PEGASUS
Embedded Linux with Java™ Technology

Technical Manual

Product Information

Full information about other Arcom products is available by contacting our Website at:
www.arcomcontrols.com

Useful Contact Information

Customer Support US
Tel: 913 549 1000
Fax: 913 549 1001
E-mail: support@arcomcontrols.com

Customer Support Europe
Tel: +44 (0)1223 412 428
Fax: +44 (0)1223 403 409
E-mail: support@arcom.co.uk

Sales offices

United States:
Arcom Control Systems Inc
7500 West 161st Street
Stilwell, KS 66085, USA
Tel: 913 549 1000
Fax: 913 549 1002

E-mail:
icpsales@arcomcontrols.com

United Kingdom:
Arcom Control Systems Ltd
Clifton Road
Cambridge CB1 7EA, UK
Tel: 01223 411 200
Fax: 01223 410 457

E-mail:
sales@arcom.co.uk

Sales hotlines

Belgium:
Groen Nummer:
Tel: 0800 7 3192
Fax: 0800 7 3191

France:
Numero Vert:
Tel: 0800 90 84 06
Fax: 0800 90 84 12

Germany:
Kostenlose Infoline:
Tel: 08001 824 511
Fax: 08001 824 512

Netherlands:
Gratis Nummer:
Tel: 0800 0221136
Fax: 0800 0221148

Italy:
Numero Verde:
Tel: 0800 790841
Fax: 0800 780841

Whilst Arcom's sales team is always available to 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, which is 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.

Arcom Control Systems Ltd operate a company-wide quality management system which has been certified by the British Standards Institution (BSI) as compliant with ISO9001:1994

Contents

Revision History

Manual	**AEL Revision**	**Date**	Comments
Issue A	V2 Iss3	19 June 2002	First full release of manual

Preface:

Disclaimer

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

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

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

Anti-Static Handling

This development kit contains CMOS devices, which could be damaged in the event of static electricity being discharged through them. At all times, please observe anti-static precautions when handling boards. This includes storing boards in appropriate anti-static packaging and wearing a wrist strap when handling boards.

Packaging

Please ensure that should a board need to be returned to Arcom Control Systems, it is adequately packed, preferably in the original packing material.

Technical Support

Arcom Control Systems has a team of technical support engineers who will be able to provide assistance if you have any problems with this product. Please contact :

US	support@arcomcontrols.com	Tel +1 913 549 1000
Europe	support@arcom.co.uk	Tel +44(0)1223 412 428

All technical support relating to the IBM WSDD 4.0 development environment should be directed to Arcom Control Systems.

Important - Please Read

Licensing

The Arcom Embedded Linux with Java Technology Development Kit contains components licensed from different sources. Some of these are open source licenses. Under certain circumstances you may be required to release source code to any modifications you have made to these modules if you further distribute them. Please consult Appendix A and the `/Reference/Licenses` directory on the CD to ensure you are familiar with the requirements of each license.

WebSphere® Studio Device Developer Tools and WebSphere Micro Environment Runtime License

This Arcom Embedded Linux with Java Technology Development Kit is supplied with a licensed copy of the 'Individual Package' version of the IBM WebSphere Studio Device Developer development tools. It also includes a runtime license for the IBM WebSphere Micro Environment which includes the J9 Java Virtual Machine and Class Libraries (including J2ME configurations and profiles as well as custom configurations, MicroView, IBM Service Management Framework, Bare Metal Graphics, P3ML, Flash Manager and the runtime framework of the IBM Component Distribution System.) Products developed using the WebSphere Studio Device Developer tools are subject to a license fee and an optional support agreement. Please contact Arcom for more information and all related technical support issues.

Please review the license documents included on the Development Kit CD before using this Development Kit:

```
/Licenses/WSDD.txt
```

OpenSSH License

OpenSSH is not covered by any restrictive license. It can be used for any and all purposes, and that explicitly includes commercial use. The license for OpenSSH is included in the distribution.

GCJ Java Runtime License

The Arcom Embedded Linux with Java Technology Development Kit contains the GNU GCJ Java run time libraries.

The libgcj library is licensed under the terms of the GNU General Public License (`/Licenses/GPL.txt` on the CD), with this special exception:

```
As a special exception, if you link this library with other files
to produce an executable, this library does not by itself cause
the resulting executable to be covered by the GNU General Public
License.  This exception does not however invalidate any other
reasons why the executable file might be covered by the GNU
General Public License.
```

Trademarks and Attributions

Linux is a registered trademark of Linus Torvalds.

Java and all Java-based trademarks are trademarks of Sun Microsystems in the United States, other countries or both.

WebSphere is a registered trademark of IBM Corporation in the United States and other countries.

Red Hat™ is a registered trademark of Red Hat, Inc. This product contains copies of the Red Hat Linux v7.2 installation CDROMs which are not a product of Red Hat, Inc. and are not endorsed by Red Hat, Inc. They are a product of Arcom Control Systems and we have no relationship with Red Hat, Inc. The CDs are identical in every respect to a standard Red Hat Linux v7.2 CD set.

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.

Overview

Arcom's Embedded Linux is a standard Linux distribution specially reduced to fit on the on-board flash array of the PEGASUS PC/104 single board computer. Key software includes

- Linux Kernel 2.4.18
- GNU C library 2.2.4
- JFFS2 (compressed Journaling Flash File System) installed on the onboard flash array
- THTTPD webserver version 2.20c
- Net kit version 0.17 FTP and telnet servers and clients
- Bourne Again Shell (BASH) version 2.05
- OpenSSH (Secure telnet replacement) version 3.1p1
- IBM J9 version 1.5 with jclMax runtime
- And many other standard Linux utilities

This development kit is based on Arcom's Embedded Linux with the IBM J9 Java Virtual Machine and class libraries included in the standard distribution. The core of IBM WebSphere Device Developer 4.0 IDE is included on the development kit CD to enable Java application development on a host system.

Note: Arcom Embedded Linux is a standard Linux distribution optimized for embedded systems. It is based upon the standard Linux kernel and user space tools. Arcom provide free first line technical support for this product.

This manual provides information about the specifics of the Arcom Embedded Linux distribution as well as tutorials on the key technologies featured. The Linux RUTE manual (`Manual/rute.pdf` on the CD-ROM) contains a much more general overview of how to use a Linux system. There are also links to several useful websites in "Appendix B - Useful Web Links".

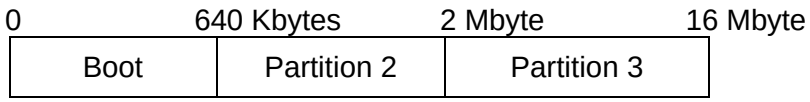
Note: It is not intended that the PEGASUS board be used to build applications. It is recommended that you design and build applications on a alternate host system and download applications to the PEGASUS target system. Arcom Control Systems suggests using a host system with an installation of Red Hat 7.2.

File System Layout

The entire Linux file system is spread across 3 separate partitions for added data security.

Mount Point	Location	File System	Initial Mount Type	Size
/	Flash partition 3	jffs2	Read Only	14 Mbytes
/var	Flash partition 2	jffs2	Read/Write	1.25 Mbytes
/var/tmp	RAM	tmpfs	Read/Write	Dynamic up to a (configurable) 4 Mbyte limit

The onboard flash partitions are positioned as follows in the on-board flash array



The tmpfs filesystem mounted on `/var/tmp` stores files in RAM. The amount of RAM used is not fixed and depends on the total size of all the files stored in the filesystem. To prevent it using all available RAM the maximum size has been set to 4Mbyte.

If you wish to increase the size of the RAM disk edit `/etc/fstab` and increase the size parameter for `/var/tmp`. After you have done this reboot the system.

Boot Times

As supplied the PEGASUS board will boot in around 30s. This can be reduced by disabling unused services.

BIOS Settings

In order to allow the user to add an IDE device the BIOS settings will need to be modified.

To access the BIOS settings, during the POST (Power On Self Test) memory check the user can press the <Del> key when the console is the PC Keyboard and video monitor, or the <Control-C> key when the console is a serial link. This causes the BIOS setup screen to load.

In Serial Console Mode, in order to go Up/Down within this screen the user would have to press <Control-E>/<Control-X>, to go to the next cell to press <Tab> and to select press <Space>. In a VGA Console Mode, the usual commands and Arrows should be used.

More details are given within the PEGASUS Technical Manual that could be found on the Development Kit CDROM in:

`/Manual/pegasus.pdf.`

Configuration files and boot scripts

Arcom Embedded Linux uses a System V type init process. Scripts are placed in `/etc/init.d/` with symbolic links for each runlevel in `/etc/rc?.d/`. The '**?**' may be replaced with one 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)	Serial only console
3	Normal	Serial and VGA console
4	Normal	VGA only console
5	Normal	VGA only console
6	Reboot	Run when rebooting

The default runlevel is level 2 or 3, as configured by the default kernel command line.

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

Arcom specific configuration files (used in the boot scripts) are located in `/etc/config`.

File	Description	Format
`/etc/config/console/keymap`	Default keymap loaded during boot	Symlink to the real keymap. (Usually under `/usr/lib/kbd/keymaps`).
`/etc/config/network/hostname`	Hostname	**`HOSTNAME=`***`hostname`*
`/etc/config/network/interfaces/*`	Network interface configuration. One file per interface.	**`DEVICE=`***`network device name`* **`IPADDR=`***`IP address`* **`NETMASK=`***`netmask`* **`USEDHCP`***`=[yes|no]`*
`/etc/config/network/routes/gateway`	Default gateway IP address	**`GATEWAY=`***`IP address`*
`/etc/config/hardware/serial/*`	Serial port IRQ configuration.	**`DEVICE=`***`path to device node`* **`IRQ=`***`IRQ`*

Making an Application Run Automatically at Boot

If you want an application to run automatically on boot then you need to do the following.

1. Write a script (e.g. **someapp**) that will run your application and put it in the directory `/etc/init.d`.
2. Make the script executable
   `chmod +x /etc/init.d/someapp`
3. Make a symbolic link in `/etc/rc2.d` (e.g. **`/etc/rc2.d/S99someapp`**) that points to the script in `/etc/init.d`. Using 99 ensures that you application will be started after all other services.
   `ln -s /etc/init.d/someapp /etc/rc2.d/S99someapp`

Arcom Embedded Linux with Java Technology Development Kit CDROM Contents

Directory Name	Description
Linux Specific	
Utils	Embedded Linux utilities
install	Installation files
install/boot	Kernel and Boot loader Images
Boot	Bootable CD disk image
source	Arcom Embedded Linux Source
Java Specific	
Java Examples	Java Examples for IBM J9 and GCJ
WSDD Specific	
Wsdd	IBM WebSphere Device Developer
wsddupdates	Updates needed for WSDD
General	
Acrobat Reader	Acrobat PDF Reader
Licenses	Copies of common licenses used by the software included in the development kit
Manual	Arcom and 3rd party documentation
Reference	Board reference documentation

Installation

The PEGASUS development kit CDROM contains a bootable installation image which can be used to return your PEGASUS board to the factory shipped state or to reinstall with a different set of components. It can also be used as a rescue disk to attempt to repair a damaged system.

Installing on a Headless System

Connect a serial terminal (VT100) to the first serial port (`/dev/ttyS0`, COM1), **`minicom`** running on your Linux Host PC connected via a null-modem cable is a common setup. Configure the serial terminal to 115200 baud, no parity and 8 data bits (in **`minicom`** press <Control-A> then Z to access the main menu). Setting up serial terminals is beyond the scope of this document (see the Serial Terminal HOWTO for details).

