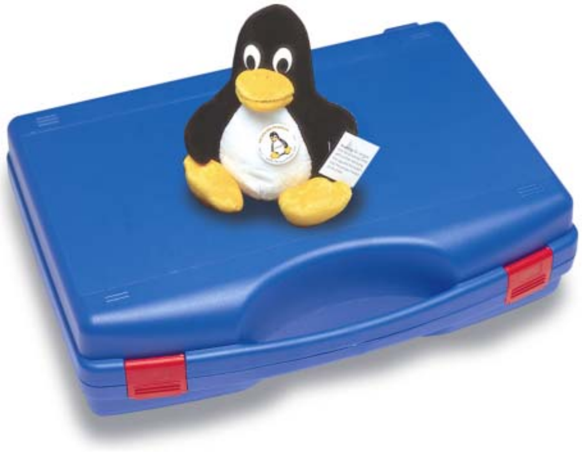
Arcom Embedded Linux

Technical Manual



Arcom
www.arcom.com

a spectris company

Definitions

Arcom is the trading name for Arcom Control Systems Inc and Arcom Control Systems Ltd.

Disclaimer

The information in this manual has been carefully checked and is believed to be accurate. Arcom assumes no responsibility for any infringements of patents or other rights of third parties, which may result from its use.

Arcom assumes no responsibility for any inaccuracies that may be contained in this document. Arcom makes no commitment to update or keep current the information contained in this manual.

Arcom reserves the right to make improvements to this document and/or product at any time and without notice.

Warranty

This product is supplied with a full 3 year warranty. Product warranty covers failure caused by any manufacturing defects. Arcom will make all reasonable effort to repair the product or replace it with an identical variant. Arcom reserves the right to replace the returned product with an alternative variant or an equivalent fit, form and functional product. Delivery charges will apply to all returned products. Please go to www.arcom.com/support for information about product return forms.

Trademarks

Linux is a registered trademark of Linus Torvalds.
Red Hat is a registered trademark of Red Hat, Inc.
ARM and StrongARM are registered trademarks of ARM, Ltd.
Intel and XScale are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
X Window System is a trademark of X Consortium Inc.
All other trademarks and copyrights referred to are the property of their respective owners.
This product includes software developed by the University of California, Berkeley and its contributors.

Revision History

Manual	*PCB*	*Date*	*Comments*
Issue A		12th July 2002	Combined Quickstart and Technical Manual.
Issue B		8th July 2003	Minor editorial changes
Issue C		3rd February 2004	Major changes for V3I5 Development Kit.
Issue D		14th June 2004	Minor changes for V316 Development Kit.
Issue E		3rd August 2004	Major updates and layout changes.

Arcom is a subsidiary of Spectris plc.
For contact details, see page 62.

Arcom operates a company-wide quality management system, which has been certified by the British Standards Institution (BSI) as compliant with ISO9001:2000

Contents

Introduction

Arcom Embedded Linux (AEL) is a standard Linux distribution produced by Arcom. It is optimized to fit within the on-board Flash of Arcom's range of Single Board Computers (SBCs). AEL is based on the standard Linux kernel and user space tools.

Arcom provide free first line technical support for this product.
See Appendix A – Contacting Arcom, page 62.

The minimum target footprint for AEL is a system with 16MB RAM and 8MB Flash memory. Additional RAM and/or Flash enables greater functionality.

The AEL distribution consists of:

- A standard Linux kernel, built to support Flash memory access.
- A number of Linux device drivers (some board-specific).
- System and application libraries.
- User space applications and utilities.

Some major components are:

- THTTPD web server.
- FTP and telnet servers and clients.
- Bourne Again Shell (BASH).
- OpenSSH (secure telnet and FTP replacement).
- XFree86/TinyX (on targets where graphics hardware is available).
- Many other standard Linux utilities.

You may well want to add specific functionalities to the base system. You can do this by adding libraries and/or applications compiled on a host Linux system. We supply all source code for the kernel, libraries and applications in the AEL distribution. This means you can completely rebuild the environment from scratch, if required.

AEL is implemented in a way that enables it to support installation on space-constrained devices, such as the on-board Flash. On certain target boards, however, you can install the distribution on IDE-compatible media. Consult the Quickstart Manual for your board to see if you can do this.

Arcom SBCs are not intended to be used to build applications. We recommend you design and build applications on a host Linux system, and then download them to the target system using FTP or SCP.

If you want to develop on the target board, it may be possible to install a host operating system (such as Red Hat Linux) with the addition of a hard disk drive. To do this, the following conditions must be satisfied:

- The target hardware must support the addition of a hard disk. For this information, see the Technical and Quickstart Manuals for your board.
- A suitable operating system for the target board's processor architecture must exist. It is not possible to develop applications under AEL directly.

Licensing AEL components

The AEL Development Kit contains components licensed by different sources. Many of these are Open Source licenses (see www.opensource.org). If you further distribute these modules you may, under certain circumstances, be required to release source code for any modifications you have made.

Please consult Appendix B - Software sources, page 63, and the relevant source packages to ensure you are familiar with the licensing requirements of any packages that you modify. Copies of the more widespread licenses are included on the Development Kit CD, in the folder /licenses.

Handling your board safely

Anti-static handling

This Development Kit contains CMOS devices. These could be damaged in the event of static electricity being discharged through them. Observe anti-static precautions at all times when handling circuit boards. This includes storing boards in appropriate anti-static packaging and wearing a wrist strap when handling them.

Packaging

Should a board need to be returned to Arcom, please ensure that it is adequately packed, preferably in the original packing material.

About this manual

This manual provides detailed information about the AEL distribution. It explains, with examples, how to use the key technologies supplied in this distribution.

AEL is available on a number of different Arcom boards, each with different devices and hardware capabilities. Certain sections of this manual may therefore not apply to particular boards. For example, boards without display hardware do not support the X Window System, so that section can be ignored.

Related documents

In addition to this manual, you can obtain useful information from a variety of sources. These include:

- An index.html file, on the Development Kit CD, which may be opened with any web browser. This contains links to many of the documents mentioned in this section and throughout this manual.
- The Linux RUTE manual. This contains a general overview of how to use a Linux system. You can find it on the Development Kit CD, in the folder /manuals/.
- The Linux online help system. The pages in this help system are referred to from time to time, in the format 'name (section)', for example 'interfaces (5) man page'. You can view this help on the host system (not on the target) by typing **man *name***, or **man *section name***. For the above example, therefore, you might type **man interfaces**, or **man 5 interfaces**. Include the section if possible, because in some cases a page with the same name may exist in a different section.
- The selection of Linux 'how to' documents covering common topics on the Development Kit CD, in the folder /manuals/HOWTO/. The full set of 'how to' documents is available at www.tldp.org.
- The Arcom Quickstart and Technical Manuals, which are also in the folder /manuals/ on the Development Kit CD.
- The websites to which you can find links in Appendix B - Software sources, page 63.

Conventions

Symbols

The following symbols are used in this guide:

Symbol	Explanation
📝	Note - information that requires your attention.
💡	Tip - a handy hint that may provide a useful alternative or save time.
💣	Caution – proceeding with a course of action may damage your equipment or result in loss of data.

Typographical conventions

This manual contains examples of commands that you can enter. These are shown as follows:

```
$ make install DESTDIR=/tmp/target-install
```

The initial symbol ($ in this case) indicates the prompt that the command is for and should not be typed.

The prompts used are explained in the following table:

Prompt	Explanation
$	Linux (BASH) as a regular user.
#	Linux (BASH) as root.
RedBoot>	RedBoot command line.
(gdb)	GNU DeBugger prompt.

Different fonts are used throughout the manual to identify different types of information, as follows:

Font	Explanation
Italics	Parts of a command that should be substituted with appropriate values.
Bold	Information that you enter yourself.
`Screen text`	Information that is displayed on screen.

Long commands that don't fit on one line, and must therefore be split across multiple lines, are indicated by a backslash (\) at the end of the line.

Development Kit CD contents

The Development Kit CD contains the following top-level folders:

Folder	Contents
packages	AEL source and binaries.
examples	Various examples.
host	Host environment.
licenses	Common licenses used by the software included in the Development Kit.
manuals	Arcom and third party documentation.
reference	Board reference documentation.

File system layout

The exact layout of the file system on an AEL system depends on the target board. In general, the Flash includes one or more smaller partitions containing the boot loader and Linux kernel images[1], along with one large partition that covers the rest of the device and contains the root (/) file system. The partition sizes are typically determined from RedBoot's Flash Image System (FIS) and can be changed from RedBoot's command line. On boards without RedBoot, the partitions are hard coded into the kernel binary, and to change them you must rebuild the kernel.

RedBoot's FIS is used to provide a partition table on boards that use RedBoot as the boot loader. In such cases, the Flash can be repartitioned from the boot loader command line.

In addition to the Flash file systems, a RAM-based file system is mounted on /var/tmp.

Journaling Flash file system

The Flash is formatted using the Journaling Flash File System (JFFS2). This places a compressed file system onto the Flash transparently to the user. Key features of JFFS2 include:

- Direct targeting of Flash devices.
- Robustness.
- Consistency across power failure.
- No integrity scan (fsck) is required at boot time after normal or abnormal shutdown.
- Explicit wear levelling.
- Transparent compression.

Flash partitions appear as pseudo-block devices with major number 31, which can be mounted using JFFS2, as follows:

```
# mount –t jffs2 /dev/mtdblock1 mount-point
```

There are a maximum of 16 partitions. These are numbered 0 to 15, and correspond to the block devices /dev/mtdblock0 through /dev/mtdblock15. In addition, each partition has a character device, /dev/mtdN. This is used to access advanced features of the Flash device, such as sector locking.

No special utility is required to make a JFFS2 file system. Simply erase the whole of the partition using **eraseall –j**, and mount as normal. The **–j** option causes an empty JFFS2 file system to be created rather than completely erasing the flash device.

JFFS2 partitions do not require an integrity check (fsck) to be performed on startup, after either normal or abnormal shutdown. The supplied /sbin/fsck.jffs2 is a dummy which always succeeds and is present to simplify the boot scripts.

[1] Sometimes the boot loader and kernel are combined into a single Flash partition.

Although JFFS2 is a journaling file system, this does not preclude the loss of data. The file system remains in a consistent state across power failure and is always mountable. However, if the board is powered down during a write, the incomplete write is rolled-back on the next boot. Any completed writes are not affected.

For more information about JFFS2, see sources.redhat.com/jffs2.

RAM file system

AEL systems make use of a RAM-based file system (tmpfs) mounted on /var/tmp. The contents of this file system are not preserved through reboot. The RAM file system grows and shrinks to accommodate only the size of the files it contains. This means there is very little overhead.

To prevent the RAM file system using the whole of RAM, the file system is constrained to use a maximum of 4MB of memory. You can change this by editing /etc/fstab and changing the **size=** parameter for /var/tmp.

Configuring AEL

Default passwords

When AEL is supplied, it is configured with two users, the super user (known as root) and a regular user called arcom. The default password for both accounts is **arcom**.

To change a user’s password, log in as that user and run the **passwd** command.

To add a user, log in as **root** and run the **adduser** command, for example:

```
# adduser abc
```

For security reasons, it is essential that you change the passwords for any deployed system.

Keyboard mapping

When supplied, AEL is configured for a US-style keyboard. This configuration is controlled by the file /etc/console/keymap.gz, which is a symbolic link to us.map.gz in the folder /usr/lib/kbd/keymaps/i386/qwerty/.

