

ioPAC 8600 CPU30 C Version Software User's Manual

Version 1.2, February 2021

www.moxa.com/product



© 2021 Moxa Inc. All rights reserved.

ioPAC 8600 CPU30 C Version Software 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

© 2021 Moxa Inc. All rights reserved.

Trademarks

The MOXA logo 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

Moxa Americas

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

Moxa Europe

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

Moxa India

Tel: +91-80-4172-9088
Fax: +91-80-4132-1045

Moxa China (Shanghai office)

Toll-free: 800-820-5036
Tel: +86-21-5258-9955
Fax: +86-21-5258-5505

Moxa Asia-Pacific

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

Table of Contents

1. Introduction.....	1-1
Overview	1-2
Scope	1-2
Related Documents	1-2
Supported Products	1-2
Software Architecture.....	1-2
Software Package.....	1-4
2. Getting Started.....	2-1
Developing the First Program	2-2
Hardware Requirements	2-2
Development Procedure	2-2
Configuring the Ethernet Interface.....	2-2
Modifying Network Settings with the Serial Console	2-2
Modifying Network Settings over the Network	2-3
Connecting an ioPAC Controller to a Linux PC.....	2-3
Installing the Toolchain on the PC.....	2-4
Configuring Cross Compiler and glibc Environment Variables.....	2-4
Developing Code and Compiling a Program	2-4
Upload and Run the Program.....	2-5
Systemctl Command	2-6
3. Controller Management.....	3-1
System Information	3-2
Firmware Upgrade and Default Settings.....	3-2
Upgrading the Firmware.....	3-2
Upgrading the Firmware from the SD Card	3-4
Loading Factory Defaults	3-4
Enabling and Disabling Daemons	3-5
System Time Setting.....	3-5
Setting the Time Manually	3-5
Using NTP Client to update the Time.....	3-5
Configuring the NTP Server	3-6
Executing Scheduled Commands with Cron Daemon	3-6
File System	3-7
Web Server.....	3-8
LOG	3-10
4. Controller Communications	4-1
Internet Configuration	4-2
Modbus TCP Master and Slave.....	4-2
DNS.....	4-2
FTP.....	4-2
Telnet	4-3
SSH.....	4-3
Iptables	4-3
NAT (SNAT)	4-4
Port Forwarding (DNAT)	4-5
PPPoE	4-5
NFS (Network File System) Client.....	4-6
Sending Mail.....	4-7
OpenVPN.....	4-7
IPSec VPN	4-10
NTP Server/Client.....	4-11
Port Trunking.....	4-11
DDNS	4-12
opkg.....	4-12
5. Controller Data Acquisition	5-1
I/O Data Access.....	5-2
Data Backup Memory	5-2
Millisecond Timestamp	5-2
MX-AOPC UA Server.....	5-2
6. Controller Programming.....	6-1
Programming Guide	6-2
Memory Usage.....	6-2
Using C/C++.....	6-2
Linux Toolchain	6-2
GDB	6-3
Library and APIs.....	6-4

A. System Commands..... A-1
Special Moxa UtilitiesA-1

1

Introduction

The following topics are covered in this chapter:

▣ **Overview**

- Scope
- Related Documents
- Supported Products

▣ **Software Architecture**

- Software Package

Overview

In this section, we describe the scope of this document, and list relevant reference documents, and supported products.

Scope

The purpose of this document is to help users set up and configure the ioPAC controller and quickly become familiar with the programming environment. The following topics are covered in this document:

- **Introduction**
- **Getting Started**
- **Controller Management**
- **Controller Communications**
- **Controller Data Acquisition**
- **Controller Programming**

The following information is provided in the Appendix:

- **System Commands**

Related Documents

Additional information about ioPAC controller features is available in the following manuals, which can be downloaded from Moxa's website:

- **ioPAC 8600 Hardware User's Manual**
- **C/C++ Sample Code Programming Guide for ioPAC Programmable Controllers**

Supported Products

In this document, we cover the following products:

- **ioPAC 8600-CPU30 Series**

Software Architecture

The ioPAC CPU30 Programmable Controller uses an ARM Cortex™ A8-based industrial-grade CPU for the system and ARM Cortex™ M4-based CPUs for the modules. It provides up to 4 GB eMMC, 512 MB SDRAM, and a microSD socket (up to 32 GB) for users to install application software and to store data directly on the controller.

The pre-installed operating system (OS) provides an open platform for software program development, which follows a standard Linux-based architecture. Software that runs on desktop PCs can be easily exported to the controller with a cross compiler. Program porting can be done with the toolchain provided by Moxa.

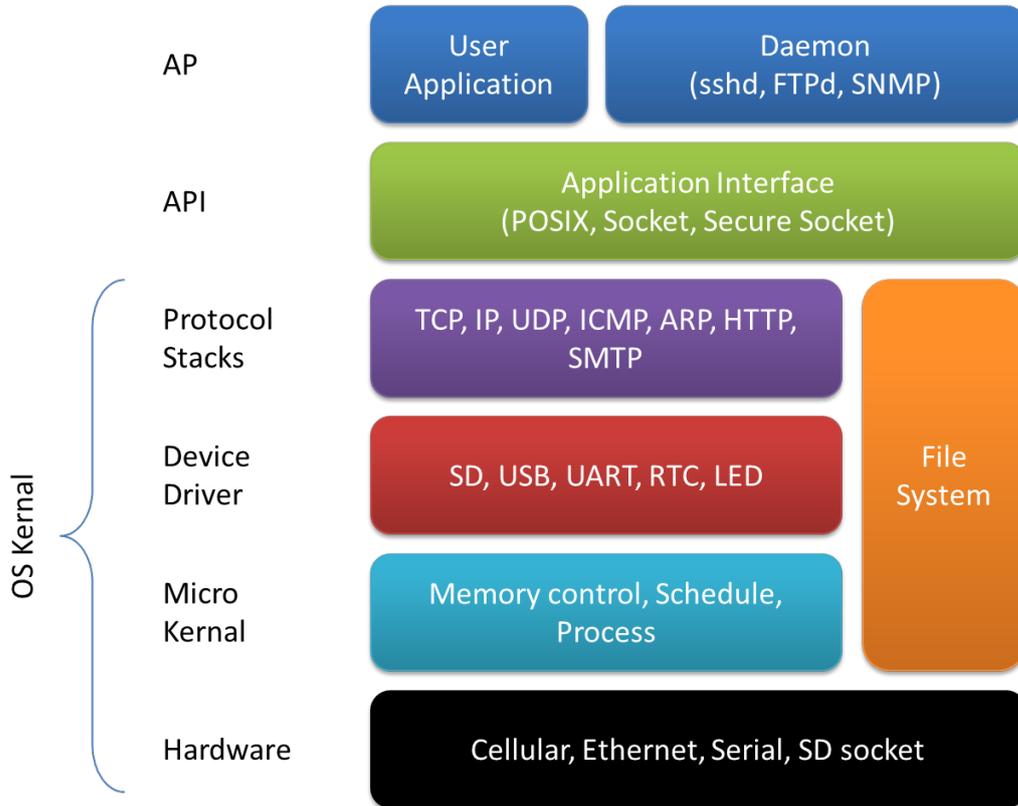
The built-in flash ROM is partitioned into the following directory partitions: Boot Loader, Kernel, Root File System, and User. In order to prevent user applications from crashing the Root File System, the controller uses a unique Root File System with Protected Configuration for emergency use. This Root File System comes with serial and Ethernet communication capability for users to load the Factory Default Image file. User settings and applications are saved in the User directory.

The ioPAC CPU30 uses OverlayFS as the file system when Linux 4.0 is implemented. For more information about OverlayFS, check the following links:

<https://github.com/torvalds/linux/commit/e9be9d5e76e34872f0c37d72e25bc27fe9e2c54c>

<https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/>

For more information about the memory map and programming, refer to the **Programming Guide** section in Chapter 6.



Software Package

System	
Boot loader	ioPAC 8600-CPU30 Boot Loader (v1.0)
Kernel	Linux 4.1.15 with Real-time patch (PREEMPT_RT)
Protocol Stack	ARP, PPP, CHAP, PAP, IPv4, ICMP, TCP, UDP, DHCP, FTP, HTTP, NTP, NFS, SMTP, SSH 1.0/2.0, SSL, Telnet, PPPoE, OpenVPN, TFTP
File System	Ext3, Ext2, Ext4, vfat, msdos, nfs, ntfs, overlay
OS Shell command	Bash
Utilities	
Busybox	Linux normal command utility collection
telnet	Telnet client program
ssh	SSH client program
ftp	FTP client program
smtpclient	Email utility
ntpdate	NTP client program
tftp	TFTP client program
ddns	NO-IP client program
Daemons	
pppd	Dial in/out over serial port daemon
PPPoE	Point-to-Point over Ethernet daemon
telnetd	Telnet server daemon
inetd	TCP server manager program
ftpd	FTP server daemon
sshd	Secure shell server
openvpn	Virtual private network
openssl	Open SSL
ntpd	NTP server daemon
HTTP server	Monkey HTTP server
systemd-journald	system log daemon
Linux Toolchain	
gcc(v5.1.1)	C/C++ PC Cross Compiler
gdb(v7.10.0)	Source Level Debug Server
glibc(v2.21)	POSIX standard C library

Getting Started

This chapter is intended as a quick start guide to help new users set up and configure the ioPAC Programmable Controller quickly, and develop a simple program to run on the ioPAC Programmable Controller.