1. Ensure the CD-ROM is connected.

> **Note:** You should not connect any device as a slave on the IDE bus unless there is also a master device, doing so will cause initialization of the IDE bus to take several seconds longer the usual. Therefore you should connect the CD-ROM as a master unless you also have a hard disk attached.

2. Power on the board.
3. Ensure the BIOS is set up so that it can boot from CD-ROM.
4. Insert the CD and reboot. You may need to hold down [ALT] or <Control-C> to cause the system to boot from the BIOS device rather than flash.
5. The CD will start and display some text and a prompt

```
Welcome to the Arcom Embedded Linux n.nn Installation CD
…
boot:
```

6. Enter ‘sermaster’ then [ENTER] if the CD-ROM drive is configured as master or ‘serslave’ then [ENTER] if the CD-ROM is configured as slave. If unsure check the jumper setting on the drive or try both.
7. After Linux has finished booting, the installation program will start. Use [TAB] and the cursor keys to select fields and [SPACE] to toggle check boxes and press buttons.
8. You will be given the option of a ‘Quick Install’ or a ‘Custom Install’. Choose ‘Quick Install’ to reprogram the flash as it left the factory and skip to step 19. Other wise choose ‘Custom Install’ and continue.
9. Select any required optional components.
10. Press the ’Next’ button to continue.
11. Configure the network. A hostname is required, although you can accept the default. If you have a DHCP server on your network then you may choose to use DHCP and have an IP address assigned at boot time, otherwise you must fill in an IP address and network mask. The default gateway and name server are optional
12. Press the ‘Next’ button to continue.
13. Ensure the ‘Boot from flash’ and ‘Use serial console’ checkboxes are enabled if you wish to boot from the onboard flash using a serial console. If you do not have a VGA card attached you may want to ensure that you only run login on ttyS0, this will prevent errors.
14. Press the ‘Next’ button to continue.

15. Edit the kernel parameters if required, although this will not normally be needed.
16. Press the ‘Next’ button to continue.
17. Normally you will want to erase both partitions, however you can choose not to if you have vital data.
18. Press the ‘Next’ button to continue.
19. Once the system has finished installing, it will reboot. Remove the installation CD.
20. The system will boot and present a login prompt. Login as ‘root’ with the password ‘arcom’ or as ‘arcom’ also with the password ‘arcom’.

Installing on a Headed System (With additional PC/104 VGA board)

1. Ensure the display, keyboard and CD-ROM drive are correctly plugged in.

> **Note:** You should not connect any device as a slave on the IDE bus unless there is also a master device, doing so will cause initialization of the IDE bus to take several seconds longer the usual. Therefore you should connect the CD-ROM as a master unless you also have a hard disk attached.

2. Power on the board.
3. Ensure the BIOS is set up so that it can boot from CD-ROM.
4. Insert the CD and reboot. Depending on whether a bootable hard disk drive is attached, you may need to hold down the [ALT] key to bypass the Linux flash boot loader.
5. The CD will start and display some text and a prompt

```
Welcome to the Arcom Embedded Linux n.nn Installation CD
…
boot:
```

6. Enter ‘master’ then [ENTER] if the CD-ROM drive is configured as master or ‘slave’ then [ENTER] if the CD-ROM is configured as slave. If unsure check the jumper setting on the drive or try both.

> **Note:** After 30 seconds without a key press the CD will continue booting using a serial terminal as the console. Reboot if this occurs.

7. Continue with step 7 given in the instructions for installing on a headless system (above).

Using the Development Kit CD as a Rescue Disk

Should the flash become unbootable the CD can be used to reinstall the flash boot loader and fix problems with the installation. The installation program runs on the first virtual terminal (VT), while a shell is available on each of terminals 2 through 5. To change to VT2 press [ALT-F2], and similar for the other terminals. To return to the installation press [ALT-F1]. Pressing ‘Back’ from the first screen of the installation program will also present a shell prompt.

For headless systems installation runs on the first serial line and a shell is available on the second serial line. Again, pressing ‘Back’ from the first prompt will drop to a shell.

The boot loader image, called `stage2.rom`, is in the directory `/cd/install/boot/`.

The kernel image is in the package `/cd/install/boot/kernel-image_2.4.18-1_pegasus.ael` in the file `/boot/vmlinuz`. You can extract it with (on a single line)

```
/cd/utils/extract-kernel /cd/install/boot/kernel-image_2.4.18-
1_pegasus.ael /var/tmp/bzImage
```

Install the boot loader with the flashboot command (on a single line)

```
flashboot  –2 stage2.rom /var/tmp/bzImage 2
console=ttyS0,115200n8 root=1f02 ro
```

Mount the flash partitions

```
mount –t jffs2 /dev/mtdblock1 /mnt/1
mount –t jffs2 /dev/mtdblock2 /mnt/2
```

Setup the network

```
ifconfig eth0 IP address netmask netmask
route add default gw IP address of default gateway
```

or

```
/etc/init.d/dhcpd start
```

The CD contains most of the utilities that are installed (ftp client, nano text editor etc).

File System

The flash can be accessed via the compressed Journaling Flash File System (JFFS2). This places a file system onto the Flash transparently to the user. Arcom Embedded Linux is supplied preinstalled on the JFFS2 file system.

Flash partitions with JFFS2 are mounted using a special pseudo-block device (major: 31)

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

The block devices are

Device	Minor number	Description
`/dev/mtdblock0`	0	1st partition (boot sector)
`/dev/mtdblock1`	1	2nd partition (`/var`)
`/dev/mtdblock2`	2	3rd partition (`/`)

The first partition (`/dev/mtdblock0`) does not have a JFFS2 partition when supplied, the user may put one on if the board is not required to boot from flash.

No special utility is required to make a JFFS2 file system. Simply erase the whole of the partition (with eraseall) and mount as normal. This will cause an empty JFFS2 file system to be created.

JFFS2 partitions do not need to be fsck'ed (this is done when mounting). The supplied `/sbin/fsck.jffs2` is a dummy which always succeeds and is present to simplify the boot scripts.

Although JFFS2 is a journaling file system this does not preclude the loss of data. While the filesystem will remain in a consistent state and always be mountable, data can be lost if the board is powered down during a write or due to the write caching strategy used by Linux for file system operations resulting in the data not being written before a power cycle.

Refer to http://sources.redhat.com/jffs2 for further information.

Installing or Uninstalling Components

As shipped the flash file system on the PEGASUS contains several optional components, including the J9 Java Virtual Machine, GCJ runtimes and the OpenBSD Secure Shell (SSH). If your application does not require one or more of these and you wish to free some space in the flash then you may remove certain components or simply reinstall (as in the section 'Installation' on page 11) and choose not to install those components. Conversely if you require a component which is not installed by default or which you did not install you may wish to add components to the board.

Removing Components from an Already Installed System

Each of the optional components is supplied with a script which is able to remove them. These scripts can be found on the CD as **`/install/comp/<`*`name`*`>/<`*`name`*`>.remove`**. (where **`<name>`** is the component name). To remove a component simply copy the script to the board (via NFS or FTP) or mount the CD and run the script. It is usual for one or two files or directories to be un-removable, often this is because the directory is not empty. In such cases you will be given a warning and can check if you want to remove the file or directory by hand.

Adding Components to an Already Installed System

Each optional component is stored in one or more packages under the directory `/install/comp/<`*`compname`*`>/`. Each package has an extension of **`.ael`** and includes a version number in the file name. To install a component simply install each package, installing only a subset of packages is possible but not advised. A utility, **`arcomunpack`**, is supplied in the `/utils/` directory on the CDROM and will install a package passed to it on the command line. An optional second parameter allows you to specify an alternative root directory.

Example

To install the ppp component (which has only a single package):

```
arcomunpack /cd/install/comp/ppp/ppp_2.4.0-1_pegasus.ael
```

To install a package to an alternative prefix:

```
arcomunpack xfree86_4.1.0-3_pegasus.ael /flash/
```

Utilities

Eraseall

The eraseall utility is found in the CD's `/utils` directory.

eraseall erases all of a given MTD device.

Usage

```
eraseall [MTD char device]
```

Note: Take care when using this command as specifying the wrong device will cause all data on that file system to be lost.

Example:
To erase all of the boot partition.

```
eraseall /dev/mtd0
```

Flashboot

The flashboot utility installs or uninstalls the flash boot loader and kernel image, it is found in the CD's `/utils` directory.

The boot loader consists of 3 main components

- stage 2 loader
- kernel command line
- kernel image.

All components must be installed correctly for the boot loader to work.

A custom, precompiled kernel image is supplied on the Development Kit CD in the package `/install/boot/kernel-image_2.4.18-3_pegasus.ael`. Rather than installing the package (which is unnecessary), you can extract only the kernel image using the **`extract-kernel`** utility found on the CD in the `/utils` directory.

This command will produce a `bzImage` which is equivalent to one you would build yourself.

```
extract-kernel kernel-image_2.4.18-3_pegasus.ael /var/tmp/bzImage
```

The kernel sources are available in the `/source` directory on the development kit CD. The configuration for the supplied kernel is contained in the file `pegasus-FLASH.conf` in the source archive.

Kernel Command Line

There are several command line options which can be passed to the kernel. More details can be found in '`Documentation/kernel-parameters.txt`'

In order for the kernel to use the serial port as its console you must pass the 'console=ttyS0,115200n8' parameter to use the first serial port at 115200 baud, no parity and 8 bits. For other options please consult the kernel parameters documentation.

If the standard partition/filesystem layout is used the kernel parameters must contain '`root=1f02`' or the root filesystem on the flash will not be mounted.

In the standard configuration the board will not be fitted with a VGA interface attached and so the board should boot in runlevel 2, this can be accomplished by prepending a '2' to the command line. Without this parameter warnings will be generated when the system tries to open a login session on a non-existent VGA console.

Stage 2 Boot loader

The stage 2 boot loader is responsible for loading the kernel from the flash drive and booting it, and can be found on the development kit CD as

- `/install/boot/stage2.rom`

Usage:

`flashboot` ***`[Options] [Kernel image] [Kernel options]`***

Options:

-b,--board=BOARD
: Board. You should always specify 'pegasus' here

-2, --stage2=LOADER
: Location of the stage 2 boot loader image. [default: `/boot/stage2.rom`]

-d, --device=DEVICE
: MTD char device to install boot loader to and image. This should always be the first MTD device or the boot loader will not work. [default: `/dev/mtd0`]

-k, --kernel-only
: Only install a kernel image and command line. The boot loader must already be installed or it will not boot.

-n, --no-erase
: Don't erase flash before installing - flash must already be erased or the images will not be written correctly.

-u, --uninstall
: Disable boot loader. This does not erase the flash and reinstalling will require erasing.

--help
: Display concise help and exit.

--version
: Output version information and exit.

Optional Splash Screen

If the board is fitted with an optional PC-104 VGA board then it is possible to have a splash screen displayed rather than the normal Linux boot messages, however this consumes space in the flash boot partition and hence reduces the maximum size of kernel which can be used.

The splash screen is either a 640 × 480, 800 × 600 or 1024 × 768, 256 color Windows format bitmap. It must be less than 96 kilobytes in size. RLE8 encoded bitmaps only are supported.

To install a splash screen use the `--splash=<BMP>` (or `-s <BMP>`) option to flashboot.