If you want to use another keyboard layout, change this link to point to another file. For example, for a UK keyboard:

```
# ln –sf /usr/lib/kbd/keymaps/i386/qwerty/uk.map.gz /etc/console/keymap.gz
```

Once you have configured the new keymap, you may reload it with:

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

Additional keyboard maps can be found in the kbd source package on the CD.

Serial port configuration

The Linux kernel that is shipped as standard with AEL already contains driver support for the standard serial 16550 UARTs (such as those found on many of Arcom's processor boards), as well as the AIM104-COM4 peripheral board and many third party serial boards. A standard AEL kernel can support many serial ports (typically up to 64) but only allocates a small number for the on-board UARTs.

Configuring serial ports using setserial

You can configure serial ports using the **setserial** tool. This section provides a brief overview of the **setserial** tool. For more detailed information, refer to the setserial(8) man page on your host system, or see setserial.sourceforge.net/setserial-man.html.

You can view the current configuration of a serial port by passing the device to **setserial**. For example, use the following command to view the configuration of COM1, which is /dev/ttyS0:

```
# setserial /dev/ttyS0
```

You can obtain more detailed information by passing the **–a** flag, as follows:

```
# setserial –a /dev/ttyS0
```

Passing additional options to setserial configures serial ports. The following table outlines the common options:

Option	Description
port *port_number*	Sets the I/O port used by this serial port.
irq *irq_number*	Sets the IRQ used by this serial port.
uart *uart_type*	Sets the UART type. Permitted types are none, 8250, 16450, 16550, 16550A, 16650, 16650V2, 16654, 16750, 16850, 16950 and 16954.
autoconfig	Causes the kernel to attempt to automatically detect the UART type. If the **auto_irq** option has been given, it also attempts to determine the IRQ. This option must be given after the **port** and **irq** (or **auto_irq**) options.
auto_irq	Causes the kernel to attempt to automatically determine the IRQ to use when performing automatic configuration. It is much safer, however, to explicitly configure the IRQ using the **irq** option. You can disable this option by prefixing it with a carat (^).
baud_base	Sets the base baud rate. This is the clock frequency divided by 16. The default base baud rate is normally either 115200 or the maximum baud base the port is capable of. In some cases, however, the default is 0. If so, you cannot set any other options until you have set a suitable baud_base. You can do this by passing baud_base as the first option.

Further options are described in the setserial (8) man page.

Configuring a new port to an additional serial board

You may have added further serial ports to your processor, for example using an Arcom AIM104-COM4 or third party serial board. If so, you can use the above commands to configure any of the unused serial ports. Simply choose a free serial port /dev/ttyS*n* and configure it as described above, in accordance with the settings you have configured on the peripheral board itself. Consult the peripheral board's documentation for details.

Automatically loading the serial port configuration on boot

At boot time the contents of the file /etc/serial.conf are parsed and the serial ports are configured accordingly. Each line of the file starts with the name of a device and is followed by one or more options that are passed to **setserial**.

For example, to set the irq of COM1 to 7, add the following to /etc/serial.conf:

```
/dev/ttyS0 irq 7
```

You can also generate a configuration line using the **–G** switch to setserial to generate an entry representing the current state of a port. For example, the following command appends an entry for /dev/ttyS1 to the configuration file:

```
# setserial –G /dev/ttyS1 >> /etc/serial.conf
```

Removing log in session from ttyS0

The default installation of AEL is configured to run a log in session on ttyS0 (COM1). This can cause problems if your application wants to use ttyS0, as the two conflict. This often manifests as the serial port changing baud rate at unusual times.

Before removing all log in sessions, either:

- Ensure you have some other way of logging into the system, such as via SSH or a local console.

-or-

- Be prepared to boot to single user mode, as described on page 22.

To remove the serial log in session, you can either:

- Edit /etc/inittab and remove the following line:

  ```
  T0:23:respawn:/sbin/getty -L ttyS0 115200 vt100
  ```

  Then signal init to reload its configuration by entering:

  ```
  # telinit q
  ```

-or-

- Move the log in session to another serial port (or add another one) by adding lines as follows:

  ```
  Tn:23:respawn:/sbin/getty –L ttySn 115200 vt100
  ```

  (Where **n** is the number of the serial port to use.)

System startup scripts

AEL uses a system V type init process. Scripts are placed in /etc/init.d/, with symbolic links for each runlevel in etc/rc?.d/.

? may be any of the following characters:

Character	Function	Description
S	Startup	Run once at boot time.
0	Halt	Run on system shutdown.
1	Single	Run on entering single user mode.
2	Normal	Serial login only.
3	Normal	Serial and VGA login.
4	Normal	VGA login only.
5	Normal	VGA login only.
6	Reboot	Run before rebooting.

The default runlevel is level 3 for targets with graphics hardware, and level 2 for others.

When the runlevel changes, the K* scripts in the /etc/rc?.d/ folder corresponding to the new runlevel are executed in alphanumerical order (with an argument of stop). The S* scripts in the same folder are then executed in alphanumerical order (with an argument of **start**).

You can start or stop a service manually by calling the script in /etc/init.d with a parameter of either **start**, **stop** or **restart**.

Calling the script with no parameters normally displays a complete list of the possible actions.

Making an application run automatically at boot

If you want an application to run automatically at boot, follow these steps:

1 Write a script that runs your application. For example, you may create a script called ‘someapp’.

2 Put the script in the following folder:

/etc/init.d

3 Make the script executable by entering the following command (replacing ‘someapp’ with the name you’ve given your script):

```
# chmod +x /etc/init.d/someapp
```

4 Make a symbolic link in /etc/rc*X*.d that points to the script in /etc/init.d. For the ‘someapp’ example, you might therefore enter:

```
# ln -s /etc/init.d/someapp /etc/rcX.d/S99someapp
```

(Where X is the runlevel number.)

For example, at runlevel 3, you would enter:

```
/etc/rc3.d/S99someapp
```

Using 99 ensures that your application starts after all other services.

Network configuration

You can view the current Ethernet configuration by running **ifconfig**, and the current default gateway with **route**.

IP address configuration

In the default install of AEL, the network device is configured to obtain an IP address automatically via DHCP (Dynamic Host Configuration Protocol). If you are running a DHCP server on your network, or you want to force a static IP address, you can reconfigure the device by editing the file /etc/network/interfaces. The format of this file is described in the interfaces(5) man page.

Each interface is defined by a line starting with the **iface** keyword. The syntax is:

```
iface NAME FAMILY METHOD
```

The parameters you can specify are explained in the following table:

Parameter	Description
NAME	The name of the interface (for example eth0). The name given to an interface in Linux is the device type with a numerical suffix. Thus, the first Ethernet device is known as eth0, while the second (if present) is eth1. The first wireless network device is called wlan0.
FAMILY	The address family for the interface (normally inet for IPV4).
METHOD	The method to be used to obtain an address for this interface. The available methods include loopback, static, manual and dhcp.

After each iface line there may be one or more lines that specify further options. In general, only interfaces that use the static method require additional options.

The following options are valid for any interface:

Option	Description
up *COMMAND*	Run command after bringing the interface up.
pre-up *COMMAND*	Run command before bringing the interface up.
down *COMMAND*	Run command before taking the interface down.
post-down *COMMAND*	Run command after taking the interface down.

For each of the above there also exists a folder, as follows:
/etc/network/if-<*option*>.d/

The scripts in each of these are run after the corresponding option, if any, has been run. All of the commands are called with several environment variables set. These variables are described in the following table:

Option	Description
IFACE	The name of the physical interface.
ADDRFAM	The address family, for example inet.
METHOD	The configuration method, for example static.
MODE	The mode, which may be either start or stop.

Statically configuring an interface

The following options are valid for an IPV4 interface that is statically configured, using the static method:

Option	Description
address *ADDRESS*	Address (dotted quad). Required.
netmask *NETMASK*	Netmask (dotted quad). Required.
broadcast *BROADCAST_ADDRESS*	Broadcast address (dotted quad).
network *NETWORK_ADDRESS*	Network address (dotted quad).
gateway ADDRESS	Default gateway (dotted quad).

For example, to configure eth0 to use the static address 192.168.1.4/24 with a default gateway of 192.168.1.1, enter the following in /etc/network/interfaces:

```
iface eth0 inet static
        address 192.168.1.4
        netmask 255.255.255.0
        gateway 192.168.1.1
```

Manually configuring an interface

The manual method causes no configuration. Such interfaces can be configured using the **up-*** and **down-*** scripts.

Configuring an interface using DHCP

The DHCP method uses an installed DHCP client to obtain configuration information. The hostname *HOSTNAME* option, which requests a specific hostname from the server, is valid for an IPV4 interface configured using DHCP.

Automatically bringing up an interface on boot

A line in /etc/network/interfaces beginning with the **auto** keyword specifies interfaces that should be brought up on boot. For example, to bring up the loopback (lo) and first Ethernet (eth0) interfaces, include the following:

```
auto lo eth0
```

You can include as many **auto** lines as you like.

Hostname

The default hostname for a board is the board type. To change the hostname, edit the file /etc/hostname.

You can use the command **hostname** to view the current hostname.

Wireless network configuration

Wireless support under AEL has been tested using a Linksys Wireless CompactFlash Card (model WCF12). This card is based on the popular PRISM chip set and any PRISM2, 2.5 or 3 based card should work. Cards based on other chip sets may or may not work.

This section explains how to configure wireless networking (WLAN) to access a network via a wireless access point under AEL. For further details, see the Wireless Tools for Linux website at www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html and the Host AP (the card drivers) website at hostap.epitest.fi/.

Configuring the wireless network device

In order to connect to a wireless network, you must obtain the following settings from your network administrator:

- ESSID network identifier, for example ABC-Wireless-Network.
- Radio channel to use (1-14, depending on local legislation and access point configuration).
- Shared keys used by your access point, if your access point is configured to use Wired Equivalent Privacy (WEP) encryption.

Configuration is carried out with extra options in /etc/network/interfaces in the appropriate iface stanza for the WLAN device (usually wlan0), as described in the previous section. The more common options are listed in the table below.

Option	Description
wireless_mode *MODE*	Sets the operating mode of the device. Common values for *MODE* are: • Adhoc. A network composed of only one cell and without an access point. • Managed. Node connects to a network composed of many access points.
wireless_essid *ESSID*	Sets the ESSID (network name).
wireless_channel *CHAN*	Specifies the radio channel (1-14) to use.
wireless_env *ENC*	Sets the level of encryption to use. *ENC* may be one of: • Off. No encryption. • Open. Encryption is used if available but non-encrypted connections are permitted. • Restricted. Encryption is required.
wireless_key*N KEY*	Sets each of the four encryption keys. *N* is 1-4 and indicates which of the keys to set. *KEY* is the encryption key composed of either 5 bytes (for 64-bit WEP) or 13 bytes (for 128-bit WEP) as hexadecimal digits.
wireless_defaultkey *N*	Uses key N (1-4) as the default key

Example configuration

For a node that is connected to the ABC-Wireless-Network using encryption and obtaining its IP information via DHCP, the following /etc/network/interfaces stanza might be used:

```
iface wlan0 inet dhcp
        wireless_mode managed
        wireless_essid ABC-Wireless-Network
        wireless_channel 11
        wireless_enc restricted
        wireless_key1 893248A248B9443FC00D423090
```

IPSEC

IPSEC is a standard for the encryption and authentication of IP traffic. To use it on boards with IPSEC support, the freeswan-utils package must be installed on the target. This package is provided on the accompanying Development Kit CD. Refer to www.freeswan.org/ for more information about its configuration and use.