The following topics are covered in this chapter:

- ❑ **Developing the First Program**
 - Hardware Requirements
 - Development Procedure
- ❑ **Configuring the Ethernet Interface**
 - Modifying Network Settings with the Serial Console
 - Modifying Network Settings over the Network
- ❑ **Connecting an ioPAC Controller to a Linux PC**
- ❑ **Installing the Toolchain on the PC**
- ❑ **Configuring Cross Compiler and glibc Environment Variables**
- ❑ **Developing Code and Compiling a Program**
- ❑ **Upload and Run the Program**
- ❑ **Systemctl Command**

Developing the First Program

In this section, we list the hardware you will need to create your first program, and give an overview of the program development procedure.

Hardware Requirements

The following hardware items are required to complete the first program:

- ❑ 1 x ioPAC Programmable Controller
- ❑ 1 x 24-110 VDC power supply
- ❑ 1 x Ethernet cable
- ❑ 1 x PC or laptop with following minimum requirements
 - CPU: Intel Pentium 4 or above
 - RAM: 512 MB (1024 MB recommended)
 - HDD: at least 200 MB of free space
 - Network Interface: 10/100M Ethernet
 - Linux Operating System (we recommend Ubuntu 14.04)

Development Procedure

Take the following steps to develop your first program:

Step 1: [Connect a Controller to a Linux PC](#)

Step 2: [Install the Toolchain onto the PC](#)

Step 3: [Configure Cross Compiler and glibc Environment Variables](#)

Step 4: [Develop the Code and Compile the Program](#)

Step 5: [Upload and Run the Program](#)

Configuring the Ethernet Interface

The network settings of the ioPAC Programmable Controller can be modified with the ioPAC's serial console or web console.

Modifying Network Settings with the Serial Console

In this section, we show how to use the serial console to configure the network interface. The default IP addresses and port numbers are 192.168.127.254/24 for interface port **eth0** and 192.168.126.254/24 for interface port **eth1**. We illustrate how to configure both static and dynamic IP addresses for interface port eth0.

Static IP address:

1. Deactivate eth0.

```
[moxa@ioPAC8600 ~]$ sudo nmcli c down eth0
Connection 'eth0' successfully deactivated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/1)
```

2. Configure a static IP and gateway for eth0.

```
[moxa@ioPAC8600 ~]$ sudo nmcli c modify eth0 ipv4.method manual ipv4.addresses
192.168.127.254/24 ipv4.gateway 192.168.127.1
```

3. Activate eth0.

```
[moxa@ioPAC8600 ~]$ sudo nmcli c up eth0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/2)
```

Dynamic IP Address:

1. Deactivate eth0.

```
[moxa@ioPAC8600 ~]# sudo nmcli c down eth0
Connection 'eth0' successfully deactivated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/1)
```

2. Configure eth0 to request an IP address dynamically.

```
[moxa@ioPAC8600 ~]# sudo nmcli c modify eth0 ipv4.method auto
```

3. Activate eth0.

```
[moxa@ioPAC8600 ~]# sudo nmcli c up eth0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/2)
```

Modifying Network Settings over the Network

In this section, we show how to use the web console to configure the network interface over an Ethernet connection. The default IP addresses and port numbers are 192.168.127.254/24 for interface port **eth0** and 192.168.126.254/24 for interface port **eth1**. We illustrate how to configure a static IP address for interface port eth0.

Static IP address:

1. Configure a static IP and gateway for eth0.

```
[moxa@ioPAC8600 ~]# sudo nmcli c modify eth0 ipv4.method manual ipv4.addresses
192.168.127.254/24 ipv4.gateway 192.168.127.1
```

2. Reboot the system.

```
[moxa@ioPAC8600 ~]# sudo reboot
```

Connecting an ioPAC Controller to a Linux PC

Take the following steps to establish a connection between the PC and the controller.

Step 1: Use an Ethernet cable to connect the PC's Ethernet port to the ioPAC controller's LAN1 or LAN2 port.

Step 2: Change the PC's IP address as indicated below:

If the ioPAC controller is connected to:	Set PC's IP address to:
LAN1 (default IP: 192.168.127.254)	192.168.127.100
LAN2 (default IP: 192.168.126.254)	192.168.126.100

Step 3: Use the **ssh** command to access the ioPAC's console.

```
[root@localhost ~]# ssh moxa@192.168.127.254
```

Step 4: When a connection between the PC and controller has been established, the following message will be shown on the screen.

```
[root@localhost ~]$ ssh moxa@192.168.127.254
moxa@192.168.127.254's password:
[moxa@ioPAC8600 ~]$
```

Installing the Toolchain on the PC

The controller's GNU toolchain requires approximately 2 GB of hard disk space on your PC. The controller's toolchain software is located on the Document and Software CD. To install the toolchain, insert the CD into your PC's CD-ROM and then issue the following commands:

```
[root@localhost ~]$ mount /dev/cdrom /mnt/cdrom
[root@localhost ~]$ unzip /mnt/cdrom/Software/Toolchain/
ioPAC_CPU30_toolchain_Vx.x.x.x_Buildxxxxxxxx.zip .
[root@localhost ~]$ sh
/mnt/cdrom/Software/Toolchain/ioPAC_CPU30_toolchain_Vx.x.x.x_Buildxxxxxxxx.sh
```

The toolchain will be installed automatically on the host Linux PC within a few minutes.

Configuring Cross Compiler and glibc Environment Variables

Before compiling the program, issue the following commands to change the path to the directory that contains the toolchain files (including the compiler, link, and library). Configuring the path allows you to run the compiler from any directory.

```
[moxa@ioPAC8600 ~]# PATH=$PATH:/usr/local/arm-linux/bin:/usr/sbin
[moxa@ioPAC8600 ~]# export PATH
[moxa@ioPAC8600 ~]# LD_LIBRARY_PATH=/usr/local/arm-linux/tools/lib:/usr/lib
[moxa@ioPAC8600 ~]# export LD_LIBRARY_PATH
```

Developing Code and Compiling a Program

Use the **hello_world.c** sample program and makefile from the Software and Document CD to learn how to develop code and compile a program on the ioPAC controller.

Type the following commands from the host PC to copy the files used for this example.

```
[moxa@ioPAC8600 ~]# cd /tmp/
root@moxa:/tmp# mkdir example
root@moxa:/tmp# cp -r /mnt/cdrom/Software/Sample/hello_world/* /tmp/example
```

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

```
[moxa@ioPAC8600 ~]# cd/tmp/example/hello_world
root@moxa:/tmp/example/hello_world# make
```

The following response should be shown on the screen.

```
root@moxa:/tmp/example/hello_world# make
/usr/local/arm-linux/bin/arm-linux-gcc -I/usr/local/arm-linux/include -
I/usr/local/arm-linux/include/RTU -o hello_world hello_world.o -L/usr/local/arm-
linux/lib -lpthread -lmxml -L/usr/local/arm-linux/lib/RTU -lmoxa_rtu -lrtu_common
-ltag -Wl,-rpath,/lib/RTU/ -Wl,--allow-shlib-undefined
```

The hello_world-release and hello_world-debug executable files are described below:

- hello_world-release: an ARM platform executable file (created specifically to run on the Moxa Controllers)
- hello_world-debug: an ARM platform GDB debug server executable file

NOTE Since Moxa's toolchain places a specially designed makefile in the `/tmp/example/hello_world` directory, be sure to type the `#make` command from within that directory. If you type the `#make` command from any other directory, the host Linux PC might use other system compilers (e.g., `cc` or `gcc`), resulting in errors.



ATTENTION

You must use the `sudo` command to run the programs that use Moxa library for the ioPAC 8600. Or, you can **switch to the root account** before you run programs that use Moxa library for the ioPAC 8600.

Upload and Run the Program

In this section, we give step-by-step instructions that describe how to upload and run the `hello_world.c` program. Before starting the process, set up the networking environment as follows:

- ioPAC 8600 IP: 192.168.127.254
- localhost IP: 192.168.127.1
- Make sure that the ioPAC 8600 and localhost can connect to each other over the network.

1. Start `vftpd` service on the ioPAC 8600.

```
[moxa@ioPAC8600 ~]$ sudo systemctl start vsftpd.service
```

2. Open the directory where the program will be saved on localhost.

```
[root@localhost ~]$ ls -hl
total 12
-rwxr-xr-x  1 root  root   10.1K Jan  8 00:34 hello_world
```

3. Upload the program from localhost to the ioPAC8600.

```
[root@localhost ~]$ ftp 192.168.127.254
Connected to 192.168.127.254.
220 (vsFTPd 3.0.3)
Name (192.168.127.254:root): moxa
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put hello_world
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
10340 bytes sent in 5.5e-05 seconds (179 Mbytes/s)
ftp> quit
221 Goodbye.
```

4. Open the directory where the program will be saved on the ioPAC8600.

```
[moxa@ioPAC8600 ~]$ ls -hl
total 12
-rw-----  1 moxa  moxa   10.1K Jan  8 00:39 hello_world
```

5. Change the program to an executable file.

```
[moxa@ioPAC8600 ~]$ chmod +x hello_world
```

6. Run the program.

```
[moxa@ioPAC8600 ~]$ ./hello_world  
Hello World
```

Systemctl Command

The following commands can be used with the ioPAC 8600 CPU30 Series to execute some services without doing any settings in advance. To control a service, type the respective command after

[moxa@ioPAC8600 ~].

