

# IA260/IA261/IA262/EM-2260 Linux User's Manual

---

First Edition, January 2009

[www.moxa.com/product](http://www.moxa.com/product)

**MOXA**®

© 2009 Moxa Inc. All rights reserved.  
Reproduction without permission is prohibited.

# IA260/IA261/IA262/EM-2260 Linux User's Manual

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

## Copyright Notice

Copyright © 2009 Moxa Inc.  
All rights reserved.  
Reproduction without permission is prohibited.

## Trademarks

MOXA is a registered trademark of Moxa Inc.  
All other trademarks or registered marks in this manual belong to their respective manufacturers.

## Disclaimer

Information in this document is subject to change without notice and does not represent a commitment on the part of Moxa.

Moxa provides this document “as is,” without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Moxa reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Moxa assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

## Technical Support Contact Information

**[www.moxa.com/support](http://www.moxa.com/support)**

### Moxa Americas:

Toll-free: 1-888-669-2872  
Tel: +1-714-528-6777  
Fax: +1-714-528-6778

### Moxa China (Shanghai office):

Toll-free: 800-820-5036  
Tel: +86-21-5258-9955  
Fax: +86-10-6872-3958

### Moxa Europe:

Tel: +49-89-3 70 03 99-0  
Fax: +49-89-3 70 03 99-99

### Moxa Asia-Pacific:

Tel: +886-2-8919-1230  
Fax: +886-2-8919-1231

# Table of Contents

<b>Chapter 1</b>	<b>Introduction .....</b>	<b>1-1</b>
	Overview.....	1-2
	Software Architecture .....	1-2
	Journaling Flash File System (JFFS2).....	1-3
	Software Package .....	1-4
<b>Chapter 2</b>	<b>Getting Started .....</b>	<b>2-1</b>
	Powering on the IA260/IA261/IA262/EM-2260 .....	2-2
	Connecting the IA260/IA261/IA262/EM-2260 to a PC .....	2-2
	Debug Port.....	2-2
	Telnet Console.....	2-3
	SSH Console .....	2-4
	VGA Console .....	2-6
	Configuring the Ethernet Interface .....	2-6
	Modifying Network Settings .....	2-6
	Modifying Network Settings by Command.....	2-7
	CF Socket for Storage Expansion .....	2-7
	Test Program—Developing Hello.c .....	2-7
	Installing the Tool Chain (Linux).....	2-8
	Checking the Flash Memory Space .....	2-8
	Compiling Hello.c .....	2-9
	Uploading and Running the “Hello” Program.....	2-9
<b>Chapter 3</b>	<b>Managing Embedded Linux .....</b>	<b>3-1</b>
	System Version Information.....	3-2
	Upgrading the Firmware.....	3-2
	Loading Factory Defaults .....	3-5
	Enabling and Disabling Daemons.....	3-5
	Setting the Run-Level .....	3-7
	Adjusting the System Time .....	3-8
	Setting the Time Manually .....	3-8
	NTP Client.....	3-9
	Updating the Time Automatically .....	3-9
	Cron—Daemon to Execute Scheduled Commands .....	3-10
<b>Chapter 4</b>	<b>Managing Communications .....</b>	<b>4-1</b>
	Telnet / FTP .....	4-2
	DNS .....	4-2
	Web Service—Apache .....	4-3
	Install PHP for Apache Web Server .....	4-5
	IPTABLES .....	4-7
	NAT.....	4-11
	NAT Example .....	4-12
	Enabling NAT at Bootup.....	4-12
	Dial-up Service—PPP.....	4-13
	PPPoE .....	4-16
	NFS (Network File System).....	4-18
	Setting up the IA260/IA261/IA262/EM-2260 as an NFS Client.....	4-18
	Mail.....	4-19

	SNMP .....	4-19
	OpenVPN.....	4-19
<b>Chapter 5</b>	<b>Development Tool Chains .....</b>	<b>5-1</b>
	Linux Tool Chain .....	5-2
	Steps for Installing the Linux Tool Chain .....	5-2
	Compilation for Applications .....	5-2
	On-Line Debugging with GDB .....	5-3
<b>Chapter 6</b>	<b>Programmer's Guide.....</b>	<b>6-1</b>
	Flash Memory Map.....	6-2
	Device API.....	6-2
	RTC (Real Time Clock) .....	6-2
	Buzzer .....	6-3
	UART.....	6-3
	MoxaCAN programming Guide .....	6-5
	Introduction .....	6-5
	Programming Guide .....	6-5
	Digital I/O .....	6-8
<b>Appendix A</b>	<b>System Commands.....</b>	<b>A-1</b>
	Linux normal command utility collection.....	A-1
	File manager.....	A-1
	Editor .....	A-1
	Network.....	A-1
	Process.....	A-2
	Other.....	A-2
	Moxa special utilities.....	A-2

The Moxa IA260/IA261/IA262/EM-2260 computers are RISC-based ready-to-run embedded computers. Available features include two or four RS-232/422/485 serial ports, two 10/100 Mbps Ethernet port, CF socket for storage expansion, VGA output, two CAN bus ports (IA262 only) and USB host making the IA260/IA261/IA262/EM-2260 ideal for your embedded applications.

The following topics are covered in this chapter:

- ❑ **Overview**
- ❑ **Software Architecture**
  - Journaling Flash File System (JFFS2)
  - Software Package

## Overview

The IA260/IA261/IA262/EM-2260 embedded computers, which are designed for industrial automation applications, feature 2 or 4 RS-232/422/485 serial ports, 2 CAN bus ports (only for the IA262), dual Ethernet ports, 8 digital input channels, 8 digital output channels, VGA output and CF socket. The computers come in a compact, IP30 protected, industrial-strength rugged chassis. The DIN-Rail vertical form factor makes it easy to install the IA260/IA261/IA262 embedded computers in small cabinets. This space-saving feature also facilitates easy wiring, and makes the IA260/IA261/IA262/EM-2260 the best choice as front-end embedded controllers for industrial applications.

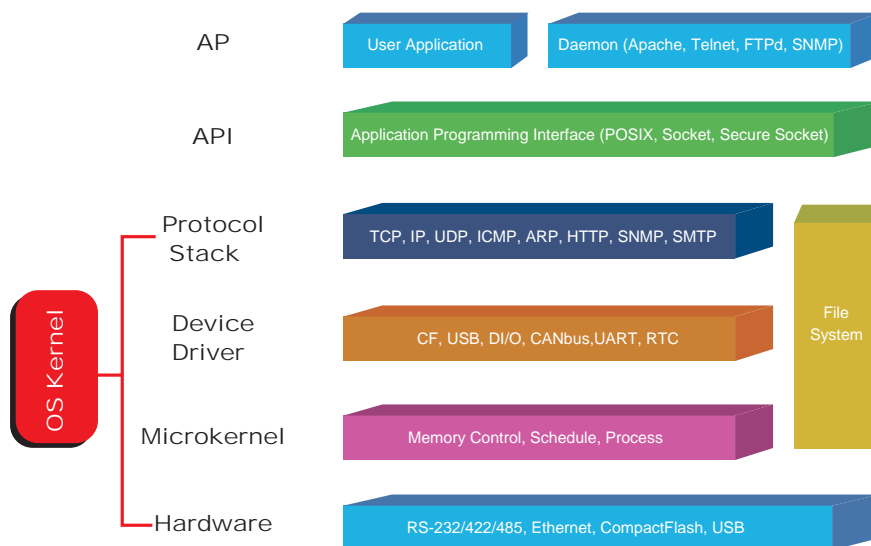
The IA260/IA261/IA262/EM-2260 computers use a Cirrus Logic 199 Mhz RISC CPU. Unlike the X86 CPU, which uses a CISC design, the RISC architecture and modern semiconductor technology provide these embedded computers with a powerful computing engine and communication functions, without generating excessive heat. A 32 MB NOR Flash ROM and a 128 MB SDRAM give you enough memory to install your application software directly on the embedded computer. In addition, dual LAN ports are built right into the RISC CPU. This network capability, in combination with the ability to control serial devices, makes the IA260/IA261/IA262/EM-2260 ideal communication platforms for data acquisition and industrial control applications.

The IA260/IA261/IA262/EM-2260's pre-installed Linux operating system (OS) provides an open software operating system for your software program development. Software written for desktop PCs can be easily ported to the computer with a GNU cross compiler, without modifying the source code. The OS, device drivers (e.g., serial and buzzer control), and your own applications, can all be stored in the NOR Flash memory.

The IA260/IA261/IA262-LX also supports a VGA output.

## Software Architecture

The Linux operating system that is pre-installed on the IA260/IA261/IA262/EM-2260 follows the standard Linux architecture, making it easy to accept programs that follow the POSIX standard. Program porting is done with the GNU Tool Chain provided by Moxa. In addition to Standard POSIX APIs, device drivers for the CF storage, buzzer and Network controls, and UART are also included in the Linux OS.



The IA260/IA261/IA262/EM-2260's built-in Flash ROM is partitioned into **Boot Loader, Linux Kernel, Root File System, and User directory** partitions.

In order to prevent user applications from crashing the Root File System, the IA260/IA261/IA262/EM-2260 uses a specially designed **Root File System with Protected Configuration** for emergency use. This **Root File System** comes with serial and Ethernet communication capability for users to load the **Factory Default Image** file. The user directory saves the user's settings and application.

To improve system reliability, the IA260/IA261/IA262/EM-2260 has a built-in mechanism that prevents the system from crashing. When the Linux kernel boots up, the kernel will mount the root file system for read-only, and then enable services and daemons. During this time, the kernel will start searching for system configuration parameters via *rc* or *inittab*.

Normally, the kernel uses the Root File System to boot up the system. The Root File System is protected, and cannot be changed by the user. This type of setup creates a "safe" zone.

For more information about the memory map and programming, refer to Chapter 5, *Programmer's Guide*.

## Journaling Flash File System (JFFS2)

The Root File System and User directory in the flash memory is formatted with the **Journaling Flash File System (JFFS2)**. The formatting process places a compressed file system in the flash memory. This operation is transparent to the user.

The Journaling Flash File System (JFFS2), which was developed by Axis Communications in Sweden, puts a file system directly on the flash, instead of emulating a block device. It is designed for use on flash-ROM chips and recognizes the special write requirements of a flash-ROM chip. JFFS2 implements wear-leveling to extend the life of the flash disk, and stores the flash directory structure in the RAM. A log-structured file system is maintained at all times. The system is always consistent, even if it encounters crashes or improper power-downs, and does not require *fsck* (file system check) on boot-up.

JFFS2 is the newest version of JFFS. It provides improved wear-leveling and garbage-collection performance, improved RAM footprint and response to system-memory pressure, improved concurrency and support for suspending flash erases, marking of bad sectors with continued use of the remaining good sectors (enhancing the write-life of the devices), native data compression inside the file system design, and support for hard links.

The key features of JFFS2 are:

- Targets the Flash ROM Directly
- Robustness
- Consistency across power failures
- No integrity scan (fsck) is required at boot time after normal or abnormal shutdown
- Explicit wear leveling
- Transparent compression

Although JFFS2 is a journaling file system, this does not preclude the loss of data. The file system will remain in a consistent state across power failures and will always be mountable. However, if the board is powered down during a write then the incomplete write will be rolled back on the next boot, but writes that have already been completed will not be affected.

**Additional information about JFFS2 is available at:**

<http://sources.redhat.com/jffs2/jffs2.pdf>

<http://developer.axis.com/software/jffs/>

<http://www.linux-mtd.infradead.org/>

## Software Package

<b>Boot Loader</b>	Moxa private
<b>Kernel</b>	Linux 2.6.23
<b>Protocol Stack</b>	ARP, PPP, CHAP, PAP, IPv4, ICMP, TCP, UDP, DHCP, FTP, SNMP V1, HTTP, NTP, NFS, SMTP, SSH 1.0/2.0, SSL, Telnet, PPPoE, OpenVPN
<b>File System</b>	JFFS2, NFS, Ext2, Ext3, VFAT/FAT
<b>OS shell command</b>	bash
Busybox	Linux normal command utility collection
<b>Utilities</b>	
tinylogin	login and user manager utility
telnet	telnet client program
ftp	FTP client program
smtpclient	email utility
scp	Secure file transfer Client Program
<b>Daemons</b>	
pppd	dial in/out over serial port daemon
snmpd	snmpd agent daemon
telnetd	telnet server daemon
inetd	TCP server manager program
ftpd	ftp server daemon
apache	web server daemon
sshd	secure shell server
openvpn	virtual private network
openssl	open SSL
<b>Linux Tool Chain</b>	
Gcc (V 4.2.1)	C/C++ PC Cross Compiler
GDB (V5.3)	Source Level Debug Server
Glibc (V2.2.5)	POSIX standard C library



# 2

## Getting Started

---

In this chapter, we explain how to connect the IA260/IA261/IA262/EM-2260, how to turn on the power, how to get started programming, and how to use the IA260/IA261/IA262/EM-2260's other functions.

The following topics are covered in this chapter:

- ❑ **Powering on the IA260/IA261/IA262/EM-2260**
- ❑ **Connecting the IA260/IA261/IA262/EM-2260 to a PC**
  - Debug Port
  - Telnet Console
  - SSH Console
  - VGA Console
- ❑ **Configuring the Ethernet Interface**
  - Modifying Network Settings
  - Modifying Network Settings by Command
- ❑ **CF Socket for Storage Expansion**
- ❑ **Test Program—Developing Hello.c**
  - Installing the Tool Chain (Linux)
  - Checking the Flash Memory Space
  - Compiling Hello.c
  - Uploading and Running the “Hello” Program

## Powering on the IA260/IA261/IA262/EM-2260

Connect the SG wire to the shielded contact located in the upper left corner of the IA260/IA261/IA262/EM-2260, and then power on the computer by connecting it to the power adaptor. It takes about 60 seconds for the system to boot up. Once the system is ready, the Ready LED will light up.

**NOTE** After connecting the IA260/IA261/IA262/EM-2260 to the power supply, it will take about 60 seconds for the operating system to boot up. The green Ready LED will not turn on until the operating system is ready.



### ATTENTION

This product is intended to be supplied by a Listed Power Unit and output marked with "LPS" and rated 12-48 VDC, 580 mA (minimum requirements).

## Connecting the IA260/IA261/IA262/EM-2260 to a PC

There are two ways to connect the IA260/IA261/IA262/EM-2260 to a PC: through the serial console port or via Telnet over the network. Or, you can use a monitor connected to the VGA output of the IA260/IA261/IA262/EM-2260 to connect directly to the computer.

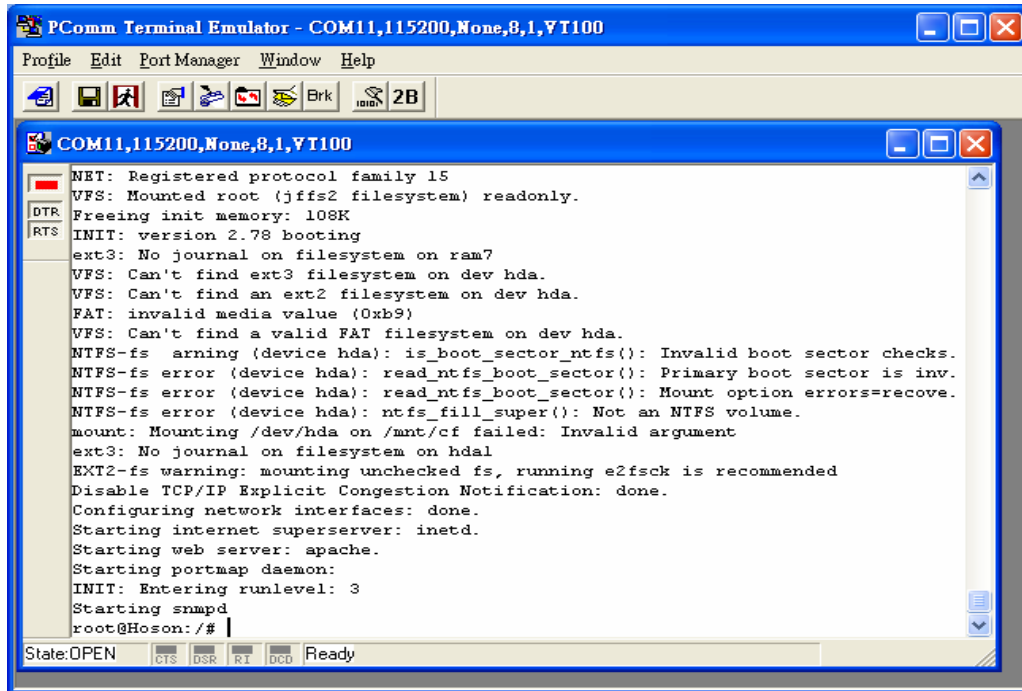
### Debug Port

The debug port gives users a convenient way of connecting to IA260/IA261/IA262/EM-2260. This method is particularly useful when using the computer for the first time. Debug port is useful for connecting IA260/IA261/IA262/EM-2260, so you do not need to know either of its two IP addresses in order to connect to the debug port.

Use the debug port settings shown below.

<b>Baudrate</b>	115200 bps
<b>Parity</b>	None
<b>Data bits</b>	8
<b>Stop bits:</b>	1
<b>Flow Control</b>	None
<b>Terminal</b>	VT100

Once the connection is established, the following window will open.