Examples:
Extract a kernel image from a package into the file bzImage

```
extract-kernel kernel-image_2.4.18-1_pegasus.ael bzImage
```

Reinstalling all of the boot loader (on a single line)

```
flashboot –2 stage2.rom bzImage 2 root=1f02 ro
console=ttyS0,115200n8
```

Reinstalling all of the boot loader with additional splash screen (on a single line)

```
flashboot –2 stage2.rom –s splash.bmp bzImage 2 root=1f02 ro
```

Changing the command line (this requires reinstalling the kernel image)

```
flashboot –k bzImage 2 root=1f02 ro
```

Bypassing the Flash Boot Loader

The following table enumerates when the board will boot from Flash and when it will boot from a BIOS configured device. A hard disk is considered to be configured if the BIOS settings for 'IDE DRIVE GEOMETRY' (under 'Basic CMOS Configuration') for IDE0 is set to anything other than 'Not Installed'. To disable a hard disk set this BIOS entry to 'Not Installed'. Note that an attached hard disk will still be visible to Linux even if the BIOS is configured to 'None', but that for the purposes of booting it does not exist.

HDD Configured	CDROM Attached	Normal Boot	[ALT] held down
×	×	Flash	None/Error
✓	×	BIOS Device	Flash
×	✓	Flash	BIOS Device
✓	✓	BIOS Device	Flash

Note: To boot from a HDD or CDROM it must be bootable, attached to the system and configured in the BIOS. The BIOS 'BOOT ORDER' (under 'Basic CMOS Configuration') must be configured to include the device which should be configured in the 'DRIVE ASSIGNMENT ORDER' section.
Note: Attempting to boot from a BIOS device when none of the devices in the BIOS boot sequence are present will result in an error.
Note: When using a serial console it is not possible to detect and [ALT] key press. You can use Control-C instead.

Using Secure Shell (SSH)

Introduction to SSH

SSH (The *S*ecure *SH*ell) is a secure replacement for several common internet protocols such as the Berkley r* tools (rlogin, rsh, rexec), ftp and telnet 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). It features several enhancements over these tools, such as:

- All traffic sent across the network (critically this includes passwords) is encrypted using strong encryption;
- Prevent spoofing and man in the middle attacks using host keys;
- Tunneling of arbitrary connections through an SSH pipe, known as *port forwarding* (in particular X11 forwarding);
- Enhanced authentication methods which improve upon normal password 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 (for example, 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 on SSH and it's advantages and usage please visit http://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.

Red Hat Linux 7.2 includes the OpenSSH server (openssh-server) and client (openssh-clients) packages, as well as the general OpenSSH package (openssh) which must be installed for either of them to work. Please see the *Official Red Hat Linux Customization Guide* for instructions on installing and deploying OpenSSH on your Red Hat Linux system.

There are also several SSH clients available for non-Linux system, including Microsoft Windows platforms, such as

- PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty), a Windows version of the ssh program
- WinSCP (http://winscp.vse.cz/eng) a graphical version of SCP for windows.

Examples of SSH Commands

The ssh Command

The **ssh** command is a secure replacement for the **rlogin**, **rexec**, **rsh** and **telnet** commands. It allows you to log in to and execute commands on a remote machine.
Logging in to a remote machine with **ssh** is similar to using **telnet**. To log in to a remote machine named penguin.example.net, type the following command at a shell prompt:

```
ssh penguin.example.net
```

The first time you ssh to a remote machine, you will see a message similar to the following:

```
The authenticity of host 'penguin.example.net' cannot be
established. DSA fingerprint is
94:68:3a:bc:f3:9a:9b:01:5d:b3:07:38:e2:11:0c.
Are you sure you want to continue connecting (yes/no)?
```

If you wish you can confirm this fingerprint matches the key used by the server and then enter 'yes' to continue. This will add the server to your list of known hosts and display the following message:

```
Warning: Permanently added 'penguin.example.net' (DSA) to the
list of known hosts.
```

Next, you'll see a prompt asking for your password for the remote machine. After entering your password, you will be at a shell prompt for the remote machine. If you use ssh without any command line options, the username that you are logged in as on the local machine is passed to the remote machine. If you want to specify a different username, use one of the following commands:

```
ssh –l username penguin.example.net
ssh username@penguin.example.net
```

The **ssh** command can be also used to execute a command on the remote machine without logging in to a shell prompt. The syntax is **ssh hostname command**. For example, if you want to execute the command **ls /usr/share/doc/** on the remote machine penguin.example.net, type the following command at a shell prompt:

```
ssh penguin.example.net ls /usr/share/doc/
```

After you enter the correct password, the contents of **/usr/share/doc/** will be displayed, and you will return to your local shell prompt.

Using the scp Command

The **scp** command can be used to transfer files between machines over a secure, encrypted connection. It is similar to **rcp**.

The general syntax to transfer a local file to a remote system is

```
scp localfile hostname:remotefile
```

As with the **ssh** command the hostname can have **username**@ prepended to use a different username on the remote machine. The **remote file** is taken relative to your home directory on the remote machine, or an absolute path can be specified.

To transfer the local file **shadowman** to your account on penguin.example.net, type the following at a shell prompt:

```
scp shadowman username@penguin.example.net:shadowman
```

This will transfer the local file **shadowman** to **~username/shadowman** on penguin.example.net.

Multiple files can be specified as the source files. For example, to transfer the contents of the directory **downloads** to an existing directory called **uploads** on the remote machine penguin.example.net, type the following at a shell prompt:

```
scp downloads/* penguin.example.net:uploads
```

All of the above commands can be reversed in order to transfer files from the remote host to the local host. You should be careful to escape any shell globbing characters (such as *). To transfer several images from a remote machine you could use the following command:

```
scp username@penguin.example.net:\*.jpg .
```

Using the sftp Command

The **sftp** utility can be used to open a secure, interactive FTP session. It is similar to **ftp** except that it uses a secure, encrypted connection. The general syntax is:

```
sftp username@hostname.com
```

Once authenticated, you can use a set of commands similar to using FTP. Refer to the **sftp** manual page for a list of these commands (execute the command '**man sftp**' at a shell prompt). The **sftp** utility is only available in OpenSSH version 2.5.0p1 and higher.

Removing insecure services

The default install of Arcom Embedded Linux includes **telnet** and **ftp** daemons which are made obsolete by the use of SSH. They remain in the distribution in order to maintain backwards compatibility. However, we recommend you disable them unless you explicitly require them.

To disable **ftpd** and **telnetd** you should edit the file **/etc/inetd.conf** and look for the following (or similar) lines

```
ftp         stream tcp nowait     root  /usr/sbin/tcpd in.ftpd –l
telnet      stream tcp nowait     root  /usr/sbin/tcpd in.telnetd
```

You should disable one or both of these lines into comments by placing a # character at the start (turning to line into a comment), and then restart **inetd** using the following command

```
/etc/init.d/inetd restart
```

It is also possible to restrict access to these services to certain IP addresses or hosts without disabling them completely using **tcpd**. Consult the **tcpd(8)** and **host_access(5)** man pages for details.

Compiling a Kernel

An Arcom Embedded Linux kernel source tree is supplied on the Arcom Development Kit CD in the `/source` directory. It consists of an upstream tarball (`kernel_2.4.18.orig.tar.gz`) and a patch (`kernel_2.4.18-3.diff.gz`) containing Arcom Control Systems' modifications.

The kernel can be recompiled on your host system or on a Red Hat hard disk installation attached to the board. The kernel cannot be recompiled from a system booted from the flash disk, since a compiler and the associated tools are not loaded on the flash file system, since they would consume valuable flash disk space.

Mount the CD

```
mount /mnt/cdrom
```

Copy the kernel source from the CD (to for example `/usr/local/src/`)

```
cp /mnt/cdrom/source/kernel_2.4.18.orig.tar.gz /usr/local/src
cp /mnt/cdrom/source/kernel_2.4.18-3.diff.gz   /usr/local/src
```

Unpack the source code

```
cd /usr/local/src
tar xzf kernel_2.4.18.orig.tar.gz
```

Apply the patch

```
cd kernel-2.4.18
zcat ../kernel_2.4.18-3.diff.gz | patch –p1
```

Configuring the kernel

```
make menuconfig
```

Select 'Load an Alternate Configuration File' from the Main menu. Several starting configurations are provided

`pegasus-FLASH.conf`	Suitable for booting from flash. (Used by the default flash install).
`cdrom.conf`	Used on the bootable development kit CDROM.

Configure the kernel as required, save and exit.

Build the source dependency information

```
make dep
```

Build the kernel image

```
make clean
make bzImage
```

If your kernel configuration includes modules then build the modules. (All of the supplied configuration are modular).

```
make modules
```

If necessary install the modules. Modules can be installed to an alternate prefix by adding **`INSTALL_MOD_PATH=<prefix>`** to the modules_install line. This should be used to install the module tree to a temporary directory before copying to the target, without this option they will be installed to `/lib/modules/2.4.18`, which could potentially overwrite your host system's modules, which could potentially damage your host system.

```
make modules_install INSTALL_MOD_PATH=/tmp/pegasus-modules
```

Your new kernel image is `arch/i386/boot/bzImage`. It should be installed using flashboot**.**

Installing Applications on the Flash File System

Application can be compiled on a RedHat 7.2 system and copied to the board using ftp or scp.

Using the application '**`dir`**' and a Pegasus board with address '**`penguin.example.net`**' as an example.

Scp the application from the host system to the board which should have its root filesystem mounted read/write (use the command '**`rw`**').

```
scp /usr/bin/dir root@penguin.example.net:/usr/bin/dir
```

On the board (as root) make the application executable

```
chmod +x /usr/bin/dir
```

Attempt to execute the application

```
/usr/bin/dir
```

The application will report that `libtermcap.so.2` is missing. You can confirm this by tracing the objects which are loaded

```
LD_TRACE_LOADED_OBJECTS=1 /usr/bin/dir
```

Reports

```
libtermcap.so.2 => not found
libc.so.6 => /lib/libc.so.6 (0x40017000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

The **`strip`** command removes and unnecessary debug and comment sections from executables and libraries. This is useful because it can greatly reduce the size of these files in situations where this information is not needed. Strip and copy any missing libraries across. The –o is important to avoid stripping the binaries on your host system, which may hinder debugging.