- list installed unit files
systemctl list-unit-files
- start the service
systemctl start [unit.service]
- stop the service
systemctl stop [unit.service]
- restart the service
systemctl restart [unit.service]
- enable the service to start on bootup
systemctl enable [unit.service]
- disable the service from starting on bootup
systemctl disable [unit.service]
- show service status
systemctl status [unit.service]

Controller Management

The following topics are covered in this chapter:

- ❑ **System Information**
- ❑ **Firmware Upgrade and Default Settings**
 - Upgrading the Firmware
 - Upgrading the Firmware from the SD Card
 - Loading Factory Defaults
- ❑ **Enabling and Disabling Daemons**
- ❑ **System Time Setting**
 - Setting the Time Manually
 - Using NTP Client to update the Time
 - Configuring the NTP Server
- ❑ **Executing Scheduled Commands with Cron Daemon**
- ❑ **File System**
- ❑ **Web Server**
- ❑ **LOG**

System Information

To determine the hardware capability of your ioPAC Programmable Controller and supported software functions, check the version numbers of your ioPAC controller hardware, kernel, and user file system. When contacting Moxa to determine the hardware version, you will need the Production S/N (serial number), which is located on the ioPAC's label.

To check the kernel version, type:

#kversion -a

```
[moxa@ioPAC8600 ~]$ sudo kversion -a
RTU version 1.0.0 build 2017/4/12:15
API version 1.0.0 build 2017/4/24:22
CPU Serial Number: 000000000000
CPU HW Version: 0.0.0.0
Backplane Serial Number: 000000000000
Backplane HW Version: 1.0.0.0
BIOS Version: 1.0.0.0
BIOS Builddate: 05/15/2017/11:42:01

Slot0 Module Info:
VID = 0x110a, PID = 0x5400, SerialNumber = *****
Hardware Version = 1.0.0, Firmware Version = 1.0.0 build 2015/2/11 18
```

NOTE The kernel version number is used for factory default configuration. You may download the latest firmware version from Moxa's website and then upgrade the ioPAC controller.

Firmware Upgrade and Default Settings

Upgrading the Firmware

The ioPAC CPU30's bios, kernel, and root file system are combined into one firmware file, which can be downloaded from Moxa's website, www.moxa.com. The name of the file has the form **FWR_ioPAC_CPU30_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 ioPAC controller using a console port or SSH console connection.



ATTENTION

Upgrading the firmware will erase all data on the flash ROM

If you are using the RAM disk to store code for your applications, beware that updating the firmware will erase all of the data on the flash ROM. You should back up your application files and data before updating the firmware.

Below we give step-by-step instructions that describe how to upgrade the firmware. Before starting the process, set up the networking environment as follows:

- ioPAC 8600: 192.168.127.254
- localhost: 192.168.127.1
- make sure the ioPAC 8600 and localhost can connect to each other over the network

1. Start the vsftpd service on the ioPAC 8600.

```
[moxa@ioPAC8600 ~]$ sudo systemctl start vsftpd.service
```

2. Open the directory where the firmware is placed on localhost.

```
[root@localhost ~]$ ls -hl
total 59564
-rw-r--r-- 1 root root 58.2M Jan 7 23:57
FWR_ioPAC8600_CPU30_V1.0.0.2_Build17070414_STD.hfm
```

3. Upload the firmware from localhost to the ioPAC8600.

```
[root@localhost ~]$ ftp 192.168.127.254
Connected to 192.168.127.254.
220 (vsFTPd 3.0.3)
Name (192.168.127.254:root): moxa
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put FWR_ioPAC8600_CPU30_V1.0.0.2_Build17070414_STD.hfm
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
60992896 bytes sent in 5.19 seconds (11.2 Mbytes/s)
ftp> quit
221 Goodbye.
```

4. Open the directory where the firmware will be saved on the ioPAC8600.

```
[moxa@ioPAC8600 ~]$ ls -hl
[moxa@ioPAC8600 ~]$ ls -hl
total 59564
-rw----- 1 moxa moxa 58.2M Jan 8 00:11
FWR_ioPAC8600_CPU30_V1.0.0.2_Build17070414_STD.hfm
```

5. Upgrade the ioPAC8600's firmware.

```
[moxa@ioPAC8600 ~]$ sudo upgradefwm FWR_ioPAC8600_CPU30_V1.0.0.2_Build17070414_STD.hfm
Upgrade firmware utility version 1.4.
Verifying firmware file context...
The firmware file context is OK.
This step will destroy all your firmware.
Continue ? (Y/N) :
Decrypt firmware file context...
CMD:gpg2 --yes --output decfile --decrypt encfile.gpg
gpg: encrypted with 1024-bit RSA key, ID DADEA8C9, created 2016-12-16
"Moxa Dac <dac@moxa.com>"
The output file [RESCUE_ioPAC8600_CPU30] file length = 34603008, flash length = 34603008,
checksum = 0x3c23296c
Now change output file offset [64] to write output file header.
...
Whole HFM CRC32:0xc85f5259(without HFM head)
Change output file offset to [0] for updating firmware header.
Finish headerlength=64, totalfileno=24, checksum=0xc85f5259,
totallength=269069568(without HFM head)
```

NOTE Make sure the power supply is operating normally while upgrading the firmware. A power failure could cause the firmware upgrade to fail, and could even damage the device.

NOTE The ioPAC 8600-CPU30 will enter recovery mode when a firmware upgrade fails. During the recovery process, the ioPAC 8600-CPU30 will do a firmware upgrade three times automatically; the RDY LED will be green and blinking. If the recovery is successful, the RDY LED will change to a solid green, at which point the ioPAC 8600-CPU30 is ready to use. If recovery fails, the RDY will change to solid red. If the recovery process fails three times, please contact Moxa for assistance.

Upgrading the Firmware from the SD Card

The ioPAC CPU30 firmware can also be upgraded from the SD card. The firmware file is encrypted and will need to be unencrypted before transferring the file to the SD card. Take the following steps to proceed:

1. Insert the microSD card into the ioPAC. See the section **Installing a microSD Card** in the ioPAC 8600 HW User's Manual for installations on how to install the microSD card.

Note: The microSD card should be FAT32 formatted.

2. Follow steps 1 to 4 in the **Upgrading the Firmware** section of this manual to upload the firmware file into the ioPAC 8600.

Note: You may use a single ioPAC 8600 device to unencrypt the firmware, and then install the unencrypted firmware on one or more other ioPAC 8600 devices.

3. Use the following commands to unencrypt the firmware:

```
[moxa@ioPAC8600 ~]$ sudo upgradehfm -b moxa-iopac-8600-series-cpu30-firmware-
v1.2.hfm
Upgrade firmware utility version 1.4.
Verifying firmware file context
...
```

4. Use the following commands to copy the unencrypted firmware file to the microSD card:

```
[moxa@ioPAC8600 ~]$ sudo cp /upgrade/rtu.hfm /mnt/sd
[moxa@ioPAC8600 ~]$ sync
[moxa@ioPAC8600 ~]$ sudo umount /mnt/sd
```

5. Power off the ioPAC 8600.
6. Turn the rotary switch to position 9
7. Power up the ioPAC 8600. The ready (RDY) LED will blink green while the firmware is upgrading.
8. The ready (RDY) LED will turn to steady green when the upgrade is complete and successful. If the RDY LED changes to red, the firmware upgrade failed. In this case, check to make sure the filename is correct (the correct filename is: rtu.hfm).
9. Turn the rotary switch to position 0 and then restart the ioPAC 8600 to finish the firmware upgrade process.

Loading Factory Defaults

Use the following procedure to reset the ioPAC to factory defaults. Note that when you reset the ioPAC, all of your tag definitions, software programs, and files will be deleted, and the service and runtime engine will be restarted.

ioPAC 8600-CPU30 Series

1. Turn the rotary switch to position 0.
2. When the system is booting up and the **RDY** LED is blinking GREEN, hold the reset button.
3. Continue to hold the reset button until the **RDY** LED turns a solid RED, and then release the reset button. It will take around 90 seconds to complete the factory reset process.
4. When the **RDY** LED starts blinking GREEN (indicating that the kernel is rebooting), factory mode is ready.

Enabling and Disabling Daemons

The ioPAC 8600-CPU30 uses `systemctl` to enable and disable daemons. Refer to the **Systemctl command** section in this manual for details.

