
- license.txt – the software license
- unwise.exe – use this should you wish to uninstall the software

Please email CMX at the following email address to report bugs or problems with the CMX-MicroNet Evaluation Version: [cmx@cmx.com](mailto:cmx@cmx.com)

## ***CMX-MicroNet Evaluation Version limitations***

The CMX-MicroNet Evaluation Version has some limitations that the full CMX-MicroNet does not have.

- The evaluation version will only run for 30 minutes or send 1,000 TCP/IP and/or UDP/IP packets before locking up. The board may be reset to run for another 30 minutes or 1,000 packets. PING reply packets and ARP packets do not count towards the packet limit.
- The options included with the evaluation version are Ethernet, HTTP Server and TFTP.
- The Ethernet MAC address is fixed at 00-12-34-56-78-9A.
- The configuration defines in mnconfig.h are fixed and may not be changed. For example, in the full CMX-MicroNet up to 127 sockets can be open at a time, but in the evaluation version only five sockets may be open at one time. Some other restrictions are that the receive buffer is 900 bytes, the TCP\_WINDOW, which is the amount of data that can be sent in a single packet, is 512 bytes, and six web pages, five GET functions and five POST functions may be added to the virtual file system. See the **Mnconfig.h** section below for a full list.
- The header files in the Netlib directory and the header files in the directories under the Mw\_hc12\demos\_ne64\netlibe\_ne64 directory may not be changed.
- RTOS support is not included.
- Web pages and graphics files are limited to 64 KB each in size.

The following are limitations common to all versions of CMX-MicroNet.

- IP options are ignored.
- ICMP only supports echo reply.
- TCP sends MSS option, received options are ignored.
- TCP respects other side's window, but uses a fixed window itself.
- Every TCP packet must get an ACK before the next one can be received. (No delayed ACKs).

## Getting Started

→ The included example programs and projects are very important in both verifying correct installation and configuration, but also in giving you a working piece of code.

→ Do not change any of the header files in the netlib directory. The netlib library has been built using those files so changing any of them would cause a conflict between the library code and the application code.

→ Applications on the NE64 may be debugged using either the serial monitor or a BDM. Each of the three supplied applications has a target for both the serial monitor and BDM. For example tcp\_app.mcp has targets called "tcp\_app – serial monitor" and "tcp\_app – BDM". Select the appropriate target before building and debugging the demo applications. Note: when using the BDM for the first time, the FLASH must be completely erased, to remove the serial monitor, before files can be successfully loaded using the debugger.

## Mnconfig.h

mnconfig.h is used to select the protocols used, number of sockets, sizes of transmit and receive buffers, etc. See the Configuration File section of the manual for details.

Here is what the mnconfig.h for the evaluation version looks like. For each of the #defines, there is an explanation in the manual which describes what the default settings are, what the setting does, etc.

```

/*****
Copyright (c) CMX Systems, Inc. 2007. All rights reserved
*****/

#ifndef MNCONFIG_H_INC
#define MNCONFIG_H_INC 1

/* Protocols */
#define MN_NUM_INTERFACES 1
#define MN_TCP 1
#define MN_UDP 1
#define MN_UDP_CHKSUM 1
#define MN_ETHERNET 1
#define MN_SLIP 0
#define MN_PPP 0
#define MN_PING_REPLY 1
#define MN_IGMP 0

/* Sockets */
#define MN_NUM_SOCKETS 5
#define MN_SOCKET_WAIT_TICKS 600
#define MN_SOCKET_INACTIVITY_TIME 0