### Telnet Console

If you know at least one of the two IP addresses and netmasks, then you can use Telnet to connect to the IA260/IA261/IA262/EM-2260's console utility. The default IP address and Netmask for each of the two ports are given below:

	Default IP Address	Netmask
LAN 1	192.168.3.127	255.255.255.0
LAN 2	192.168.4.127	255.255.255.0

Use a cross-over Ethernet cable to connect directly from your PC to the IA260/IA261/IA262/EM-2260. You should first modify your PC's IP address and netmask so that your PC is on the same subnet as one of IA260/IA261/IA262/EM-2260's two LAN ports. For example, if you connect to LAN 1, you can set your PC's IP address to 192.168.3.126 and netmask to 255.255.255.0. If you connect to LAN 2, you can set your PC's IP address to 192.168.4.126 and netmask to 255.255.255.0.

To connect to a hub or switch connected to your local LAN, use a straight-through Ethernet cable. The default IP addresses and netmasks are shown above. To log in, type the Login name and password as requested. The default values are both **root**:

**Login:** root  
**Password:** root



You can proceed with configuring the network settings of the target computer when you reach the bash command shell. Configuration instructions are given in the next section.



**ATTENTION**

**Debug Port Reminder**

Remember to choose VT100 as the terminal type. Use the cable CBL-4PINDB9F-100, which comes with the IA260/IA261/IA262/EM-2260, to connect to the serial console port.

**Telnet Reminder**

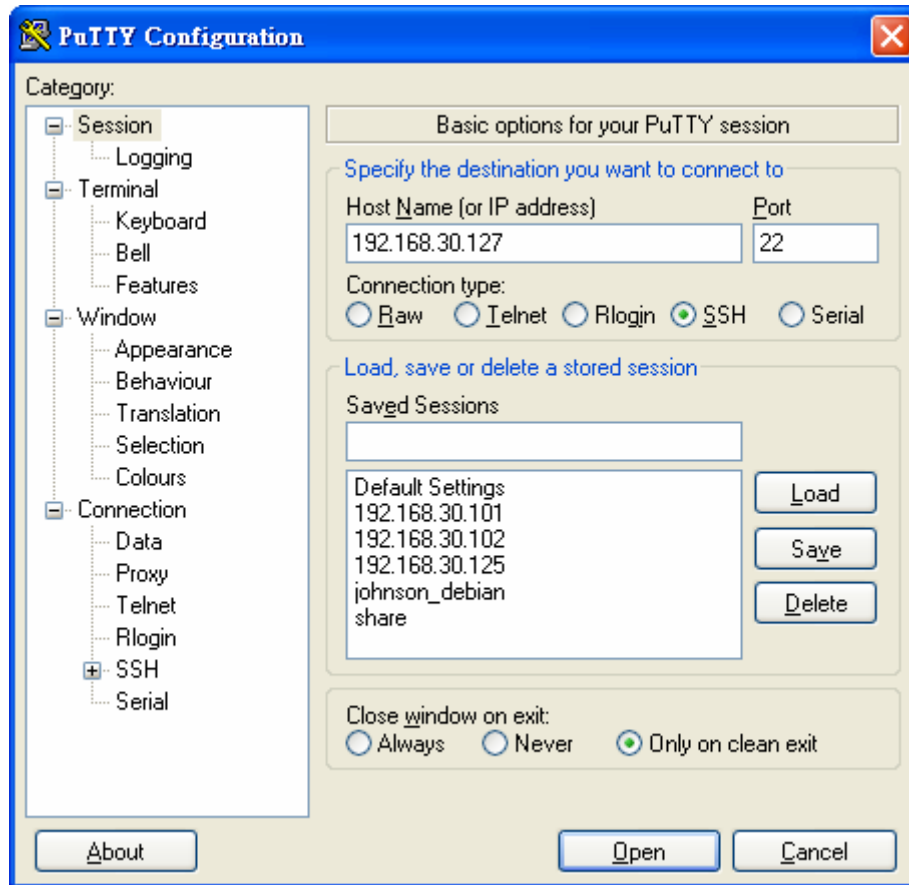
When connecting to the IA260/IA261/IA262/EM-2260 over a LAN, you must configure your PC's Ethernet IP address to be on the same subnet as the IA260/IA261/IA262/EM-2260 that you wish to contact. If you do not get connected on the first try, re-check the IP settings, and then unplug and re-plug the IA260/IA261/IA262/EM-2260's power cord.

**SSH Console**

The IA260/IA261/IA262/EM-2260 supports an SSH Console to provide users with better security options.

**Windows Users**

Click on the link <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html> to download PuTTY (free software) to set up an SSH console for the IA260/IA261/IA262/EM-2260 in a Windows environment. The following figure shows a simple example of the configuration that is required.



## Linux Users

From a Linux machine, use the “ssh” command to access the IA260/IA261/IA262/EM-2260’s console utility via SSH.

```
#ssh 192.168.3.127
```

Select **yes** to complete the connection.

```
[root@bee_notebook root]# ssh 192.168.3.127
The authenticity of host '192.168.3.127 (192.168.3.127)' can't be established.
RSA key fingerprint is 8b:ee:ff:84:41:25:fc:cd:2a:f2:92:8f:cb:1f:6b:2f.
Are you sure you want to continue connection (yes/no)? yes_
```

**NOTE** SSH provides better security compared to Telnet for accessing the IA260/IA261/IA262/EM-2260’s console utility over the network.

## VGA Console

You can connect VGA port to a LCD monitor with VGA interface. Then you connect a keyboard with USB interface to USB port. You will see a login request on the LCD monitor. To log in, type the Login name and password as requested. The default values are both **root**:

**Login:** root  
**Password:** root

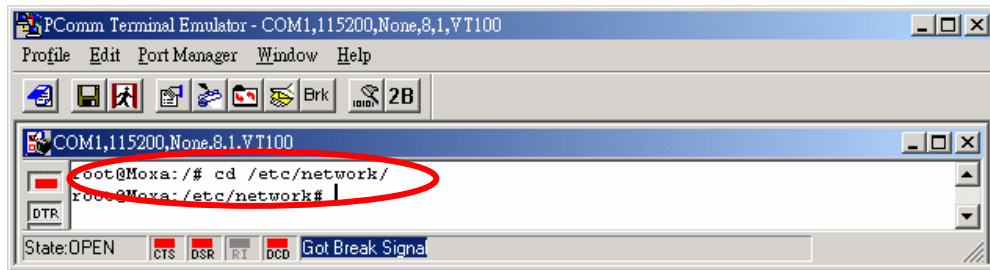
## Configuring the Ethernet Interface

The network settings of the IA260/IA261/IA262/EM-2260 can be modified with the Debug Port, or online over the network.

### Modifying Network Settings

In this section, we use the serial console to configure the network settings of the target computer.

1. Follow the instructions given in a previous section to access the Console Utility of the target computer via the serial console port, and then type **#cd /etc/network** to change directories.



2. Type **#vi interfaces** to edit the network configuration file with vi editor. You can configure the Ethernet ports of the IA260/IA261/IA262/EM-2260 for **static** or **dynamic** (DHCP) IP addresses.

#### Static IP addresses:

As shown below, 4 network addresses must be modified: **address**, **network**, **netmask**, and **broadcast**. The default IP addresses are 192.168.3.127 for LAN1 with default netmask of 255.255.255.0.

#### Dynamic IP addresses:

By default, the IA260/IA261/IA262/EM-2260 is configured for “static” IP addresses. To configure one or both LAN ports to request an IP address dynamically, replace **static** with **dhcp** and then delete the address, network, netmask, and broadcast lines.

Default Setting for LAN1	Dynamic Setting using DHCP
<pre>iface eth0 inet static address 192.168.3.127 network: 192.168.3.0 netmask 255.255.255.0 broadcast 192.168.3.255</pre>	<pre>iface eth0 inet dhcp</pre>

- After the boot settings of the LAN interface have been modified, issue the following command to activate the LAN settings immediately:

```
#/etc/init.d/networking restart
```

**NOTE** After changing the IP settings, use the **networking restart** command to activate the new IP address.

## Modifying Network Settings by Command

IP settings can be activated over the command, but the new settings will not be saved to the flash ROM without modifying the file `/etc/network/interfaces`.

For example, type the command `#ifconfig eth0 192.168.1.1` to change the IP address of LAN1 to 192.168.1.1.

```
root@moxa:~# ifconfig eth0 192.168.1.1
root@moxa:~# _
```

## CF Socket for Storage Expansion

The IA260/IA261/IA262 provides a CF socket for storage expansion. Moxa provides a CF flash disk for expansion that allows users to plug in a Compact Flash (CF) memory card of additional memory space. The CF socket is located on the front panel of the IA260/IA261/IA262. To install a CF card, you must first power off, and then plug the CF card directly into the socket.

The CF card will be mounted at `/mnt/cf`.

## Test Program—Developing Hello.c

In this section, we use the standard “Hello” programming example to illustrate how to develop a program for the IA260/IA261/IA262/EM-2260. In general, program development involves the following seven steps.

### Step 1:

Connect the IA260/IA261/IA262/EM-2260 to a Linux PC.

### Step 2:

Install Tool Chain (GNU Cross Compiler & glibc).

### Step 3:

Set the cross compiler and glibc environment variables.

### Step 4:

Code and compile the program.

### Step 5:

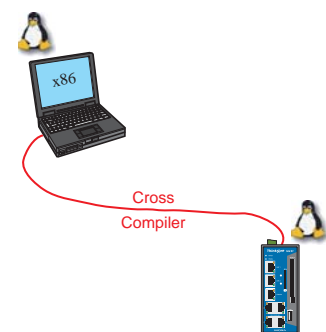
Download the program to the IA260/IA261/IA262/EM-2260  
Via FTP or NFS.

### Step 6:

Debug the program  
→ If bugs are found, return to Step 4.  
→ If no bugs are found, continue with Step 7.

### Step 7:

Back up the user directory (distribute the program to additional  
IA260/IA261/IA262/EM-2260 units if needed).



## Installing the Tool Chain (Linux)

The Linux Operating System must be pre-installed in the PC before installing the IA260/IA261/IA262/EM-2260 GNU Tool Chain. Fedora core or compatible versions are recommended. The Tool Chain requires approximately 1 GB of hard disk space on your PC. The IA260/IA261/IA262/EM-2260 Tool Chain software is located on the IA260/IA261/IA262/EM-2260 CD. To install the Tool Chain, insert the CD into your PC and then issue the following commands:

```
#mount /dev/cdrom /mnt/cdrom
#sh /mnt/cdrom/tool-chain/linux/arm-linux_x.x.sh (where x.x indicates the version of
the Tool Chain)
```

The Tool Chain will be installed automatically on your Linux PC within a few minutes. Before compiling the program, be sure to set the following path first, since the Tool Chain files, including the compiler, link, and library, are located in this directory.

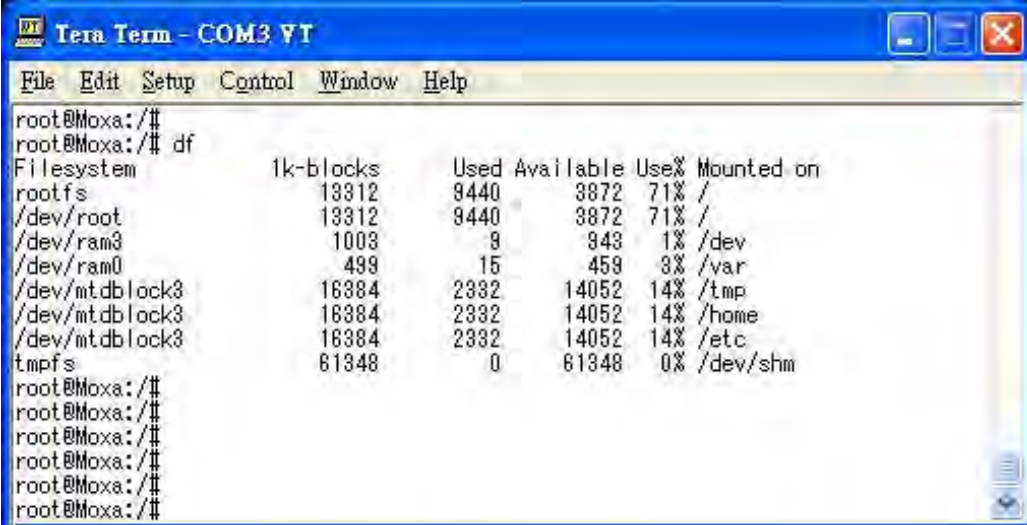
```
PATH=/usr/local/arm-linux/bin:$PATH
```

Setting the path allows you to run the compiler from any directory.

## Checking the Flash Memory Space

If the flash memory is full, you will not be able to save data to the Flash ROM. Use the following command to calculate the amount of "Available" flash memory:

```
/>df -h
```



```
root@Moxa:/#
root@Moxa:/# df
Filesystem          1k-blocks    Used Available Use% Mounted on
rootfs              13312       9440      3872   71% /
/dev/root           13312       9440      3872   71% /
/dev/ram3            1003         9         943    1% /dev
/dev/ram0            499          15         459    3% /var
/dev/mtdblock3     16384       2332     14052   14% /tmp
/dev/mtdblock3     16384       2332     14052   14% /home
/dev/mtdblock3     16384       2332     14052   14% /etc
tmpfs               61348         0      61348    0% /dev/shm
root@Moxa:/#
root@Moxa:/#
root@Moxa:/#
root@Moxa:/#
root@Moxa:/#
root@Moxa:/#
```

If there isn't enough "Available" space for your application, you will need to delete some existing files. To do this, connect your PC to the IA260/IA261/IA262/EM-2260 with the console cable, and then use the console utility to delete the files from the IA260/IA261/IA262/EM-2260's flash memory. To check the amount of free space available, look at the directories in the read/write directory **/dev/mtdblock3**. Note that the directories **/home** and **/etc** are both mounted on the directory **/dev/mtdblock3**.

**NOTE** If the flash memory is full, you will need to free up some memory space before saving files to the Flash ROM.



## Compiling Hello.c

The package CD contains several example programs. Here we use **hello.c** as an example to show you how to compile and run your applications. Type the following commands from your PC to copy the files used for this example from the CD to your computer's hard drive:

```
# cd /tmp/
# mkdir example
# cp -r /mnt/cdrom/example/* /tmp/example
```

To compile the program, go to the **hello** subdirectory and issue the following commands:

```
#cd example/hello
#make
```

You should receive the following response:

```
[root@localhost hello]# make
/usr/local/arm-linux/bin/arm-linux-gcc -o hello-release hello.c
/usr/local/arm-linux/bin/arm-linux-strip -s hello-release
/usr/local/arm-linux/bin/arm-linux-gcc -ggdb -o hello-debug hello.c
[root@localhost hello]# _
```

Both **hello-release** and **hello-debug** will be generated, which are described below:

**hello-release**—an ARM platform execution file (created specifically to run on the IA260/IA261/IA262/EM-2260)

**hello-debug**—an ARM platform GDB debug server execution file (see Chapter 5 for details about the GDB debug tool)

NOTE	Since Moxa's tool chain places a specially designed <b>Makefile</b> in the directory <b>/tmp/example/hello</b> , be sure to type the <b>#make</b> command from within that directory. This special Makefile uses the arm-linux-gcc compiler to compile the hello.c source code for the ARM environment. If you type the <b>#make</b> command from within any other directory, Linux will use the x86 compiler (for example, cc or gcc).
------	---

## Uploading and Running the "Hello" Program

Use the following commands to upload **hello-release** to the IA260/IA261/IA262/EM-2260 via FTP.

1. From the PC, type:

```
#ftp 192.168.3.127
```

2. Use the bin command to set the transfer mode to Binary mode, and then use the put command to initiate the file transfer:

```
ftp> bin
ftp> put hello-release
```

3. From the IA260/IA261/IA262/EM-2260, type:

```
# chmod +x hello-release
# ./hello-release
```

The word **Hello** will be printed on the screen.

```
root@Moxa:~# ./hello-release
Hello
```

## Managing Embedded Linux

---

This chapter includes information about version control, deployment, updates, and peripherals. The information in this chapter will be particularly useful when you need to run the same application on several IA260/IA261/IA262/EM-2260 units.

The following topics are covered in this chapter:

- ❑ **System Version Information**
  - Upgrading the Firmware
  - Loading Factory Defaults
- ❑ **Enabling and Disabling Daemons**
- ❑ **Setting the Run-Level**
- ❑ **Adjusting the System Time**
  - Setting the Time Manually
  - NTP Client
  - Updating the Time Automatically
- ❑ **Cron—Daemon to Execute Scheduled Commands**

## System Version Information

To determine the hardware capability of your IA260/IA261/IA262/EM-2260, and what kind of software functions are supported, check the version numbers of your IA260/IA261/IA262/EM-2260's kernel, and user file system. Contact Moxa to determine the hardware version. You will need the **Production S/N** (Serial number), which is located on the IA260/IA261/IA262/EM-2260's bottom label.