System Time Setting

Setting the Time Manually

The ioPAC Programmable Controller has two time settings: the system time, and the RTC (Real Time Clock) time kept by the ioPAC hardware. Use the **#date** command to query the current system time or set a new system time. Use the **#hwclock** command 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 YYYY.MM.DD-hh:mm:ss

YYYY = Year

MM = Month

DD = Date

hh = hour

mm = minute

ss = second

Use the following command to set the RTC time:

#hwclock -w

Below, we illustrate how to update the system time and set the RTC time.

```
[moxa@ioPAC8600 ~]$ date; sudo hwclock
Sat Jan 1 00:00:13 UTC 2000
Sat Jan 1 00:00:13 2000 0.000000 seconds
[moxa@ioPAC8600 ~]$ sudo date 2017.01.01-00:00:00; sudo hwclock -w
Sun Jan 1 00:00:00 UTC 2017
[moxa@ioPAC8600 ~]$ date; sudo hwclock
Sun Jan 1 00:00:05 UTC 2017
Sun Jan 1 00:00:05 2017 0.000000 seconds
```

Using NTP Client to update the Time

The ioPAC Programmable Controller has a built-in NTP (Network Time Protocol) client that is used to initialize a time request to a remote NTP server. Use the following command to synchronize the time with the NTP server.

```
[moxa@ioPAC8600 ~]# sudo ntpdate time.nist.gov
```

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

Configuring the NTP Server

To configure the ioPAC's NTP server, first configure the time sync server at the path `/etc/ntp.conf`, and then use the `/etc/init.d/ntpd start` command to start the NTP server.

```
[moxa@ioPAC8600 ~]# sudo systemctl start ntpd.service
```

NOTE `ntpd` and `ntpdate` cannot be used at the same time because both services share the same UDP port 123. Remember to use `hwclock-w` to update the RTC time. Otherwise, the time sync settings will be lost after you restart the device.

Executing Scheduled Commands with Cron Daemon

Crond wakes up every minute and checks each command to see if it should be run at that time. Crontab files have the following format:

Definition	min	hour	date	month	week	command
Range	0-59	0-23	1-31	1-12	0-7 (0 or 7 is Sunday)	

The following steps illustrate how to configure Crond to execute the `date` command once every minute.

1. Start the cron service.

```
[moxa@ioPAC8600 ~]$ sudo systemctl start crond.service
```

2. Set the `date` command to run once every minute in `crontab` file format.

```
[moxa@ioPAC8600 ~]$ crontab -e
```

```
* * * * * /usr/bin/date
```

3. Check the results of the `date` execution.

```
[moxa@ioPAC8600 ~]$ sudo journalctl -u crond.service
```

```
-- Logs begin at Mon 2017-01-02 22:06:08 UTC, end at Mon 2017-01-02 22:23:24 UTC. --
Jan 02 22:07:39 ioPAC8600 systemd[1]: Started Periodic Command Scheduler.
Jan 02 22:10:00 ioPAC8600 crond[353]: Mon Jan  2 22:10:00 UTC 2017
Jan 02 22:11:00 ioPAC8600 crond[353]: Mon Jan  2 22:11:00 UTC 2017
Jan 02 22:12:00 ioPAC8600 crond[353]: Mon Jan  2 22:12:00 UTC 2017
Jan 02 22:13:00 ioPAC8600 crond[353]: Mon Jan  2 22:13:00 UTC 2017
```

File System

ioPAC controllers support two file system tools: **e2fsprogs** and **dosfstools**. The following examples illustrate how to use these tools.

Example 1: Using e2fsprogs to format an SD card to ext4

1. Display all storage mount points.

```
[moxa@ioPAC8600 ~]# df -Th
```

Filesystem	Type	Size	Used	Available	Use%	Mounted on
None	devtmpfs	234.2M	0	234.2M	0%	/dev
Overlay	overlay	1.9G	22.4M	1.8G	1%	/
Tmpfs	mpfs	246.7M	20.0K	246.6M	0%	/dev/shm
Tmpfs	tmpfs	246.7M	580.0K	246.1M	0%	/run
/dev/mmcblk1p1	vfat	7.2G	4.0K	7.2G	0%	/mnt/sd

2. Unmount the SD card.

```
[moxa@ioPAC8600 ~]# sudo umount /mnt/sd
```

3. Format the SD card to ext4.

```
[moxa@ioPAC8600 ~]# sudo mke2fs -t ext4 /dev/mmcblk1p1
mke2fs 1.42.13 (17-May-2015)
/dev/mmcblk1p1 contains a vfat file system
Proceed anyway? (y,n) y
Discarding device blocks: done
Creating filesystem with 1898496 4k blocks and 475136 inodes
Filesystem UUID: 7815f0b8-f62b-4b7b-ac1c-0a8bbe1df59c
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

4. Mount the SD card.

```
[moxa@ioPAC8600 ~]# sudo mount /dev/mmcblk1p1 /var/sd
```

5. Check that the SD card is formatted to ext4.

```
[moxa@ioPAC8600 ~]# df -Th
```

Filesystem	Type	Size	Used	Available	Use%	Mounted on
None	devtmpfs	234.2M	0	234.2M	0%	/dev
Overlay	overlay	1.9G	22.4M	1.8G	1%	/
Tmpfs	tmpfs	246.7M	20.0K	246.6M	0%	/dev/shm
Tmpfs	tmpfs	246.7M	580.0K	246.1M	0%	/run
/dev/mmcblk1p1	ext4	7.2G	4.0K	7.2G	0%	/mnt/sd

Example 2: Using dosfstools to format an SD card to ext4

1. Display all storage mount points.

```
[moxa@ioPAC8600 ~]# df -Th
```

Filesystem	Type	Size	Used	Available	Use%	Mounted on
None	devtmpfs	234.2M	0	234.2M	0%	/dev
Overlay	overlay	1.9G	22.4M	1.8G	1%	/
Tmpfs	tmpfs	246.7M	20.0K	246.6M	0%	/dev/shm
Tmpfs	tmpfs	246.7M	580.0K	246.1M	0%	/run
/dev/mmcblk1p1	ext4	7.2G	4.0K	7.2G	0%	/mnt/sd

2. Unmount the SD card.

```
[moxa@ioPAC8600 ~]# sudo umount /mnt/sd
```

3. Format the SD card to vfat.

```
[moxa@ioPAC8600 ~]# sudo mkdosfs /dev/mmcblk1p1
mkfs.fat 3.0.28 (2015-05-16)
```

4. Mount the SD card.

```
[moxa@ioPAC8600 ~]# sudo mount /dev/mmcblk1p1 /var/sd
```

5. Check that the SD card is formatted to ext4.

```
[moxa@ioPAC8600 ~]# df -Th
```

Filesystem	Type	Size	Used	Available	Use%	Mounted on
None	devtmpfs	234.2M	0	234.2M	0%	/dev
Overlay	overlay	1.9G	22.4M	1.8G	1%	/
Tmpfs	tmpfs	246.7M	20.0K	246.6M	0%	/dev/shm
Tmpfs	tmpfs	246.7M	580.0K	246.1M	0%	/run
/dev/mmcblk1p1	vfat	7.2G	4.0K	7.2G	0%	/mnt/sd

Web Server

ioPAC controllers have a built-in web server tool for developing applications. In this section, we explain how to use the web server tool.

http

```
[moxa@ioPAC8600 ~]# sudo systemctl start monkey.service
```

https

1. Include the monkey-tls.so plugin.

```
[moxa@ioPAC8600 ~]# sudo vi /etc/monkey/plugins.load
```

```
[PLUGINS]
# Load /usr/lib/monkey-auth.so
# Load /usr/lib/monkey-cgi.so
# Load /usr/lib/monkey-cheetah.so
# Load /usr/lib/monkey-dirlisting.so
# Load /usr/lib/monkey-fastcgi.so
# Load /usr/lib/monkey-logger.so
# Load /usr/lib/monkey-mandrill.so
Load /usr/lib/monkey-tls.so
# Load /usr/lib/monkey-duda.so
```

2. Modify the listen port.

```
[moxa@ioPAC8600 ~]# sudo vi /etc/monkey/monkey.conf

[SERVER]
  Listen 443 tls
  Workers 0
  Timeout 15
  PidFile /var/run/monkey.pid
  UserDir public_html
  Indexfile index.html index.htm index.php
  HideVersion Off
  Resume On
  User www-data
  KeepAlive On
  KeepAliveTimeout 5
  MaxKeepAliveRequest 1000
  MaxRequestSize 32
  SymLink Off
  DefaultMimeType text/plain
  FDT On
  OverCapacity Resist
  # FDLimit 4096
```

3. Open the tls directory.

```
[moxa@ioPAC8600 ~]# cd /etc/monkey/plugins/tls/
```

4. Check the tls.conf settings.

```
[moxa@ioPAC8600 tls]# cat tls.conf

[TLS]
  CertificateFile srv_cert.pem
  # CertificateChainFile srv_cert_chain.pem
  RSAKeyFile rsa_key.pem
  DHParameterFile dhparam.pem
```

5. Use **rsa_key.pem** to generate RSAKeyFile.

```
[moxa@ioPAC8600 tls]# sudo openssl genrsa -out rsa_key.pem 1024
Generating RSA private key, 1024 bit long modulus
..+++++
...+++++
e is 65537 (0x10001)
```

6. Use *srv_cert.pem* to generate CertificateFile.