Configuring and loading kernel modules

The Linux kernel supports configuring and loading device drivers as external modules. This means much of the system need only be resident in RAM when absolutely necessary.

Modules are loaded using the **modprobe** utility, which in turn calls the lower level **insmod** utility. You need not normally call either of these utilities manually, since the kernel automatically calls modprobe when it is asked to open a device for which no module is currently loaded. The modules that are available via modprobe on the current system are usually found in the /lib/modules/*VERSION*/ folder corresponding to the running kernel.

You can load a module that is installed in the /lib/modules/*VERSION*/ folder by calling modprobe with just the module name and without the **.o** suffix. For example, to load the **ppp_generic.o** module, which is installed under /lib/modules/*VERSION*/, enter the following command:

```
# modprobe ppp_generic
```

Modules can be loaded from elsewhere by using the **insmod** utility directly and giving a full path to the module. For example, to load the module /tmp/mymodule.o, enter the following:

```
# insmod /tmp/mymodule.o
```

You can view the list of currently loaded modules using the command **lsmod**.

Passing parameters to modules

Many kernel modules can take parameters that allow you to tweak various options. The available parameters (along with other interesting information) can be listed using the **modinfo** utility, as follows:

```
# modinfo ppp_generic
# modinfo /tmp/mymodule.o
```

In many cases, the parameters of a particular module are documented more fully within the kernel source tree.

Configuring modprobe

The **modprobe** utility (but not the **insmod** utility) can be configured to automatically pass parameters when a module is loaded. This is useful when the kernel calls modprobe for you but you still want to pass parameters to the module.

The modprobe configuration is stored in /etc/modules.conf (generic configuration) and /etc/modules.conf.board (board-specific configuration). Each entry has the following format:

```
options MODULE PARAMTERS
```

For example, to pass debug=1 whenever loading the **ppp_generic** module, use:

```
options ppp_generic debug=1
```

In addition, modprobe can alias one module name to another. This is useful because the kernel often calls modprobe with an abstract service name rather than a specific module. For example, when an attempt is made to access the first Ethernet device, the kernel calls modprobe to load eth0 (rather than a specific Ethernet driver module). In such cases eth0 is configured as an alias for the actual Ethernet module. For example (if smc91x is the driver for eth0 on your system):

```
alias eth0 smc91x
```

The full modules.conf format is in the modules.conf(5) man page.

Automatically loading modules at boot time

In most cases, the kernel loads the correct modules automatically or they are loaded by the hotplug daemon. In some circumstances, however, this is not the case and it is desirable to load the module automatically and unconditionally at boot time. You can do this by listing the modules you wish to be loaded, one per line, in /etc/modules. Each line must consist of a module name (or alias). This may be followed by one or more optional parameters.

Removing kernel modules

A kernel module that is not in use by the system can be removed using the **rmmod** command.

System recovery and single user mode

If a configuration error has been made that prevents you from logging in, you may be able to boot into single user mode in order to repair the problem. A target board can be booted into single user mode by adding the word **single** to the kernel command line. This is done using the RedBoot **exec** command described on page 60.

If it is not possible to recover the system using this method, you may have to reload the default Development Kit image, as described in Updating the entire Flash on page 60.

Secure Shell (SSH)

Introduction to SSH

SSH (Secure SHell) is a secure replacement for several common Internet protocols, all of which have security flaws when used in a non-trusted network environment (primarily the plaintext exchange of passwords across a non-trusted network). These include, for example, the Berkley r* tools (rlogin, rsh, rexec), FTP and telnet.

SSH has several advantages over these tools. These include:

- All traffic sent across the network is encrypted using strong encryption. Critically, this includes passwords.
- Prevention of spoofing and man-in-the-middle attacks using host keys.
- Tunnelling of arbitrary connections through an SSH pipe, known as port forwarding (in particular X11 forwarding).
- Enhanced authentication methods that improve on normal-based mechanisms.

The server also benefits from SSH, especially if it is running a number of services. If you use port forwarding, otherwise insecure protocols (such as POP) can be encrypted for secure communication with remote machines. SSH makes it relatively simple to encrypt different types of communication normally sent insecurely over public networks.

For more information about SSH, see www.openssh.org.

A large number of client and server programs can use the SSH protocol, including many open source and freely available applications. Several different SSH client versions are available for almost every major operating system in use today.

Please see the documentation for your host system for an explanation of how to install and deploy OpenSSH on your host system.

Several SSH clients are also available for non-Linux systems. These include Microsoft Windows platforms, such as:

- **PuTTY:** A Windows version of the ssh program. This is provided on the Development Kit CD, in the folder /host/windows, and at www.chiark.greenend.org.uk/~sgtatham/putty.

- **WinSCP:** A graphical version of SCP for windows. This is also provided in /host/windows, and at winscp.vse.cz/eng.

Using SSH commands

The ssh command

The **ssh** command enables you to remotely log in to a machine. For example to log in to the machine ael.example.net, you would enter the following command:

$ **ssh ael.example.net**

The first time you log in to a machine, a message similar to the following (but with a different fingerprint) is displayed:

```
The authenticity of host 'ael.example.net (10.2.4.14)' can't be
established.
RSA key fingerprint is
e0:79:67:58:78:e4:bc:0a:6a:e2:f8:62:f8:62:f8:ea:fa:bc
Are you sure you want to continue connecting (yes/no)?
```

This gives you the opportunity to verify that the machine you are logging into is the machine you are expecting by confirming that the host's cryptographic key is correct. Verify that the fingerprint is correct before typing **yes** to continue logging in.

You can obtain the host fingerprint of a system by asking the administrator or by logging in at the console and running the command:

$ **ssh-keygen –l –f /etc/sshd/ssh_host_key.pub**

There are likely to be several ssh_host_*key.pub files. Examine them all.

Once you verify the fingerprint of a system, it is written to ~/.ssh/known_hosts, so you are not asked this question again. If the remote host's fingerprint changes for any reason, ssh displays an error message. If, on investigation, you determine that the change is legitimate, you can edit ~/.ssh/known_hosts to remove the incorrect fingerprint. For example, the remote operating system may be reinstalled, causing the SSH server's host key to change.

After you have accepted the remote system's fingerprint, you are asked to authenticate yourself. You normally do this by entering your password. However, there are alternative authentication methods, such as public key authentication. See Public key authentication methods, page 26.

The **ssh** command logs in to the remote machine using the current user name from your local host. To log in as a different user, for example as the user 'arcom', you may use one of these two command forms:

$ **ssh arcom@ael.example.net**

$ **ssh –l arcom ael.example.net**

All the commands in the SSH suite accept both these forms for specifying a log in user. The examples in this manual use the user@host form.

As well as logging in to a remote machine, ssh can run a command on a remote machine without the need to log in and run it manually. To do this, append the command to the ssh command line. For example, to examine the contents of the /bin/ folder on a remote system you might enter:

```
$ ssh arcom@ael.example.net ls /bin/
```

When you use **ssh** to run a remote command, quote shell meta characters that you want to be passed to the remote system rather than processed locally.

The scp command

The **scp** command is similar to the regular **cp** command, except that it uses the SSH protocol and allows for the source or destination file to be located on a remote system. You can specify a remote file as follows:

user@host:file

The ***user@host*** part is the same as for the ssh command, described in the previous section. If you don't specify a ***file***, or you give a relative path (i.e. one which doesn't start with /), the default is the remote user's home folder. If you forget the colon (:), you copy to or from a local file named *user@host*. This is rarely required.

For example, the following command copies 'my-file' to /home/arcom on the remote system ael.example.net:

```
$ scp my-file arcom@ael.example.net:
```

The following command copies the same file to /etc/ on the remote host:

```
$ scp my-file arcom@ael.example.net:/etc/my-file
```

To retrieve a remote file, reverse the order of the operands. For example, the following retrieves /home/arcom/my-file from the remote system:

```
$ scp arcom@ael.example.net:my-file .
```

The sftp command

The **sftp** command behaves like the regular **ftp** command, except that it uses the SSH protocol to provide the strong authentication and encryption that regular FTP lacks. You can specify a user and hostname using the ***user@host*** syntax described in the previous section, for example:

```
$ sftp arcom@ael.example.net
```

Public key authentication methods

In addition to regular password authentication, SSH also offers public key authentication. Public key authentication works by generating a public/private key pair. The public part may be passed around freely in order to transfer it to a remote system running SSH. The private part must remain a secret, since anyone possessing the private key can log in to any system configured with the public key.

A private key designated for interactive use normally has a pass-phrase that adds an extra layer of security. However, a private key designated for non-interactive use (such as in a script on a remote system), does not.

The **ssh-agent** command enables you to only enter your pass-phrase once per session. See Further information about SSH, page 27.

Generating keys

Public/private key pairs are generated using the **ssh-keygen** utility. This utility asks for a destination (press **Enter** to accept the default) and a pass-phrase (which may be left blank). If you accepted the default location, you have a DSA public/private key pair in ~/.ssh/id_dsa and ~/.ssh/id_dsa.pub.

SSH can also use RSA keys by passing rsa rather than dsa to ssh-keygen. RSA keys are stored in id_rsa and id_rsa.pub.

The file id_dsa is your private key and you must keep this safe. Your public key is in id_dsa.pub. You can copy this to a remote machine and append it to the file ~/.ssh/authorized_keys to enable public key authentication on that host.

Once you have configured the remote host, you can continue to use all the SSH utilities as normal. The authorized_keys file can also be used to restrict the commands that may be run when that key is used. This means you can restrict the commands that can be run by an interactive script (where the key has an empty pass-phrase). See Further information about SSH, page 27.

Port forwarding

SSH has the ability to tunnel TCP/IP connections from the local machine to the remote host and vice-versa. This is useful to provide a secure wrapper around an insecure protocol.

For example, to create a secure tunnel from the local machine to a mail server running remotely on port 25, you could enter the following:

```
$ ssh –L 2525:localhost:25 user@mail.example.net
```

localhost refers to the local host from the perspective of the remote host. Anyone connecting to port 2525 on your local host is therefore forwarded through the SSH tunnel to port 25 on mail.example.net.

You need not use **localhost**. For example, the following command enables anyone with access to your local machine to access mail.example.net, which we assume is secured behind the example.net firewall:

```
$ ssh –L 2525:mail.example.net:25 firewall.example.net
```

You can also forward a port on the remote machine to any machine accessible from your local machine. For example, to tunnel port 2525 on the remote machine to a mail server on your local network, enter the following:

```
$ ssh –R 2525:mail.example.net:25 ael.example.net
```

Anyone connecting to port 2525 on ael.example.net is forwarded to mail.example.net on your local network.

Further information about SSH

The preceding sections provide simple examples of what you can achieve using SSH. Further information about using SSH is on the OpenSSH website, at www.openssh.org.

Removing legacy services

The default install of AEL includes telnet and FTP daemons that are made obsolete by the use of SSH. They remain in the distribution to maintain backwards compatibility. However, we recommend that you remove them unless you explicitly need them.

You can remove the telnet and FTP servers using the **dpkg** packaging tool, as follows:

```
# dpkg –r ftpd
# dpkg –r telnetd
```

For more information, see Package management, on the next page.

You can also use tcpd to restrict access to these services by certain IP addresses or hosts without disabling them completely. Consult the tcpd(8) and host_access(5) man pages for details.

Package management

A default installation of AEL can contain several optional packages, such as the J9 Java Virtual Machine, the OpenBSD Secure Shell (SSH), Web and FTP servers. If your application does not require these and you want to free some space in the Flash, you may remove some packages. Conversely, if you require a package that is not installed by default, you can add packages to the board.