```
strip /lib/libtermcap.so.2 –o /tmp/libtermcap.so.2
scp /tmp/libtermcap.so.2
root@penguin.example.net:/lib/libtermcap.so.2
```

To make the system aware of new libraries you should run the **`ldconfig`** command

```
/sbin/ldconfig
```

Run the application to test

```
dir
```

Java Overview

Java is a programming language designed with the goal of reducing complexity for the programmer. It wraps complex tasks, such as multithreading and network programming in language features and class libraries that can help to make these tasks trivial. It also tackles cross platform programming, dynamic code loading and security.

Java is more portable than some other languages due to its object code. Compilers for most languages generate native machine code for the target processor, which then runs at the native speed of the system. The Java compiler generates object code (known as byte code) for a theoretical machine known as the 'Java Virtual Machine'. The Java Runtime Environment includes an emulator for this virtual machine as well as a set of class libraries. This means that byte code will run on any computer for which the Java Runtime Environment is available, no matter where it was compiled. The trade off is in performance: the interpreter adds a level of overhead to the program.

The IBM J9 Java environment includes a SmartLinker, which is run on the host system and links only the required classes from the class library with your application to create a **JXE** executable. A **JXE** is a self contained Java program and hence the IBM J9 environment requires only a Java Virtual Machine and not a potentially large Java Runtime Environment to be installed on the target.

Arcom Embedded Linux and Java

The Arcom Embedded Linux with Java Technology Development Kit is supplied with the IBM J9 Java Virtual Machine and runtime libraries pre-installed on the PEGASUS, these take up about 2 megabytes of flash. Java programs can be developed on a host system using the IBM WebSphere Device Developer IDE supplied on the development kit CD.

Alternative Class Libraries

WebSphere supports several levels of class library, trading off library size against features. By default the PEGASUS board comes installed with the jclMax configuration which provides maximum features but at a space cost. Therefore you should in general select 'Max Class Library (jclMax)' when creating a new project (see below).

If you wish to use an alternative class library, to save space for example, then you may choose to base your project on another class library. If you do this then you may choose to remove the jclMax support libraries from `/opt/wsdd4.0` on the target. You will also need to copy new support libraries from your host system. These support libraries can be found in `/opt/wsdd/wsdd4.0/ive/bin/` on a host system which has the WSDD environment installed (see below). In general it is easiest to select which libraries you need simply by running your application, observing the error messages and installing the requested libraries.