```
[moxa@ioPAC8600 tls]# sudo openssl req -new -x509 -key rsa_key.pem -out
srv_cert.pem -days 1095

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:TW
State or Province Name (full name) [Some-State]:TAIPEI
Locality Name (eg, city) []:TAIPEI
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MOXA
Organizational Unit Name (eg, section) []:DAC
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

7. Use *dhparam.pem* to generate DHParameterFile.

```
[moxa@ioPAC8600 tls]# sudo openssl dhparam -out dhparam.pem 2048
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time
.....+*+*+*
```

8. Start **monkey service**.

```
[moxa@ioPAC8600 ~]# sudo systemctl start monkey.service
```

LOG

The following example shows the log of **sshd.service**.

```
[moxa@ioPAC8600 ~]# sudo journalctl -u sshd.service

-- Logs begin at Fri 2000-01-14 21:34:32 UTC, end at Fri 2000-01-14 21:35:42 UTC
. --
Jan 14 21:34:38 ioPAC8600 systemd[1]: Starting OpenSSH server daemon...
Jan 14 21:34:48 ioPAC8600 ssh-keygen[271]: ssh-keygen: generating new host keys:
RSA DSA ECDSA ED25519
Jan 14 21:34:48 ioPAC8600 systemd[1]: Started OpenSSH server daemon.
Jan 14 21:34:48 ioPAC8600 sshd[324]: Server listening on 0.0.0.0 port 22.
```

Additional information is available here:

<https://www.freedesktop.org/software/systemd/man/journalctl.html>

Controller Communications

In this chapter, we explain how to configure the ioPAC controller various communication functions.

The following topics are covered in this chapter:

❑ Internet Configuration

- Modbus TCP Master and Slave
- DNS
- FTP
- Telnet
- SSH
- Iptables
- NAT (SNAT)
- Port Forwarding (DNAT)
- PPPoE
- NFS (Network File System) Client
- Sending Mail
- OpenVPN
- IPsec VPN
- NTP Server/Client
- Port Trunking
- DDNS
- opkg

Internet Configuration

Modbus TCP Master and Slave

Since the Modbus TCP communication protocol is used for many industrial applications, the ioPAC Programmable Controller has built-in Modbus TCP master and slave functions. For detailed information, refer to the following Moxa document:

C/C++ Sample Code Programming Guide for ioPAC Programmable Controllers.

DNS

The ioPAC controllers can be configured as a DNS client but not as a DNS server.

1. Deactivate eth0.

```
[moxa@ioPAC8600 ~]# sudo nmcli c down eth0
Connection 'eth0' successfully deactivated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/1)
```

2. Configure eth0 to request an IP address dynamically.

```
[moxa@ioPAC8600 ~]# sudo nmcli c modify eth0 ipv4.dns "8.8.8.8 8.8.4.4"
```

3. Active eth0.

```
[moxa@ioPAC8600 ~]# sudo nmcli c up eth0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/2)
```

FTP

In this section, we explain how to use the three FTP commands supported by the ioPAC: **ncftp**, **tftp**, and **sftp**. In the following example, 192.168.127.1 is the IP address of the FTP server.

ncftp

Connect to an FTP server.

```
[moxa@ioPAC8600 ~]# ncftp -u <ftp username> -p <ftp password> 192.168.127.1
```

Download a remote file from an FTP server.

```
[moxa@ioPAC8600 ~]# ncftpget -u <ftp username> -p <ftp password> 192.168.127.1
<local-directory> <Remote FILE>
```

Upload a local file to an FTP server.

```
[moxa@ioPAC8600 ~]# ncftpput -u <ftp username> -p <ftp password> 192.168.127.1
<remote-directory> <Local FILE>
```

tftp

Download a remote file from an FTP server.

```
[moxa@ioPAC8600 ~]# tftp -gr <Remote FILE> 192.168.127.1
```

Upload a local file to an FTP server.

```
[moxa@ioPAC8600 ~]# tftp -pl <Local FILE> 192.168.127.1
```

sftp

Connect to an FTP server.

```
[moxa@ioPAC8600 ~]# sftp <ftp username>@192.168.127.1
```

Telnet

The ioPAC 8600-CPU30 has a built-in telnet server (telnetd) and client (telnet) service. You can use the service to connect other hosts to the ioPAC 8600-CPU30. Since the telnet server is disabled by default, you must enable the daemon before using it.

Enable the telnet server.

```
[moxa@ioPAC8600 ~]# sudo systemctl start telnetd.service
```

Connect to otherhost from the ioPAC 8600-CPU30.

```
[moxa@ioPAC8600 ~]# telnet 192.168.127.1
```

SSH

The ioPAC 8600-CPU30 has a built-in SSH server (sshd) and client (ssh) service. You can use the service to connect the ioPAC 8600 CPU30 to otherhost.

Enable the SSH server.

```
[moxa@ioPAC8600 ~]# sudo systemctl start sshd.service
```

Connect to otherhost from the ioPAC 8600-CPU30.

```
[moxa@ioPAC8600 ~]# ssh <ssh username>@192.168.127.1
```

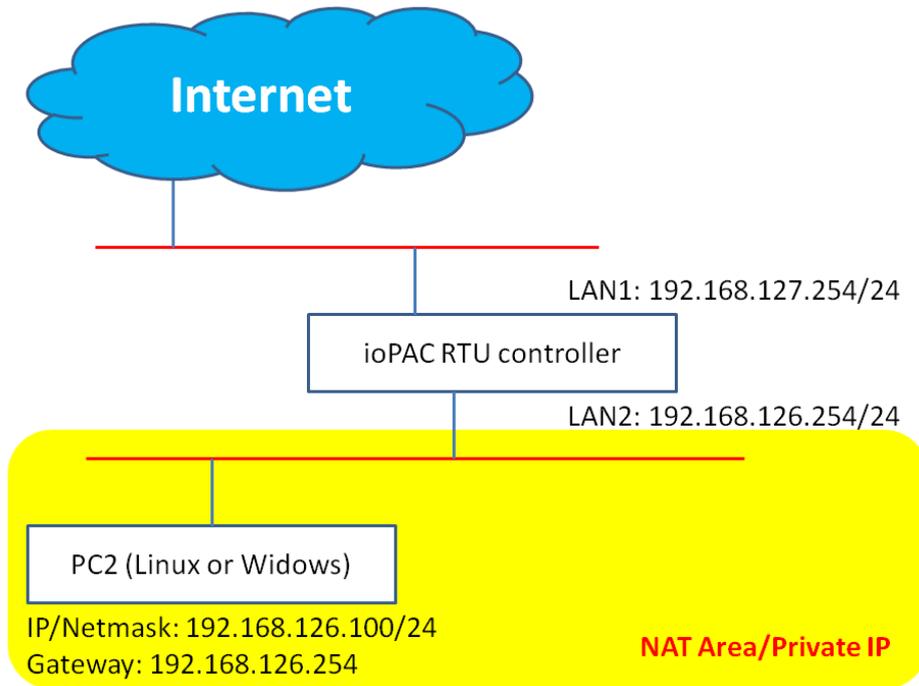
Iptables

The iptables command 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, and 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."

NAT (SNAT)

The 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 is the outside network. Typically, ioPAC programmable controllers are connected to several devices on a network. They map local inside network addresses to one or more global outside IP addresses, and un-map global IP addresses on incoming packets back into local IP addresses. Take the following step to configure SNAT.



Step 1: Enable IP Forwarding.

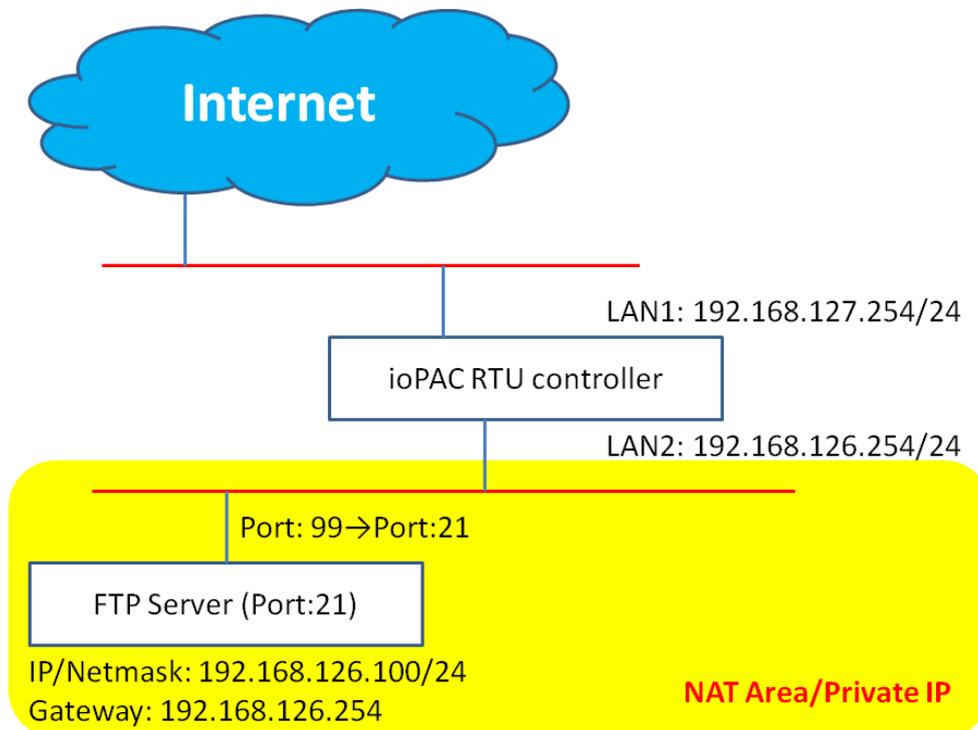
```
[moxa@ioPAC8600 ~]# sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Step 2: Config SNAT to NAT table.