AEL uses the dpkg package management system to manage packages on the target system. An application is typically packaged as a single .deb package file, which can be installed as described in Adding packages, below.

A library is normally split into two packages, the runtime package libFOO and the development package libFOO-dev. The libFOO package must be installed on the target system, while the libFOO-dev package must not. To compile and link applications against a library, both the library package itself and the development package must also be present in the host environment.

All of the library runtime and development packages that are included on the Development Kit CD (even those that are not included in the default target installation) are installed into the host environment by the installer. If you install a library package from another source, such as one provided by Arcom technical support, you must install the runtime and development packages into the host environment yourself. See Installing additional packages into the host environment, page 33.

You can view a list of packages installed on the target by running the following command on the target:

$ **dpkg –l**

When you run this command, the following information is displayed:

```
…
ii    libc6           2.3.1-5
…
ii    bash            2.05b-3
…
```

Removing packages

Packages can be removed using **dpkg**. For example, enter the following to remove the **ftpd** package:

# **dpkg –r ftpd**

Adding packages

Additional software components in .deb packages are on the Development Kit CD, in the /packages folder. To add a package to the target system, follow these steps:

1 Transfer the required package to a temporary folder on the target, for example:

$ **scp /mnt/cdrom/packages/ntp/ntpd_4.1.1-1_arm.deb \ root@target.example.net:/tmp**

2 Install the package on the target by entering the following:

# **dpkg –i /tmp/ntpd_4.1.1-1_arm.deb**

The X Window System

AEL makes use of the TinyX server that is designed for embedded systems and makes use of the Linux frame buffer driver. If your target board has graphics capabilities, it is already installed with the TinyX server and a small number of other utilities.

You can start the TinyX X server on the primary frame buffer device by running the command **startx**.

If you are running on a VGA CRT display device and want to run the TinyX X server at a different video mode to the board default, you can use the **fbset** command to set the mode before running **startx**. The list of modes available is stored in the file /etc/fb.modes on the target, and includes a selection of standard VGA modes. For example, to run X at 800x600 at 75Hz, enter:

```
# fbset 800x600-75
$ startx
```

If your target board supports a non-CRT device such as an LCD display, please consult the Quickstart Manual for details about configuring this device.

Window manager

By default, AEL comes installed with the matchbox window manager. The matchbox window manager is specifically designed to require very few resources. It aims to target systems with little screen real estate and limited input mechanisms (such as a touchscreen). To support these aims, all windows remain maximized at all times (unlike other window managers, which allow you to arbitrarily choose position and size).

AEL does not come supplied with any other window managers. You may, however, choose to compile and install another window manager. To use a different window manager, you can set the environment variable X_WINDOW_MANAGER to the full path to the window manager binary, as follows:

```
$ export X_WINDOW_MANAGER=/usr/bin/another-wm
$ startx
```

Using a touchscreen via a TSC1 controller board

A touchscreen and a TSC1 touchscreen controller board are included in many AEL Development Kits. If your Development Kit includes these, your target board already has the TSC1D touchscreen controller daemon software installed. The TinyX X server in the Development Kit includes an input driver that automatically makes use of the TSC1D daemon if it is running. You can start the daemon by typing **tsc1d &** before launching X.

If you obtained your touchscreen and TSC1 separately from your Development Kit, you must calibrate them using tsc1cal before you can use the touchscreen.

Development Kits that are supplied with a TSC1 are already calibrated. However, if you reinstall the system you may need to recalibrate.

To calibrate your touchscreen, follow these steps:

1 Connect the TSC1 controller board to your target board.

In its default configuration, AEL runs a serial log in on ttyS0 (COM1). We therefore recommend that you connect the TSC1 controller to ttyS1 (COM2). Some Development Kits that are supplied with an Industrial Compact Enclosure (ICE) containing the processor board and the TSC1 controller may use a different serial port. See the Quickstart Manual for your board for more details.

2 Ensure that the tsc1d touchscreen daemon is not running by typing:

```
# killall tsc1d
```

3 Run the touchscreen calibration program **tsc1cal**.

The default serial port is always /dev/ttyS1, even if the ICE box is supplied with the TSC1 attached to some other port. If you are not using the default serial port, pass the correct **--device /dev/ttyS*n*** parameter:

```
# tsc1cal –-device /dev/ttyS1 –-generate /etc/tsc1d.conf
```

---

4 Follow the on screen instructions by touching the squares as they appear.

When you have finished, tsc1cal exits and displays a calibration string. This is a string of five integers separated by commas. The **--generate** option also writes your configuration to the correct location.

The tsc1d has now been calibrated for your touchscreen. You can start it in the background by typing **tsc1d &**, and then run X by typing **startx**.

For more information on tsc1d and tsc1cal, see the tsc1d README file in the tsc1d source, which is in /packages/tsc1d on the Development Kit CD.

Developing software for AEL

To ensure that an application is able to run correctly when installed on a target board, you must ensure that it is compiled and linked against the libraries that are present on the target system. This is particularly true when the processor architecture of the target board differs from the processor architecture of the host system, but is also true if the processor architecture is the same. The act of compiling for a target system that differs from the host system is known as cross compilation.

The AEL host environment contains a suite of cross compilers and other tools, as well as the libraries and headers that are necessary to compile applications for use on AEL. It also contains various tools that are useful when working with AEL target systems.

Host system requirements

The AEL host environment requires a host Linux distribution that is compatible with the Linux Standard Base[2] (LSB) version 1.3. The LSB is an attempt by Linux distribution vendors to specify a set of basic functionality that is present on any Linux distribution.

Each Linux distribution vendor who supports the LSB provides a package which ensures that the LSB functionality is present on the system. Many Linux distributions do not include LSB support in the base installation, so you may need to add it.

The LSB support package is necessarily tightly coupled with the Linux distribution and version. It is therefore important that you install the LSB package for the exact distribution version you are using as supplied by the distribution vendor. An LSB support package downloaded from anywhere else is unlikely to function correctly.

[2] www.linuxbase.org

The distributions confirmed as supporting the Arcom host environment using the LSB packages supplied by the Linux distribution vendor are listed in the following table.

Arcom recommend Fedora Core 1 as a host distribution.

Distribution	Vendor	LSB package	Notes
RedHat 7.3 (VALHALLA)	Red Hat, Inc.	redhat-lsb or on CD #3	[0]
RedHat 8.0 (PSYCHE)	Red Hat, Inc.	redhat-lsb or on CD #3	
RedHat 9 (SHRIKE)	Red Hat, Inc.	redhat-lsb or on CD #3	
Fedora Core 1 (YARROW)	Fedora Project	redhat-lsb or on CD #3	[1], [2]
Fedora Core 2 (TETTNANG)	Fedora Project	redhat-lsb or on CD #3	[2]
Mandrake Linux 10.0	mandrakesoft	lsb on CD #1	[3]
SuSE Linux 9.0	SuSE/Novell	lsb or on CD #1	[4]
Debian GNU/Linux 3.0 (WOODY)	Debian	lsb	[5], [6]
Debian GNU/Linux 'Testing' (currently SARGE)	Debian	lsb	[6], [7]
Debian GNU/Linux 'Unstable' (SID)	Debian	lsb	[6], [7]

The numbers in the **Notes** column refer to the following information:

[0] Red Hat 7.3 is shipped with version 1.1 of the LSB. It is possible to install the Arcom host environment on this platform by passing the following additional arguments to the Arcom host environment installation program: **--ignore-lsb-version** and **--ignore-package-dependencies**

[1] Red Hat discontinued their Red Hat Linux product after version 9. In its place they started the Fedora project. Therefore Fedora Core 1 can be considered the successor to RedHat 9.

[2] You can install Fedora supplied packages (such as the redhat-lsb package) using the **yum** tool. Simply run the command "**yum install <*PACKAGES*>**". **yum** requires an active Internet connection.

[3] You can install Mandrakesoft supplied packages (such as the lsb package) using the **urpmi** tool. Simply run the command "**urpmi <*PACKAGES*>**".

[4] You can install SuSE supplied packages (such as the lsb package) using the **yast** tool. Simply run the command "**yast --install <*PACKAGE*>**".

[5] Debian GNU/Linux 3.0 requires that the **--ignore-required-package-versions** argument be passed to the Arcom host environment installation program.

[6] You can install Debian supplied packages (such as the lsb package) using the **apt-get** tool. Simply run the command "**apt-get install <*PACKAGES*>**". **apt-get** requires an active Internet connection.

[7] Debian “Testing” and “Unstable” were known to work at the time of writing (July 2004). However, due to the continual updates to these distributions, this may change at any given time.

If your host distribution is not listed in the preceding table, you may still be able to install the host environment, as your distribution may be derived from one of the distributions that are listed.

If you are still unable to install the host environment but you know that your chosen distribution supports the LSB version 1.3, please contact Arcom to discuss support for your chosen distribution. Alternatively, install a supported distribution that includes support for the LSB.

The AEL host environment requires up to 150MB of disk space.

Installing the AEL host environment

To install the host environment, mount the CD and run (as root) the script **install**, which is located in the top folder of the CD. Pass any options required for your distribution, as described in the preceding section. For example (assuming your distribution is configured to mount CDs on /mnt/cdrom):

```
# mount /mnt/cdrom
# /mnt/cdrom/install [OPTIONS]
```

Some distributions are configured by default so that programs cannot be executed from removable media such as a CD. If so, a permission denied error is displayed when you enter the above command. In such cases, use the following command:

```
# perl /mnt/cdrom/install
```

Once you have installed the host environment, you must add the folder /opt/arcom/bin to your path and the folder /opt/arcom/share/man to your manual path. You can do this temporarily for the current log in session by running the following commands:

```
$ export PATH="/opt/arcom/bin:$PATH"
$ export MANPATH=”/opt/arcom/share/man:$MANPATH”
```

Alternatively, you can cause these commands to take effect for all log in sessions for a particular user by adding them to the file **.bash_profile** in that user’s home directory.

Installing additional packages into the host environment

When the host environment is installed, all of the library runtime and development packages that are supplied on the Development Kit CD are installed into it. If you obtain a library package from some other source, such as Arcom Technical Support, you must install the runtime and development packages yourself.

This section explains installing packages in the host environment, whilst Package management, on page 28, explains installing packages in the target device. The installation into both the host and target environments of applications and binaries that you have built yourself (i.e. not using packages) is explained later in this section.

Many of the packages you want to install in the host environment are in the form of .deb packages, suitable for installation on the target device. These must be converted into

LSB-compliant .rpm packages suitable for installation in the host environment. A small minority of packages (for example cross compilers) are strictly intended for the host environment only, and so are supplied directly as LSB .rpm packages.

The host environment contains a tool, **ael-cross-rpm**, that is used to convert a .deb package into an LSB-compliant .rpm package. A .deb package 'foo' is converted by the tool into an RPM package 'lsb-arcom-ARCH-linux-foo', with the correct metadata for an LSB package and the correct file system layout for the Arcom host environment.

A .deb package or set of .deb packages can be converted to LSB compliant .rpm packages by passing them to ael-cross-rpm. To do this, enter:

```
$ ael-cross-rpm libc6_2.3.1-3_arm.deb libc6-dev_2.3.1-3_arm.deb
```

When you enter this command, the following is displayed:

```
libc6_2.3.1-3_arm.deb
libc6-dev_2.3.1-3_arm.deb
```

You can create two LSB .rpm packages in the current folder by entering:

```
$  ls *.rpm
```

When you enter this command, the following is displayed:

```
lsb-arcom-arm-linux-libc6-2.3.1-3.noarch.rpm
lsb-arcom-arm-linux-libc6-dev-2.3.1-3.noarch.rpm.
```

If **ael-cross-rpm** determines that a .deb package contains no files that would be useful in the host environment, it does not produce any output for that package.