```

```
/* Send and Recv buffers */
#define MN_ETH_RECV_BUFF_SIZE      900
#define MN_ETH_XMIT_BUFF_SIZE      570
#define MN_UART_RECV_BUFF_SIZE     600
#define MN_UART_XMIT_BUFF_SIZE     600

/* TCP/IP options */
#define MN_TIME_TO_LIVE             64
#define MN_ETH_TCP_WINDOW          512
#define MN_UART_TCP_WINDOW         512
#define MN_TCP_RESEND_TICKS        600
#define MN_TCP_RESEND_TRYS         12
#define MN_PING_GLEANING            0
#define MN_PING_BUFF_SIZE          32
#define MN_ALLOW_BROADCAST         0
#define MN_ALLOW_MULTICAST         0
#define MN_MULTICAST_TTL           1
#define MN_IGMP_LIST_SIZE          4
#define MN_IP_FRAGMENTATION        0
#define MN_IP_REASSEMBLY_TIME      60
#define MN_NUM_REASSEMBLIES        1
#define MN_REASSEMBLY_BUFF_SIZE    3000
#define MN_COPY_FRAG_DATA          1

/* Ethernet */
#define MN_POLLED_ETHERNET         0
#define MN_ETHER_WAIT_TICKS        1
#define MN_USE_HW_CHKSUM           0

/* ARP */
#define MN_ARP                     1
#define MN_ARP_WAIT_TICKS          600
#define MN_ARP_TIMEOUT              0
#define MN_ARP_AUTO_UPDATE          0
#define MN_ARP_CACHE_SIZE           4
#define MN_ARP_KEEP_TICKS           6000
#define MN_ARP_RESEND_TRYS          6

/* DHCP */
#define MN_DHCP                     0
#define MN_DHCP_RESEND_TRYS         4
#define MN_DHCP_DEFAULT_LEASE_TIME  36000

/* BOOTP */
#define MN_BOOTP                    0
#define MN_BOOTP_RESEND_TRYS        6
#define MN_BOOTP_REQUEST_IP         1

/* DNS */
#define MN_DNS                      0
#define MN_DNS_WAIT_TICKS           300
#define MN_DNS_SEND_TRYS            3
#define MN_USE_PPP_DNS              0
#define MN_DNS_RECV_BUFF_SIZE       512
#define MN_DNS_XMIT_BUFF_SIZE        64

/* PPP options */
#define MN_USE_PAP                   1
#define MN_PAP_USER_LEN              10
#define MN_PAP_PASSWORD_LEN         10
#define MN_PAP_NUM_USERS             1
#define MN_PPP_RESEND_TICKS          300
#define MN_PPP_RESEND_TRYS           6
#define MN_PPP_TERMINATE_TRYS        2
```

```
#define MN_FAST_FCS 1

/* Modem */
#define MN_MODEM 0
#define MN_DIRECT_CONNECT 1

/* HTTP */
#define MN_HTTP 1
#define MN_SERVER_SIDE_INCLUDES 1
#define MN_INCLUDE_HEAD 0
#define MN_URI_BUFFER_LEN 52
#define MN_BODY_BUFFER_LEN 52
#define MN_HTTP_BUFFER_LEN 512

/* FTP */
#define MN_NEED_MEM_POOL 0
#define MN_MEM_POOL_SIZE 4096
#define MN_FTP_SERVER 0
#define MN_FTP_MAX_PARAM 24
#define MN_FTP_BUFFER_LEN 512
#define MN_FTP_USER_LEN 10
#define MN_FTP_PASSWORD_LEN 10
#define MN_FTP_NUM_USERS 1
#define MN_FTP_CLIENT 0
#define MN_FTPC_USER_LEN 10
#define MN_FTPC_PASSWORD_LEN 10
#define MN_FTPC_ACCOUNT_LEN 0
#define MN_FTPC_CMD_BUFF_SIZE 64
#define MN_FTPC_REPLY_BUFF_SIZE 512
#define MN_FTPC_FILE_BUFFER_LEN 512
#define MN_FTPC_WAIT_TICKS 600

/* TFTP */
#define MN_TFTP 1
#define MN_TFTP_RESEND_TRYIS 3
#define MN_TFTP_USE_FLASH 1

/* SMTP */
#define MN_SMTP 0
#define MN_SMTP_BUFFER_LEN 512
#define MN_SMTP_AUTH 0

/* SNTP */
#define MN_Sntp 0
#define MN_Sntp_WAIT_TICKS 300
#define MN_Sntp_SEND_TRYIS 3

/* POP3 */
#define MN_POP3 0
#define MN_POP3_CMD_BUFFER_LEN 25
#define MN_POP3_REPLY_BUFFER_LEN 512
#define MN_POP3_LINE_BUFFER_LEN 250
#define MN_POP3_WAIT_TICKS 600

/* SNMP */
#define MN_Snmp 0
#define MN_Snmp_VERSION2C 0
#define MN_Snmp_TRAP 0
#define MN_Snmp_IN_BUFF_SIZE 484
#define MN_Snmp_OUT_BUFF_SIZE 484

/* Virtual File System */
#define MN_VIRTUAL_FILE 1
#define MN_NUM_VF_PAGES 6
```

```
#define MN_VF_NAME_LEN      20
#define MN_FUNC_NAME_LEN   20
#define MN_NUM_POST_FUNCS  5
#define MN_NUM_GET_FUNCS   5
#define MN_USE_LONG_FSIZE  0
#define MN_USE_EFFS_THIN   0
#define MN_USE_EFFS        0
#define MN_USE_EFFSM       0
#define MN_USE_EFFS_FAT    0

#endif /* ifndef MNCONFIG_H_INC */
```