```
[moxa@ioPAC8600 ~]# sudo iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 192.168.127.254
```

Port Forwarding (DNAT)

Port forwarding allows remote computers (e.g., computers on the Internet) to connect to a specific computer or service within a private local-area network (LAN). The following is an FTP port forwarding example.



Step 1: Enable IP Forwarding.

```
[moxa@ioPAC8600 ~]# sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Step 2: Config DNAT to NAT table.

```
[moxa@ioPAC8600 ~]# sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 99
-j DNAT --to-destination 192.168.126.100:21
```

PPPoE

1. Connect the ioPAC Programmable Controller LAN port to an ADSL modem with a cross-over cable, hub, or switch.
2. Log in to the ioPAC Programmable Controller as the root user.
3. Deactivate eth0.

```
[moxa@ioPAC8600 ~]# sudo nmcli c down eth0
Connection 'eth0' successfully deactivated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/1)
```

4. Add a pppoe connection with ifname (eth0) and ISP account for NetworkManager.

```
[moxa@ioPAC8600 ~]# sudo nmcli c add type pppoe ifname eth0 username <PPPoE username>
password <PPPoE password>
```

5. Check the connection name (pppoe-eth0) of the pppoe added in the previous step.

```
[moxa@ioPAC8600 ~]# sudo nmcli c show
NAME                UUID                                TYPE          DEVICE
pppoe-eth0          57c602de-7f2d-43c7-b51d-fd674cb0a5a5  pppoe         --
eth0                7286c513-4600-47e1-838f-9061b2f95e47  802-3-ethernet eth0
eth1                8eac9c2f-4184-4bf6-9161-057acc3b3494  802-3-ethernet --
```

6. Activate pppoe-eth0.

```
[moxa@ioPAC8600 ~]# sudo nmcli c up pppoe-eth0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/2)
```

NFS (Network File System) Client

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 ioPAC Programmable Controllers, without worrying about the amount of disk space that will be available. The ioPAC Programmable Controller supports NFS protocol for clients.

NOTE More information about NFS is available at:
<http://www.tldp.org/HOWTO/NFS-HOWTO/index.html>
<http://nfs.sourceforge.net/nfs-howto/client.html>
<http://nfs.sourceforge.net/nfs-howto/server.html>

Setting Up the ioPAC Programmable Controller as an NFS Client

Use the following procedure to mount a remote NFS Server.

- ioPAC8600: 192.168.127.254
- NFS_Server: 192.168.127.1
- make sure ioPAC8600 and NFS_Server can connect to each other over the network.

1. Find the NFS Server's shared directory.

```
[moxa@ioPAC8600 ~]# showmount -e 192.168.127.1
Export list for 192.168.127.1:
/home/nfs *
```

2. Establish a mount point on the NFS client site.

```
[moxa@ioPAC8600 ~]# mkdir /root/nfs
```

3. Mount the remote directory to a local directory.

```
[moxa@ioPAC8600 ~]# sudo mount -t nfs 192.168.127.1:/home/nfs /root/nfs
```

Sending Mail

msmtp 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.

NOTE More information about **smtplib** is available at:
<https://wiki.archlinux.org/index.php/Msmtp>

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

Example:

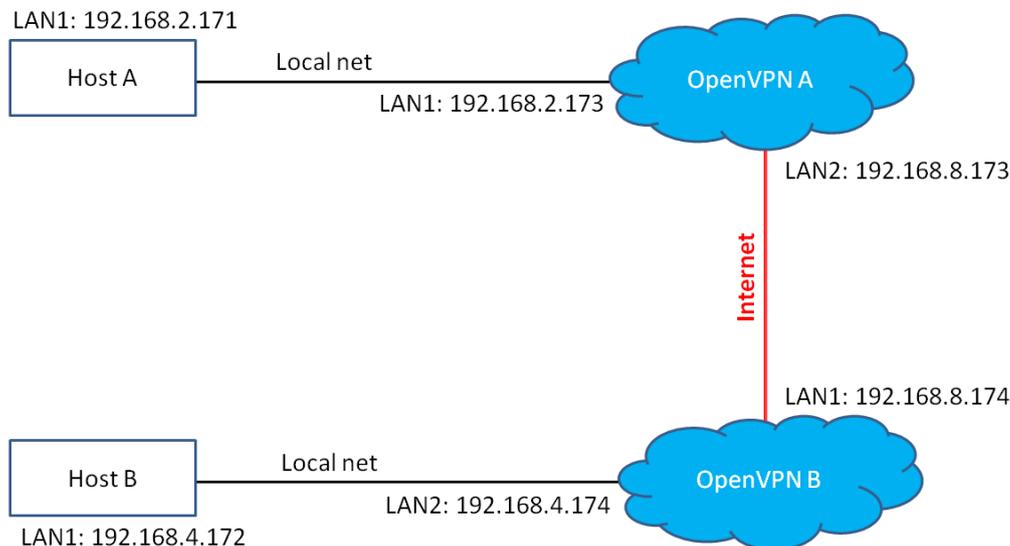
msmtp --host=hostname --read-envelope-from address_receiver@moxa.com

OpenVPN

OpenVPN provides two types of tunnels for users to implement VPNs: **Routed IP Tunnels** and **Bridged Ethernet Tunnels**.

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.

Example: Configure OpenVPN 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 connected to another device (such as a firewall or DSL box) first.

Routed VPN Tunnels

1. Generate a static secret key, **secrouter.key**, on OpenVPN A.

```
[moxa@ioPAC8600 ~]$ sudo openvpn --genkey --secret /etc/openvpn/secrouter.key
```

2. Copy **secrouter.key** from OpenVPN A to OpenVPN B over a pre-existing secure channel.

```
[moxa@ioPAC8600 ~]$ sudo scp /etc/openvpn/secrouter.key moxa@192.168.8.174:~
moxa@192.168.8.174's password:
```

3. Log in to OpenVPN B and then copy **secrouter.key** to the directory **/etc/openvpn/**.

```
[moxa@ioPAC8600 ~]$ sudo cp secrouter.key /etc/openvpn/
[moxa@ioPAC8600 ~]$ rm -f secrouter.key
```

4. Modify the configuration file **tun.conf**.

```
[moxa@ioPAC8600 ~]$ sudo vi /etc/openvpn/tun.conf

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

script-security 3 system
ifconfig 192.168.2.173 192.168.4.174
up /etc/openvpn/tun.sh

#comp-lzo
```

5. Modify the executable script file **tun.sh**.

```
[moxa@ioPAC8600 ~]$ sudo vi /etc/openvpn/tun.sh

#!/bin/sh
route add -net 192.168.4.0 netmask 255.255.255.0 gw $5
```

6. Execute the following script to set up a routed VPN tunnel environment.

```
[moxa@ioPAC8600 ~]$ sudo /etc/openvpn/openvpn-bridge env
Setup openvpn environment.
load modules tun ok.
enable ip forwarding ok.
```

7. Start OpenVPN with **tun.conf**.

```
[moxa@ioPAC8600 ~]$ sudo openvpn /etc/openvpn/tun.conf &
```

Bridged Ethernet Tunnels

1. Generate a static secret key, **secrouter.key**, on OpenVPN A.

```
[moxa@ioPAC8600 ~]$ sudo openvpn --genkey --secret /etc/openvpn/secrouter.key
```

2. Copy **secrouter.key** from OpenVPN A to OpenVPN B over a pre-existing secure channel.

```
[moxa@ioPAC8600 ~]$ sudo scp /etc/openvpn/secrouter.key moxa@192.168.8.174:~
moxa@192.168.8.174's password:
```

3. Log in to OpenVPN B and then copy **secrouter.key** to the directory **/etc/openvpn/**.

```
[moxa@ioPAC8600 ~]$ sudo cp secrouter.key /etc/openvpn/
[moxa@ioPAC8600 ~]$ rm -f secrouter.key
```

4. Modify the configuration file **tap0-br.conf**.

```
[moxa@ioPAC8600 ~]$ sudo vi /etc/openvpn/tun.conf

[moxa@ioPAC8600 ~]$ sudo vi /etc/openvpn/tap0-br.conf

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
ping-restart 120

script-security 3 system
up /etc/openvpn/tap0-br.sh

#comp-lzo
```

5. Modify the executable script file **tap0-br.sh**.

```
[moxa@ioPAC8600 ~]$ sudo vi /etc/openvpn/tap0-br.sh

#!/bin/sh
route add -net 192.168.4.0 netmask 255.255.255.0 dev br0
```

6. Execute the following script to set up a bridge Ethernet tunnel environment.

```
[moxa@ioPAC8600 ~]$ sudo /etc/openvpn/openvpn-bridge start
Setup openvpn bridge.
Setup openvpn environment.
load modules tun ok.
enable ip forwarding ok.
Thu Jan 20 16:46:30 2000 TUN/TAP device tap0 opened
Thu Jan 20 16:46:30 2000 Persist state set to: ON
```

7. Start OpenVPN with **tap0-br.conf**.

```
[moxa@ioPAC8600 ~]$ sudo openvpn /etc/openvpn/tap0-br.conf &
```

IPSec VPN

In this section, we explain how to use IPsec, another widely used VPN protocol.

1. Enable IP Forwarding.

```
[moxa@ioPAC8600 ~]# sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

2. Modify the configuration file *ipsec.conf*.