Once you have created the LSB .rpm packages, you must install them on your host system. The method for doing this varies between Linux distributions.

On a distribution that uses the RPM package manager, for example Red Hat Linux, you can install the LSB .rpm packages directly using the **rpm** tool:

```
#  rpm -ivh lsb-arcom-arm-linux-libc6-2.3.1-3.noarch.rpm
```

When you enter this command, the following is displayed:

```
Preparing...                 #################################
                            [100%]
  1:lsb-arcom-arm-linux-lib #################################  [100%]
```

On distributions that use other package managers you must use another tool to install the LSB .rpm package. For example, Debian GNU/Linux provides the tool **alien** that supports installing LSB .rpm packages:

```
#  alien -ik lsb-arcom-arm-linux-libc6-2.3.1-3.noarch.rpm
```

Like AEL, Debian uses dpkg as its package management tool. However you cannot install a .deb built for AEL directly in your Debian system (or vice-versa). Doing so could damage your host system.

You must convert the AEL .deb to an LSB RPM using **ael-cross-rpm** (so that the meta data and file system layout can be modified to be suitable for the host rather than target environments) and then install the LSB .rpm package using **alien**.

Other host distributions have a similar method for installing LSB packages. For details, consult the documentation for your host distribution.

Obtaining help

You can view help about many of the utilities provided by the host environment using the **man** utility. For example, to get help on the arm-linux-gcc compiler, run the following command:

```
$ man arm-linux-gcc
```

Cross compiling applications and libraries

This section includes a number of examples in which we use the arm-linux cross compiler. For other boards, substitute the appropriate prefix from the table below.

Compiling a simple C application is simply a matter of using the cross compiler instead of the regular compiler:

```
$ arm-linux-gcc -o example -Wall -g -O2 example.c
```

Tools available in the host environment

The majority of the cross compilation tools are the same as their native compilation counterparts, with an additional prefix that specifies the target system. The prefixes for the various architectures are described in the following table:

Architecture	Prefix	Example processors
Intel x86	i386-linux-	AMD SC520, National Geode GX1
ARM and XScale (little-endian)	arm-linux-	Intel PXA255
ARM and XScale (big-endian)	armbe-linux-	Intel IXP425

The following cross compilation tools are provided:

Tool	Description
ar	Manage archives (static libraries).
as	Assembler.
c++, g++	C++ compiler.
cpp	C pre-processor.
gcc	C compiler.
gdb	Debugger.
ld	Linker.
nm	List symbols from object files.
objcopy	Copy and translate object files.
objdump	Display information about object files.
ranlib	Generate indexes to archives (static libraries).
readelf	Display information about ELF files.
size	List object file section sizes.
strings	Print strings of printable characters from files (usually object files).
strip	Remove symbols and sections from object files (usually debugging information).

Common open source build systems and cross compilation

The majority of open source software available uses configure scripts as part of the build process. Passing **--host=arm-linux** on the command line in addition to your normal options is often all that is required. ‘Host’ in this context refers to the system that the final application is to run on. Newer configure scripts accept a **--target=arm-linux** option as an alternative, as follows:

```
$ ./configure --host=arm-linux [other options]
```

Not all configure scripts follow this behavior. In particular, build systems not generated with the autoconf and automake tools are likely to not behave as expected. Fortunately, a great many open source projects do use these tools.

Building libraries

Building libraries is similar to building applications. The libraries must be configured and built to run on the target board. This means that the --prefix must be /usr, so that libraries expect to be installed to /usr/lib/. However, the library and headers must also be installed on the build system in /opt/arcom/*TARGET*/lib and /opt/arcom/*TARGET*/include (where *TARGET* is the cross compile prefix, such as arm-linux or i386-linux), so that the cross compiler and linker can use them.

With a standard automake and autoconf build system, this can be achieved by entering:

```
# make install prefix=/opt/arcom/arm-linux
```

Alternatively, you could install to /tmp/myapp-tmp and move the libraries and headers into /opt/arcom/*TARGET* by hand.

Installing applications and libraries on the target

Installing an application or library that uses the automake tool is normally achieved by calling the install target, as follows:

```
$ make install
```

This causes the application or library to be installed into the host file system, potentially causing enormous damage to the system.

Consider the consequences of replacing libc on an X86 system with a libc cross compiled for an ARM system.

Perform the build and install of applications for the target as a non-root user on the host system, as this can prevent the worst disasters.

Fortunately, automake provides a variable called DESTDIR that is used as the base folder for installation. DESTDIR is normally an empty string, but you can define it to install into a temporary folder, as follows:

```
$ make install DESTDIR=/tmp/target-install
```

You can now remove from tmp/target-install any files that you do not want to install on the target, such as documentation and static libraries (*.a), and transfer it to the board:

```
$ cd /tmp/target-install
```

You may also want to use the strip command to remove unnecessary symbols from any application binaries or libraries:

```
$ arm-linux-strip --strip-unneeded ./usr/bin/app
$ arm-linux-strip --strip-unneeded ./lib/libtmp.so
```

Custom build systems and cross compilation

There are projects that use build systems which differ from the de-facto standard discussed above. There is no simple recipe for building applications with non-standard build systems. You must consult build instructions, README files and similar documentation, study the build system makefiles, and so on.

The RUTE Linux Tutorial (which is on the Development Kit CDROM) contains sections about the use of make and makefiles.

You can normally edit the makefile to prefix all references to tools mentioned in Tools available in the host environment (page 35) with the correct cross compilation prefix. A makefile often defines variables such as CC and CXX to contain the C and C++ compilers respectively. In general, setting environment variables to override these before running **make** meets with some success:

```
$ CC=arm-linux-gcc CXX=arm-linux-c++ make
```

You must install to a temporary folder so as not to overwrite your build system's native libraries and binaries. A custom build system may not make use of the DESTDIR variable. Consult any documentation you have and examine the makefile for install targets or similar.

Cross compilation example

A trivial example application that utilizes a shared library is included on the Development Kit CD. The two source tarballs are in /examples/c/trivial/trivial-app-1.0.tar.gz and /examples/c/trivial/libtrivial-1.0.tar.gz. This example uses a standard build system based on the **autoconf**, **automake** and **libtool** utilities. The procedure is therefore most relevant to real applications using the same build system.

The **autoconf**, **automake** and **libtool** utilities are commonly used by Open Source projects. They can be found at www.gnu.org/software/autoconf/, www.gnu.org/software/automake/, and www.gnu.org/software/libtool/, respectively.

Building the shared library

To build the shared library, follow these steps:

1 Unpack the library distribution using the following commands:

```
$ tar xzf libtrivial-1.0.tar.gz
$ cd libtrivial-1.0
```

2 Configure the library as follows:

```
$ ./configure --prefix=/usr --host=arm-linux
$ make
```

Note the use of the **--host** option. For more information about this option, see Common open source build systems and cross compilation, page 36.

3 Install the library on the host (build) system by entering the following command:

```
# make install prefix=/opt/arcom/arm-linux
```

4 Install for the target (this is transferred to the target later), as follows:

```
$ make install-strip DESTDIR=/tmp/trivial-app
```

Building the application

To build the application, follow these steps:

1 Unpack the application distribution:

```
$ tar xzf trivial-app-1.0.tar.gz
$ cd trivial-app-1.0
```

2 Configure the application:

```
$ ./configure --prefix=/usr --host=arm-linux
$ make
```

Again, note the use of the **--host** option. For more information about this option, see Common open source build systems and cross compilation, page 36.

3 Install for the target:

```
$ make install-strip DESTDIR=/tmp/trivial-app
```

Installing on the target

To install the application on the target, follow these steps:

1 Generate the tarball to be installed based on the installed files:

```
$ cd /tmp/trivial-app
$ tar czvf ../trivial-app.tar.gz *
$ cd ..
```

For a real application, you may want to remove files that are not required on the target (such as documentation) from the temporary folder before you build the tarball.

2 Transfer the installation tarball to the target board (named penguin.example.net in this example) using scp:

```
$ scp trivial-app.tar.gz root@penguin.example.net:/tmp
```

3 Enter the following command on the target board (as root):

```
# cd /
# tar xzvf /tmp/trivial-app.tar.gz
```

4 Update the shared library cache:

```
# ldconfig
```

5 Run the application:

```
$ trivial-app
```

The following message is displayed:

```
This is a trivial application.
This is a trivial function in a trivial shared library.
```

Debugging applications on the target

The AEL host environment includes the GNU Debugger (GDB). GDB enables you to control the execution of your application, examine program state and view the application code. GDB is a symbolic debugger. This means that you can debug your application using the function and variable names that were used in the source code.

The main GDB application binary is automatically installed on your host system as part of the host environment. The recommended method of debugging target applications using GDB is over an Ethernet connection. To do this you must install a small server binary, 'gdbserver', on the target board. This server is extremely small and consumes minimal memory. It is provided on the Development Kit CD in the gdbserver package, which is in the /packages/gdb/ folder. For an explanation of how to install it, see Adding packages, page 28.

It is possible to run GDB directly on the target board by installing the GDB package from the Development Kit CD. We do not recommend this because GDB is relatively large and memory-hungry. Furthermore, you would not benefit from full symbolic debugging as your application source code is not available on the target.

This section describes how you can use GDB to debug an application running on a remote target. Please consult the GDB manual for more information about using GDB. The manual is available on the Development Kit CD in /manuals/gdb.pdf, or from the GDB website at www.gnu.org/software/gdb.

Compiling an application for debugging

Before you can debug a program using the full symbolic information, you must compile and link the application with full symbolic information. This is done by adding the **–g** option to both compile and link commands. For example, to compile a simple hello world application, enter the following:

```
$ arm-linux-gcc –g –o hello hello.c
```

Once compiled, copy the binary to your target system using scp:

```
$ scp hello root@penguin.example.net:/tmp/hello
```

It is possible to debug optimized programs with GDB, but you may find that optimizations such as those performed by the –O2 interfere with GDB's understanding of program flow and variable location. We recommend that you disable optimizations in your application while you are debugging it.

Starting GDB and GDB server

Now that the binary has been compiled with debugging information and copied to the board, you can start the gdbserver and initiate a GDB session on your target system. To do this, follow these steps:

1. Start the gdbserver process on your target system, giving a port number to listen on (9000 in the following example) followed by the program to be debugged:

   ```
   $ gdbserver :9000 /tmp/hello “hello world”
   ```

   Arguments can be passed to the application by adding them to the gdbserver command line. The above command starts **gdbserver**, which then loads **/tmp/hello**, passing the string **“hello world”** as an argument. It then stops before running the application and waits for a remote GDB session to be initiated.

2. Initiate a GDB session on the host and pass the application binary as a parameter to GDB:

   ```
   $ arm-linux-gdb hello
   ```

   GDB starts, displaying a banner followed by a (gdb) prompt.

3. Connect to the remote system, assuming it is 192.168.1.4, by typing:

   ```
   (gdb) target extended-remote 192.168.1.4:9000
   ```

   This connects to the gdbserver process you started in step 1. The **extended-remote** option offers several advantages over the normal **remote** option, including the ability to restart the program being debugged without restarting gdbserver.