To check the kernel version, type:

```
#kversion
```

```
192.168.3.127 - PuTTY
root@moxa:~# kversion
Version 1.0
root@moxa:~# █
```

## Upgrading the Firmware

The IA260/IA261/IA262/EM-2260's bios, kernel, and root file system are combined into one firmware file, which can be downloaded from Moxa's website ([www.moxa.com](http://www.moxa.com)). The name of the file has the form `ia260/ia261/ia262/em2260-x.x.x.hfm`, with "x.x.x" indicating the firmware version. To upgrade the firmware, download the firmware file to a PC, and then transfer the file to the IA260/IA261/IA262/EM-2260 via a Debug Port or Telnet Console connection.



### ATTENTION

#### **Upgrading the firmware will erase some data on the Flash ROM**

If you are using the **ramdisk** to store code for your applications, beware that updating the firmware will erase some of the data on the Flash ROM. We strongly suggest that you should back up your application files and data before updating the firmware.

Since different Flash disks have different sizes, it's a good idea to check the size of your Flash disk before upgrading the firmware, or before using the disk to store your application and data files. Use the `#df -h` command to list the size of each memory block and how much free space is available in each block.

```

192.168.3.127 - PuTTY
root@Moxa:~# df -h
Filesystem      Size  Used Available Use% Mounted on
/dev/mtdblock2 8.0M   6.0M   2.0M   75% /
/dev/ram0       499.0k 16.0k  458.0k   3% /var
/dev/mtdblock3 6.0M   488.0k 5.5M    8% /tmp
/dev/mtdblock3 6.0M   488.0k 5.5M    8% /home
/dev/mtdblock3 6.0M   488.0k 5.5M    8% /etc
tmpfs          30.4M   0      30.4M   0% /dev/shm
root@Moxa:~# upramdisk
root@Moxa:~# df -h
Filesystem      Size  Used Available Use% Mounted on
/dev/mtdblock2 8.0M   6.0M   2.0M   75% /
/dev/ram0       499.0k 16.0k  458.0k   3% /var
/dev/mtdblock3 6.0M   488.0k 5.5M    8% /tmp
/dev/mtdblock3 6.0M   488.0k 5.5M    8% /home
/dev/mtdblock3 6.0M   488.0k 5.5M    8% /etc
tmpfs          30.4M   0      30.4M   0% /dev/shm
/dev/ram1      16.0M   1.0k  15.1M   0% /mnt/ramdisk
root@Moxa:~# cd /mnt/ramdisk
root@Moxa:/mnt/ramdisk#

```

The following instructions give the steps required to save the firmware file to the IA260/IA261/IA262/EM-2260's RAM disk and how to upgrade the firmware.

1. Type the following commands to enable the RAM disk:

```
#upramdisk
#cd /mnt/ramdisk
```

2. Type the following commands to use the IA260/IA261/IA262/EM-2260's built-in FTP client to transfer the firmware file (`ia260/ia261/ia262/em2260-x.x.x.hfm`) from the PC to the IA260/IA261/IA262/EM-2260:

```
/mnt/ramdisk> ftp <destination PC's IP>
Login Name: xxxx
Login Password: xxxx
ftp> bin
ftp> get ia260/ia261/ia262/em2260-x.x.x.hfm
```

```

192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# ftp 192.168.3.193
Connected to 192.168.3.193 (192.168.3.193).
220 TYPSoft FTP Server 1.10 ready...
Name (192.168.3.193:root): root
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd newsw
250 CWD command successful. "/C:/ftproot/newsw/" is current directory.
ftp> bin
200 Type set to I.
ftp> ls
200 Port command successful.
150 Opening data connection for directory list.
drw-rw-rw-  1 ftp ftp    0 Nov 30 10:03 .
drw-rw-rw-  1 ftp ftp    0 Nov 30 10:03 .
-rw-rw-rw-  1 ftp ftp  13167772 Nov 29 10:24
ia260/ia261/ia262/em2260-x.x.x.hfm
226 Transfer complete.
ftp> get ia260/ia261/ia262/em2260-x.x.x.hfm
local: ia246/ia261/ia262/em2260-x.x.x.hfm
remote: ia260/ia261/ia262/em2260-x.x.x.hfm
200 Port command successful.
150 Opening data connection for ia260/ia261/ia262/em2260-x.x.x.hfm
226 Transfer complete.
13167772 bytes received in 2.17 secs (5925.8 kB/s)
ftp>

```

3. Next, use the **upgradefm** command to upgrade the kernel and root file system:

```
#upgradefm ia260/ia261/ia262/em2260-x.x.x.hfm
```

```

192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# upgradefm ia260/ia261/ia262/em2260-x.x.x.hfm
Moxa IA26x upgrade firmware utility version 1.0.
To check source firmware file context.
The source firmware file context is OK.
This step will destroy all your firmware.
Continue ? (Y/N) : Y
Now upgrade the file [kernel].
Format MTD device [/dev/mtd1] . . .
MTD device [/dev/mtd1] erase 128 Kibyte @ 1C0000 - 100% complete.
Wait to write file . . .
Compleleted 100%
Now upgrade the file [usrdisk].
Format MTD device [/dev/mtd2] . . .
MTD device [/dev/mtd2] erase 128 Kibyte @ 800000 - 100% complete.
Wait to write file . . .
Compleleted 100%
Upgrade the firmware is OK.

```



#### ATTENTION

The **upgradefm** utility will reboot your target after the upgrade is OK.

## Loading Factory Defaults

To load the factory default settings, you must press the reset-to-default button for more than 5 seconds. All files in the /home & /etc directories will be deleted. Note that while pressing the reset-to-default button, the Ready LED will blink three times, and then turn off. It takes about one minute for the factory defaults to fully load. For users who have logged onto the computer, please note that they will be forcibly logged out when the factory defaults are loaded and the systems start to reboot.

## Enabling and Disabling Daemons

The following daemons are enabled when the IA260/IA261/IA262/EM-2260 boots up for the first time.

**snmpd** .....SNMP Agent daemon  
**telnetd** .....Telnet Server daemon and Client  
**inetd** .....Internet Daemons  
**ftpd**.....FTP Server daemon and Client  
**sshd** .....Secure Shell Server daemon  
**httpd** .....Apache WWW Server daemon

Type the command “ps ” to list all processes currently running.

```

192.168.3.127 - PuTTY
root@Moxa:~# cd /etc
root@Moxa:/etc# ps
  PID  Uid      VmSize  Stat  Command
    1  root          532  S    init [3]
    2  root          SWN   [ksoftirqd/0]
    3  root          SW<   [events/0]
    4  root          SW<   [khelper]
   13  root          SW<   [kblockd/0]
   14  root          SW    [khubd]
   24  root          SW    [pdflush]
   25  root          SW    [pdflush]
   27  root          SW<   [aio/0]
   26  root          SW    [kswapd0]
  604  root          SW    [mtdblockd]
  609  root          SW    [pccardd]
  611  root          SW    [pccardd]
  625  root          SWN   [jffs2_gcd_mtd3]
  673  root          500  S    /bin/inetd
  679  root         3004  S    /usr/bin/httpd -k start -d /etc/apache
  682  bin           380  S    /bin/portmap
  685  root         1176  S    /bin/sh --login
  690  root          464  S    /bin/snmpd
  694  nobody       3012  S    /usr/bin/httpd -k start -d /etc/apache
  695  nobody       3012  S    /usr/bin/httpd -k start -d /etc/apache
  696  nobody       3012  S    /usr/bin/httpd -k start -d /etc/apache
  697  nobody       3012  S    /usr/bin/httpd -k start -d /etc/apache
  698  nobody       3012  S    /usr/bin/httpd -k start -d /etc/apache
  701  root          352  S    /bin/reportip
  714  root         1176  S    -bash
  726  root          436  S    /bin/telnetd
  727  root         1180  S    -bash
  783  root          628  R    ps
root@Moxa:/ect#

```

To run a private daemon, you can edit the file `rc.local`, as follows:

```
#cd /etc/rc.d
#vi rc.local
```

```
192.168.3.127 - PuTTY
root@Moxa:~# cd /etc/rc.d
root@Moxa:/etc/rc.d# vi rc.local
```

Next, use `vi` to open your application script. We use the example program `tcps2-release`, and put it to run in the background.

```
192.168.3.127 - PuTTY
# !/bin/sh
# Add you want to run daemon
/home/tcps2-release &
```

The enabled daemons will be available after you reboot the system.

```
192.168.3.127 - PuTTY
root@Moxa:~# ps
  PID  Uid    VmSize  Stat  Command
    1  root      532  S    init [3]
    2  root          SWN  [ksoftirqd/0]
    3  root          SW<  [events/0]
    4  root          SW<  [khelper]
   13  root          SW<  [kblockd/0]
   14  root          SW   [khubd]
   24  root          SW   [pdflush]
   25  root          SW   [pdflush]
   27  root          SW<  [aio/0]
   26  root          SW   [kswapd0]
  604  root          SW   [mtdblockd]
  609  root          SW   [pccardd]
  611  root          SW   [pccardd]
  625  root          SWN  [jffs2_gcd_mtd3]
  673  root      500  S    /bin/inetd
  674  root     1264  S    /root/tcps2-release
  679  root     3004  S    /usr/bin/httpd -k start -d /etc/apache
  682  bin       380  S    /bin/portmap
  685  root     1176  S    /bin/sh --login
  690  root      464  S    /bin/snmpd
  694  nobody   3012  S    /usr/bin/httpd -k start -d /etc/apache
  695  nobody   3012  S    /usr/bin/httpd -k start -d /etc/apache
  696  nobody   3012  S    /usr/bin/httpd -k start -d /etc/apache
  697  nobody   3012  S    /usr/bin/httpd -k start -d /etc/apache
  698  nobody   3012  S    /usr/bin/httpd -k start -d /etc/apache
  701  root      352  S    /bin/reportip
  714  root     1176  S    -bash
  726  root      436  S    /bin/telnetd
  727  root     1180  S    -bash
  783  root      628  R    ps
root@Moxa:~#
```

## Setting the Run-Level

In this section, we outline the steps you should take to set the Linux run-level and execute requests. Use the following command to enable or disable settings:

```
192.168.3.127 - PuTTY
root@Moxa:/etc/rc.d/rc3.d# ls
S19nfs-common S25nfs-user-serverS99showreadyled
S20snmpd      S55ssh
S24pcmcia    S99rmnologin
root@Moxa:/etc/rc.d/rc3.d# █
```

```
#cd /etc/rc.d/init.d
```

Edit a shell script to execute `/root/tcps2-release` and save to `tcps2` as an example.

```
#cd /etc/rc.d/rc3.d
#ln -s /etc/rc.d/init.d/tcps2 S60tcps2
```

SxxRUNFILE stands for

S: start the run file while linux boots up.

xx: a number between 00-99. Smaller numbers have a higher priority.

RUNFILE: the file name.

```
192.168.3.127 - PuTTY
root@Moxa:/etc/rc.d/rc3.d# ls
S19nfs-common S25nfs-user-serverS99showreadyled
S20snmpd      S55ssh
S24pcmcia    S99rmnologin
root@Moxa:/etc/rc.d/rc3.d# ln -s /root/tcps2-release S60tcps2
root@Moxa:/etc/rc.d/rc3.d# ls
S19nfs-common S25nfs-user-serverS99rmnologin
S20snmpd      S55ssh        S99showreadyled
S24pcmcia    S60tcps2
root@Moxa:/etc/rc.d/rc3.d# █
```

KxxRUNFILE stands for

K: start the run file while linux shuts down or halts.

xx: a number between 00-99. Smaller numbers have a higher priority.

RUNFILE: the file name.

To remove the daemon, remove the run file from the `/etc/rc.d/rc3.d` directory by using the following command:

```
#rm -f /etc/rc.d/rc3.d/S60tcps2
```



## Adjusting the System Time

### Setting the Time Manually

The IA260/IA261/IA262/EM-2260 has two time settings. One is the system time, and the other is the RTC (Real Time Clock) time kept by the IA260/IA261/IA262/EM-2260's hardware. Use the `#date` command to query the current system time or set a new system time. Use `#hwclock` to query the current RTC time or set a new RTC time.

Use the following command to query the system time:

```
#date
```

Use the following command to query the RTC time:

```
#hwclock
```

Use the following command to set the system time:

```
#date MMDDhhmmYYYY
```

MM = Month

DD = Date

hhmm = hour and minute

YYYY = Year

Use the following command to set the RTC time:

```
#hwclock -w
```

The following figure illustrates how to update the system time and set the RTC time.

```
192.168.3.127 - PuTTY
root@Moxa:~# date
Fri Jun 23 23:30:31 CST 2000
root@Moxa:~# hwclock
Fri Jun 23 23:30:35 2000 -0.557748 seconds
root@Moxa:~# date 120910002004
Thu Dec 9 10:00:00 CST 2004
root@Moxa:~# hwclock -w
root@Moxa:~# date ; hwclock
Thu Dec 9 10:01:07 CST 2004
Thu Dec 9 10:01:08 2004 -0.933547 seconds
root@Moxa:~# █
```

## NTP Client

The IA260/IA261/IA262/EM-2260 has a built-in NTP (Network Time Protocol) client that is used to initialize a time request to a remote NTP server. Use `#ntpdate <server name>` to update the system time.

```
#ntpdate time.stdtime.gov.tw
#hwclock -w
```

Visit <http://www.ntp.org> for more information about NTP and NTP server addresses.

```
10.120.53.100 - PuTTY
root@Moxa:~# date ; hwclock
Sat Jan 1 00:00:36 CST 2000
Sat Jan 1 00:00:37 2000 -0.772941 seconds
root@Moxa:~# ntpdate time.stdtime.gov.tw
 9 Dec 10:58:53 ntpdate[207]: step time server 220.130.158.52 offset 155905087.9
84256 sec
root@Moxa:~# hwclock -w
root@Moxa:~# date ; hwclock
Thu Dec 9 10:59:11 CST 2004
Thu Dec 9 10:59:12 2004 -0.844076 seconds
root@Moxa:~# █
```

**NOTE** Before using the NTP client utility, check your IP and DNS settings to make sure that an Internet connection is available. Refer to Chapter 2 for instructions on how to configure the Ethernet interface, and see Chapter 4 for DNS setting information.

## Updating the Time Automatically

In this subsection, we show how to use a shell script to update the time automatically.

### Example shell script to update the system time periodically

```
#!/bin/sh
ntpdate time.nist.gov # You can use the time server's ip address or domain
# name directly. If you use domain name, you must
# enable the domain client on the system by updating
# /etc/resolv.conf file.
hwclock --systohc
sleep 100 # Updates every 100 seconds. The min. time is 100 seconds. Change
# 100 to a larger number to update RTC less often.
```

Save the shell script using any file name. E.g., `fixtime`

### How to run the shell script automatically when the kernel boots up

Copy the example shell script `fixtime` to directory `/etc/init.d`, and then use `chmod 755 fixtime` to change the shell script mode. Next, use vi editor to edit the file `/etc/inittab`. Add the following line to the bottom of the file:

```
ntp : 2345 : respawn : /etc/init.d/fixtime
```

Use the command `#init q` to re-init the kernel.

## Cron—Daemon to Execute Scheduled Commands

Start Cron from the directory `/etc/rc.d/rc.local`. It will return immediately, so you don't need to start it with `'&'` to run in the background.

The Cron daemon will search `/etc/cron.d/crontab` for crontab files, which are named after accounts in `/etc/passwd`.

Cron wakes up every minute, and checks each command to see if it should be run in the current minute. Modify the file `/etc/cron.d/crontab` to set up your scheduled applications. Crontab files have the following format:

min	hour	date	month	week	user	command
0-59	0-23	1-31	1-12	0-6 (0 is Sunday)		

The following example demonstrates how to use Cron.

### How to use cron to update the system time and RTC time every day at 8:00

#### STEP1: Write a shell script named `fixtime.sh` and save it to `/home/`.

```
#!/bin/sh
ntpdate time.nist.gov
hwclock --systohc
exit 0
```

#### STEP2: Change mode of `fixtime.sh`

```
#chmod 755 fixtime.sh
```

#### STEP3: Modify `/etc/cron.d/crontab` file to run `fixtime.sh` at 8:00 every day.

Add the following line to the end of the crontab:

```
* 8 * * * root /home/fixtime.sh
```

#### STEP4: Enable the cron daemon manually.

```
#/etc/init.d/cron start
```

#### STEP5: Enable cron when the system boots up.

Add the following line in the file `/etc/rc.d/rc.local`

```
#/etc/init.d/cron start
```

## Managing Communications

---

In this chapter, we explain how to configure the IA260/IA261/IA262/EM-2260's various communication functions.

The following topics are covered in this chapter:

- Telnet / FTP**
- DNS**
- Web Service—Apache**
- Install PHP for Apache Web Server**
- IPTABLES**
- NAT**
  - NAT Example
  - Enabling NAT at Bootup
- Dial-up Service—PPP**
- PPPoE**
- NFS (Network File System)**
  - Setting up the IA260/IA261/IA262/EM-2260 as an NFS Client
- Mail**
- SNMP**
- OpenVPN**

## Telnet / FTP

In addition to supporting Telnet client/server and FTP client/server, the IA260/IA261/IA262/EM-2260 also supports SSH and sftp client/server. To enable or disable the Telnet/ftp server, you first need to edit the file `/etc/inetd.conf`.

### Enabling the Telnet/ftp server

The following example shows the default content of the file `/etc/inetd.conf`. The default is to enable the Telnet/ftp server:

```
discard dgram udp wait root /bin/discard
discard stream tcp nowait root /bin/discard
telnet stream tcp nowait root /bin/telnetd
ftp stream tcp nowait root /bin/ftpd -l
```

### Disabling the Telnet/ftp server

Disable the daemon by typing '#' in front of the first character of the row to comment out the line.