## ***The tcp\_app application***

This program can be configured to run as either a TCP echo client or TCP echo server by changing the following define in tcp\_app.c:

```
#define SERVER_MODE      0 /* set to 1 if a server, or 0 if a client */
```

We recommend starting with the example set for TCP client mode, as shown above.

You may also need to edit callback.c in the demo\_apps directory to change the default network gateway IP address and subnet mask. Application specific functions (callbacks) in this file can be changed, if required. If a gateway is not used (gateway IP address is set to 255.255.255.255) then the IP address of the PC and the IP address of the board must be set so they are on the same network. e.g. both are 192.168.2.xxx. These IP addresses are in tcp\_app.c.

The program, when configured for CLIENT\_MODE, will attempt to connect via TCP/IP to a TCP echo server (such as the included TCP echo server program tcp\_svr.exe that runs on a PC) at the destination IP location indicated in tcp\_app.c in the eth\_dest\_addr array and using the echo service port 7. It sends data continuously to this address and then receives it back from the echo server.

You could then do the opposite, use the PC tcp\_cli.exe program, and set the example to run as a server. In this case, the PC will send data to the example program running on the board, which will then echo it back to the PC. When using tcp\_cli.exe you must supply a parameter that is the same as the IP address set in the eth\_src\_addr array in tcp\_app.c. e.g.

```
Tcp_cli 192.168.2.3
```

Tcp\_cli.exe and Tcp\_svr.exe both display the data they receive on the screen.

See the **Using the included DevTech projects** section for more details on using the tcp\_app application.

## ***The web1 application***

The web1 application demonstrates how to use the HTTP server to serve up a simple web page. The web page, index.htm, and its graphics file are in the demo\_apps\web1\_pages directory. The HTML2C utility has been used to convert those files into .c and .h files that are included in the project. Note that the main web page must be called index.htm or index.html.

If server-side-includes, POST functions and JAVA applets are not used, a web server application can be created in just a few steps.

- Convert web pages to .c and .h files using the HTML2C utility.
- #include the created .h files in your application after #include "micronet.h"
- In the main function call mn\_init() before calling any other CMX-MicroNet functions.
- Call functions mn\_assign\_interface() and mn\_open\_interface() to set up the IP addresses
- Add the web pages to the virtual file system with the mn\_vf\_set\_entry() function call and the parameters defined in the .h files created by HTML2C.
- Call the mn\_server() function. This function normally does not return.

See the **Using the included DevTech projects** section for more details on using the web1 application.