   Once the remote connection is established any of the normal GDB commands may be used to debug the application, such as setting a breakpoint on the main() function and continuing:

   ```
   (gdb) b main
   (gdb) c
   ```

Compiling a kernel

Overview

This section explains how to build a new Linux kernel, along with the associated modules, and install them on your target board.

To cross compile the kernel, you must have the cross compilation environment installed. For more information, see Installing the AEL host environment, page 33.

An archive containing the patched Linux kernel source tree used in the Development Kit is installed by the AEL host environment as /opt/arcom/*TARGET*/source/linux-source-*VERSION*.tar.gz. Refer to your board's Quickstart manual for additional board-specific instructions.

Unpacking and configuring the kernel

The kernel must be recompiled on your host system.

To unpack and configure the kernel, follow these steps:

1. Unpack the source code:

   ```
   tar zxf /opt/arcom/TARGET/source/linux-source-VERSION.tar.gz
   $  cd linux-source-VERSION
   ```

   The folder linux-source-*VERSION* is created.

2. Configure your kernel image. You can get a default configuration by running the following command:

   ```
   $  make TARGET_config
   ```

   Where ***TARGET*** is the name of the board. Consult your Quickstart Manual to determine the correct default configuration for your target board.

3. Optional. Tweak the options to your satisfaction using the kernel configuration tools, once you have a default configuration for your board. For example you can use the **menuconfig** tool, as follows:

   ```
   $  make menuconfig
   ```

4. Save your configuration once you have made any changes required, and exit.

Building the kernel

Once the kernel has been configured it can be built using the ael-kernel-build tool. This tool builds the kernel and associated modules, as well as any external kernel modules, as required.

The basic invocation of the **ael-kernel-build** utility is as follows:

ael-kernel-build [*OPTIONS*]… <*TARGETS*>

The main **OPTIONS** that you can specify when using ael-kernel-build are explained in the following table:

Option	Description
--architecture=*ARCH*	Required. Specifies the processor architecture to build the kernel for. Use one of the architecture names from the table below.
--revision=*REVISION*	Specifies the version number to be used in the packages that are created. This is typically a single integer or a word combined with an integer, for example ‘1’ or ‘customer.1’. The numeric part of the revision should be incremented with each release.

The architectures for which you can build the kernel are listed in the following table:

Processor architecture	Architecture name	Example processors
Intel x86	i386	AMD SC520, National Geode GX1
ARM and XScale (little-endian)	arm	Intel PXA255
ARM and XScale (big-endian)	armbe	Intel IXP425

The possible **TARGETS** are described in the following table. In most cases, you use **image** to build a kernel image and the in-tree modules, and perhaps **modules** to build any out-of-tree modules.

Target	Description
image	Builds a package containing a kernel image and a package containing the in-tree kernel modules.
clean	Cleans the kernel build tree.
modules	Builds packages for any external modules for which a suitable source package is installed in the host environment.
modules-clean	Cleans the build trees for any external modules that have been built.

For more information on ael-kernel-build and the available options and targets, consult the ael-kernel-build man page.

Installing the new kernel and modules

Once you have built a new kernel and modules you must install them. The method used to install a new kernel varies from board to board.

On some boards the Linux kernel image .deb package can be installed using the **dpkg** command, as described in Package management, on page 28. If this is true for your board, the existing kernel image is in the /boot folder on the target. As well as installing the kernel image, you may also need to update the RedBoot boot script to load the new kernel. You do this by either editing the boot script itself using the **fconfig** command (see page 52) or by changing the alias used by the boot script to specify the kernel filename.

On other boards the kernel is stored in a FIS partition named ‘kernel’. If this is true for your board, a partition named ‘kernel’ is in the output from the RedBoot **fis list** command (described on page 57). In this case you can update the image with the new kernel, following the instructions in Managing images in Flash on page 57. The kernel image to load can be extracted from the Linux image package (where it is called /boot/vmlinuz-VERSION), or directly from the arch/<ARCH>/boot subfolder of the build tree (where it is named zImage or bzImage, depending on the platform).

See your Quickstart Manual for more details.

The Linux modules package can always be installed using the standard package management tools. See Package management, page 28.

Common embedded software development tasks

Accessing the physical address space

When developing software for AEL, it is sometimes necessary to access the processor's physical address space from within your application. For example, you may need to tweak memory mapped register settings within the CPU or to access some external peripheral such as a GPIO pin.

A Linux application may only access the virtual address space that the Linux kernel has created for it. This virtual address space often does not contain mappings to the physical regions that you want to use. An application can create a mapping to an arbitrary physical address by opening the /dev/mem special device file, which provides access to the entire physical address space, and then performing an mmap() operation on the file handle.

For security reasons, the application must be running as root in order to do this.

Getting the specifics of this operation correct can be tricky, so we provide a library (libdevmem) that handles this complexity for you. The library provides a number of functions:

```
libdevmem_handle libdevmem_open(unsigned long base,
               unsigned long length);
int libdevmem_close(libdevmem_handle handle);
TYPE_t libdevmem_read_TYPE(libdevmem_handle handle,
              unsigned long offset);
void libdevmem_write_TYPE(libdevmem_handle handle,
             unsigned long offset,
             TYPE_t value);
```

The following table explains the functions provided by libdevmem (which are also described in /opt/arcom/*TARGET*/include/libdevmem.h):

Function	Description
libdevmem_open	Maps a physical address into the current virtual address space, returning a **libdevmem_handle** that serves as a handle onto the mapping. Takes the physical address to map and the length of the region to map and returns. Returns NULL on error and sets errno accordingly.
libdevmem_close	Unmaps an existing mapping. Takes a handle as returned by **libdevmem_close**. Returns 0 on success, –1 on error, and sets errno accordingly.
libdevmem_read_*TYPE*	Reads a value from an offset within a mapping. Takes a handle and an offset and returns a ***TYPE*_t** that is the value that was read. ***TYPE*** may be one of uint8, uint16 or uint32.
libdevmem_write_*TYPE*	Writes a value to an offset within a mapping. Takes a handle, an offset and a ***TYPE*_t** that is the value to write. ***TYPE*** may be one of uint8, uint16 or uint32.

An application that wishes to use libdevmem must include the libdevmem.h. It must also include libdevmem when linking:

```
$  arm-linux-gcc –o application application.o -ldevmem
```

The library is currently available only as a static library.

Using an Arcom PC/104 I/O board

AEL comes pre-installed with drivers for a selection of Arcom's PC/104 I/O boards. The following AIM104 peripheral boards are supported:

Board	Module	Device node
AIM104-RELAY8/IN8	aim104-relay.o	/dev/arcom/aim104/relay8/{0..7}
AIM104-IN16	aim104-in16.o	/dev/arcom/aim104/in16/{0..7}
AIM104-OUT16	aim104-out16.o	/dev/arcom/aim104/out16/{0..7}
AIM104-IO32	aim104-io32.o	/dev/arcom/aim104/io32/{0..7}
AIM104-MULTI-IO	aim104.multi-io.o	/dev/arcom/aim104/multi-io/{0..7}

Peripheral boards containing serial ports, such as the AIM104-COM4 and the AIM104-COM8, are supported by the standard 16550 UART driver in the Linux kernel. These boards can be configured using **setserial**, as described on page 12.

The drivers are supplied as kernel modules that must be loaded to access the device. Each module can support up to eight individual boards that are configured by passing a list of I/O addresses using the **io_base=** module parameter. This is a comma-separated list of addresses. For example, if you have two AIM104-RELAY8/IN8 boards with base addresses configured as 0x180 and 0x184:

```
modprobe aim104-relay8 io_base=0x180,0x184
```

Once the kernel driver has been loaded then the **libaim104** library can be used to access the AIM104 boards. The library provides functions for each of the peripheral boards. The functions are defined in the C header file arcom/libaim104.h.

Each function takes a file handle obtained by opening the device node listed in the above table. All functions return 0 or a positive value on success, and a negative value on failure. The possible libaim104 error codes are explained in the following table:

Error code	Explanation
AIM104_SUCCESS = 0	Success. No error occurred.
AIM104_EBAD_CHANNEL = -10	The given channel is invalid.
AIM104_EIO = -20	Low-level I/O error. Check errno for details.
AIM104_ERANGE = -30	The given value was out of range.

AIM104-RELAY8/IN8

The library libaim104 provides several functions on AIM104-RELAY8/IN8:

```
int aim104_relay8_enable_relays(int fd, int enable);
int aim104_relay8_set_all(int fd, unsigned char set);
int aim104_relay8_set_masked(int fd, unsigned char mask,
             unsigned char set);
int aim104_relay8_inputs(int fd);
int aim104_relay8_relay_status(int fd);
```

The libaim104 functions on AIM104-RELAY8/IN8 are explained below:

Function	Explanation
aim104_relay8_enable_relays	Enables all relays if the **enable** parameter is true.
aim104_relay8_set_all	Sets all 8 relays to the state given by **set**.
aim104_relay8_set_masked	Sets all relays selected by the **mask** parameter to the state given by the **set** parameter. A relay is selected if there is a 1 in the corresponding bit position within **mask**.
aim104_relay8_inputs	Returns the state of the 8 digital I/O lines as a bit mask.
aim104_relay8_relay_status	Returns the current state of the 8 relays as a bit mask.

AIM104-IN16

The function provided by the library libaim104 on AIM104-IN16 is:

```
int aim104_in16_inputs(int fd, int channel);
```

The function **aim104_in16_inputs** returns the state of digital I/O lines 0-7 as a bit mask if **channel** is 0, or lines 8-15 if **channel** is 1.

AIM104-OUT16

The functions provided by the library libaim104 on AIM104-OUT16 are:

```
int aim104_out16_enable_outputs(int fd, int enable);
int aim104_out16_set_all(int fd, int channel,
           unsigned char set);
int aim104_out16_set_masked(int fd, int channel,
            unsigned char mask,
            unsigned char set);
int aim104_out16_output_status(int fd, int channel);
```

When the parameter **channel** is zero, it selects I/O lines 0-7. When **channel** is one, it selects I/O lines 8-15.

The functions provided by libaim104 on AIM104-OUT16 are explained below:

Function	Explanation
aim104_out16_enable_outputs	Enables the outputs if **enable** is 1, disables otherwise.
aim104_out16_set_all	Sets outputs selected by **channel** to the state given as set.
aim104_out16_set_masked	Sets outputs on **channel** selected by the **mask** parameter to the state given by the **set** parameter. A relay is selected if there is a 1 in the corresponding bit position within **mask**.
aim104_out16_output_status	Returns the current status of the outputs selected by **channel**.

AIM104-IO32

The functions provided by the library libaim104 on AIM104-IO32 are:

```
int aim104_io32_enable_outputs(int fd, int enable);
int aim104_io32_set_all(int fd, int channel, unsigned char set);
int aim104_io32_inputs(int fd, int channel);
```

The parameter **channel** selects I/O lines 0-7 when zero, I/O lines 8-15 when one, I/O lines 16-23 when two and I/O lines 24-31 when 3.

The functions provided by libaim104 on AIM104-IO3 are explained below:

Function	Explanation
aim104_io32_enable_outputs	Enables the outputs if **enable** is 1, disables otherwise.
aim104_io32_set_all	Sets the 8 outputs selected by **channel** to the state given by set.
aim104_io32_inputs	Returns the status of the inputs selected by channel.