## DNS

The IA260/IA261/IA262/EM-2260 supports DNS client (but not DNS server). To set up DNS client, you need to edit three configuration files: `/etc/hosts`, `/etc/resolv.conf`, and `/etc/nsswitch.conf`.

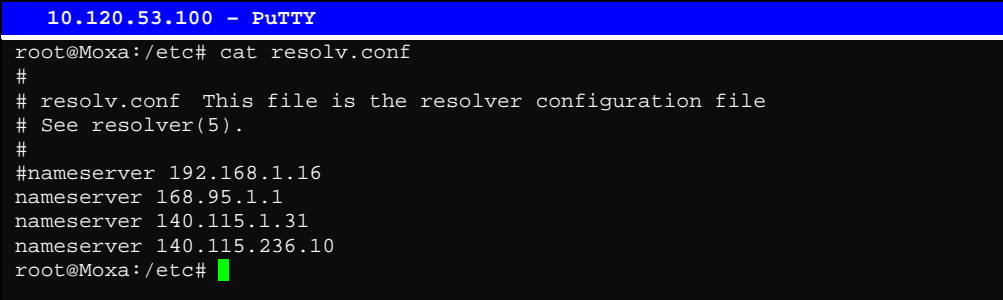
### `/etc/hosts`

This is the first file that the Linux system reads to resolve the host name and IP address.

### `/etc/resolv.conf`

This is the most important file that you need to edit when using DNS for the other programs. For example, before you use `#ntpdate time.nist.gov` to update the system time, you will need to add the DNS server address to the file. Ask your network administrator which DNS server address you should use. The DNS server's IP address is specified with the "nameserver" command. For example, add the following line to `/etc/resolv.conf` if the DNS server's IP address is 168.95.1.1:

```
nameserver 168.95.1.1
```



```
10.120.53.100 - PuTTY
root@Moxa:/etc# cat resolv.conf
#
# resolv.conf This file is the resolver configuration file
# See resolver(5).
#
#nameserver 192.168.1.16
nameserver 168.95.1.1
nameserver 140.115.1.31
nameserver 140.115.236.10
root@Moxa:/etc#
```

### `/etc/nsswitch.conf`

This file defines the sequence to resolve the IP address by using `/etc/hosts` file or `/etc/resolv.conf`.

## Web Service—Apache

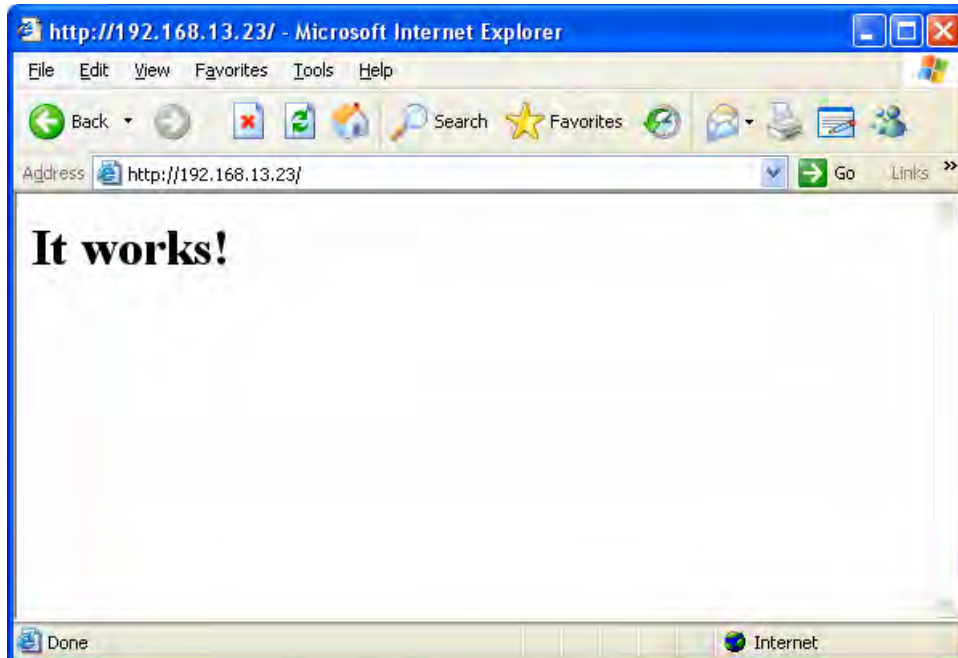
The Apache web server's main configuration file is `/etc/apache/conf/httpd.conf`, with the default homepage located at `/home/httpd/htdocs/index.html`. Save your own homepage to the following directory:

`/home/httpd/htdocs/`

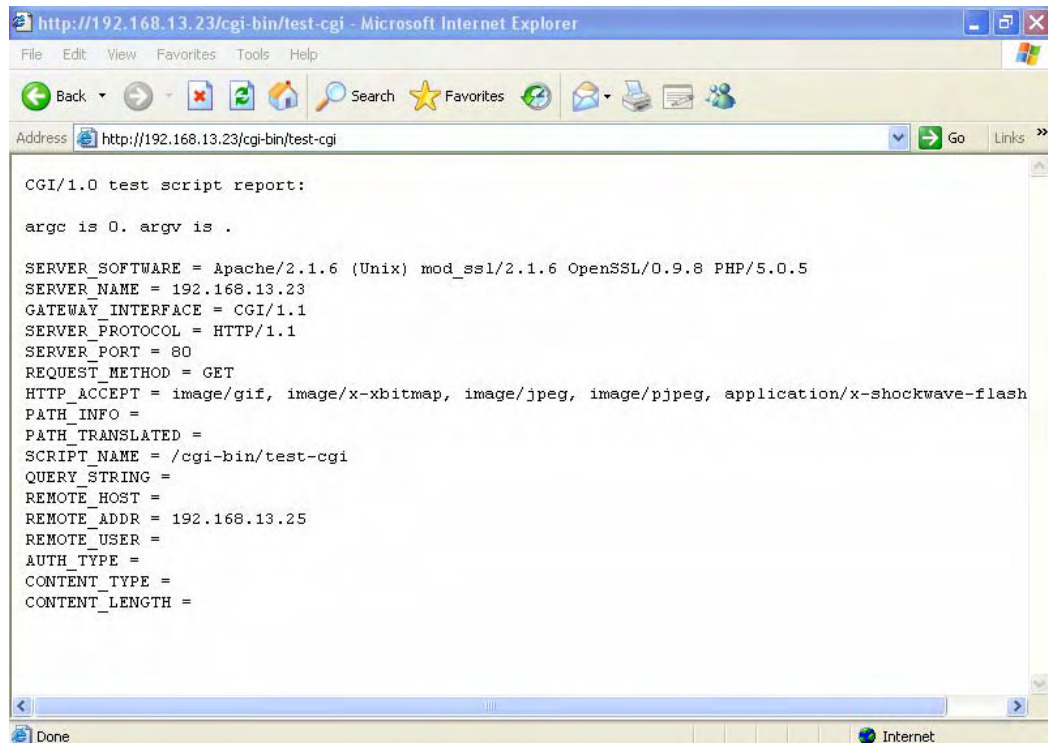
Save your CGI page to the following directory:

`/home/httpd/cgi-bin/`

Before you modify the homepage, use a browser (such as Microsoft Internet Explorer or Mozilla Firefox) from your PC to test if the Apache Web Server is working. Type the IP address of LAN1 or LAN2 in the browser's address box to open the homepage. E.g , type `http://192.168.13.23` in the address box.



To open the default CGI page, type **http://192.168.13.23/cgi-bin/test-cgi** in your browser's address box.



**NOTE** The CGI function is enabled by default. If you want to disable the function, modify the file `/etc/apache/conf/httpd.conf`. When you develop your own CGI application, make sure your CGI file is executable.

```
192.168.3.127 - PuTTY
root@Moxa: /home/httpd/cgi-bin# ls -al
drwxr-xr-x  2 root  root    0 Aug 24 1999 .
drwxr-xr-x  5 root  root    0 Nov  5 16:16 ..
-rwxr-xr-x  1 root  root   757 Aug 24 1999 test-cgi
root@Moxa: /home/httpd/cgi-bin#
```

## Install PHP for Apache Web Server

This embedded computer supports the PHP option. However, since the PHP file is 3 MB, it is not installed by default. To install it yourself, first make sure there is enough free space (at least 3 MB) on your embedded flash ROM).

Step 1: Check that you have enough free space

```
192.168.3.127 - PuTTY
root@Moxa:/bin# df -h
Filesystem      Size      Used Available Use% Mounted on
/dev/mtdblock2  8.0M      6.0M      2.0M    75% /
/dev/ram0        499.0k    17.0k     457.0k   4% /var
/dev/mtdblock3  6.0M     488.0k    5.5M    8% /tmp
/dev/mtdblock3  6.0M     488.0k    5.5M    8% /home
/dev/mtdblock3  6.0M     488.0k    5.5M    8% /etc
tmpfs           30.4M      0         30.4M   0% /dev/shm
root@Moxa:/bin#
```

To check that the /dev/mtdblock3 free space is greater than 3 MB.

Step 2: Type 'upramdisk' to get the free space ram disk to save the package.

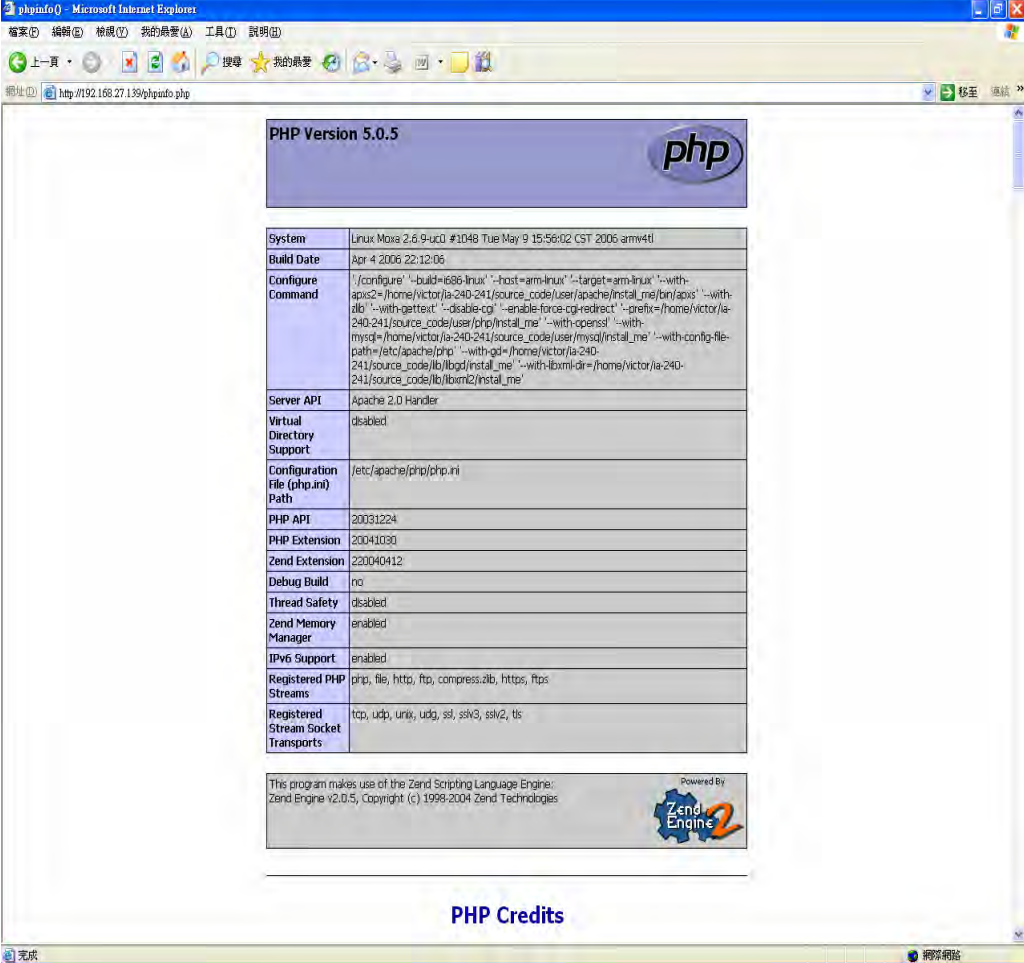
```
192.168.3.127 - PuTTY
root@Moxa:/bin# upramdisk
root@Moxa:/bin# df -h
Filesystem      Size      Used Available Use% Mounted on
/dev/mtdblock2  8.0M      6.0M      2.0M    75% /
/dev/ram0        499.0k    18.0k     456.0k   4% /var
/dev/mtdblock3  6.0M     488.0k    5.5M    8% /tmp
/dev/mtdblock3  6.0M     488.0k    5.5M    8% /home
/dev/mtdblock3  6.0M     488.0k    5.5M    8% /etc
tmpfs           30.4M      0         30.4M   0% /dev/shm
/dev/ram1        16.0M      1.0k     15.1M   0% /var/ramdisk
root@Moxa:/bin#
```

Step 3: Download the PHP package from the CD-ROM. You can find the package in CD ROM/utility\_tools/libmysqlclientx\_x.x.xx\_XXXXXX.ipk and ROM/utility\_tools/libphpx\_x.x.x\_XXXXXX.ipk (where x indicates the version and CPU model)

Step 4: Run 'ipkg-cl install libmysqlclient5\_5.1.23\_arm.ipk' and then run 'ipkg-cl install libphp5\_5.2.5\_arm.ipk' to install php

Step 5: Test it. Use the browser to access <http://192.168.3.127/phpinfo.php>






PHP Version 5.0.5

System	Linux Moira 2.6.9-uc0 #1048 Tue May 9 15:56:02 CST 2006 armv4tl
Build Date	Apr 4 2006 22:12:06
Configure Command	./configure '--build=i686-linux' '--host=arm-linux' '--target=arm-linux' '--with-apxs2=/home/victor/ia-240-241/source_code/user/apache/install_me/bin/apxs' '--with-xml' '--with-gettext' '--disable-cg' '--enable-force-cgi-redirect' '--prefix=/home/victor/ia-240-241/source_code/user/php/install_me' '--with-openssl' '--with-mysqld=/home/victor/ia-240-241/source_code/user/mysqld/install_me' '--with-config-file-path=/etc/apache/php' '--with-gd=/home/victor/ia-240-241/source_code/lib/gd/install_me' '--with-ibxml-dir=/home/victor/ia-240-241/source_code/lib/libxml2/install_me'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/apache/php/php.ini
PHP API	20031224
PHP Extension	20041030
Zend Extension	220040412
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	ftp, file, http, ftp, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls

This program makes use of the Zend Scripting Language Engine:  
Zend Engine v2.0.5, Copyright (c) 1999-2004 Zend Technologies

Powered By  


PHP Credits

If you want to uninstall PHP, follow steps 2 to 4 but select the uninstall option.

## IPTABLES

IPTABLES is an administrative tool for setting up, maintaining, and inspecting the Linux kernel's IP packet filter rule tables. Several different tables are defined, with each table containing built-in chains and user-defined chains.

Each chain is a list of rules that apply to a certain type of packet. Each rule specifies what to do with a matching packet. A rule (such as a jump to a user-defined chain in the same table) is called a "target."

The IA260/IA261/IA262/EM-2260 supports 3 types of IPTABLES table: **Filter** tables, **NAT** tables, and **Mangle** tables:

A. **Filter Table**—includes three chains:

**INPUT** chain—filters all incoming traffic destined for the local host. Note that all incoming packets destined for this host pass through this chain, no matter what interface or direction they came from.

**OUTPUT** chain—filters packets sent from the local host.

**FORWARD** chain—routes and filters forwarded packets only. Note that all forwarded traffic passes through this chain (not only in one direction), so you need to consider this factor when writing your rule-set.

B. **NAT Table**—includes three chains:

**PREROUTING** chain—transfers the destination IP address (DNAT)

**POSTROUTING** chain—works after the routing process and before the Ethernet device process to transfer the source IP address (SNAT)

**OUTPUT** chain—produces local packets

*Sub-tables*

**Source NAT (SNAT)**—changes the first source packet IP address.

**Destination NAT (DNAT)**—changes the first destination packet IP address.

**MASQUERADE**—a special form for SNAT. If one host can connect to Internet, then other computers that connect to this host can connect to the Internet when the computer does not have an actual IP address.