```
[moxa@ioPAC8600 ~]# sudo vi /etc/ipsec.conf

version 2.0

config setup
    dumpdir=/var/run/pluto/
    nat_traversal=yes
    oe=off
    protostack=auto
    #plutostderrlog=/dev/null

conn ipsec
    authby=secret
    left=192.168.8.173
    leftsourceip=192.168.2.173
    leftnexthop=0.0.0.0
    leftsubnet=192.168.2.0/24
    right=192.168.8.174
    rightsubnet=192.168.4.0/24
    type=tunnel
    auth=esp
    esp=3des-sha1
    rekeymargin=9m
    rekeyfuzz=100%
    keyingtries=%forever
    keyexchange=ike
    ikelifetime=1h
    keylife=480m
    ike=3des-sha1-modp1024
    auto=start
    pfs=no
    dpddelay=30
    dpdtimeout=120
    dpdaction=restart
```

3. Replace the local (192.168.8.173) and remote (192.168.8.174) IP addresses with the correct addresses for your location. The pre-shared key will be supplied by the VPN provider and will need to be placed in this file in cleartext form.

```
[moxa@ioPAC8600 ~]# sudo vi /etc/ipsec.secrets

192.168.8.173 192.168.8.174 : PSK "your_pre_shared_key"
```

4. Start `ipsec`.

```
[moxa@ioPAC8600 ~]# sudo ipsec setup start
```

NTP Server/Client

Refer to the **System Time Setting** section in Chapter 3 for more information.

Port Trunking

Port Trunking is used to combine two network connections, connected to two different ports on the ioPAC, for redundancy or increased throughput.

1. Deactivate `eth0` and `eth1`.

```
[moxa@ioPAC8600 ~]# sudo nmcli c down eth0
Connection 'eth0' successfully deactivated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/1)

[moxa@ioPAC8600 ~]# sudo nmcli c down eth1
Connection 'eth1' successfully deactivated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/2)
```

2. Add a bond connection with `ifname(bond0)` and bonding mode for NetworkManager.

```
[moxa@ioPAC8600 ~]# sudo nmcli c add type bond ifname bond0 mode active-backup
Connection 'bond-bond0' (9edd06fe-575d-42e0-94d7-e183ab8559cd) successfully
added.
```

3. Add two bond-slave connections with `ifname(eth0 and eth1)` for the master `bond0` added in the previous step.

```
[moxa@ioPAC8600 ~]# sudo nmcli c add type bond-slave ifname eth0 master bond0
Connection 'bond-slave-eth0' (3a13902c-a5c4-4867-a27c-f2fa0b79898d)
successfully added.

[moxa@ioPAC8600 ~]# sudo nmcli c add type bond-slave ifname eth1 master bond0
Connection 'bond-slave-eth1' (067120e7-8828-40d9-8f0c-f494d54de862)
successfully added.
```

4. Activate `bond-slave-eth0` and `bond-slave-eth1`.

```
[moxa@ioPAC8600 ~]# sudo nmcli c up bond-slave-eth0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/6)

[moxa@ioPAC8600 ~]# sudo nmcli c up bond-slave-eth1
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/7)
```

5. Activate port trunking.

```
[moxa@ioPAC8600 ~]# sudo nmcli c up bond-bond0
Connection successfully activated (master waiting for slaves) (D-Bus active
path: /org/freedesktop/NetworkManager/ActiveConnection/8)
```

DDNS

The ioPAC CPU30 can be connected by DDNS service.

1. Go to no-ip.com to sign up and apply for a Dynamic DNS.

The screenshot shows a web form titled "Add Hostname". It contains the following fields and options:

- Hostname:** A text input field containing "ioPAC8600".
- Domain:** A dropdown menu showing "ddns.net".
- Record Type:** A dropdown menu showing "A" with a "More Records" button next to it.
- Enable Wildcard:** A checkbox that is currently unchecked, with a note: "Upgrade to Enhanced to enable wildcard hostnames."
- IPv4 Address:** A text input field containing "61.221.166.100".
- Buttons:** "Cancel" and "Add Hostname" (highlighted in green).

2. Modify inadyn.conf for NO-IP.

```
[moxa@ioPAC8600 ~]# sudo vi /etc/inadyn.conf

# Basic configuration file for inadyn
#
# /etc/inadyn.conf
pidfile /var/run/inadyn.pid
update_period_sec 600 # Check for a new IP every 600 seconds
username test        # replace 'test' with your username
password test        # replace 'test' with your password
dyndns_system default@no-ip.com # replace w/ your provider

alias ioPAC8600.ddns.net
# replace 'test.homeip.net' with yourdomainname for actual (non-test) use
```

3. Start the inadyn service.

```
[root@ioPAC8600 ~]# nmcli c up eth0
[moxa@ioPAC8600 ~]# sudo systemctl start inadyn.service
```

opkg

opkg (Open PacKaGe) is a lightweight package management system based on opkg. You can use it to install the ipk from a third party. Before using it, use the following command to place the ipk in the correct directory.

```
[moxa@ioPAC8600 ~]# ls -hl
total 8
-rw-r--r--  1 moxa  moxa      5.0K Jan 16 19:22 now_time_1.0.0_armv7.ipk
```

Install

```
[moxa@ioPAC8600 ~]# sudo opkg install now_time_1.0.0_armv7.ipk
Installing now_time (1.0.0) to root...
Configuring now_time.
```

Remove

1. Check which package was installed.

```
[moxa@ioPAC8600 ~]# sudo opkg list-installed | grep now_time
now_time - 1.0.0
```

2. Remove the package.

```
[moxa@ioPAC8600 ~]# sudo opkg remove now_time
Removing package now_time from root...
```

Controller Data Acquisition

In this chapter, we explain how to use the ioPAC Programmable Controller to implement data acquisition applications.

The following topics are covered in this chapter:

- ❑ **I/O Data Access**
- ❑ **Data Backup Memory**
- ❑ **Millisecond Timestamp**
- ❑ **MX-AOPC UA Server**

I/O Data Access

ioPAC Programmable Controllers support many kinds of I/O modules that collect physical data from sensors. You can use 85/86M series I/O modules for ioPAC 8600-CPU30 series products.

Moxa provides C/C++ programming tools for users to develop their own projects for data acquisition. The C/C++ library and APIs can be found under **C/C++ Sample Code Programming Guide for ioPAC Programmable Controllers** → **io, io_event, and misc.**

Data Backup Memory

ioPAC Programmable Controllers support data backup memory, which is important for applications that require any kind of data backup in the event of a power failure. Use the following APIs to access the memory. For more detailed information and examples, see the Moxa document **C/C++ Sample Code Programming Guide for ioPAC Programmable Controllers** → **misc.**

Millisecond Timestamp

A timestamp is a sequence of characters or encoded information identifying when a certain event occurred, usually expressing the date and time of day and sometimes accurate to a small fraction of a second. The ioPAC 85M-1602-T and 85M-3801/11-T modules have a dual CPU architecture, which means that each I/O module has its own CPU, which can easily record events in milliseconds. Using this function enables the ioPAC to collect data more accurately by providing millisecond timestamps for event and historical data analysis. Thanks to the millisecond timestamp function, operators can distinguish the sequence of data and know how to handle this event accordingly.

For more detailed information and examples, refer to **C/C++ Sample Code Programming Guide for ioPAC Programmable Controllers** → **IO.**

MX-AOPC UA Server

MX-AOPC UA Server is a software package provided by Moxa that operates as an OPC driver for an HMI or SCADA system. It can be used to establish a seamless connection from Moxa's ioPAC/ioLogik series products to SCADA systems. MX-AOPC UA Server meets the latest standard of OPC UA, which allows connections to various kinds of devices and OPC client machines.

Moxa has pioneered the concept of "active type" OPC software in the automation industry. The patented Active OPC Server supports a non-polling architecture alongside the standard OPC protocol, giving users the alternative of active, push-based communication from Moxa's controllers and remote I/O devices. This adaptation of "push" technology means that I/O status will be updated at the MX-AOPC UA Server only when there is an I/O status change, a pre-configured interval is reached, or when a request is issued by a user. This application of push technology cuts metadata overhead, resulting in faster I/O response times and more accurate data collection than traditional pull-based architectures. With Moxa's "active technology" advantage, users can now instantly receive alarms and real-time updates.

Use the **AOPC APIs** to configure and access the AOPC tag information. For detailed information and examples, refer to **C/C++ Sample Code Programming Guide for ioPAC Programmable Controllers** → **aopc.**

Controller Programming

In this chapter, we describe how to install a toolchain on the host computer used to develop your applications. Cross-platform development and debugging are also discussed.

The following topics are covered in this chapter:

▣ **Programming Guide**

- Memory Usage

▣ **Using C/C++**

- Linux Toolchain
- GDB
- Library and APIs

Programming Guide

Memory Usage

Use the **df** and **free** commands to check memory information.