## ***The web2 application***

The web2 application serves up a web page with two frames, two server-side-includes, a form and a JAVA applet. If your OS does not have a JAVA virtual machine go to <http://java.sun.com/getjava> and download the JAVA Runtime Environment (JRE) for your OS.

You need to have a TFTP server running on your network if you wish to try uploading a web page using the CMX-MicroNet TFTP client through the web page form. The array ip\_tftp\_addr in web2.c must be changed to reflect the IP address of the TFTP server used. For this application there is one free entry in the Virtual File System so only one web page may be uploaded to the board.

Server-side-includes are a way of inserting dynamic data into a web page. A special tag is placed into the web page specifying a GET function to be called by CMX-MicroNet. A GET function must be placed into the virtual file system with a call to function mn\_gf\_set\_entry(). This user-defined function passes the data to be placed into the web page to the HTTP server, which then replaces the tag with the passed data. For example in bot.htm there is the tag:

```
<param name=Tick value="<!--#exec cgi="getTickVal"-->">
```

When the HTTP server sees the "<!--#exec cgi=" string it looks for a function name inside the following quotes. It then looks up the function name in the virtual file system and runs the associated function, which in this case is get\_tick\_func. A GET function must take a pointer to a pointer to a buffer as a parameter and return the number of bytes placed in the buffer as a word16 variable. See web2.c and the CMX-MicroNet manual for details.

Forms in web pages are handled through POST functions. The ACTION attribute of the form is set to the name of a user-defined POST function. A POST function must be placed into the virtual file system with a call to function mn\_pf\_set\_entry(). When the submit button of the form is clicked the function associated with the POST function name is executed. A POST function is passed a pointer to the socket associated with the web page. The mn\_http\_find\_value function can be called to get the value(s) of the variable(s) in the POST request. In a POST function either the mn\_http\_set\_message function must be called to send a message back to the browser or the mn\_http\_set\_file called to send a web page back to the browser. In the web2 example it looks for a variable called filename. If filename is found, the file is retrieved using function mn\_tftp\_get\_file from the TFTP server defined in ip\_tftp\_addr, and placed into FLASH at address 0x4000 using the TFTP FLASH callback functions in tftpflash.c. If successful the file is added to the Virtual File System. If there already is an entry for that file it will be deleted first. This way the same web page may be modified and uploaded repeatedly. See web2.c, bot.htm and the CMX-MicroNet manual for details.

The web server can also serve up JAVA applets. The applet .class files are converted to .c and .h files using HTML2C and added to the virtual file system the same as other web pages. A powerful feature is the ability of JAVA applets to establish a TCP connection with CMX-MicroNet

thus allowing immediate bi-directional communication between the applet and the board. In the web2 example a socket is opened up for listening on port 2000 before the HTTP server is started. The JAVA applet opens up a TCP connection at startup and then listens for data coming from the board. Function `mn_app_server_idle()` in `callback2.c` is called whenever the web server is not busy processing packets. . In the web2 example this function has been modified to make sure there is a socket on port 2000 available to listen for incoming connections and every five seconds the system timer tick value is sent to all sockets with a destination port of 2000. Note that multiple JAVA applets may connect to the board at the same time. See `web2demo.java`, `web2.c` and `callback2.c` for more details.

See the **Using the included DevTech projects** section for more details on using the web2 application.

## ***Using the included DevTech projects***

The following project files are provided in the `Mw_hc12\demos_ne64` directory.

TCP echo client or server	<code>Tcp_app\tcp_app.mcp</code>
Simple web server	<code>Web1\web1.mcp</code>
More advanced web server	<code>Web2\web2.mcp</code>

→ Open the projects using Codewarrior Development Studio for Freescale HC12 v4.6.

→ Select the appropriate target, serial monitor or BDM.

→ Before running the project, make sure `callback.c` has the desired IP network settings.

If you change one of the web pages in the web server examples then you must run the HTML2C utility found in the `util` directory to create new `.c` and `.h` files. For example if `index.htm` is modified you would run:

```
Html2c index.htm
```

That will create `index.c` and `index.h`. See the Virtual File System section of the CMX-MicroNet manual for more information on using the Virtual File System.