AIM104-MULTI-IO

The functions provided by the library libaim104 on AIM104-MULTI-IO are:

```
int aim104_multi_io_inputs(int fd);
int aim104_multi_io_ADC(int fd, int channel, int single_ended);
int aim104_multi_io_DAC(int fd, int channel,
           unsigned short output);
```

The functions provided by libaim104 on AIM104-MULTI-IO are explained below:

Function	Explanation
aim104_multi_io_inputs	Returns the status of the 8 digital I/O lines as a bit mask.
aim104_multi_io_ADC	Returns a value read from one of the ADC inputs. For an ADC input connected in single ended mode **single_ended** must be set to true/non-zero and **channel** must be between 0 and 15 inclusive. For an ADC input connected in differential mode, **single_ended** must be set to false/zero and **channel** must be between 0 and 7 inclusive.
aim104_multi_io_DAC	Writes the 12 bit unsigned value (0-4095) output to DAC channel 0 or 1.

Transferring files to and from the target board via a serial connection

If you want to transfer a relatively small file to or from a target board running Linux without setting up an Ethernet connection, you may want to do it via the serial line.

If you have many files to transfer or the files are large, we recommend you configure an Ethernet connection and use the **scp** tool as described in Secure Shell, page 23. A serial connection typically runs at less than 115200Kb per second, compared with up to 10 or 100Mb for an Ethernet connection.

AEL includes the **lrzsz** utility, which is capable of performing X-, Y- and Z-modem transfers over a serial line. The protocol you choose depends on the protocols supported by your terminal emulator. If possible, use Z-modem in preference to Y-modem, and use X-modem only if nothing better is available. Z- and Y-modem are capable of transferring multiple files in one session, while X-modem can only transfer a single file at a time.

Before transmitting or receiving the file, you must run the appropriate utility on the target system. The following table describes X-, Y- and Z-modem transfer utilities:

Protocol	Receive on the target system	Transmit from the target system
Z-modem	rz	sz [FILES...]
Y-modem	rb	sb [FILES...]
X-modem	rx [FILE]	sx [FILE]

To receive files from the host system on the target system, use the **rz**, **rb** or **rx** utilities. The **rz** and **rb** utilities do not take any parameters, while the **rx** utility requires you to give the name of the file to be received.

To transmit files from the target to the host system, use the **sz**, **sb** or **sx** utilities. The **sz** and **sb** utilities can take any number of files to send as parameters, while the **sx** utility can only transmit a single file.

Once you have started a utility on the target system, initiate the appropriate type of file transfer from the terminal emulator on your host. To do this in minicom, press **Ctrl-A** followed by **S** to send or **Ctrl-A** followed by **R** to receive.

Extracting an image of the on-board Flash

An image can be extracted from the on-board Flash and sent via an Ethernet connection to a host system using the netcat tool (nc).

If you intend to download the image to other target devices, remove the SSH cryptographic keys before taking an image. Otherwise, all of your targets will have the same private keys. The SSH keys are stored in /etc/ssh in the following files:

ssh_host_dsa_key
ssh_host_dsa_key.pub
ssh_host_rsa_key
ssh_host_rsa_key.pub
ssh_host_key
ssh_host_key.pub

A unique new set of keys is generated on each board the first time it is booted.

For example, to extract the first Flash partition from a board to a host with IP address 192.168.1.5, follow these steps:

1 Start a netcat process on the host system listening on a port (4000 in this example) by entering:

```
$ nc –1 –p 4000 > Flash0.img
```

2 Extract the Flash image and send it to the host system by running the following on the target system you want to take an image of:

```
# dd if=/dev/mtd0 | nc 192.168.1.5 4000
```

If you want to take an image of the entire Flash device, rather than just individual partitions, you can repeat the above procedure for each Flash partition.

A complete list of the Flash partitions on your board is provided in the special file /proc/mtd.

You can combine the images on the host using the following command:

```
$ cat Flash0.img Flash1.img ... FlashN.img > Flash.img
```

Single partitions or images of an entire Flash part can be downloaded onto a target board, as described in Managing images in Flash, page 57.

RedBoot

RedBoot is a complete bootstrap environment for embedded systems. Based on the eCos Hardware Abstraction Layer, RedBoot inherits the eCos qualities of reliability, compactness, configurability and portability.

The primary function of RedBoot is to bring up the board to the point where control can be handed off to another operating system, such as Linux. However, RedBoot also offers the facility to download binary images via a serial or Ethernet connection and to update the on-board Flash array.

Ethernet download and debug support is included. RedBoot can configure its IP parameters via BOOTP, DHCP or statically via the Flash configuration block. Images can be downloaded via Ethernet using TFTP or HTTP, or over a serial connection using X- or Y-modem.

The RedBoot command line

RedBoot provides an interactive command line interface which allows management of Flash images, image download, RedBoot configuration, and so on. The command line interface is accessible via the serial console or as a telnet connection via Ethernet. If, however, the target has been configured with a boot script that launches an application or operating system, the Ethernet console is not available and so you must use the serial console. Any boot script that has been configured can be aborted by pressing **Ctrl-C** on the serial console during the early stages of the boot process. Once the board has dropped to a RedBoot prompt, you may use either the serial console or telnet via port 9000.

Configuring and using RedBoot

This section explains how to download images, update the Flash and execute applications and operating systems. For information about the more advanced features of RedBoot, refer to the eCos Reference manual, which is on the Development Kit CD in the folder /manuals/.

You can find out about an individual RedBoot command by typing:

```
RedBoot> help <command name>
```

The RedBoot Flash configuration block

RedBoot contains a Flash configuration system that includes information such as a boot script and networking configuration. The options can be examined and modified using the **fconfig** command.

If called with no parameters, **fconfig** prompts you for each available option in turn. Alternatively, when entering **fconfig**, you can include the following parameters:

```
fconfig [-i] [-l] [-n] [-d] nickname [value]
```

The following table explains the parameters you can specify when using **fconfig**:

Parameter	Action
-i	Initialize the Flash configuration block to default values.
-l	List the settings in the Flash configuration block.
-n	Use nicknames rather than full names.
nickname	Only act on the named configuration item.
value	Value to set to.

The most interesting configuration options are those that allow the use of BOOTP or a static network configuration to configure the on-board Ethernet and run a script at boot. Refer to the Quickstart manual for your board for more information about a suitable script to use.

Aliases

RedBoot can store aliases (simple macros) with the **fconfig** configuration parameters. Aliases are defined with the **alias** command:

```
RedBoot> alias zimage /boot/vmlinuz
RedBoot> alias cmdline "\"console=ttyS0,115200n8 root=/dev/mtdblock1 ro\""
```

Note the use of quotes and backslashes.

Configuring an IP address

You can use **fconfig** to configure RedBoot with an IP address using either BOOTP/DHCP, or a static address. In addition, you can use the **ip_address** command to specify an IP address from the command line.

When entering the **ip_address** command, you can specify the following parameters:

```
ip_address [-b] [-l <local IP address>[/<mask_length>]] [-h <server address>]
```

These parameters are explained in the following table:

Parameter	Action
-b	Obtains an IP address via BOOTP.
-l <*local IP address*>[/<*mask length*>]	Sets the local IP address to the address entered. You can also enter the network **mask length**.
- h <server address>	Specifies the default **server address** used by the **load** command.

For example, to configure the static IP address 10.2.2.4 with a network mask of 255.255.0.0 (hence length 16) and default server of 10.2.1.1:

```
RedBoot> ip_address –l 10.2.2.4/16 –h 10.2.1.1
```

Alternatively, to configure an IP address via BOOTP or DHCP and override the supplied server address with 10.2.1.1:

```
RedBoot> ip_address –b –h 10.2.1.1
```

When run with no parameters, the **ip_address** command reports the current IP configuration:

```
RedBoot> ip_address
IP: 10.2.40.120/255.255.0.0, Gateway: 10.2.1.1
Default server: 10.2.1.1
```

Loading images into RAM

An image may be loaded into RAM over the serial line (not recommended for large images as it is slow), over the Ethernet connection or from a JFFS2 file system in the on-board Flash. When loading over the serial line, the X- or Y-modem protocol may be used. When loading via Ethernet, the TFTP or HTTP protocols can be used. Before an image can be loaded via Ethernet, an IP address must be configured using either the **fconfig** or **ip_address** commands. Before an image can be loaded from a JFFS2 file system it must be mounted using the **mount** command, as described in the next section.

When downloading an image via any method, you must provide an address in RAM where there is enough free space to contain the image. You can see the free regions of RAM in the output of the **version** command. Alternatively, the macro **%{FREEMEMLO}** evaluates to the base of free memory. This is useful because it avoids the need to hardcode addresses in your boot scripts. The following examples use the **%{FREEMEMLO}** macro.

Images are loaded into RAM using the **load** command. When entering the **load** command, you can specify the following parameters:

```
load [-r] [-b <base address>] [-h <hostname>] [-m <method>] <filename>
```

These parameters are explained in the following table:

Parameter	Action
-r	Loads a raw image. The default is to load an SREC format image.
-m <method>	Specifies the transfer method. This can be **http**, **tftp**, **file**, **xmodem** or **ymodem**.
-h <host>	Specifies the TFTP or HTTP server address, if you are using one of these methods. The server address given to the **ip_address** command, the address supplied by the BOOTP server or the server address in the Flash configuration block is used (in that order).
-b <base address>	Specifies the address to load the image to.
<filename>	The name of the file to load, if you are loading via TFTP or HTTP or from a file on a JFFS2 file system.

In almost every situation you can use the **–r** and **–b** parameters to load a raw image to a specific address. Loading an SREC format image is beyond the scope of this manual and is described in the eCos Reference Manual.

For example, the following command loads the raw file application.img from the default TFTP server to address 0x200000:

```
RedBoot> load -r -b 0x200000 -m tftp application.img
```

Download via serial connection

Images are downloaded over the serial line using the X- or Y-modem protocols. You can initiate a transfer by entering the following command:

```
RedBoot> load –r –b %{FREEMEMLO} –m ymodem
```

Once you have done this, you must start a Y-modem upload from within your serial terminal emulation program. Under minicom, this is done by pressing **Ctrl-A** followed by **S**.

Download via Ethernet

If you do not have either a web (HTTP) or a TFTP server on your network, refer to your host system documentation for information about setting one up. If the image you want to download is small, it may be quicker and easier to perform the download over a serial connection.

> The following instructions assume that you have placed the image you want to download into the root folder of the TFTP or HTTP server, in a file named image.img.

To load the image, follow these steps:

1. Configure the network interface if you have not already configured the board with an IP address. To do this, enter:

   RedBoot> **ip –l *IP_ADDRESS* -h *SERVER_IP_ADDRESS***

2. Load the image over TFTP or HTTP as follows:

   - If using TFTP, enter:

     RedBoot> **load –r –b %{FREEMEMLO} –m tftp image.img**
   - If using HTTP, enter:

     RedBoot> **load –r –b %{FREEMEMLO} –m http image.img**

Loading from a JFFS2 file system

RedBoot provides a **mount** command that performs a similar function to the Linux mount command. Only JFFS2 file systems are currently supported by RedBoot. However, JFFS2 must still be specified using the **–t** parameter.

RedBoot> **mount –t jffs2 –f *<partition>***

The **–f** parameter specifies the FIS partition that contains the JFFS2 file system to access. You can obtain a list of the currently defined partitions using the **fis list** command described below. Only one file system may be mounted at a time. The **umount** command unmounts the currently mounted file system:

RedBoot> **umount**

The **load** command is used to load files from the file system:

RedBoot> **load –r –b %{FREEMEMLO} –m file /dir/image.img**

Managing images in Flash

Flash is managed under RedBoot using the FIS (Flash Image System) command. This command has several subcommands that can be used to update the entire Flash from an image in RAM, or to create and manage images (partitions) within the Flash.