```
[moxa@ioPAC8600 ~]$ df -Th
Filesystem      Type      Size      Used      Available  Use%      Mounted on
none           devtmpfs  234.2M    0          234.2M    0%        /dev
overlay        overlay   1.9G      25.9M     1.8G      1%        /
tmpfs          tmpfs     246.7M    24.0K     246.6M    0%        /dev/shm
tmpfs          tmpfs     246.7M    568.0K    246.1M    0%        /run
tmpfs          tmpfs     246.7M    0          246.7M    0%        /sys/fs/cgroup
tmpfs          tmpfs     246.7M    0          246.7M    0%        /tmp
/dev/mtdblock1 vfat      501.0K    512       500.5K    0%        /var/retain
[moxa@ioPAC8600 ~]$ free -ht
              total      used      free      shared      buff/cache  available
Mem:          493M        11M       450M       592K        31M         462M
Swap:         0B          0B         0B
Total:        493M        11M       450M
```

If the user file system is incorrect, the kernel will change the root file system to the kernel and use the default Moxa file system. To finish the boot process, run the init program.

- NOTE**
1. The user file system is a complete file system. You can create and delete directories and files (including source code and executable files) as needed.
 2. You can create the user file system on the host PC or the target platform and copy it to the ioPAC Programmable Controller.
 3. Continuously writing data to flash is not recommended, since doing so will decrease the flash's lifespan.

Using C/C++

Linux Toolchain

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

- Arch linux Rolling release (2017-02-01)
- Ubuntu 14.04

Installing the Linux Toolchain

Refer to the **Installing the Toolchain on the PC** section in Chapter 2.

Compiling Applications

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

```
#arm-linux-gnueaibhf -o example -Wall -g -O2 example.c
```

```
#arm-linux-gnueaibhf -s example
```

```
#arm-linux-gnueaibhf -ggdb -o example-debug example.c
```

Most of the cross compiler tools are the same as their native compiler counterparts, but have an additional prefix that specifies the target system. In the case of the ioPAC 8600-CPU30, the prefix is **arm-linux-gnueaibhf**.

For example, the native C compiler is **gcc** and the cross C compiler for ARM in the ioPAC is **arm-linux-gcc**.

Moxa provides cross compiler tools for the following native compilers. When using these commands, simply add the **arm-linux-gnueaibhf** prefix to the command.

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)

GDB

The toolchain also provides a debugging mechanism to help you develop your program. Take the following steps to use GDB for debugging.

1. Place the executable file and source code in the same directory.

```
[moxa@ioPAC8600 ~]# ls -hl
total 16
-rwxr-xr-x  1  moxa   moxa   10.0K  Jan  4  06:46 test
-rw-r--r--  1  moxa   moxa    95    Jan  4  06:45 test.c
```

2. Debug the program.

```
[moxa@ioPAC8600 ~]# gdb -q test
Reading symbols from test...done.
(gdb) l
1      int main()
2      {
3          int a, b, c;
4          a=5;
5          b=10;
6          c=15;
7          c+=a;
8          c-=b;
9          return 0;
10     }
(gdb) b 7
Breakpoint 1 at 0x103b2: file test.c, line 7.
(gdb) r
Starting program: /root/test

Breakpoint 1, main () at test.c:7
7      c+=a;
(gdb) display c
1: c = 15
(gdb) n
8      c-=b;
1: c = 20
(gdb) n
9      return 0;
1: c = 10
(gdb) c
Continuing.
[Inferior 1 (process 474) exited normally]
(gdb) q
```

Library and APIs

For detailed information about the library and APIs, see **C/C++ Sample Code Programming Guide for ioPAC Programmable Controllers**.

System Commands

Special Moxa Utilities

Command	Description
kversion	Show kernel version
setdef	Reset to factory defaults and reboot
upgradefm	Firmware upgrade utility

To view the supported system commands, use the **help** or **busybox --help** commands, as illustrated in the following examples:

help:

```
[moxa@ioPAC8600 ~]$ help
GNU bash, version 4.3.42(1)-release (arm-buildroot-linux-gnueabi)
These shell commands are defined internally. Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

job_spec [&]                history [-c] [-d offset] [n] or hist>
(( expression ))           if COMMANDS; then COMMANDS; [ elif C>
. filename [arguments]     jobs [-lnprs] [jobspec ...] or jobs >
:                            kill [-s sigspec | -n signum | -sigs>
[ arg... ]                 let arg [arg ...]
[[ expression ]]           local [option] name[=value] ...
alias [-p] [name[=value] ... ]  logout [n]
bg [job_spec ...]          mapfile [-n count] [-O origin] [-s c>
bind [-lpsvPSVX] [-m keymap] [-f file> popd [-n] [+N | -N]
break [n]                  printf [-v var] format [arguments]
builtin [shell-builtin [arg ...]]  pushd [-n] [+N | -N | dir]
caller [expr]              pwd [-LP]
case WORD in [PATTERN [| PATTERN]...)> read [-ers] [-a array] [-d delim] [->
cd [-L|[-P [-e]] [-@]] [dir]      readarray [-n count] [-O origin] [-s>
command [-pVv] command [arg ...]  readonly [-aAf] [name[=value] ...] o>
compgen [-abcdefgjkusv] [-o option] > return [n]
complete [-abcdefgjkusv] [-pr] [-DE] > select NAME [in WORDS ... ;] do COMM>
comppopt [-o|+o option] [-DE] [name ..> set [-abefhkmnptuvxBCHP] [-o option->
continue [n]                shift [n]
coproc [NAME] command [redirections]  shopt [-pqsu] [-o] [optname ...]
declare [-aAfFgIlNrtux] [-p] [name[=v> source filename [arguments]
dirs [-clpv] [+N] [-N]        suspend [-f]
disown [-h] [-ar] [jobspec ...]  test [expr]
echo [-neE] [arg ...]         time [-p] pipeline
enable [-a] [-dnps] [-f filename] [na> times
eval [arg ...]                trap [-lp] [[arg] signal_spec ...]
exec [-cl] [-a name] [command [argume> true
exit [n]                      type [-afptP] name [name ...]
export [-fn] [name[=value] ...] or ex> typeset [-aAfFgIlrtux] [-p] name[=va>
false                          ulimit [-SHabcdeilmnpqrstuvXT] [lim>
fc [-e ename] [-lnr] [first] [last] o> umask [-p] [-S] [mode]
fg [job_spec]                 unalias [-a] name [name ...]
for NAME [in WORDS ... ] ; do COMMAND> unset [-f] [-v] [-n] [name ...]
for (( exp1; exp2; exp3 )); do COMMAN> until COMMANDS; do COMMANDS; done
function name { COMMANDS ; } or name > variables - Names and meanings of so>
getopts optstring name [arg]    wait [-n] [id ...]
hash [-lr] [-p pathname] [-dt] [name > while COMMANDS; do COMMANDS; done
help [-dms] [pattern ...]      { COMMANDS ; }
```

busybox --help:

```
[moxa@ioPAC8600 ~]$ busybox --help
BusyBox v1.24.1 (2017-07-03 18:15:00 CST) multi-call binary.
BusyBox is copyrighted by many authors between 1998-2015.
Licensed under GPLv2. See source distribution for detailed
copyright notices.

Usage: busybox [function [arguments]...]
or: busybox --list[-full]
or: busybox --install [-s] [DIR]
or: function [arguments]...

BusyBox is a multi-call binary that combines many common Unix
utilities into a single executable. Most people will create a
link to busybox for each function they wish to use and BusyBox
will act like whatever it was invoked as.

Currently defined functions:
[, [[, addgroup, adduser, ar, arp, arping, ash, awk, basename, blkid,
bunzip2, bzip, cat, catv, chat, chatr, chgrp, chmod, chown, chpasswd,
chroot, chrt, chvt, cksum, clear, cmp, cp, cpio, crond, crontab, cut,
date, dc, dd, deallocvt, delgroup, deluser, devmem, df, diff, dirname,
dmesg, dnsd, dnsdomainname, dos2unix, dpkg, dpkg-deb, du, dumpkmap,
echo, egrep, eject, env, ether-wake, expr, false, fbset, fdflush,
fdformat, fdisk, fgrep, find, flock, fold, free, freeramdisk, fsck,
fstrip, fuser, getopt, getty, grep, gunzip, gzip, halt, hdparm, head,
hexdump, hostid, hostname, hwclock, i2cdetect, i2cdump, i2cget, i2cset,
id, ifconfig, ifdown, ifup, inetd, init, insmod, install, ip, ipaddr,
ipcrm, ipcs, iplink, iproute, iptunnel, kill, killall,
killall5, last, less, linux32, linux64, linuxrc, ln, loadfont,
loadkmap, logger, login, logname, losetup, ls, lsattr, lsmod, lsof,
lspci, lsusb, lzcat, lzma, makedevs, md5sum, mdev, msg, microcom,
mkdir, mkfifo, mknod, mkswap, mktemp, modprobe, more, mount,
mountpoint, mt, mv, nameif, netstat, nice, nohup, nslookup, od, openvt,
passwd, patch, pidof, ping, pipe_progress, pivot_root, poweroff,
printenv, printf, ps, pwd, rdate, readlink, readprofile, realpath,
reboot, renice, reset, resize, rm, rmdir, rmmode, route, run-parts,
runlevel, sed, seq, setarch, setconsole, setkeycodes, setlogcons,
setserial, setsid, sh, shasum, sha256sum, sha3sum, sha512sum, sleep,
sort, start-stop-daemon, strings, stty, su, sulogin, swapoff, swapon,
switch_root, sync, sysctl, tail, tar, tee, telnet, telnetd, test, tftp,
time, top, touch, tr, traceroute, true, truncate, tty, udhcpc, uevent,
umount, uname, uniq, unix2dos, unlink, unlzma, unxz, unzip, uptime,
usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc,
wget, which, who, whoami, xargs, xz, xzcat, yes, zcat
```