**REDIRECT**—a special form of DNAT that re-sends packets to a local host independent of the destination IP address.

C. **Mangle Table**—includes the following chains:

**INPUT**—mangles packets after they have been routed, but before they are actually sent to the processing machine.

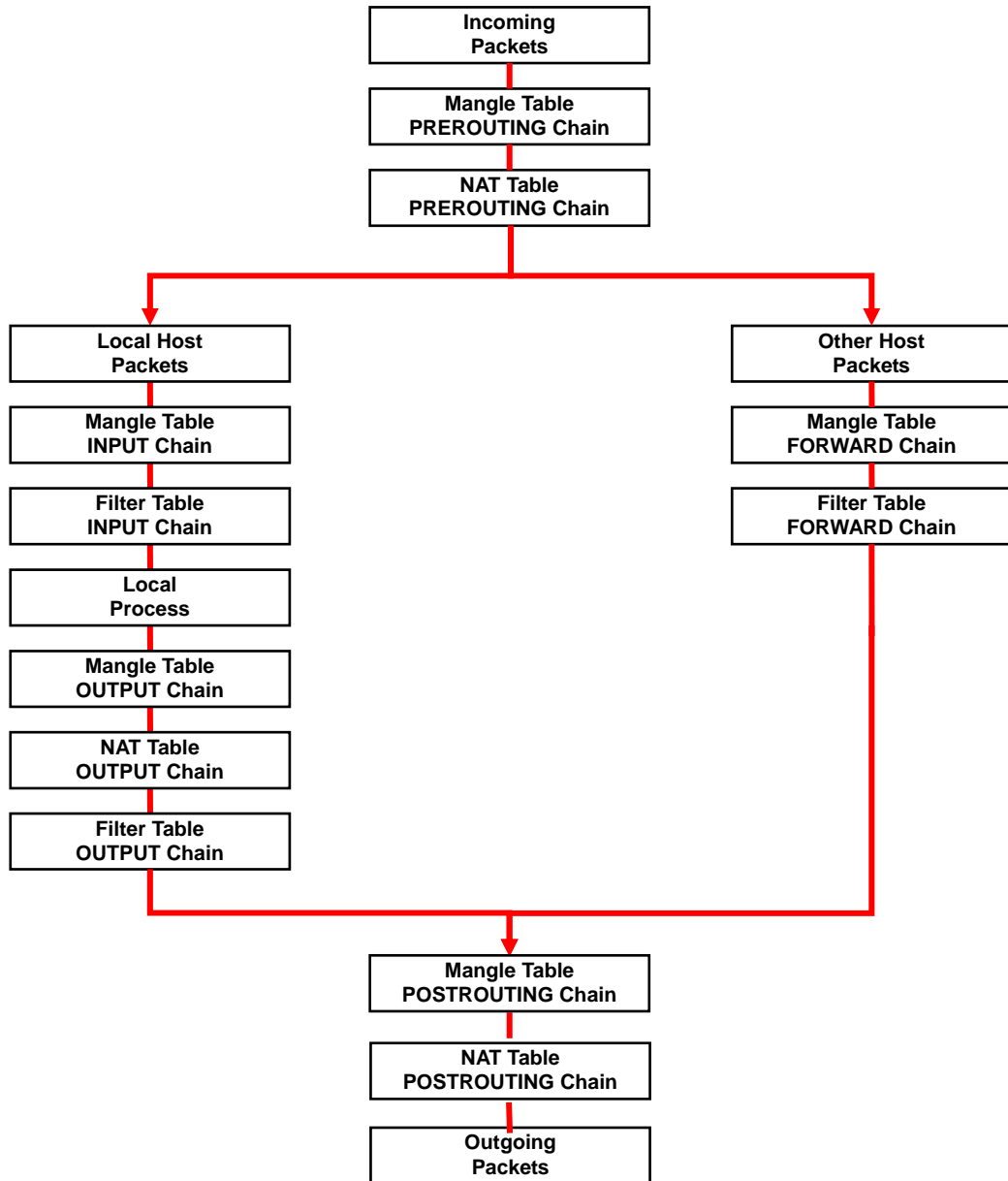
**FORWARD**—mangles the packet after the initial routing decision, but before the last routing decision prior to sending the packet the out.

**PREROUTING** chain—pre-processes packets before the routing process.

**OUTPUT** chain—processes packets after the routing process.

It has three extensions—TTL, MARK, and TOS.

The following figure shows the IPTABLES hierarchy.



The IA260/IA261/IA262/EM-2260 supports the following sub-modules. Be sure to use the module that matches your application.

ip_queue	ipt_REDIRECT	ipt_ah	iptable_filter
ip_tables	ipt_REJECT	ipt_ecn	iptable_mangle
ipt_CLUSTERIP	ipt_SAME	ipt_iprange	iptable_nat
ipt_ECN	ipt_TOS	ipt_owner	iptable_raw
ipt_LOG	ipt_TTL	ipt_recent	
ipt_MASQUERADE	ipt_ULOG	ipt_tos	
ipt_NETMAP	ipt_addrtype	ipt_ttl	

**NOTE** The IA260/IA261/IA262/EM-2260 does NOT support IPv6 and ipchains.

The basic syntax to enable and load an IPTABLES module is as follows:

```
#lsmod
#modprobe ip_tables
#modprobe iptable_filter
```

Use **lsmod** to check if the ip\_tables module has already been loaded in the IA260/IA261/IA262/EM-2260. Use **modprobe** to insert and enable the module.

Use the following command to load the modules (iptable\_filter, iptable\_mangle, iptable\_nat):

```
#modprobe iptable_filter
```

**NOTE** IPTABLES plays the role of packet filtering or NAT. Take care when setting up the IPTABLES rules. If the rules are not correct, remote hosts that connect via a LAN or PPP may be denied access. We recommend using the Serial Console to set up the IPTABLES.

Click on the following links for more information about iptables.

<http://www.linuxguruz.com/iptables/>  
<http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html>

Since the IPTABLES command is very complex, to illustrate the IPTABLES syntax we have divided our discussion of the various rules into three categories: **Observe and erase chain rules**, **Define policy rules**, and **Append or delete rules**.

## Observe and erase chain rules

### Usage:

```
# iptables [-t tables] [-L] [-n]
  -t tables:  Table to manipulate (default: 'filter'); example: nat or filter.
  -L [chain]: ListList all rules in selected chains. If no chain is selected, all chains are listed.
  -n:        Numeric output of addresses and ports.

# iptables [-t tables] [-FZX]
  -F:  Flush the selected chain (all the chains in the table if none is listed).
  -X:  Delete the specified user-defined chain.
  -Z:  Set the packet and byte counters in all chains to zero.
```

**Examples:**

```
# iptables -L -n
```

In this example, since we do not use the `-t` parameter, the system uses the default 'filter' table. Three chains are included: INPUT, OUTPUT, and FORWARD. INPUT chains are accepted automatically, and all connections are accepted without being filtered.

```
#iptables -F
#iptables -X
#iptables -Z
```

**Define policy for chain rules****Usage:**

```
# iptables [-t tables] [-P] [INPUT, OUTPUT, FORWARD, PREROUTING, OUTPUT, POSTROUTING]
[ACCEPT, DROP]
```

`-P:` Set the policy for the chain to the given target.

`INPUT:` For packets coming into the IA260/IA261/IA262/EM-2260.

`OUTPUT:` For locally-generated packets.

`FORWARD:` For packets routed out through the IA260/IA261/IA262/EM-2260.

`PREROUTING:` To alter packets as soon as they come in.

`POSTROUTING:` To alter packets as they are about to be sent out.

**Examples:**

```
#iptables -P INPUT DROP
#iptables -P OUTPUT ACCEPT
#iptables -P FORWARD ACCEPT
#iptables -t nat -P PREROUTING ACCEPT
#iptables -t nat -P OUTPUT ACCEPT
#iptables -t nat -P POSTROUTING ACCEPT
```

In this example, the policy accepts outgoing packets and denies incoming packets.

**Append or delete rules:****Usage:**

```
# iptables [-t table] [-AI] [INPUT, OUTPUT, FORWARD] [-io interface] [-p tcp, udp, icmp,
all] [-s IP/network] [--sport ports] [-d IP/network] [--dport ports] -j [ACCEPT. DROP]
```

`-A:` Append one or more rules to the end of the selected chain.

`-I:` Insert one or more rules in the selected chain as the given rule number.

`-i:` Name of an interface via which a packet is going to be received.

`-o:` Name of an interface via which a packet is going to be sent.

`-p:` The protocol of the rule or of the packet to check.

`-s:` Source address (network name, host name, network IP address, or plain IP address).

`--sport:` Source port number.

`-d:` Destination address.

`--dport:` Destination port number.

`-j:` Jump target. Specifies the target of the rules; i.e., how to handle matched packets. For example, ACCEPT the packet, DROP the packet, or LOG the packet.

**Examples:**

Example 1: Accept all packets from lo interface.

```
# iptables -A INPUT -i lo -j ACCEPT
```

Example 2: Accept TCP packets from 192.168.0.1.

```
# iptables -A INPUT -i eth0 -p tcp -s 192.168.0.1 -j ACCEPT
```

Example 3: Accept TCP packets from Class C network 192.168.1.0/24.

```
# iptables -A INPUT -i eth0 -p tcp -s 192.168.1.0/24 -j ACCEPT
```

Example 4: Drop TCP packets from 192.168.1.25.

```
# iptables -A INPUT -i eth0 -p tcp -s 192.168.1.25 -j DROP
```

Example 5: Drop TCP packets addressed for port 21.

```
# iptables -A INPUT -i eth0 -p tcp --dport 21 -j DROP
```

Example 6: Accept TCP packets from 192.168.0.24 to IA260/IA261/IA262/EM-2260's port 137, 138, 139

```
# iptables -A INPUT -i eth0 -p tcp -s 192.168.0.24 --dport 137:139 -j ACCEPT
```

Example 7: Log TCP packets that visit IA260/IA261/IA262/EM-2260's port 25.

```
# iptables -A INPUT -i eth0 -p tcp --dport 25 -j LOG
```

Example 8: Drop all packets from MAC address 01:02:03:04:05:06.

```
# iptables -A INPUT -i eth0 -p all -m mac --mac-source 01:02:03:04:05:06 -j DROP
```

NOTE: In Example 8, remember to issue the command `#modprobe ipt_mac` first to load module `ipt_mac`.

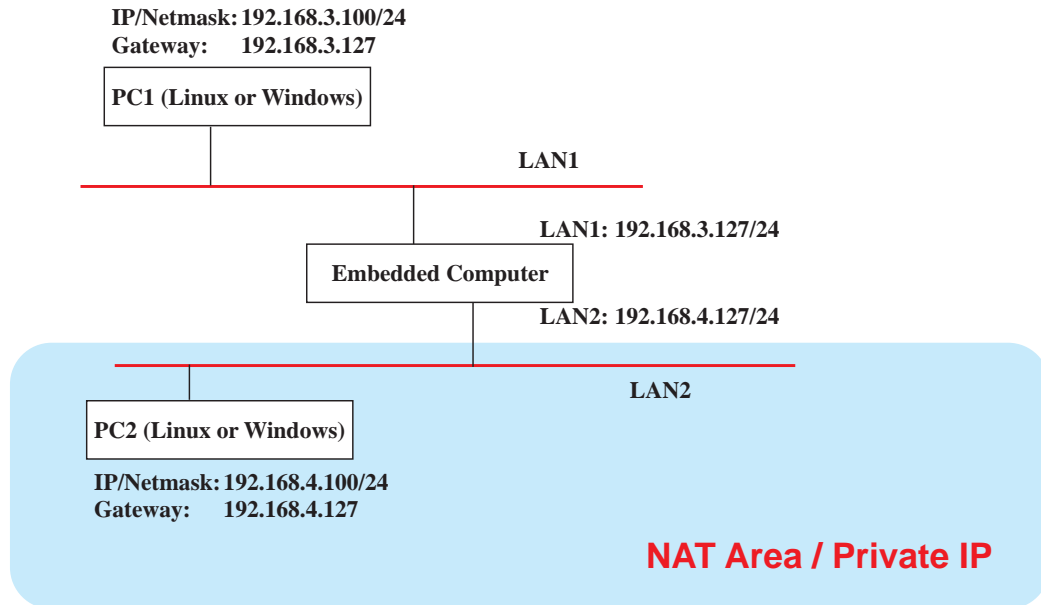
## NAT

NAT (Network Address Translation) protocol translates IP addresses used on one network to different IP addresses used on another network. One network is designated the inside network and the other designated the outside network. Typically, the IA260/IA261/IA262/EM-2260 connects several devices on a network and maps local inside network addresses to one or more global outside IP addresses, and un-maps the global IP addresses on incoming packets back into local IP addresses.

NOTE	Click on the following link for more information about iptables and NAT: <a href="http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html">http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html</a>
------	---

## NAT Example

The IP address of LAN1 is changed to 192.168.3.127 (you will need to load the module `ipt_MASQUERADE`):



1. `#echo 1 > /proc/sys/net/ipv4/ip_forward`
2. `#modprobe ip_tables`
3. `#modprobe iptable_filter`
4. `#modprobe ip_conntrack`
5. `#modprobe iptable_nat`
6. `#modprobe ipt_MASQUERADE`
7. `#iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 192.168.3.127`
8. `#iptables -t nat -A POSTROUTING -o eth0 -s 192.168.3.0/24 -j MASQUERADE`

## Enabling NAT at Bootup

In most real world situations, you will want to use a simple shell script to enable NAT when the IA260/IA261/IA262/EM-2260 boots up. The following script is an example.

```
#!/bin/bash
# If you put this shell script in the /home/nat.sh
# Remember to chmod 744 /home/nat.sh
# Edit the rc.local file to make this shell startup automatically.
# vi /etc/rc.d/rc.local
# Add a line in the end of rc.local /home/nat.sh
EXIF='eth0' #This is an external interface for setting up a valid IP address.
EXNET='192.168.4.0/24' #This is an internal network address.
# Step 1. Insert modules.
# Here 2> /dev/null means the standard error messages will be dump to null device.
modprobe ip_tables 2> /dev/null
modprobe ip_nat_ftp 2> /dev/null
modprobe ip_nat_irc 2> /dev/null
modprobe ip_conntrack 2> /dev/null
modprobe ip_conntrack_ftp 2> /dev/null
modprobe ip_conntrack_irc 2> /dev/null
# Step 2. Define variables, enable routing and erase default rules.
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
export PATH
```

```

echo "1" > /proc/sys/net/ipv4/ip_forward
/bin/iptables -F
/bin/iptables -X
/bin/iptables -Z
/bin/iptables -F -t nat
/bin/iptables -X -t nat
/bin/iptables -Z -t nat
/bin/iptables -P INPUT ACCEPT
/bin/iptables -P OUTPUT ACCEPT
/bin/iptables -P FORWARD ACCEPT
/bin/iptables -t nat -P PREROUTING ACCEPT
/bin/iptables -t nat -P POSTROUTING ACCEPT
/bin/iptables -t nat -P OUTPUT ACCEPT
# Step 3. Enable IP masquerade.

```

## Dial-up Service—PPP

PPP (Point to Point Protocol) is used to run IP (Internet Protocol) and other network protocols over a serial link. PPP can be used for direct serial connections (using a null-modem cable) over a Telnet link, and links established using a modem over a telephone line.

Modem / PPP access is almost identical to connecting directly to a network through the IA260/IA261/IA262/EM-2260's Ethernet port. Since PPP is a peer-to-peer system, the IA260/IA261/IA262/EM-2260 can also use PPP to link two networks (or a local network to the Internet) to create a Wide Area Network (WAN).

NOTE	<p>Click on the following links for more information about PPP:</p> <p><a href="http://tldp.org/HOWTO/PPP-HOWTO/index.html">http://tldp.org/HOWTO/PPP-HOWTO/index.html</a></p> <p><a href="http://axion.physics.ubc.ca/ppp-linux.html">http://axion.physics.ubc.ca/ppp-linux.html</a></p>
------	---

The `pppd` daemon is used to connect to a PPP server from a Linux system. For detailed information about `pppd` see the man page.

### Example 1: Connecting to a PPP server over a simple dial-up connection

The following command is used to connect to a PPP server by modem. Use this command for old ppp servers that prompt for a login name (replace *username* with the correct name) and password (replace *password* with the correct password). Note that *debug* and *defaultroute 192.1.1.17* are optional.

```
#pppd connect 'chat -v " " ATDT5551212 CONNECT" " ogin: username word: password'
/dev/ttyM0 115200 debug crtscts modem defaultroute
```

If the PPP server does not prompt for the username and password, the command should be entered as follows. Replace *username* with the correct username and replace *password* with the correct password.

```
#pppd connect 'chat -v " " ATDT5551212 CONNECT" " ` user username password password
/dev/ttyM0 115200 crtscts modem
```

The `pppd` options are described below:

```
connect 'chat etc...'
```

This option gives the command to contact the PPP server. The 'chat' program is used to dial a remote computer. The entire command is enclosed in single quotes because `pppd` expects a one-word argument for the 'connect' option. The options for 'chat' are given below:

```
-v
verbose mode; log what we do to syslog
" "
```



Double quotes—don't wait for a prompt, but proceed with the following instead (note that you must include a space before the second quotation mark).

**ATDT5551212**

Dial the modem, and proceed with the following.

**CONNECT**

Wait for an answer.

**" "**

Send a return (null text followed by the usual return)

**ogin: username word: password**

Log in with *username* and *password*.

Refer to the chat man page, chat.8, for more information about the chat utility.

**/dev/**

Specify the callout serial port.

**115200**

The baudrate.

**debug**

Log status in syslog.

**crtscts**

Use hardware flow control between computer and modem (at 115200 this is a must).

**modem**

Indicates that this is a modem device; pppd will hang up the phone before and after making the call.

**defaultroute**

Once the PPP link is established, make it the default route; if you have a PPP link to the Internet, this is probably what you want.