To access the web pages using one of the HTTP server examples, in the browser's address box enter `http://` followed by the board's IP address defined in `callback.c`. e.g.

<http://192.168.2.3>

## ***Debugging***

Besides using the included PC-side TCP/UDP client/server test programs, we highly recommend the use of a packet sniffer. These allow you to see all transmitted frames and see exactly what is going on. Some of the freeware ones, like Wireshark (formerly known as Ethereal), are surprisingly good.

### **Freeware Packet Sniffers for Windows**

AnalogX PacketMon - [www.analogx.com](http://www.analogx.com)

Anasil - [www.sniff-tech.com](http://www.sniff-tech.com)

CommView - [www.tamosoft.com](http://www.tamosoft.com)



Wireshark - [www.wireshark.org](http://www.wireshark.org)  
Sniff'em - [www.sniff-em.com](http://www.sniff-em.com)

**Commercial**

Klos Technologies' SerialView, PacketView [www.klos.com](http://www.klos.com)

Windows Packet sniffing library for C#, C++, VB - <http://www.packet-sniffing.com>

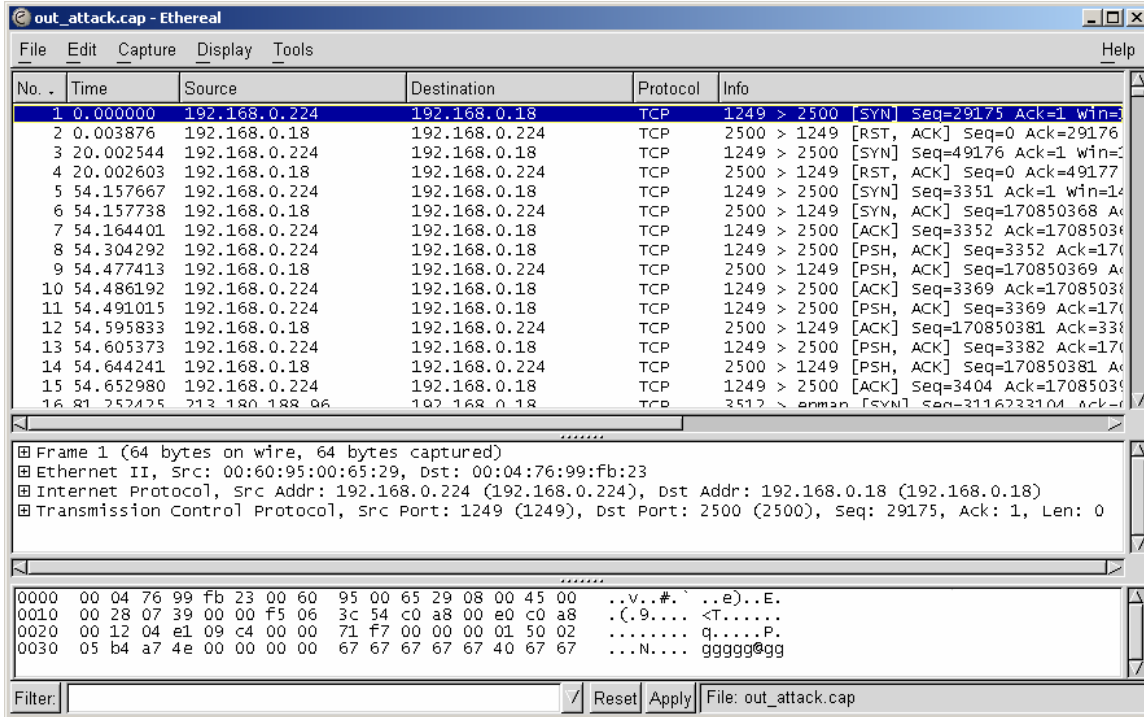


Figure 1 Ethereal freeware packet sniffer