Installing IBM WebSphere Device Developer IDE

Included on the Development Kit CD is the IBM WebSphere Device Developer (WSDD) IDE that should be used to build, smart link and test Java applications on your Red Hat host system before downloading to the PEGASUS board. Before installing the IDE

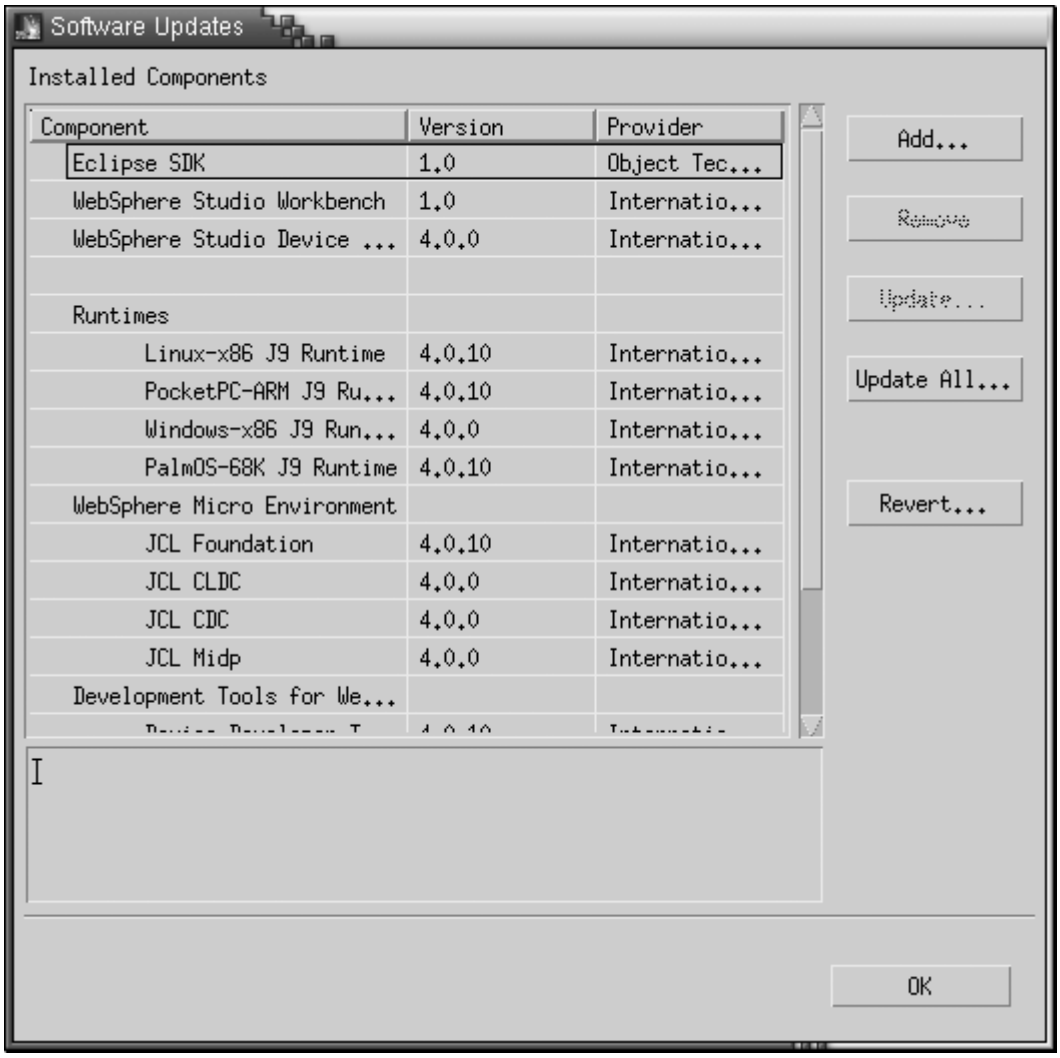
please read the IBM license for J9/WSDD, this is on the CD in the file `Licenses/WSDD.txt`.

To install the IDE on your host system follow these instructions (you will need to be the root user):

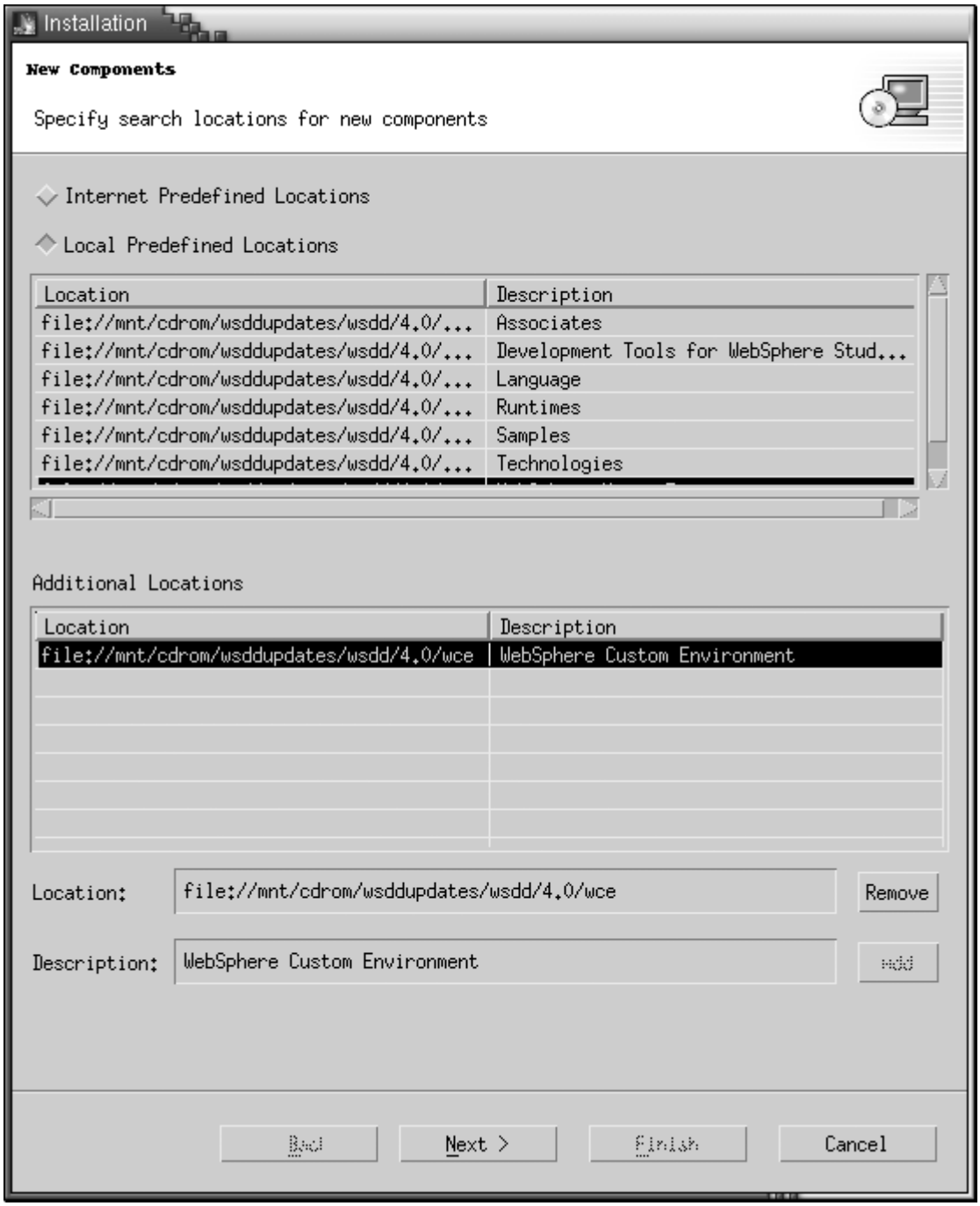
1. Mount the CD
   `mount /mnt/cdrom`
2. Change to the directory on the CD containing the packages
   `cd /mnt/cdrom/wsdd`
3. Install the software (you will need to have root user privileges).
   `rpm –ivh main-ide-4.0-2171.i386.rpm`

In order to make use of WSDD for developing embedded applications for J9 the IDE needs to be updated with the WebSphere Custom Environment (WCE) package, which provides the jclMax build time support. This is also supplied on the Development Kit CDROM. To install the WCE updates follow these instructions (as root):

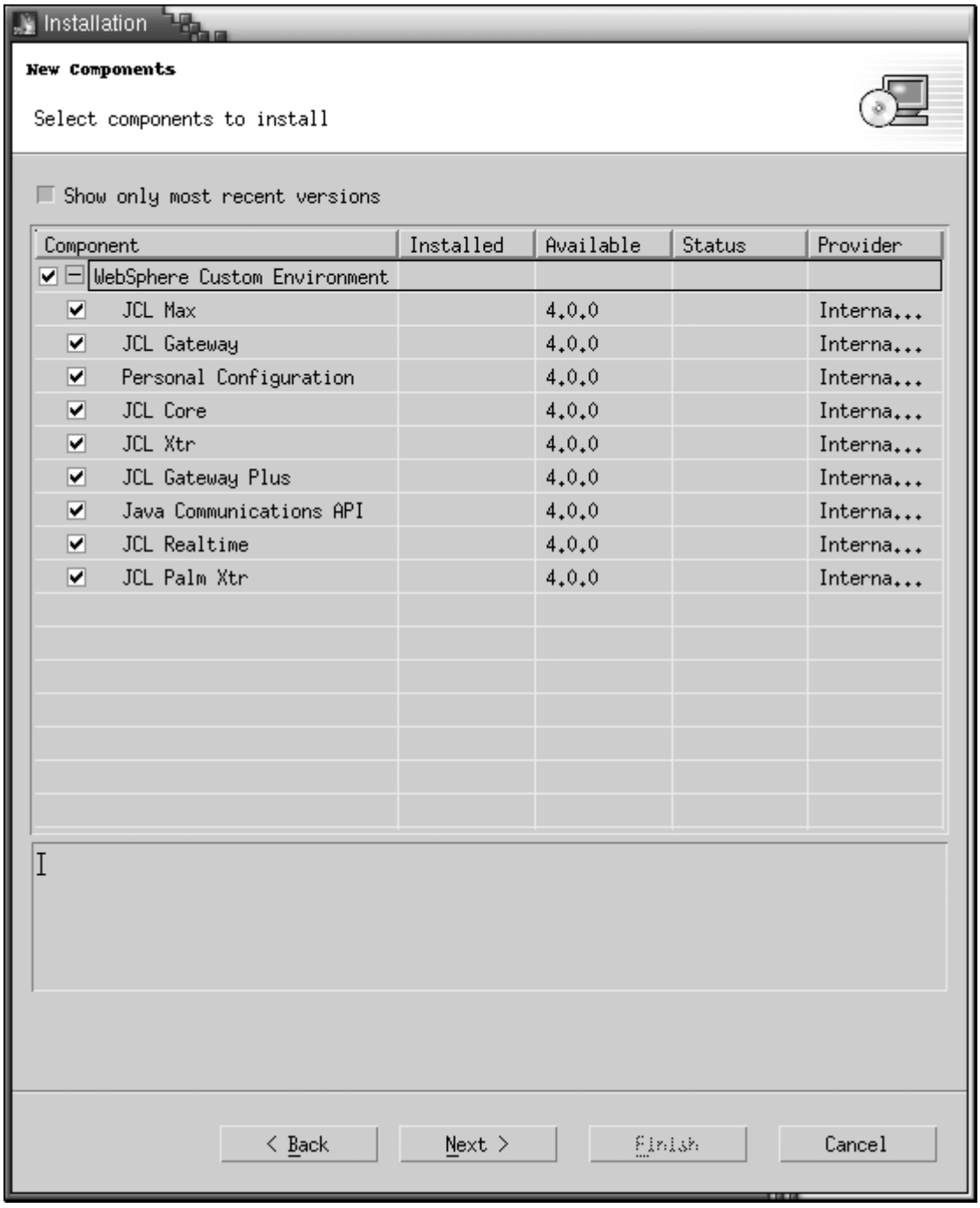
1. Mount the CD
   `mount /mnt/cdrom`
2. Launch the WSDD IDE
   `/opt/wsdd/eclipse/wsdd &`
3. Select `Help -> Device Developer Updates` from the menu, this will present the `Software Updates` dialog:



4. Click on the Add… button. This will present the new components dialog.
5. Towards the bottom enter the location as `file://mnt/cdrom/wsddupdates/wsdd/4.0/wce` and the description as '`WebSphere Custom Environment`'. Press the `Add` Button and select the newly created entry in '`Additional Locations`', then press `Next`.



6. You will be presented with a dialog allowing you to select which WCE components you wish to install. This tutorial requires the `JCL Max` configuration, however it is suggested that all components are installed

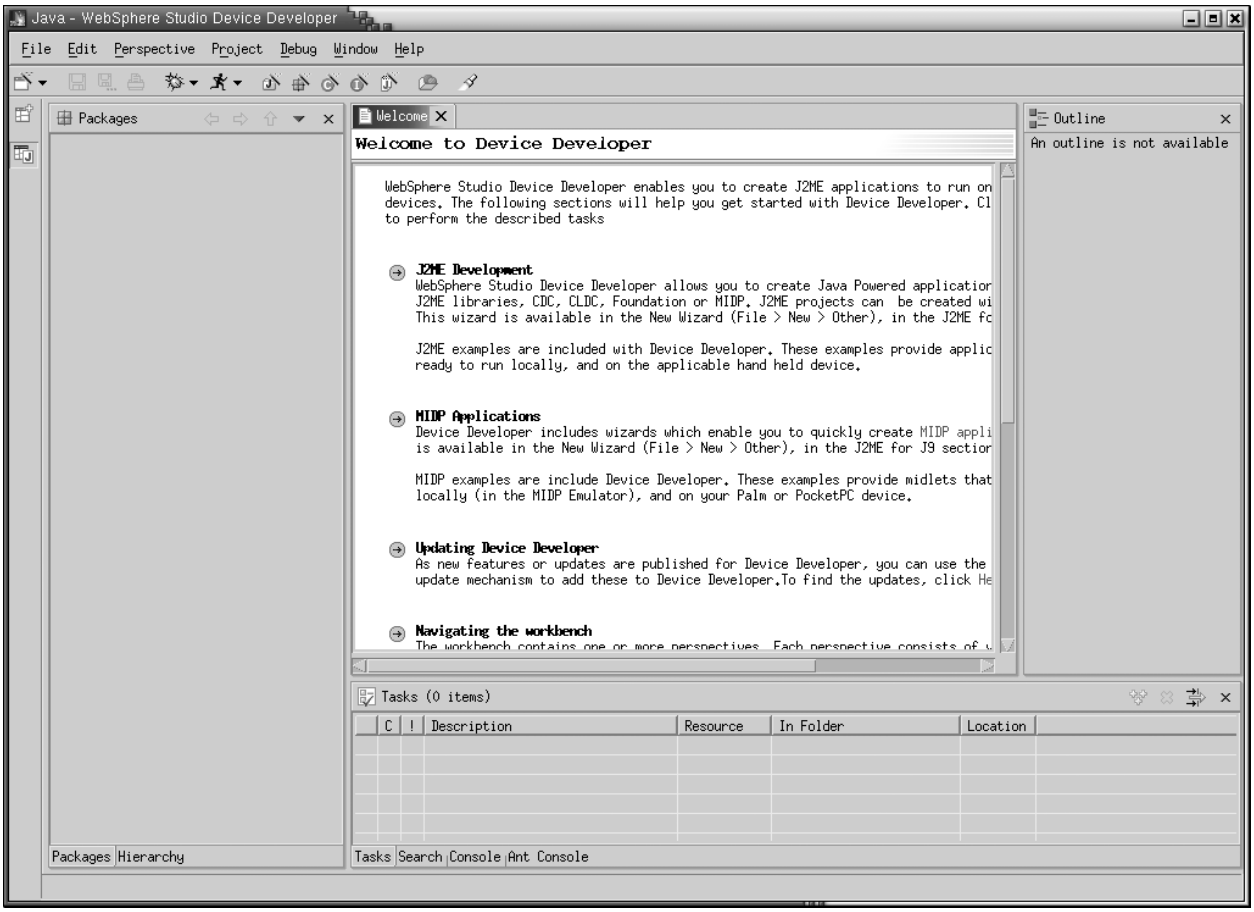


7. Click `Finish` to install the components and then Exit WSDD.

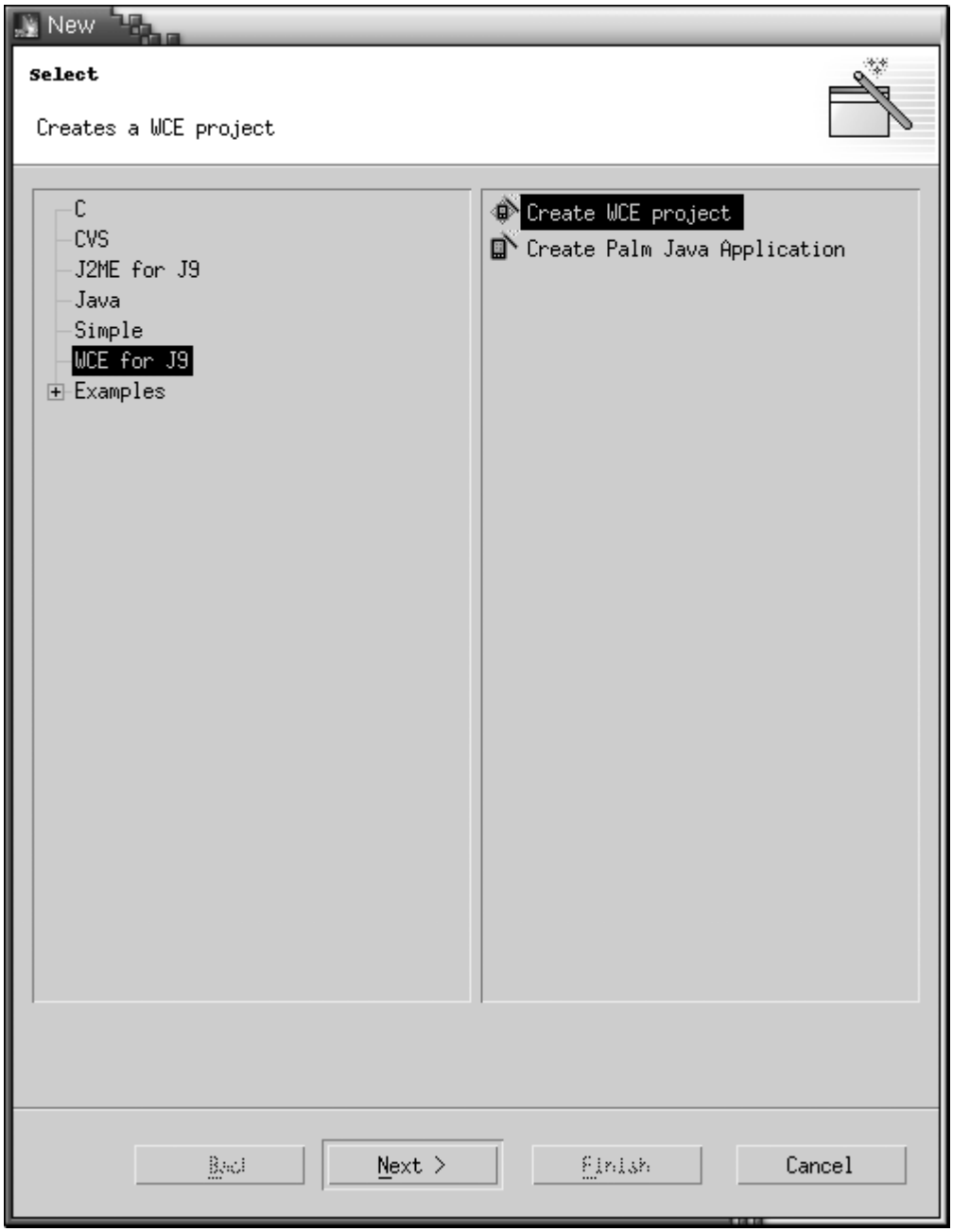
Building and Running Applications Using IBM WSDD

Full instructions for building and running applications can be found in the WSDD documentation installed in `/opt/wsdd/wsdd4.0/doc/` by the RPM. The following is an example of how to create a simple application.

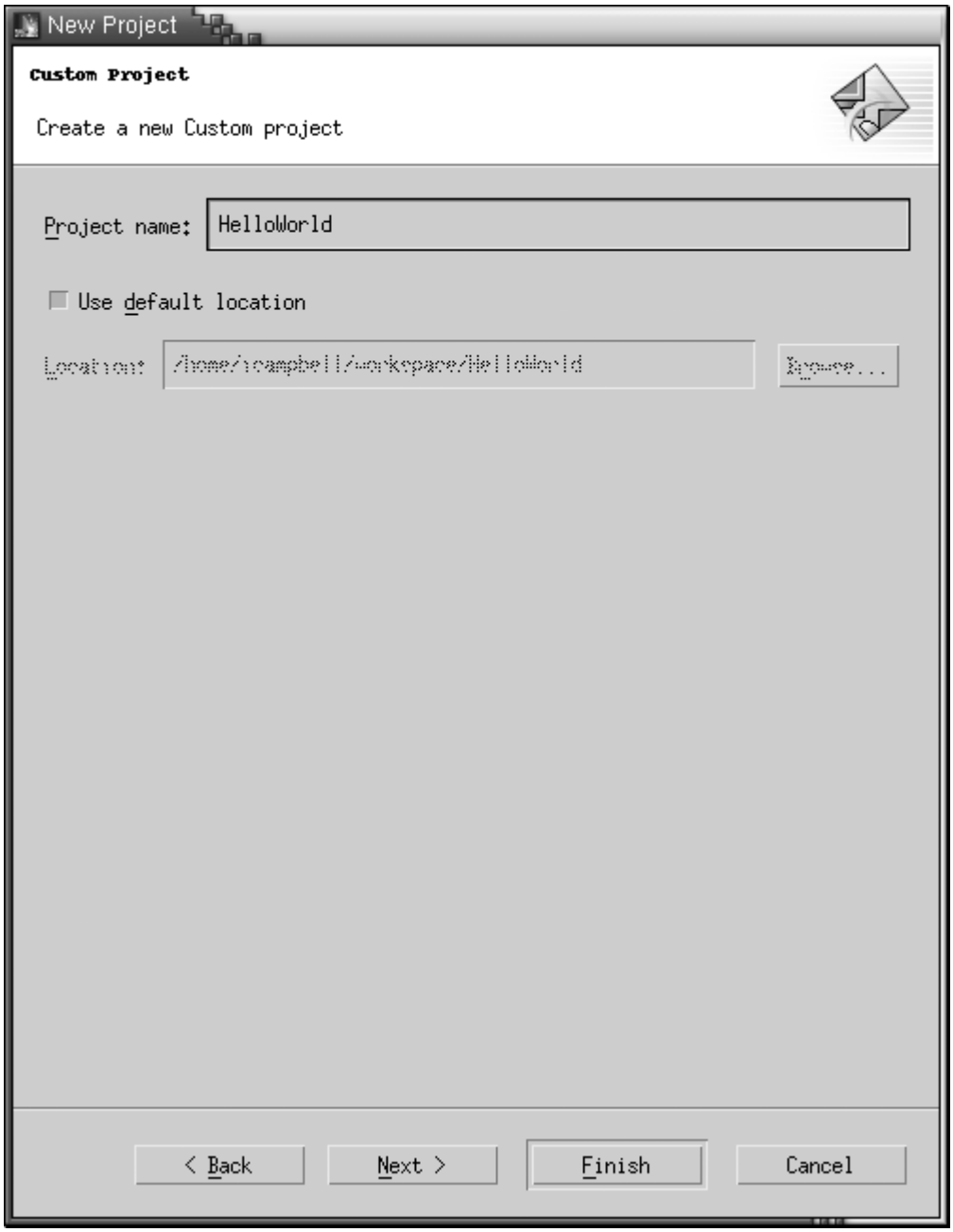
1. Run the WSDD IDE
   `/opt/wsdd/eclipse/wsdd &`
   The main WSDD workspace window should now be displayed.



2. Select '`File -> New -> Other`…' from the menu.
3. In the resulting dialog select '`WCE for J9`' and '`Create WCE Project`' and click `Next`.

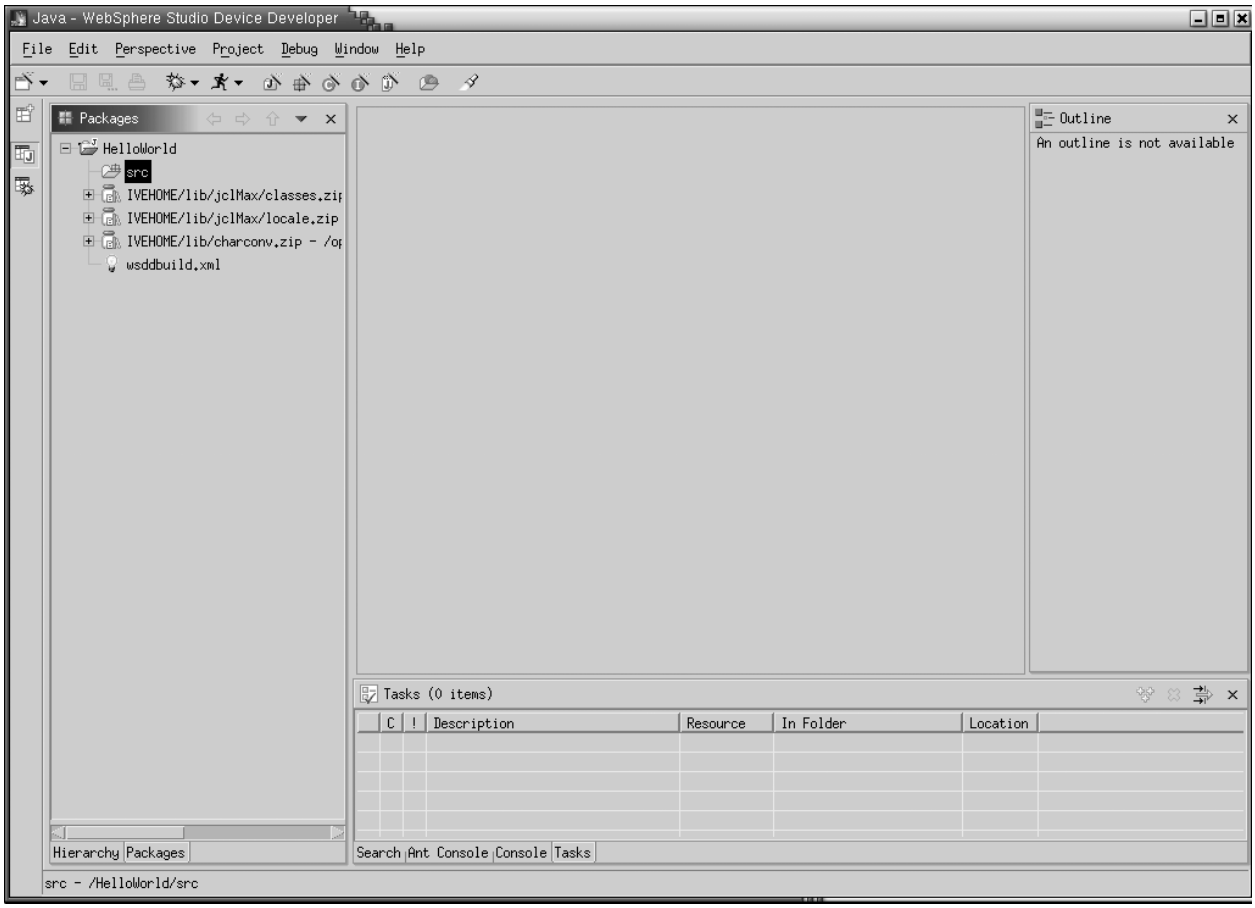


4. Enter the project name as '`HelloWorld`' and click `Next`.



5. Select '`Max Class Library (JCL Max)`' as the class library and press `Next`.
6. Accept the defaults in all other windows by pressing `Finish`.