**192.1.1.17**

This is a degenerate case of a general option of the form *x.x.x.x:y.y.y.y*. Here *x.x.x.x* is the local IP address and *y.y.y.y* is the IP address of the remote end of the PPP connection. If this option is not specified, or if just one side is specified, then *x.x.x.x* defaults to the IP address associated with the local machine's hostname (located in **/etc/hosts**), and *y.y.y.y* is determined by the remote machine.

## Example 2: Connecting to a PPP server over a hard-wired link

If a username and password are not required, use the following command (note that *noipdefault* is optional):

```
#pppd connect `chat -v" " " " ` noipdefault /dev/ttyM0 19200 crtscts
```

If a username and password is required, use the following command (note that *noipdefault* is optional, and *root* is both the username and password):

```
#pppd connect `chat -v" " " " ` user root password root noipdefault /dev/ttyM0 19200 crtscts
```

## How to check the connection

Once you've set up a PPP connection, there are some steps you can take to test the connection. First, type:

```
/sbin/ifconfig
```

(The file **ifconfig** may be located elsewhere, depending on your distribution.) You should be able to see all the network interfaces that are UP. ppp0 should be one of them, and you should recognize the first IP address as your own, and the "P-t-P address" (or point-to-point address) the address of your server. Here's what it looks like on one machine:

```
lo      Link encap Local Loopback
        inet addr 127.0.0.1  Bcast 127.255.255.255  Mask 255.0.0.0
        UP LOOPBACK RUNNING  MTU 2000  Metric 1
        RX packets 0 errors 0 dropped 0 overrun 0

ppp0    Link encap Point-to-Point Protocol
        inet addr 192.76.32.3  P-t-P 129.67.1.165  Mask 255.255.255.0
        UP POINTOPOINT RUNNING  MTU 1500  Metric 1
        RX packets 33 errors 0 dropped 0 overrun 0
        TX packets 42 errors 0 dropped 0 overrun 0
```

Now, type:

```
ping z.z.z.z
```

where z.z.z.z is the address of your name server. This should work. Here's what the response could look like:

```
Moxa:~$p ping 129.67.1.165
PING 129.67.1.165 (129.67.1.165): 56 data bytes
64 bytes from 129.67.1.165: icmp_seq=0 ttl=225 time=268 ms
64 bytes from 129.67.1.165: icmp_seq=1 ttl=225 time=247 ms
64 bytes from 129.67.1.165: icmp_seq=2 ttl=225 time=266 ms
^C
--- 129.67.1.165 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 247/260/268 ms
Moxa:~$
```

Try typing:

```
netstat -nr
```

This should show three routes, similar to the following:

```
Kernel routing table
Destination      Gateway          Genmask          Flags      Metric    Ref     Use
iface
129.67.1.165     0.0.0.0         255.255.255.255 UH         0         0       6
ppp0
127.0.0.0        0.0.0.0         255.0.0.0       U          0         0       0 lo
0.0.0.0          129.67.1.165   0.0.0.0         UG         0         0      6298
ppp0
```

If your output looks similar but doesn't have the destination 0.0.0.0 line (which refers to the default route used for connections), you may have run pppd without the 'defaultroute' option. At this point you can try using Telnet or ftp, bearing in mind that you'll have to use numeric IP addresses unless you've set up /etc/resolv.conf correctly.

## Setting up a Machine for Incoming PPP Connections

This first example applies to using a modem, and requiring authorization with a username and password.

```
pppd/dev/ttyM0 115200 crtscts modem 192.168.16.1:192.168.16.2 login auth
```

You should also add the following line to the file `/etc/ppp/pap-secrets`:

```
* * "" *
```

The first star (\*) lets everyone login. The second star (\*) lets every host connect. The pair of double quotation marks ("" ) is to use the file `/etc/passwd` to check the password. The last star (\*) is to let any IP connect.

The following example does not check the username and password:

```
pppd/dev/ttyM0 115200 crtscts modem 192.168.16.1:192.168.16.2
```

## PPPoE

1. Connect IA260/IA261/IA262/EM-2260's LAN port to an ADSL modem with a cross-over cable, HUB, or switch.
2. Log into the IA260/IA261/IA262/EM-2260 as the root user.
3. Edit the file `/etc/ppp/chap-secrets` and add the following:

```
"username@hinet.net"*"password"*
```

```
# Secrets for authentication using CHAP
# client server secret IP addresses
"username@hinet.net" * "password" *
:
:
:
:
"chap-secrets" line 1 of 3 --33%--
```

"username@hinet.net" is the username obtained from the ISP to log in to the ISP account.  
 "password" is the corresponding password for the account.

4. Edit the file `/etc/ppp/pap-secrets` and add the following:

```
"username@hinet.net"*"password"*
```

```
# password if you don't use the login option of pppd! The mgetty Debian
# package already provides this option; make sure you don't change that.

# INBOUND connections
# Every regular user can use PPP and has to use passwords from /etc/passwd
hostname      "*"
"username@hinet.net" *    "password" *
hostname      "*"
# UserIDs that cannot use PPP at all. Check your /etc/passwd and add any
# other accounts that should not be able to use pppd!
guest  hostname      "*" -
master hostname      "*" -
root   hostname      "*" -
support hostname      "*" -
stats  hostname      "*" -

# OUTBOUND connections

# Here you should add your userid password to connect to your providers via
# PAP. The * means that the password is to be used for ANY host you connect
# to. Thus you do not have to worry about the foreign machine name. Just
# replace password with your password.
"pap-secrets" line 1 of 42 --2%--
```

`"username@hinet.net"` is the username obtained from the ISP to log in to the ISP account.  
`"password"` is the corresponding password for the account.

5. Add one of two files: `/etc/ppp/options.eth0` or `/etc/ppp/options.eth1`. The choice depends on which LAN is connected to the ADSL modem. If you use LAN1 to connect to the ADSL modem, then add `/etc/ppp/options.eth0`. If you use LAN2 to connect to the ADSL modem, then add `/etc/ppp/options.eth1`. The file context is shown below:

```
name username@hinet.net
mtu 1492
mru 1492
defaultroute
noipdefault
~
~
~
~
~
~
~
~
~
~
~
"options.ixp0" line 1 of 5 --20%--
```

Type your username (the one you set in the `/etc/ppp/pap-secrets` and `/etc/ppp/chap-secrets` files) after the "name" option. You may add other options as desired.

6. Set up DNS

If you are using DNS servers supplied by your ISP, edit the file `/etc/resolv.conf` by adding the following lines of code:

```
nameserver ip_addr_of_first_dns_server
nameserver ip_addr_of_second_dns_server
```

For example:

```
nameserver 168.95.1.1
nameserver 139.175.10.20
```

7. Use the following command to create a pppoe connection:

```
pppd eth0
```

The eth0 is what is connected to the ADSL modem LAN port. The example above uses LAN1.

To use LAN2, type:

```
pppd eth1
```

8. Type **ifconfig ppp0** to check if the connection is OK or has failed. If the connection is OK, you will see information about the ppp0 setting for the IP address. Use ping to test the IP.

9. If you want to disconnect it, use the kill command to kill the pppd process.

## NFS (Network File System)

The Network File System (NFS) is used to mount a disk partition on a remote machine, as if it were on a local hard drive, allowing fast, seamless sharing of files across a network. NFS allows users to develop applications for the IA260/IA261/IA262/EM-2260, without worrying about the amount of disk space that will be available. The IA260/IA261/IA262/EM-2260 supports NFS protocol for client.

NOTE Click on the following links for more information about NFS:  
<http://www.tldp.org/HOWTO/NFS-HOWTO/index.html>

## Setting up the IA260/IA261/IA262/EM-2260 as an NFS Client

The following procedure is used to mount a remote NFS Server.

1. To know the NFS Server's shared directory.
2. Establish a mount point on the NFS Client site.
3. Mount the remote directory to a local directory.

```
#mkdir -p /home/nfs/public
#mount -t nfs NFS_Server(IP):/directory /mount/point
```

**Example**

```
#mount -t nfs 192.168.3.100:/home/public /home/nfs/public
```

## Mail

Smtplib is a minimal SMTP client that takes an email message body and passes it on to an SMTP server. It is suitable for applications that use email to send alert messages or important logs to a specific user.

To send an email message, use the 'smtplib' utility, which uses SMTP protocol. Type **#smtplib -help** to see the help message.

### Example:

```
smtplib -s test -f sender@company.com -S IP_address receiver@company.com  
< mail-body-message
```

- s: The mail subject.
- f: Sender's mail address
- S: SMTP server IP address

The last mail address **receiver@company.com** is the receiver's e-mail address. **mail-body-message** is the mail content. The last line of the body of the message should contain ONLY the period '.' character.

You will need to add your hostname to the file **/etc/hosts**.

## SNMP

The IA260/IA261/IA262/EM-2260 has built-in SNMP V1 (Simple Network Management Protocol) agent software. It supports RFC1317 RS-232 like group and RFC 1213 MIB-II.

NOTE Click on the following links for more information about MIB II and RS-232 like groups:  
<http://www.faqs.org/rfcs/rfc1213.html>  
<http://www.faqs.org/rfcs/rfc1317.html>

→ IA260/IA261/IA262/EM-2260 does NOT support SNMP trap.

## OpenVPN

OpenVPN provides two types of tunnels for users to implement VPNS: **Routed IP Tunnels** and **Bridged Ethernet Tunnels**. To begin with, check to make sure that the system has a virtual device **/dev/net/tun**. If not, issue the following command:

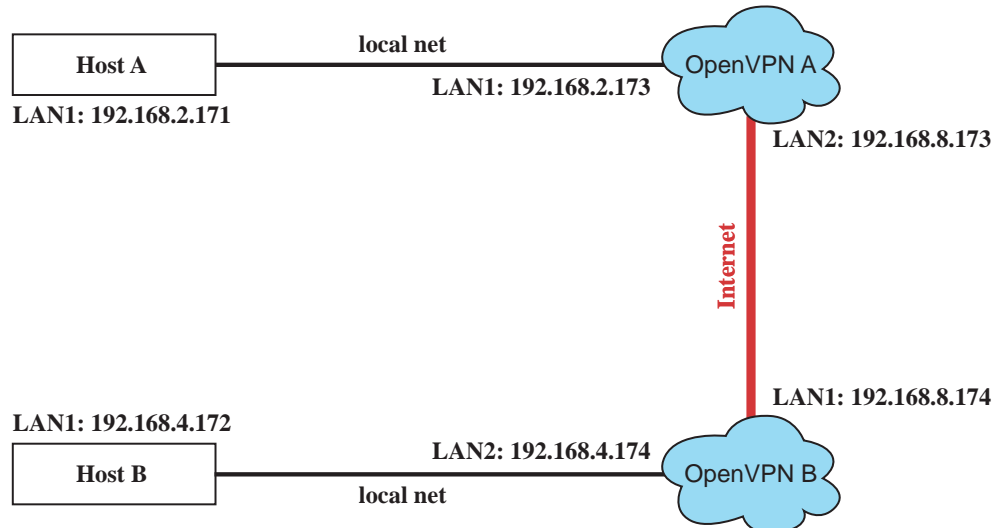
```
# mknod /dev/net/tun c 10 200
```

An Ethernet bridge is used to connect different Ethernet networks together. The Ethernets are bundled into one bigger, "logical" Ethernet. Each Ethernet corresponds to one physical interface (or port) that is connected to the bridge.

On each OpenVPN machine, you should generate a working directory, such as **/etc/openvpn**, where script files and key files reside. Once established, all operations will be performed in that directory.

## Setup 1: Ethernet Bridging for Private Networks on Different Subnets

1. Set up four machines, as shown in the following diagram.



Host A (B) represents one of the machines that belongs to OpenVPN A (B). The two remote subnets are configured for a different range of IP addresses. When this setup is moved to a public network, the external interfaces of the OpenVPN machines should be configured for static IPs, or connect to another device (such as a firewall or DSL box) first.

```
# openvpn --genkey --secret secrouter.key
```

Copy the file that is generated to the OpenVPN machine.

2. Generate a script file named **openvpn-bridge** on each OpenVPN machine. This script reconfigures interface "eth1" as IP-less, creates logical bridge(s) and TAP interfaces, loads modules, enables IP forwarding, etc.

```
#-----Start-----
#!/bin/sh

iface=eth1 # defines the internal interface
maxtap=`expr 1` # defines the number of tap devices. I.e., # of tunnels

IPADDR=
NETMASK=
BROADCAST=

# it is not a great idea but this system doesn't support
# /etc/sysconfig/network-scripts/ifcfg-eth1
ifcfg_vpn()
{
while read f1 f2 f3 f4 r3
do
if [ "$f1" = "iface" -a "$f2" = "$iface" -a "$f3" = "inet" -a "$f4" = "static" ];then
i=`expr 0`
while :
do
if [ $i -gt 5 ]; then
break
fi
i=`expr $i + 1`
read f1 f2
case "$f1" in
```

```

        address ) IPADDR=$f2
        ;;
        netmask ) NETMASK=$f2
        ;;
        broadcast ) BROADCAST=$f2
        ;;
    esac
done
    break
fi
done < /etc/network/interfaces
}

# get the ip address of the specified interface
mname=
module_up()
{
    oIFS=$IFS
    IFS='
'
    FOUND="no"
    for LINE in `lsmod`
    do
        TOK=`echo $LINE | cut -d' ' -f1`
        if [ "$TOK" = "$mname" ]; then
            FOUND="yes";
            break;
        fi
    done
    IFS=$oIFS

    if [ "$FOUND" = "no" ]; then
        modprobe $mname
    fi
}

start()
{
    ifcfg_vpn
    if [ ! \(` -d "/dev/net" \) ]; then
        mkdir /dev/net
    fi

    if [ ! \(` -r "/dev/net/tun" \) ]; then
        # create a device file if there is none
        mknod /dev/net/tun c 10 200
    fi

    # load modules "tun" and "bridge"
    mname=tun
    module_up
    mname=bridge
    module_up
    # create an ethernet bridge to connect tap devices, internal interface
    brctl addbr br0
    brctl addif br0 $iface
    # the bridge receives data from any port and forwards it to other ports.

    i=`expr 0`
    while :
    do
        # generate a tap0 interface on tun
        openvpn --mktun --dev tap${i}

        # connect tap device to the bridge
        brctl addif br0 tap${i}

        # null ip address of tap device

```



```

    ifconfig tap${i} 0.0.0.0 promisc up

    i=`expr $i + 1`
    if [ $i -ge $maxtap ]; then
        break
    fi
done

# null ip address of internal interface
ifconfig $iface 0.0.0.0 promisc up

# enable bridge ip
ifconfig br0 $IPADDR netmask $NETMASK broadcast $BROADCAST

ipf=/proc/sys/net/ipv4/ip_forward
# enable IP forwarding
echo 1 > $ipf
echo "ip forwarding enabled to"
cat $ipf
}

stop() {
    echo "shutdown openvpn bridge."
    ifcfg_vpn
    i=`expr 0`
    while :
    do
        # disconnect tap device from the bridge
        brctl delif br0 tap${i}
        openvpn --rmtun --dev tap${i}

        i=`expr $i + 1`
        if [ $i -ge $maxtap ]; then
            break
        fi
    done
    brctl delif br0 $iface
    brctl delbr br0
    ifconfig br0 down
    ifconfig $iface $IPADDR netmask $NETMASK broadcast $BROADCAST
    killall -TERM openvpn
}

case "$1" in
start)
    start
    ;;
stop)
    stop
    ;;
restart)
    stop
    start
    ;;
*)
    echo "Usage: $0 [start|stop|restart]"
    exit 1
esac
exit 0
#----- end -----

```

Create link symbols to enable this script at boot time:

```

# ln -s /etc/openvpn/openvpn-bridge /etc/rc.d/rc3.d/S32vpn-br # for example
# ln -s /etc/openvpn/openvpn-bridge /etc/rc.d/rc6.d/K32vpn-br # for example

```

3. Create a configuration file named **A-tap0-br.conf** and an executable script file named **A-tap0-br.sh** on OpenVPN A.

```
# point to the peer
remote 192.168.8.174
dev tap0
secret /etc/openvpn/secrouter.key
cipher DES-EDE3-CBC
auth MD5
tun-mtu 1500
tun-mtu-extra 64
ping 40
up /etc/openvpn/A-tap0-br.sh

#-----Start-----
#!/bin/sh
# value after "-net" is the subnet behind the remote peer
route add -net 192.168.4.0 netmask 255.255.255.0 dev br0
#----- end -----
```

Create a configuration file named **B-tap0-br.conf** and an executable script file named **B-tap0-br.sh** on OpenVPN B.

```
# point to the peer
remote 192.168.8.173
dev tap0
secret /etc/openvpn/secrouter.key
cipher DES-EDE3-CBC
auth MD5
tun-mtu 1500
tun-mtu-extra 64
ping 40
up /etc/openvpn/B-tap0-br.sh

#----- Start-----
#!/bin/sh
# value after "-net" is the subnet behind the remote peer
route add -net 192.168.2.0 netmask 255.255.255.0 dev br0
#----- end -----
```

**Note:** Select cipher and authentication algorithms by specifying “cipher” and “auth”. To see which algorithms are available, type:

```
# openvpn --show-ciphers
```

4. Start both of OpenVPN peers,

```
# openvpn --config A-tap0-br.conf&
# openvpn --config B-tap0-br.conf&
```

If you see the line “Peer Connection Initiated with 192.168.8.173:5000” on each machine, the connection between OpenVPN machines has been established successfully on UDP port 5000.