Each FIS partition table entry has the following properties:

Property	Description
Name	A descriptive name for the image.
Flash Base	The offset of the image in Flash.
Memory Base	The virtual address of the address in memory that the image should be loaded to by the **fis load** command.
Size	The total length of the image.
Data Length	The length of the data currently stored in the image.
Entry Point	The entry point of the image. This is normally equal to the Flash base (for execute-in-place images) or the memory base.

The following table describes the most commonly used FIS subcommands:

Subcommand	Action
init	Initializes the FIS partition table.
list	Lists the current contents of the FIS partition table.
create	Creates a new FIS partition.
load	Loads a partition or other region from Flash into RAM.
lock	Locks a partition or other region (if available in hardware).
unlock	Unlocks a partition or other region (if available in hardware).
write	Writes data from RAM to Flash ignoring the any partitions.

Full documentation of the FIS system can be found in the eCos Reference Manual.

Initializing the FIS partition table

Arcom hardware is shipped with a valid FIS partition table. If you have erased the entire Flash device or want to reinitialize the partition table, you can use the **fis init** command.

Examining the FIS partition table

The current Flash partition list can be viewed using the **fis list** command:

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
FIS directory     0x00000000  0x00000000  0x0001F000  0x00000000
RedBoot config    0x0001F000  0x0001F000  0x00001000  0x00000000
```

Creating a new FIS partition table entry

Images are created in the FIS table using the **fis create** command. When entering this command, you can specify the following parameters:

fis create -b <memory base> -l <image length> [-s <data length>] [-f <flash address>] [-e <entry point>] [-r <ram address>] [-n] <name>

The following table explains the parameters you can specify when using the **fis create** command:

Parameter	Explanation
-b <memory base address>	The address in memory of the image to write to Flash. This defaults to the last image loaded by the **load** command.
-l <image length>	The length of the image in Flash. This defaults to the length of an existing image with the same name or to the size of the last image loaded with the **load** command (rounded up to a whole erase block). This corresponds to the **Size** parameter described on page 57.
-s <data length>	The length of the data stored in this image, which defaults to the length of the last image loaded with the **load** command (rounded up to a whole erase block). This corresponds to the **Data Length** parameter described on page 57.
-f <flash address>	The offset in Flash where this image resides. FIS defaults to trying any region of unallocated Flash large enough for the image being created. This corresponds to the **Flash Base** parameter described on page 57.
-e <entry point>	The entry point of this image, which defaults to the base address of the last image loaded with the **load** command. This corresponds to the **Entry Point** parameter described on page 57.
-r <ram address>	The address this image is to be loaded to. This defaults to the address given by **-b**. This corresponds to the **Memory Base** parameter described on page 57.
-n	If given, **fis create** only updates the image table and not the image itself.
<name>	The name of the image.

For example, the following command creates an image table entry named ‘application’:

```
RedBoot> fis create -f 0x00002000 -l 0xfe000 -b 0x200000 -e 0x200000 -n application
```

This does not update the image itself. The image resides from offset 0x00002000 to 0x00100000 in Flash and has a memory address and entry point of 0x200000.

More commonly, only the **–f** and **–l** options are required. For example to create an image named ‘kernel’ at offset 0x200000 in Flash with length 0x100000.

```
RedBoot> fis create -f 0x200000 -l 0x0x100000 kernel
```

The image data and other parameters are taken from the preceding **load** command.

Updating a single FIS image

Once the image has been loaded into RAM you can use the **fis create** command to update it. The location of the image in RAM and the size is remembered from the last **load** command. For example, to update a partition named ‘filesystem’, enter the following (assuming you have already loaded the image into RAM as described previously):

```
RedBoot> fis create filesystem
```

Loading a Flash image into RAM

An image that is stored in the Flash can be loaded into RAM using the **fis load** command. By default the image is loaded to the RAM address stored in the partition table entry (supplied by the **–r** or **–b** parameters to **fis create**). The RAM address can be overridden using the **–b** parameter. For example, to load the image named ‘kernel’ into RAM at the address given by the Memory Base property in the partition table entry:

```
RedBoot> fis load kernel
```

To load the image named ‘kernel’ to the base of available memory instead of the address stored in the partition table:

```
RedBoot> fis load –b %{FREEMEMLO} kernel
```

Unlocking the Flash

Some Flash devices require the Flash to be unlocked before writing. The **fis unlock** command is therefore provided on boards with these devices. This command takes either a Flash offset to start unlocking from (using the **–f** parameter) and a length in bytes to unlock (using the **–l** parameter), or the name of an existing Flash image to unlock.

To unlock the entire Flash enter the following (where **FLASH_SIZE** is the size in bytes of the Flash device):

```
RedBoot> fis unlock –f 0x0 –l <FLASH_SIZE>
```

To unlock a FIS partition named ‘filesystem’:

```
RedBoot>  fis unlock filesystem
```

Updating the entire Flash

As well as updating individual portions of the Flash, it may also be desirable to reload the entire Flash array, for example to reload the Flash image the board shipped with.

To update the entire Flash you must first load the image into RAM as described in the section Loading images into RAM on page 54. Once the image is in RAM you may need to unlock the Flash, as described above, before using the **fis write** command to write the image into Flash.

```
RedBoot> load –r –b %{FREEMEMLO} flash.img
RedBoot> fis unlock –f 0 –l <FLASH_SIZE>
RedBoot> fis write –f 0x0 –b %{FREEMEMLO} –l <FLASH_SIZE>
```

FLASH_SIZE is the size of the Flash in bytes. For an F16 board this would be 0x1000000.

Executing an image

There are two commands that can be used to execute an application or another operating system under RedBoot. They are **go** and **exec**. Typically the **go** command is used to execute an application while **exec** is primarily designed to launch Linux.

go

The **go** command jumps directly to the supplied virtual address. If no address is given, it either jumps to the entry address of the most recently loaded image (which could be specified by the FIS table entry), or it may default to the load address of the image.

The MMU is left enabled and set up and the processor is left in privileged mode. For example, if your application binary is loaded and linked to run at virtual address 0x200000, the following command jumps to the application:

```
RedBoot> go 0x200000
```

At this point, control is transferred directly to your application at 0x200000. The application may choose to continue using the existing MMU settings or may tear down the current configuration and reinitialize using the desired options.

exec

The **exec** command disables the MMU before jumping to the physical address supplied. The primary purpose of this command is to execute a Linux kernel image, but it could be used to launch any application that had been designed to be launched in this way.

When entering the **exec** command, you can specify the following parameters:

```
exec [-b <virtual address>] [-l <length>] [-c “command line”] [<entry point>]
```

These parameters are explained in the following table:

Parameter	Action
-b <virtual address>	The virtual base address of the image. This defaults to the base address of the last loaded image.
-l <length>	The length of the image, in bytes. This defaults to the length of the last image loaded.
-c “command line”	The command line to pass to Linux (or the application).
<entry point>	The physical address of the entry point of the application.

Appendix A - Contacting Arcom

Arcom sales

Arcom's sales team is always available to assist you in choosing the board that best meets your requirements. Contact your local sales office or hotline.

Sales office US

Arcom
7500W 161st Street
Overland Park
Kansas
66085
USA

Tel:	913 549 1000
Fax:	913 549 1002
E-mail:	us-sales@arcom.com

Sales office Europe

Arcom
Clifton Road
Cambridge
CB1 7EA
UK

Tel:	01223 411 200
Fax:	01223 410 457
E-mail:	euro-sales@arcom.com

Full information about all Arcom products is available on our Web site at www.arcom.com.

While Arcom's sales team can assist you in making your decision, the final choice of boards or systems is solely and wholly the responsibility of the buyer. Arcom's entire liability in respect of the boards or systems is as set out in Arcom's standard terms and conditions of sale. If you intend to write your own low level software, you can start with the source code on the disk supplied. This is example code only to illustrate use on Arcom's products. It has not been commercially tested. No warranty is made in respect of this code and Arcom shall incur no liability whatsoever or howsoever arising from any use made of the code.

Technical support

Arcom has a team of technical support engineers who can provide assistance if you have any problems with your board.

Technical support US

Tel:	913 549 1010
Fax:	913 549 1001
E-mail:	us-support@arcom.com

Technical support Europe

Tel:	+44 (0)1223 412 428
Fax:	+44 (0)1223 403 409
E-mail:	euro-support@arcom.com

Appendix B - Software sources

The source for a component consists of the following:

- An upstream source tarball named *PACKAGE_VERSION*.orig.tar.gz (where *VERSION* is the upstream version number).

- A patch containing Arcom's modifications named *PACKAGE_VERSION-REVISION*.diff.gz (where *REVISION* is Arcom's revision of the component).

A small number of packages do not include a patch as there are no modifications to upstream. This is common when Arcom is the upstream author.

Source and binaries for a given component can normally be found on the Development Kit CD, in the folder /packages/PACKAGE/. The file /packages/index.html contains an index of all of the packages available on the Development Kit.

Source code to any open source components of AEL that are not included on the CD can be supplied by Arcom on request.

Appendix C - Reference information

Sources of further information are listed below:

Information	Where found
General Linux information	www.linux.org
Linux kernel	www.kernel.org
ARM (and XScale) Linux kernel	www.arm.linux.org.uk
GNU GCC	www.gnu.org/software/gcc
Linux documentation project	www.tldp.org
Linux Standard Base project	www.linuxbase.org
The BSD license	www.opensource.org/licenses/bsd-license.html
GNU General Public License (GPL)	www.gnu.org/copyleft/gpl.html
GNU Lesser General Public License (LGPL)	www.gnu.org/copyleft/lgpl.html
The MIT license	www.opensource.org/licenses/mit-license.html
RUTE: The Rute user's tutorial and exposition	http://rute.sourceforge.net
eCos and RedBoot	http://sources.redhat.com/ecos/docs-latest
	http://sources.redhat.com/ecos/docs-latest/ref/redboot.html

Appendix D - Acronyms and abbreviations

AEL	Arcom Embedded Linux
BASH	Bourne Again SHell
BOOTP	BOOTstrap Protocol
BSD	Berkeley Software Design
COM	Communication port
CPU	Central Processing Unit (PXA255)
CMOS	Complementary Metal Oxide Semiconductor
DHCP	Dynamic Host Configuration Protocol
FIS	Flash Image System
FTP	File Transfer Protocol
GCC	GNU Compiler Collection
GDB	GNU DeBugger
GPIO	General Purpose Input/Output
GPL	General Public License
HTTP	Hyper Text Transfer Protocol
ICE	In-Circuit-Emulator
IO	Input/Output
IPSEC	IP SECurity
IPV4	Internet Protocol Version 4
IRQ	Interrupt ReQuest line
JFFS2	Journaling Flash File System 2
LCD	Liquid Crystal Display
LGPL	Lesser General Public License
LSB	Linux Standard Base
OS	Operating System
RAM	Random Access Memory
RTC	Real Time Clock
RUTE	Rute Users Tutorial and Exposition
SBC	Single Board Computer
SDRAM	Synchronous Dynamic Random Access Memory
SRAM	Static Random Access Memory
SSH	Secure SHell
SSID	Service Set IDentifier
TCP/IP	Transmission Control Protocol/Internet Protocol
TFTP	Trivial File Transfer Protocol
TSC1	TouchScreen Controller 1
UART	Universal Asynchronous Receiver / Transmitter
USB	Universal Serial Bus
VGA	Video Graphics Adapter, display resolution 640 x 480 pixels
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network

Index