The WebSphere IDE should now be displayed with a perspective (or view) open on the new project.

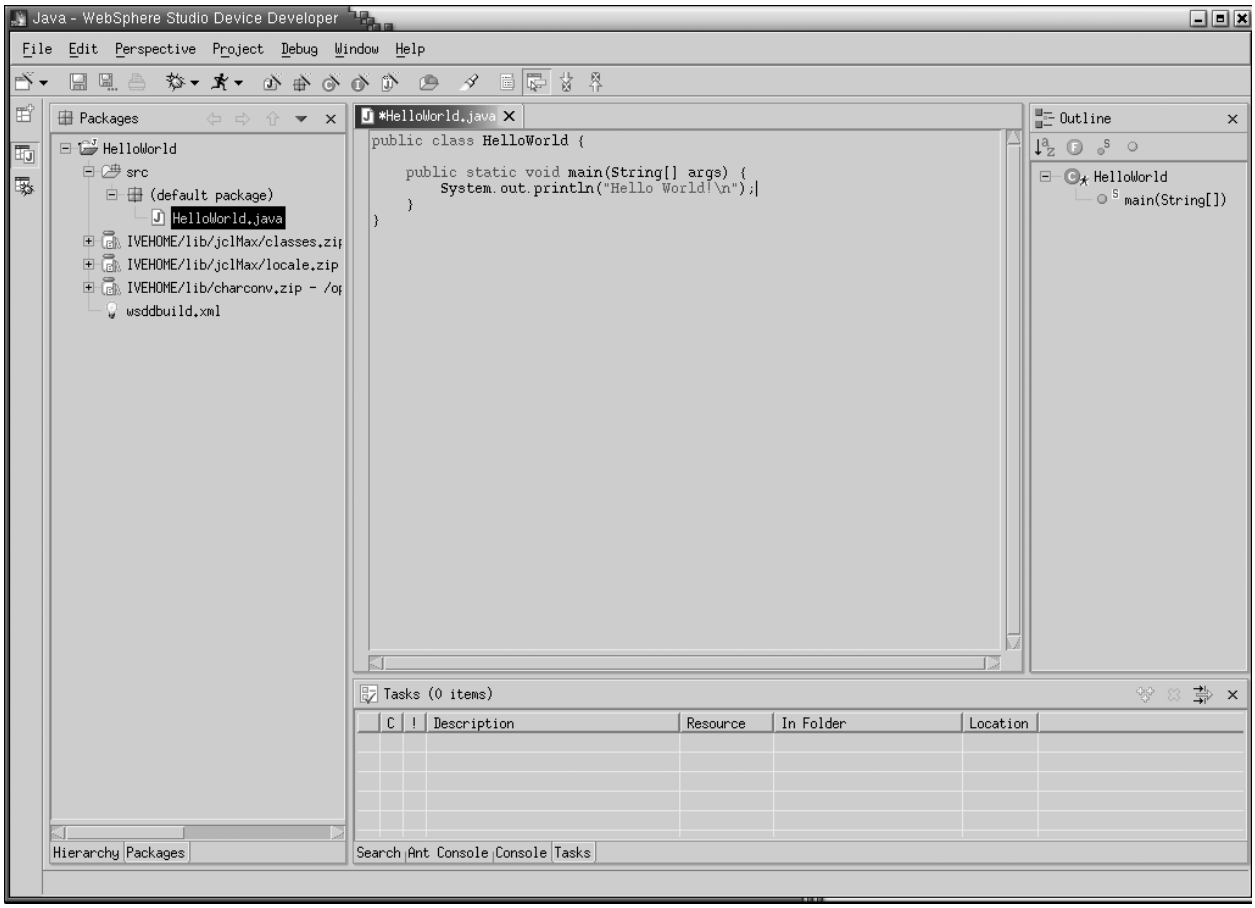


Adding a Class to Your Application

Next we will add a simple class to the application.

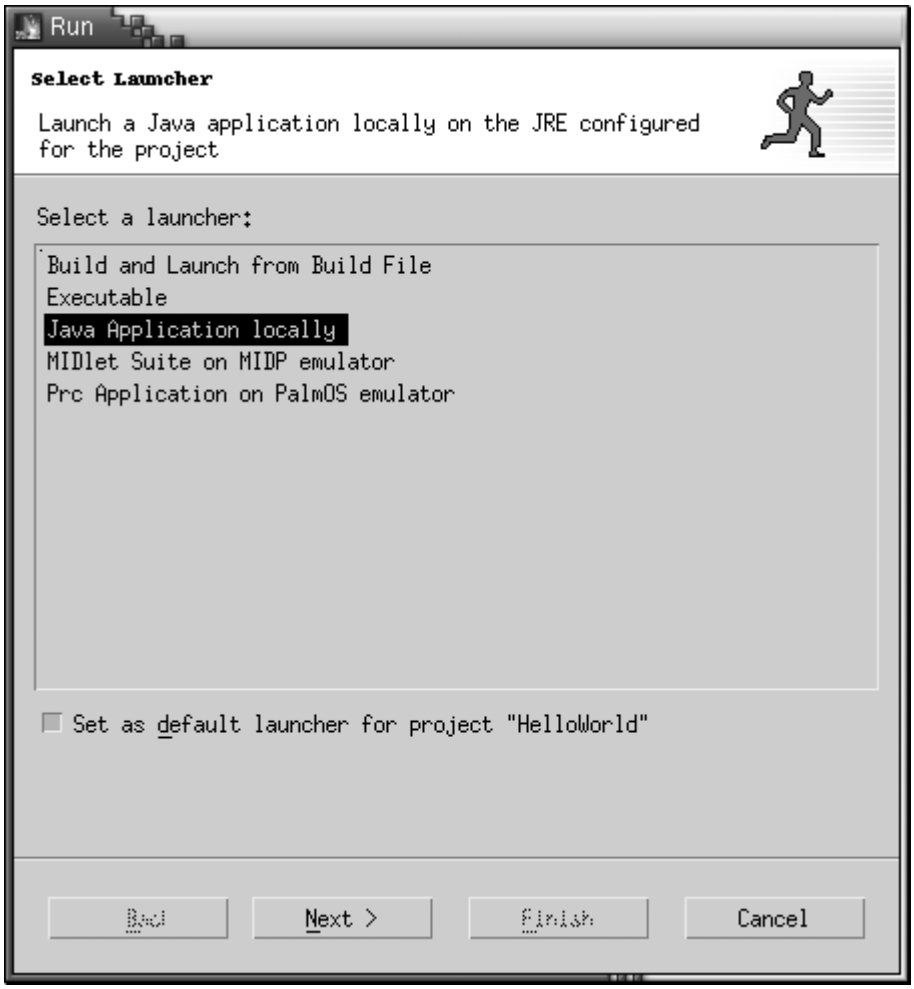
1. Select '`File -> New -> Java Class`'. Ensure that the folder is '`/HelloWorld/src`' and name the class '`HelloWorld`'. Tick the box to create a method stub for '`public static void main(String[] args)`'. Click `Finish` to create the new class.

2. In the main WSDD window double click ‘`HelloWorld -> src -> (default package) -> HellowWorld.java`’ from the ‘`Packages`’ panel and add the following code inside of the `main` method:
`System.out.println(“Hello World!\n”);`

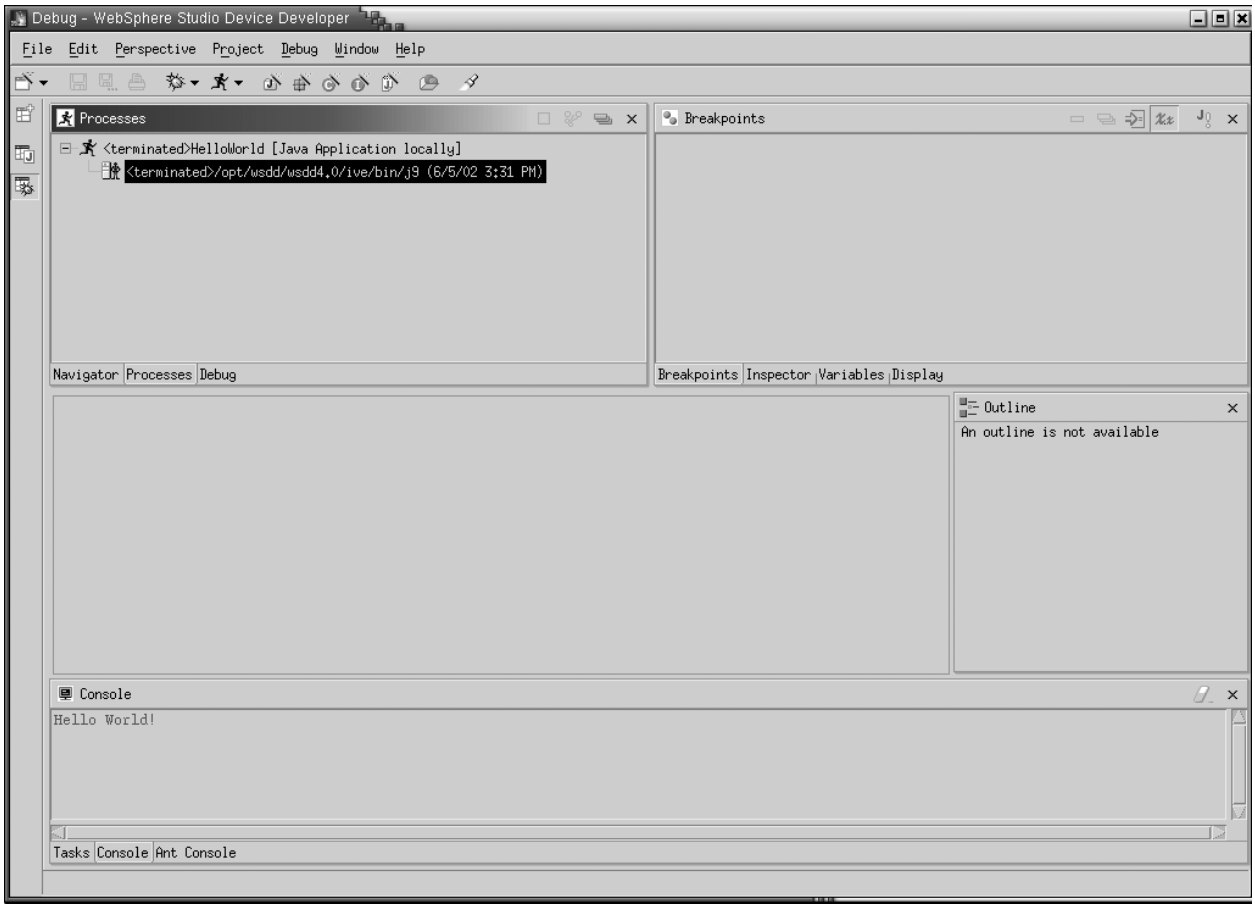


Building and Running Your Application

1. Click '`File -> Save HelloWorld.java`' and then '`Project -> Build All`'.
2. Click on the 'Run' Button (), Select '`Java Application Locally`' and press `Next`.



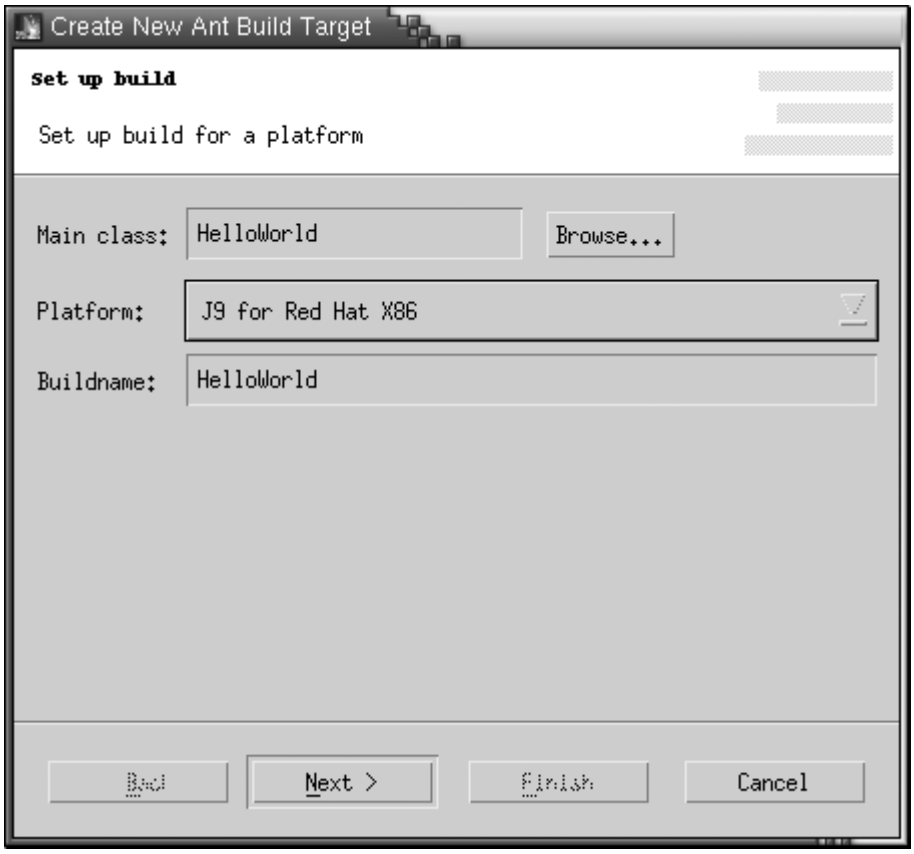
3. Select the ‘`HelloWorld`’ class and click `Finish`. You will be placed into a debugging perspective and should see ‘`Hello World!`’ in the console panel.



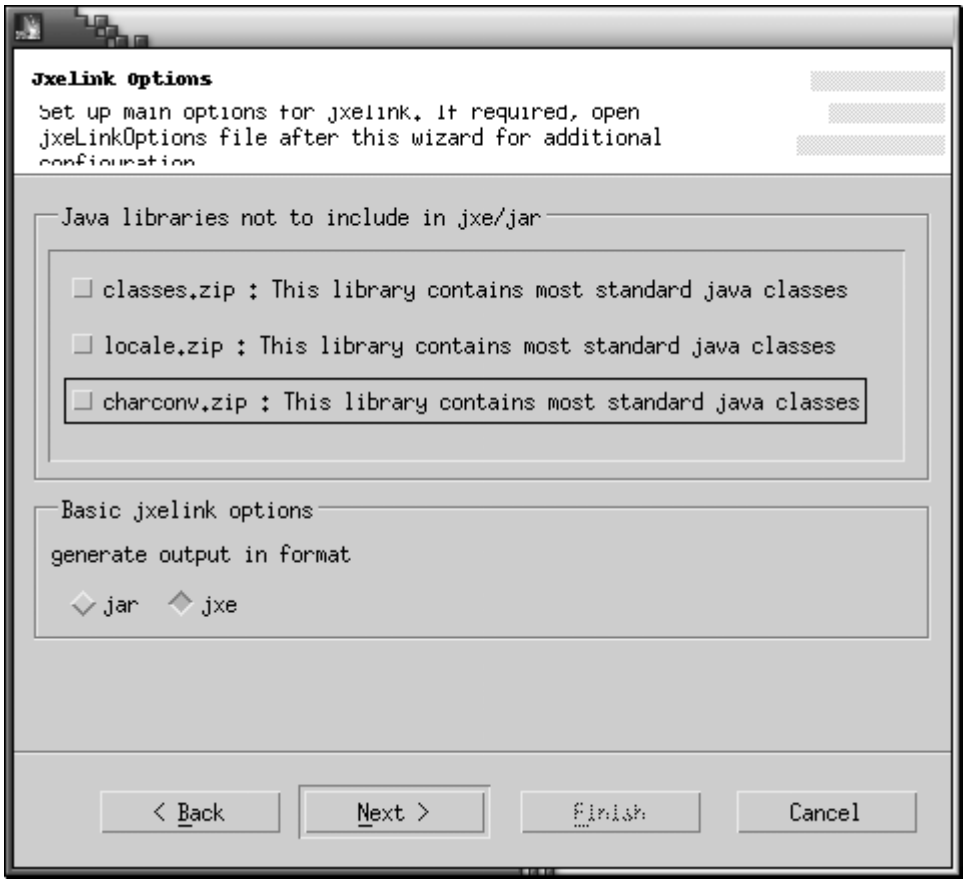
Creating a HelloWorld.jxe

To enable your programs to run on the J9 Java Virtual Machine on the target board you need to create a .JXE file. The WSDD SmartLinker is used to do this.

1. Click the Java perspective button () at the top left, under the toolbar.
2. Double Click on the '`wsddbuild.txt`' to access the build configuration panel.
3. Click on the '`Add Build`' button.
4. Ensure the main class is '`HelloWorld`' and the platform is '`J9 for Red Hat X86`'. Name the build '`HelloWorld`' and click `Next`.



5. We want to build everything into the JXE so uncheck all the check boxes under ‘Java libraries not to include in JXE/JAR’. Ensure the output format is JXE. Press `Next`.



6. Select '`JXE or Jar Application Locally`' as the target. Select '`Run and Debug`' as launch mode. Click `Finish`.

7. In the Build pane, select the new '`lnxx86/HelloWorld`' and click '`Perform Build`'. The SmartLinker will run.

Test HelloWorld.jxe

1. At the bottom of the build pane, click the 'Launches' tab. The pane will change to reflect the launches which you have configured.
2. Select '`Run and Debug local platform/HelloWorld.jxe as Jxe or Jar application locally`' and click '`Run`' from the bottom of the pane.
3. The JXE will run, and produce the same output as the previous test run.

Running the JXE on the Target

Using FTP, transfer `workspace/HelloWorld/lnxx86/HelloWorld.jxe` (relative to your home directory) to a suitable directory on the target (e.g. `/home/arcom`). See the Quick Start Manual for details on using FTP, or better, SCP.

1. Run the HelloWorld.jxe with the J9 Java Virtual Machine
   `j9 –jxe:/home/arcom/HelloWorld.jxe`

GCJ

GCJ is an ahead-of-time optimizing compiler for the Java Programming Language. It can compile:

- Java source code directly to native machine code,
- Java source code to Java byte code (class files),
- Java byte code to native machine code.

Compiled applications are linked with the GCJ runtime, libgcj, which provides the core class libraries, a garbage collector, and a byte code interpreter. libgcj can dynamically load and interpret class files, resulting in applications consisting of a mixture of compiled and interpreted classes.

Most of the APIs specified by 'The Java Class Libraries' Second Edition and the 'Java 2 Platform supplement' are supported, including collections, networking, reflection and serialization. AWT and RMI are currently unsupported, but work to implement them is in progress.

Advantage of Using GCJ

- Native code typically executes faster than byte code executing on a Java Virtual Machine.
- The Java code can call C/C++ libraries and code through the Cygnus Native Interface (CNI) and C via JNI.
- GCJ can compile byte code to native code.

Disadvantages of Using GCJ

- Java programs compiled to native code are not portable (although GCJ does include a byte code compiler and interpreter).
- At present some library classes are missing.
- GCJ runtime libraries are quite large (4.5 Mbytes).

Installing GCC on the Host System

GCJ is a part of the GNU Compiler Collection (GCC). Before installing GCC on your host system please read the GNU General Public License which is on the development kit CD as '`Reference/Licenses/GPL.txt`'. When you have unpacked the source tarball you should also read '`gcc-3.0/libjava/LIBGCJ_LICENSE`'.

You need to install GCC on to your host system to allow you to design and build applications which can then be transferred to, run and tested on the target PEGASUS board.

1. Copy the GCC tarballs from the development kit CD directory '`utils/gcc`' to an appropriate directory (e.g. '`~/gcj`'), on the host system.
2. Unzip and untar all the archives by typing the following
   ```
   tar xfz gcc-core-3.0.tar.gz
   tar xfz gcc-g++-3.0.tar.gz
   tar xfz gcc-java-3.0.tar.gz
   ```

3. Apply the Arcom patch
   `patch –p0 –i gcc-3.0-ael.patch`
4. Create a build directory '`gcc-3.0-build`' and change to that directory
   ```
   mkdir gcc-3.0-build
   cd gcc-3.0-build
   ```
5. Configure using the configure script
   `../gcc-3.0/configure --prefix=/opt/gcc-3.0 --enable-threads=posix`

   Note: If you specify an alternate prefix, ensure that you don't overwrite your current gcc.

6. Build GCC
   `make`
7. Install (you must have root user privileges)
   `make install`

GCJ Examples

On the development kit CD are some example applications for GCJ in the '`Java_Examples/gcj/`' directory.

- **HelloWorld** - directory for HelloWorld example
- **cni** – directory for a CNI example

There are Makefiles and README files for each of these examples in the appropriate directory.

Ensure that you set the **`LD_LIBRARY_PATH`** environment variable