5. On each OpenVPN machine, check the routing table by typing the command:

```
# route
```

Destination	Gateway	Genmsk	Flags	Metric	Ref	Use	Iface
192.168.4.0	*	255.255.255.0	U	0	0	0	br0
192.168.2.0	*	255.255.255.0	U	0	0	0	br0
192.168.8.0	*	255.255.255.0	U	0	0	0	eth0

Interface **eth1** is connected to the bridging interface **br0**, to which device **tap0** also connects, whereas the virtual device **tun** sits on top of **tap0**. This ensures that all traffic from internal networks connected to interface **eth1** that come to this bridge write to the TAP/TUN device that the OpenVPN program monitors. Once the OpenVPN program detects traffic on the virtual device, it sends the traffic to its peer.

- To create an indirect connection to Host B from Host A, you need to add the following routing item:

```
route add -net 192.168.4.0 netmask 255.255.255.0 dev eth0
```

To create an indirect connection to Host A from Host B, you need to add the following routing item:

```
route add -net 192.168.2.0 netmask 255.255.255.0 dev eth0
```

Now ping Host B from Host A by typing:

```
ping 192.168.4.174
```

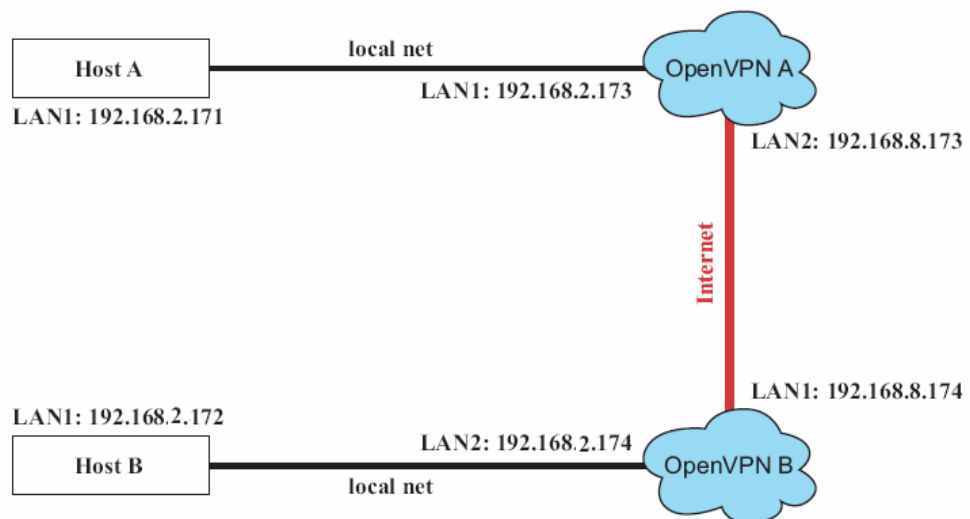
A successful ping indicates that you have created a VPN system that only allows authorized users from one internal network to access users at the remote site. For this system, all data is transmitted by UDP packets on port 5000 between OpenVPN peers.

- To shut down OpenVPN programs, type the command:

```
# killall -TERM openvpn
```

## Setup 2: Ethernet Bridging for Private Networks on the Same Subnet

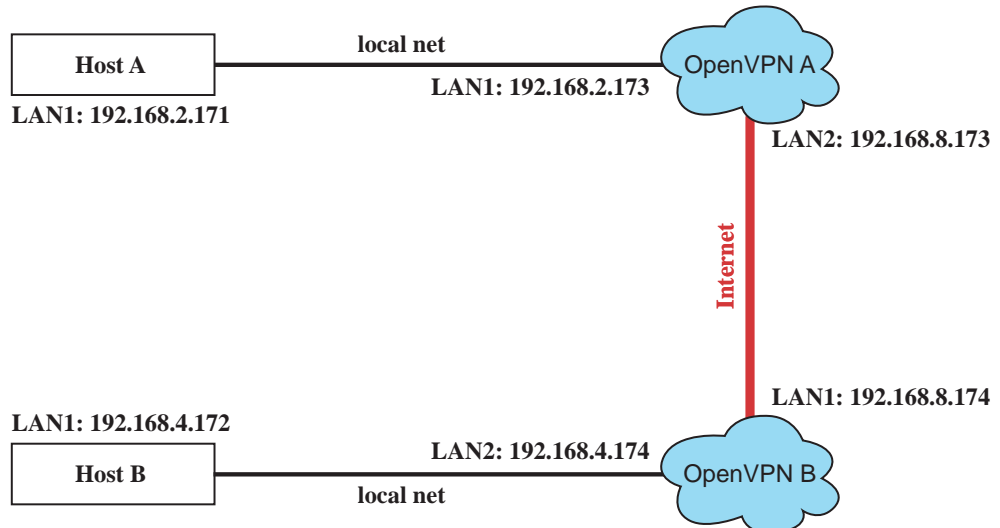
- Set up four machines as shown in the following diagram:



- The configuration procedure is almost the same as for the previous example. The only difference is that you will need to indicate the **#up** parameter. “/etc/openvpn/A-tap0-br.conf” and “/etc/openvpn/B-tap0-br.conf”.

### Setup 3: Routed IP

1. Set up the four machines as shown in the following diagram:



2. Create a configuration file named "A-tun.conf" and an executable script file named "A-tun.sh".

```
# point to the peer
remote 192.168.8.174
dev tun
secret /etc/openvpn/secrouter.key
cipher DES-EDE3-CBC
auth MD5
tun-mtu 1500
tun-mtu-extra 64
ping 40
ifconfig 192.168.2.173 192.168.4.174
up /etc/openvpn/A-tun.sh
```

```
#----- Start-----
#!/bin/sh
# value after "-net" is the subnet behind the remote peer
route add -net 192.168.4.0 netmask 255.255.255.0 gw $5
#----- end -----
```

Create a configuration file named **B-tun.conf** and an executable script file named **B-tun.sh** on OpenVPN B:

```
remote 192.168.8.173
dev tun
secret /etc/openvpn/secrouter.key
cipher DES-EDE3-CBC
auth MD5
tun-mtu 1500
tun-mtu-extra 64
ping 40
ifconfig 192.168.4.174 192.168.2.173
up /etc/openvpn/B-tun.sh
```

```
#----- Start-----
#!/bin/sh
# value after "-net" is the subnet behind the remote peer
route add -net 192.168.2.0 netmask 255.255.255.0 gw $5
#----- end -----
```

Note that the parameter "ifconfig" defines the first argument as the local internal interface and the second argument as the internal interface at the remote peer.

Note that \$5 is the argument that the OpenVPN program passes to the script file. Its value is the second argument of **ifconfig** in the configuration file.

3. Check the routing table after you run the OpenVPN programs, by typing the command:

```
# route
```

Destination	Gateway	Genmsk	Flags	Metric	Ref	Use	Iface
192.168.4.174	*	255.255.255.255	UH	0	0	0	tun0
192.168.4.0	192.168.4.174	255.255.255.0	UG	0	0	0	tun0
192.168.2.0	*	255.255.255.0	U	0	0	0	eth1
192.168.8.0	*	255.255.255.0	U	0	0	0	eth0

## Development Tool Chains

---

This chapter describes how to install a tool chain in the host computer that you use to develop your applications. In addition, the process of performing cross-platform development and debugging are also introduced. For clarity, the IA260/IA261/IA262/EM-2260 embedded computer is called a target computer.

The following functions are covered in this chapter:

- ❑ **Linux Tool Chain**
  - Steps for Installing the Linux Tool Chain
  - Compilation for Applications
  - On-Line Debugging with GDB

## Linux Tool Chain

The Linux tool chain contains a suite of cross compilers and other tools, as well as the libraries and header files that are necessary to compile your applications. These tool chain components must be installed in your host computer (PC) running Linux. We have confirmed that the following Linux distributions can be used to install the tool chain.

**Fedora 7, Debian 4**

### Steps for Installing the Linux Tool Chain

The tool chain needs about 1 GB of hard disk space. To install it, follow the steps.

1. Insert the package CD into your PC and then issue the following commands:  

```
#mount /dev/cdrom /mnt/cdrom
#sh /mnt/cdrom/tool-chain/linux/arm-linux_x.x.sh (where x.x indicates the version
of the Tool Chain)
```
2. Wait for the installation process to complete. This should take a few minutes.
3. Add the directory **/usr/local/arm-linux/bin** to your path. You can do this for the current login by issuing the following commands:  

```
#export PATH="/usr/local/arm-linux/bin:$PATH"
```

Alternatively, you can add the same commands to **\$HOME/.bash\_profile** to make it effective for all login sessions.

### Compilation for Applications

To compile a simple C application, use the cross compiler instead of the regular compiler:

```
#arm-linux-gcc -o example -Wall -g -O2 example.c
#arm-linux-strip -s example
#arm-linux-gcc -ggdb -o example-debug example.c
```

Most of the cross compiler tools are the same as their native compiler counterparts, but with an additional prefix that specifies the target system. In the case of x86 environments, the prefix is **i386-linux-** and in the case of IA260/IA261/IA262/EM-2260 ARM boards, it is **arm-linux-**.

For example, the native C compiler is **gcc** and the cross C compiler for ARM in the IA260/IA261/IA262/EM-2260 is **arm-linux-gcc**.

The following cross compiler tools are provided:

ar	Manages archives (static libraries)
as	Assembler
c++, g++	C++ compiler
cpp	C preprocessor
gcc	C compiler
gdb	Debugger
ld	Linker
nm	Lists symbols from object files
objcopy	Copies and translates object files
objdump	Displays information about object files
ranlib	Generates indexes to archives (static libraries)
readelf	Displays information about ELF files
size	Lists object file section sizes
strings	Prints strings of printable characters from files (usually object files)
strip	Removes symbols and sections from object files (usually debugging information)

## On-Line Debugging with GDB

The tool chain also provides an on-line debugging mechanism to help you develop your program. Before performing a debugging session, add the option **-ggdb** to compile the program. A debugging session runs on a client-server architecture on which the server **gdbserver** is installed in the target computer and the client **ddd** is installed in the host computer. We'll assume that you have uploaded a program named **hello-debug** to the target computer and started debugging the program.

1. Log on to the target computer and run the debugging server program.

```
#gdbserver 192.168.4.142:2000 hello-debug
Process hello-debug created; pid=38
```

The debugging server listens for connections at network port 2000 from the network interface 192.168.4.142. The name of the program to be debugged follows these parameters. For a program requiring arguments, add the arguments behind the program name.

2. In the host computer, change the directory to where the program source resides.

```
cd /my_work_directory/myfilesystem/testprograms
```

3. Execute the client program.

```
#ddd --debugger arm-linux-gdb hello-debug &
```

4. Enter the following command at the GDB, DDD command prompt.

```
Target remote 192.168.4.99:2000
```

The command produces a line of output on the target console, similar to the following.

```
Remote debugging using 192.168.4.99:2000
```

192.168.4.99 is the machine's IP address, and 2000 is the port number. You can now begin debugging in the host environment using the interface provided by DDD.

5. Set a break point on main function by double clicking, or by entering **b main** on the command line.
6. Click the **cont** button.



# 6

## Programmer's Guide

---

This chapter includes important information for programmers.

The following functions are covered in this chapter:

- Flash Memory Map**
- Device API**
- RTC (Real Time Clock)**
- Buzzer**
- UART**
- MoxaCAN Programming Guide**
- Digital I/O**

## Flash Memory Map

Partition sizes are hard coded into the kernel binary. To change the partition sizes, you will need to rebuild the kernel. The flash memory map is shown in the following table.

Address	Size	Contents
0x00000000 – 0x0005FFFF	384 KB	Boot Loader—Read ONLY
0x00040000 – 0x002FFFFFFF	2.75 MB	Kernel object code—Read ONLY
0x00300000 – 0x00FFFFFFF	13 MB	Root file system (JFFS2) —Read ONLY
0x01000000 – 0x01FFFFFFF	16 MB	User directory (JFFS2) —Read/Write

Mount the user file system to **/mnt/usrdisk** with the root file system. Check to see if the user file system was mounted correctly. If the user file system is okay, the kernel will change the root file system to **/mnt/usrdisk**. If the user file system is not okay, the kernel will use the default Moxa file system. To finish the boot process, run the init program.

- |      |  |
|------|--|
| NOTE | <ol style="list-style-type: none"> <li>1. The default Moxa file system only enables the network and CF. It lets users recover the user file system when it fails.</li> <li>2. The user file system is a complete file system. Users can create and delete directories and files (including source code and executable files) as needed.</li> <li>3. Users can create the user file system on the PC host or target platform, and then copy it to the IA260/IA261/IA262/EM-2260.</li> </ol> |
|------|--|

## Device API

The IA260/IA261/IA262/EM-2260 supports control devices with the **ioctl** system API. You will need to **include <moxadvice.h>**, and use the following **ioctl** function.

```
int ioctl(int d, int request,...);
Input: int d - open device node return file handle
      int request - argument in or out
```

Use the desktop Linux's man page for detailed documentation:

```
#man ioctl
```

## RTC (Real Time Clock)

The device node is located at **/dev/rtc**. The IA260/IA261/IA262/EM-2260 supports Linux standard simple RTC control. You must **include <linux/rtc.h>**.

1. Function: **RTC\_RD\_TIME**  

```
int ioctl(fd, RTC_RD_TIME, struct rtc_time *time);
```

 Description: read time information from RTC. It will return the value on argument 3.
2. Function: **RTC\_SET\_TIME**  

```
int ioctl(fd, RTC_SET_TIME, struct rtc_time *time);
```

 Description: set RTC time. Argument 3 will be passed to RTC.

## Buzzer

The device node is located at `/dev/console`. The IA260/IA261/IA262/EM-2260 supports Linux standard buzzer control, with the IA260/IA261/IA262/EM-2260's buzzer running at a fixed frequency of 100 Hz. You must **include** `<sys/kd.h>`.

Function: KDMKTONE

```
ioctl(fd, KDMKTONE, unsigned int arg);
```

Description: The buzzer's behavior is determined by the argument **arg**. The "high word" part of the argument gives the length of time the buzzer will sound, and the "low word" part gives the frequency.

The buzzer's on/off behavior is controlled by software. If you call the "ioctl" function, you **MUST** set the frequency at 100 Hz. If you use a different frequency, the system could crash.

## UART

The normal tty device node is located at `/dev/ttyM0 ... ttyM3`.

The IA260/IA261/IA262/EM-2260 supports Linux standard termios control. The Moxa UART Device API allows you to configure ttyM0 to ttyM3 as RS-232, RS-422, 4-wire RS-485, or 2-wire RS-485. IA260/IA261/IA262/EM-2260 supports RS-232, RS-422, 2-wire RS-485, and 4-wire RS485.

You must **include** `<moxadevice.h>`.

```
#define RS232_MODE    0
#define RS485_2WIRE_MODE 1
#define RS422_MODE    2
#define RS485_4WIRE_MODE 3
```

1. Function: MOXA\_SET\_OP\_MODE

```
int ioctl(fd, MOXA_SET_OP_MODE, &mode)
```

### Description

Set the interface mode. Argument 3 mode will pass to the UART device driver and change it.

2. Function: MOXA\_GET\_OP\_MODE

```
int ioctl(fd, MOXA_GET_OP_MODE, &mode)
```

### Description

Get the interface mode. Argument 3 mode will return the interface mode.

There are two Moxa private ioctl commands for setting up special baudrates.

Function: MOXA\_SET\_SPECIAL\_BAUD\_RATE

Function: MOXA\_GET\_SPECIAL\_BAUD\_RATE

If you use this ioctl to set a special baudrate, the termios cflag will be B4000000, in which case the B4000000 define will be different. If the baudrate you get from termios (or from calling `tcgetattr()`) is B4000000, you must call ioctl with MOXA\_GET\_SPECIAL\_BAUD\_RATE to get the actual baudrate.

### Example to set the baudrate

```
#include <moxadevice.h>
#include <termios.h>
struct termios term;
int fd, speed;
fd = open("/dev/ttyM0", O_RDWR);
tcgetattr(fd, &term);
term.c_cflag &= ~(CBAUD | CBAUDEX);
term.c_cflag |= B4000000;
tcsetattr(fd, TCSANOW, &term);
speed = 500000;
ioctl(fd, MOXA_SET_SPECIAL_BAUD_RATE, &speed);
```

### Example to get the baudrate

```
#include <moxadevice.h>
#include <termios.h>
struct termios term;
int fd, speed;
fd = open("/dev/ttyM0", O_RDWR);
tcgetattr(fd, &term);
if ( (term.c_cflag & (CBAUD|CBAUDEX)) != B4000000 ) {
    // follow the standard termios baud rate define
} else {
    ioctl(fd, MOXA_GET_SPECIAL_BAUD_RATE, &speed);
}
```

### Baudrate inaccuracy

Divisor = 921600/Target Baud Rate. (Only Integer part)

ENUM = 8 \* (921600/Target - Divisor) (Round up or down)

**Inaccuracy** = ((Target Baud Rate - 921600/(Divisor + (ENUM/8))) / Target Baud Rate) \* 100%

E.g.,

To calculate 500000 bps

Divisor = 1, ENUM = 7,

Inaccuracy = 1.7%

\* Inaccuracy should be less than 2% for work reliably.

### Special Note

1. If the target baudrate is not a special baudrate (e.g. 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600), the termios cflag will be set to the same flag.
2. If you use stty to get the serial information, you will get speed equal to 0.

# MoxaCAN Programming Guide

## Introduction

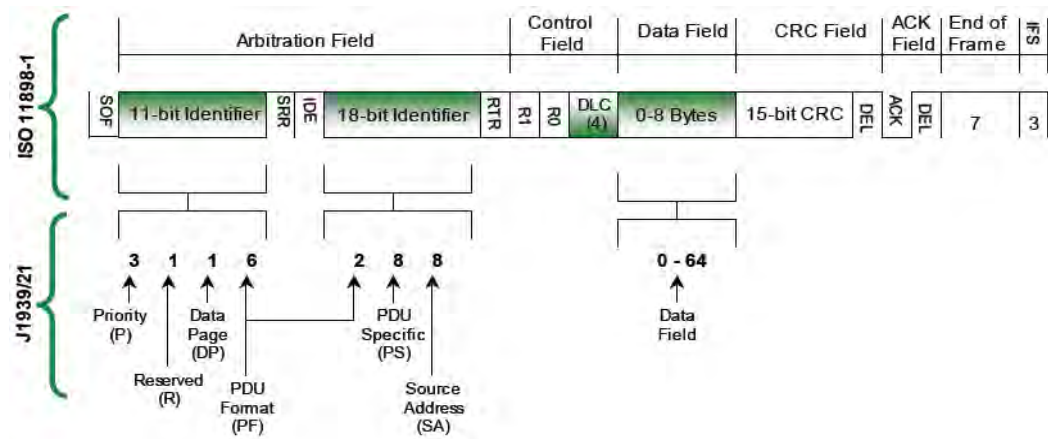
CAN is a broadcast serial bus standard for connecting electronic control units (ECUs).

Each node is able to send and receive messages, but not simultaneously: a message (consisting primarily of an ID—usually chosen to identify the message-type/sender—and up to eight message bytes) is transmitted serially onto the bus, one bit after another. This signal-pattern codes the message (in NRZ) and is sensed by all nodes.

Moxa embedded computers provide the CAN bus interface for industrial CAN communication. Users can use the file control interface to read, write, or control the CAN interface as a file for easy CAN programming.

## Programming Guide

The CAN data format looks like this:



Define the structure for CAN programming as follows:

```
moxacan_ioctl.h
```

```
#ifndef _MOXACAN_IOCTL_H
#define _MOXACAN_IOCTL_H
```

```
#ifndef BIT
#define BIT(x) (1<<(x))
#endif // BIT
```

```
#define MAX_CAN_DATA_LENGTH      8
```

```
typedef union __attribute__((packed)) can_data_union {
    unsigned char array[MAX_CAN_DATA_LENGTH];
    struct byte_struct {
        unsigned char    byte1;
        unsigned char    byte2;
        unsigned char    byte3;
        unsigned char    byte4;
        unsigned char    byte5;
    };
};
```

```

        unsigned char    byte6;
        unsigned char    byte7;
        unsigned char    byte8;
    } byte;
} can_data_u;

typedef struct user_frame_struct {
    int    id;
    int    info;    // RTR & extend ID flag
#define USER_RTR    BIT(0)
#define USER_EXTEND_ID    BIT(1)
    int    data_length;
    can_data_u    data;
} user_frame_t;

// ioctl command code
#define MOXACAN_IOCTL_SET_BAUD_RATE    0x100
#define MOXACAN_IOCTL_GET_BAUD_RATE    0x101

#endif // _MOXACAN_IOCTL_H

```

Then we can include this header for CAN programming

We can open /dev/can[n] to use the CAN device. Then read(), write() or ioctl() the device files. If you complete the using, you should close() this file to release this device.

```

#include    <stdio.h>
#include    <stdlib.h>
#include    <fcntl.h>
#include    <string.h>
#include    "moxacan_ioctl.h"

#define PORT1_NAME    "/dev/can0"
#define PORT2_NAME    "/dev/can1"
int    main(int argc, char *argv[])
{
    int    fd1, fd2, i, baudrate;
    user_frame_t    txframe, rxframe;

    fd1 = open(PORT1_NAME, O_RDWR);
    if ( fd1 < 0 ) {
        printf("Open %s fail !\n", PORT1_NAME);
        return -1;
    }
    fd2 = open(PORT2_NAME, O_RDWR);
    if ( fd2 < 0 ) {
        printf("Open %s fail !\n", PORT2_NAME);
        return -1;
    }
    baudrate = 100000;
    ioctl(fd1, MOXACAN_IOCTL_SET_BAUD_RATE, &baudrate);
    ioctl(fd2, MOXACAN_IOCTL_SET_BAUD_RATE, &baudrate);
}

```

```

    memset(&txframe, 0, sizeof(txframe));
    memset(&rxframe, 0, sizeof(rxframe));

txframe.id = 12;
txframe.data_length = MAX_CAN_DATA_LENGTH;
txframe.info |= USER_EXTEND_ID;
for ( i=0; i<MAX_CAN_DATA_LENGTH; i++ )
    txframe.data.array[i] = 0x30+i;

i = write(fd1, &txframe, sizeof(txframe));
printf("Write port1 data return = %d\n", i);
printf("Write data contexts are following :\n");
for ( i=0; i<MAX_CAN_DATA_LENGTH; i++ )
    printf("%c", txframe.data.array[i]);
printf("\n");

usleep(1);
i = read(fd2, &rxframe, sizeof(rxframe));
printf("Read port2 data return = %d\n", i);
printf("Receive frame id = %d, length = %d\n", rxframe.id, rxframe.data_length);

printf("Receive data contexts are following :\n");
for ( i=0; i<MAX_CAN_DATA_LENGTH; i++ )
    printf("%c", rxframe.data.array[i]);
printf("\n");

close(fd1);
close(fd2);
return 0;
}

```

#### Makefile

```

CROSS_COMPILE=arm-linux-
CC=$(CROSS_COMPILE)gcc
LD=$(CROSS_COMPILE)ld
STRIP=$(CROSS_COMPILE)strip
#STRIP=$(CROSS_COMPILE)strip -s

TARGET=tcan
TARGET1=canttrw

all:    debug release

debug: $(TARGET).c $(TARGET1).c moxacan_ioctl.h
      $(CC) -ggdb -o $(TARGET)-debug $(TARGET).c
      $(CC) -ggdb -o $(TARGET1)-debug $(TARGET1).c

release: $(TARGET).c $(TARGET1).c moxacan_ioctl.h
      $(CC) -o $(TARGET)-release $(TARGET).c
      $(CC) -o $(TARGET1)-release $(TARGET1).c

```

```
$(STRIP) $(TARGET)-release $(TARGET1)-release
```

clean:

```
rm -f *.o $(TARGET)-debug $(TARGET)-release $(TARGET1)-debug $(TARGET1)-release
```

## Digital I/O

Digital Output channels can be set to high or low. The channels are controlled by the function call **set\_dout\_state()**. The digital input channels can be used to detect the state change of the digital input signal. The DI channels can also be used to detect whether or not the state of a digital signal changes during a fixed period of time. This can be done by the function call, **set\_din\_event()**.

Moxa provides 5 function calls to handle the digital I/O state change and event handling.

### Application Programming Interface

Return error code definitions:

```
#define DIO_ERROR_PORT -1 // no such port
#define DIO_ERROR_MODE -2 // no such mode or state
#define DIO_ERROR_CONTROL -3 // open or ioctl fail
#define DIO_ERROR_DURATION -4 // The value of duration is not 0 or not in the range, 40
<= duration <= 3600000 milliseconds (1 hour)
#define DIO_ERROR_DURATION_20MS -5 // The value of duration must be a multiple of 20
ms
#define DIO_OK 0
```

The definition for DIN and DOUT:

```
#define DIO_HIGH 1
#define DIO_LOW 0
```

```
int set_dout_state(int doport, int state)
```

Description: To set the DOUT port to high or low state.

Input: int doport - which DOUT port you want to set. Port starts from 0 to 3.

int state - to set high or low state; DIO\_HIGH (1) for high, DIO\_LOW (0) for low.

Output: none.

Return: reference the error code.

```
int get_din_state(int diport, int *state)
```

Description: To get the DIN port state.

Input: int diport - get the current state of which DIN port. Port numbering is from 0 to 3.

int \*state - save the current state.

Output: state - DIO\_HIGH (1) for high, DIO\_LOW (0) for low.

Return: reference the error code.

```
int get_dout_state(int doport, int *state)
```



Description: To get the DOUT port state.

Input: int doport - get the current state of which DOUT port.

int \*state - save the current state.

Output: state - DIO\_HIGH (1) for high, DIO\_LOW (0) for low.

Return: reference the error code.

int set\_din\_event(int diport, void (\*func)(int diport), int mode, long int duration)

Description: Set the event for DIN when the state is changed from high to low or from low to high.

Input: int diport - the port that will be used to detect the DIN event.

Port numbering is from 0 to 3.

void (\*func) (int diport) - Not NULL

> Returns the call back function. When the event occurs, the call back function will be invoked.

NULL

> Clears this event

int mode DIN\_EVENT\_HIGH\_TO\_LOW

(1): from high to low

DIN\_EVENT\_LOW\_TO\_HIGH

(0): from low to high

DIN\_EVENT\_CLEAR

(-1): clear this event

unsigned long duration - 0: detect the din event > DIN\_EVENT\_HIGH\_TO\_LOW or  
DIN\_EVENT\_LOW\_TO\_HIGH> without duration

- Not 0

> detect the din event

DIN\_EVENT\_HIGH\_TO\_LOW or

DIN\_EVENT\_LOW\_TO\_HIGH with

duration. The value of "duration" must be a

multiple of 20 milliseconds. The range of

"duration" is 0, or  $40 \leq \text{duration} \leq 3600000$

milliseconds. The error of the measurement is

24 ms. For example, if the DIN duration is

200 ms, this event will be generated when the

DIN pin stays in the same state for a time

between 176 ms and 200 ms.

Output: none.

Return: reference the error code.

```
int get_din_event(int diport, int *mode, long int *duration)
```

Description: To retrieve the DIN event configuration, including mode (DIN\_EVENT\_HIGH\_TO\_LOW or DIN\_EVENT\_LOW\_TO\_HIGH), and the value of "duration."

Input: **int diport** - which DIN port you want to retrieve.

- The port whose din event setting we wish to retrieve

**int \*mode** - save which event is set.

**unsigned long \*duration** - the duration of the DIN port is kept in high or low state.

- return to the current duration value of diport

Output: **mode** DIN\_EVENT\_HIGH\_TO\_LOW

(1): from high to low

DIN\_EVENT\_LOW\_TO\_HIGH(0): from low to high

DIN\_EVENT\_CLEAR(-1): clear this event

**duration** The value of duration should be 0 or 40 <= duration

<= 3600000 milliseconds.

Return: reference the error code.

## Special Note

Don't forget to link the library **libmoxalib** for DI/DO programming, and also include the header file **moxadevice.h**. The DI/DO library only can be used by one program at a time.

## Examples

### DIO Program Source Code File Example

File Name: tdio.c

Description: The program indicates to connect DO1 to DI1, change the digital output state to high or low by manual input, then detect and count the state changed events from DI1.

```
#include <stdio.h>
#include <stdlib.h>
#include <moxadevice.h>
#include <fcntl.h>
#ifdef DEBUG
#define dbg_printf(x...) printf(x)
#else
#define dbg_printf(x...)
#endif
#define MIN_DURATION 40
static char *DataString[2]={"Low ", "High "};
static void hightolowevent(int diport)
{
printf("\nDIN port %d high to low.\n", diport);
```

```

}
static void lowtohighevent(int diport)
{
printf("\nDIN port %d low to high.\n", diport);
}
int main(int argc, char * argv[])
{
int i, j, state, retval;
unsigned long duration;
while( 1 ) {
printf("\nSelect a number of menu, other key to exit. \n\
1. set high to low event \n\
2. get now data. \n\
3. set low to high event \n\
4. clear event \n\
5. set high data. \n\
6. set low data. \n\
7. quit \n\
8. show event and duration \n\
Choose : ");
retval =0;
scanf("%d", &i);
if ( i == 1 ) { // set high to low event
printf("Please keyin the DIN number : ");
scanf("%d", &i);
printf("Please input theDINduration, thisminimunvalue must be over%d : ", MIN_DURATION);
scanf("%lu", &duration);
retval=set_din_event(i, hightolowevent, DIN_EVENT_HIGH_TO_LOW, duration);
} else if ( i == 2 ) { // get now data
printf("DIN data : ");
for ( j=0; j<4; j++ ) {
get_din_state(j, &state);
printf("%s", DataString[state]);
}
printf("\n");
printf("DOUT data : ");
for ( j=0; j<MAX_DOUT_PORT; j++ ) {
get_dout_state(j, &state);
printf("%s", DataString[state]);
}
printf("\n");
}
}

```

```
    } else if ( i == 3 ) { // set low to high event
printf("Please keyin the DIN number : ");
scanf("%d", &i);
printf("Please input the DIN duration, this minimum value must be over %d :", MIN_DURATION);
scanf("%lu", &duration);
retval = set_din_event(i, lowtohighevent, DIN_EVENT_LOW_TO_HIGH, duration);
    } else if ( i == 4 ) { // clear event
printf("Please keyin the DIN number : ");
scanf("%d", &i);
retval=set_din_event(i, NULL, DIN_EVENT_CLEAR, 0);
    } else if ( i == 5 ) { // set high data
printf("Please keyin the DOUT number : ");
scanf("%d", &i);
retval=set_dout_state(i, 1);
    } else if ( i == 6 ) { // set low data
printf("Please keyin the DOUT number : ");
scanf("%d", &i);
retval=set_dout_state(i, 0);
    } else if ( i == 7 ) { // quit
break;
    } else if ( i == 8 ) { // show event and duration
printf("Event:\n");
for ( j=0; j<MAX_DOUT_PORT; j++ ) {
retval=get_din_event(j, &i, &duration);
switch ( i ) {
case DIN_EVENT_HIGH_TO_LOW :
printf("(htl,%lu)", duration);
break;
case DIN_EVENT_LOW_TO_HIGH :
printf("(lth,%lu)", duration);
break;
case DIN_EVENT_CLEAR :
printf("(clr,%lu)", duration);
break;
default :
printf("err " );
break;
}
}
printf("\n");
    } else {
```

```
printf("Select error, please select again !\n");
}
switch(retval) {
case DIO_ERROR_PORT:
printf("DIO error port\n");
break;
case DIO_ERROR_MODE:
printf("DIO error mode\n");
break;
case DIO_ERROR_CONTROL:
printf("DIO error control\n");
break;
case DIO_ERROR_DURATION:
printf("DIO error duratoin\n");
case DIO_ERROR_DURATION_20MS:
printf("DIO error! The duratoin is not a multiple of 20 ms\n");
break;
}
}
return 0;
}
```

### DIO Program Make File Example

```
FNAME=tdio
CC=arm-linux-gcc
STRIP=arm-linux-strip
release:
$(CC) -o $(FNAME) $(FNAME).c -lmoxalib -lpthread
$(STRIP) -s $(FNAME)
debug:
$(CC) -DDEBUG -o $(FNAME)-dbg $(FNAME).cxx -lmoxalib -lpthread
clean:
/bin/rm -f $(FNAME) $(FNAME)-dbg *.o
```

# A

## System Commands

---

### Linux normal command utility collection

#### File manager

1. **cp** copy file
2. **ls** list file
3. **ln** make symbolic link file
4. **mount** mount and check file system
5. **rm** delete file
6. **chmod** change file access permissions
7. **chown** change file owner
8. **chgrp** change file group
9. **sync** sync file system, let system file buffer be saved to hardware
10. **mv** move file
11. **pwd** display the current working directory
12. **df** list now file system space
13. **mkdir** make new directory
14. **rmdir** delete directory

#### Editor

1. **vi** text editor
2. **cat** dump file context
3. **zcat** compress or expand files
4. **grep** search string on file
5. **cut** get string on file
6. **find** find where the files are
7. **more** dump file by one page
8. **test** test file exist or not
9. **sleep** sleep (seconds)
10. **echo** echo string

#### Network

1. **ping** ping to test network
2. **route** routing table manager
3. **netstat** display network status
4. **ifconfig** set network ip address
5. **tracerout** trace route
6. **tftp** Trivial File Transfer Protocol client
7. **telnet** remote login utility
8. **ftp** File Transfer Protocol utility

## Process

1. **kill** kill process
2. **ps** display now running process

## Other

1. **dmesg** dump kernel log message
2. **stty** to set serial port
3. **mknod** make device node
4. **free** display system memory usage
5. **date** print or set the system date and time
6. **env** run a program in a modified environment
7. **clear** clear the terminal screen
8. **reboot** reboot / power off/on the server
9. **halt** halt the server
10. **du** estimate file space usage
11. **gzip, gunzip** compress or expand files
12. **hostname** show system's host name

## Moxa special utilities

1. **kversion** show kernel version
2. **upramdisk** mount ramdisk
3. **downramdisk** unmount ramdisk
4. **setinterface** configure serial port mode as RS-232, RS-422, or RS-485
5. **upgradehfm** firmware upgrade utility
6. **reportip** UC Finder host program
7. **setdef** reset to default script