```
export LD_LIBRARY_PATH=”/opt/gcc-3.0/lib”
```

and set the **`PATH`**

```
export PATH=”/opt/gcc-3.0/bin:$PATH”
```

before compiling the examples.
If you don't the executable will not run on the Target Board.

For more detailed information on compiling and linking with GCJ see '`/Reference/GCJ/gcj.html`' on the development kit CD or http://www.gnu.org/software/gcc/java/.

Running the Application on the Target

1. Transfer the executable HelloWorld to your target board using FTP.
2. Make the application executable
   `chmod +x ./HelloWorld`
3. Run the application
   `./HelloWorld`

Debugging GCJ Applications.

Gdb 5.0 includes support for debugging gcj-compiled Java programs. For more information see http://www.gnu.org/software/gcc/java/gdb.html.

Notes on Using Cygnus Native Interface (CNI)

When building an application which using CNI (to implement native methods) the resulting executable is linked to the C++ standard library. This library (`/opt/gcc-3.0/lib/libstdc++.so.3`) needs to be transferred to the target system for the application to run.

For more details on CNI consult the CNI reference manual `/Reference/GCJ/cni.pdf` on the development kit CD.

Appendix A - Sources for the Software Contained in Arcom Embedded Linux.

The following lists some of the packages included in Arcom Embedded Linux, their source, any patches applied and their licensing terms. Copies of the more widespread licenses are included on the development kit CD in the directory `/Reference/Licenses`, they are also listed in Appendix B. For less widespread licenses please consult the source distribution for that package.

Linux Kernel 2.4.18
Original source
/source/kernel_2.4.18.orig.tar.gz
ftp://ftp.kernel.org/pub/linux/kernel/2.4/linux-2.4.18.tar.gz
Arcom Patch
/source/kernel_2.4.18-3.diff.gz
License
GPL

GNU Libc Library 2.2.4
Original source
/source/glibc_2.4.4.orig.tar.gz
Arcom Patch
/source/glibc_2.4.4-2.diff.gz
License
LGPL

GNU GCC 3.0
Original Source
/source/gcc_3.0.orig.tar.gz
Arcom Patch
/source/gcc_3.0-1.diff.gz
License
GPL + ‘special linking exception’ (See ‘GCJ Java Runtime License’ on page 4)

Bash 2.05
Original Source
/source/bash_2.05.orig.tar.gz
Arcom Patch
/source/bash_2.05-3.diff.gz
License
GPL

Modutils 2.4.12
Original Source
/source/modutils_2.4.12.orig.tar.gz
Arcom Patch
/source/modutils_2.4.12-2.diff.gz
License
GPL

TinyLogin 0.80
Original Source
/source/tinylogin_0.80.orig.tar.gz
Arcom Patch
/source/tinylogin_0.80-2.diff.gz
License
GPL

BusyBox 0.60.2
Original Source
/source/busybox_0.60.2.orig.tar.gz
Arcom Patch
/source/busybox_0.60.2-1.diff.gz
License
GPL

Linux Ftpd 0.17
Original Source
/source/linux-ftpd_0.17.orig.tar.gz
Arcom Patch
/source/linux-ftpd_0.17-1.diff.gz
License
BSD

Netkit Ftp 0.17
Original Source
/source/netkit-ftp_0.17.orig.tar.gz
Arcom Patch
/source/netkit-ftp_0.17-1.diff.gz
License
BSD

Thttpd 2.19
Original Source
/source/thttpd_2.20c.orig.tar.gz
Arcom Patch
/source/thttpd_2.20c-2.diff.gz
License
BSD like

SysVInit 2.78
Original Source
/source/sysvinit_2.78.orig.tar.gz
Arcom Patch
/source/sysvinit_2.78-2.diff.tar.gz
License
GPL

Pppd 2.4.0
Original Source
/source/ppp_2.4.0.orig.tar.gz
Arcom Patch
/source/ppp_2.4.0-2.diff.gz
License
Mixed GPL/BSD

Sysklogd 1.4
Original Source
/source/sysklogd_1.4.orig.tar.gz
Arcom Patch
/source/sysklogd_1.4-2.diff.gz
License
GPL

XFree86/TinyX 4.1.0
Original Source
/source/xfree86_4.1.0.orig.tar.gz
Arcom Patch
/source/xfree86_4.1.0-4.diff.tar.gz
License
MIT

Contact Arcom if modifications are required to the following software: flashboot, check-libs, Stage 1 flash boot loader (stage1.rom), BIOS images and BIOS update utilities.

Appendix B - Useful Web Links

General Linux Information
http://www.linux.org/

GNU GCC
http://www.gnu.org/software/gcc

GNU GCJ
http://www.gnu.org/software/gcc/java

Linux Documentation project
http://www.linuxdoc.org/

The BSD License
http://www.opensource.org/licenses/bsd-license.html

GNU General Public License (GPL)
http://www.gnu.org/copyleft/gpl.html

GNU Lesser General Public License (LGPL)
http://www.gnu.org/copyleft/lgpl.html

The MIT License
http://www.opensource.org/licenses/mit-license.html

RUTE: The Rute User's Tutorial and Exposition
http://rute.sourceforge.net

Frequently Asked Questions

Question: **I have Linux pre-loaded onto an PEGASUS board. How do I boot from an attached floppy or CD-ROM drive ?**

Answer: During the LINUX boot process, you may need to hold down the [ALT] key to boot from the BIOS selected device, depending on the configuration of the attached peripherals.
Note: The supplied development kit PSU is not recommended for systems fitted with a CD-ROM drive, a standard PC/AT power supply is recommended .

Question: **How do I remove Linux from a configured PEGASUS board ?**

Answer: At the Linux prompt type :
dd if=/dev/zero of=/dev/mtd0 count=1
This will destroy the start of the BIOS extension as stored in flash. When the system is rebooted, the LINUX BIOS extension will not be found so any attached bootable media will be used instead.

Question: **I see long timeouts and error messages such as 'hda: no response (status = 0xa1), resetting drive' when initializing the IDE controller. What can I do?**

Answer: You probably have the IDE bus setup with a device acting as a slave but no master. This causes Linux to spend several seconds looking for the master device when initializing the IDE controller. The solution is to reconfigure the slave device to be a master, often this will mean changing the jumpers on the CD-ROM or hard disk drive.

Question: **I periodically see error messages such as 'INIT: Id "1" respawning too fast: disabled for 5 minutes'.**

Answer: This is caused by a service spawned by init dieing repeatedly. A common cause of this is to run a login process on a VGA virtual console when there is no VGA card attached. This happens in runlevels 3, 4 and 5, but not in runlevel 2. You can make runlevel 2 default by editing /etc/inittab and replacing 'id:3:initdefault' with 'id:2:initdefault'.