Linux on the Sun JavaStation NC HOWTO

Table of Contents

Linux on the Sun JavaStation NC HOWTO....1
Robert S. Dubinski....1

1. META Information....2
1.1. The Purpose of this Document....2
1.2. Acknowledgments....2
1.2.1. Document Contributors....3
1.3. History....3
1.4. Document Copyright and Licenses....5
1.5. Location of the Latest Version and Source....5
1.6. Reporting Bugs Found In or Additions to the HOWTO....6
1.7. TODO List for this HOWTO....6

2. What is a JavaStation?....7
2.1. What is a JavaStation NC?....7
2.2. Definition of an NC including the Differentiation from PC's....7
2.3. Description of the JavaStation Model Line including Hardware Specs....8
2.3.1. JavaStation−1 ["Mr. Coffee"] ["the brick"] [Sun Option No. JJ−xx]....8
2.3.2. JavaStation−NC [" JavaStation−10"] [" Krups"] ["the tower"] ["the percolator"] [Sun Option No. JK−xx]....9
2.3.3. JavaStation−E ["Espresso"] [Sun Option No. JE−xx]....10
2.3.4. JavaEngine−1 ["JE−1"]....11
2.3.5. The "Dover" JavaStation model....11
2.3.6. The Generation 3 "Super JavaStation"....12
2.3.7. The Pre−Mr. Coffee JavaStation Prototype....12
2.3.8. The Pre−Mr. Coffee JavaStation/Fox....13
2.4. Reasons for Running Linux and NC Myths Dispelled....13
2.5. Why JavaStations are No Longer Produced....14
2.6. Where to Purchase a JavaStation....17

3. Background Requirements for Linux on a JavaStation....18
3.1. Complete Hardware Requirements....18
3.2. Network Service Requirements....18
3.3. Understand the JavaStation Boot Sequence....18
3.4. Additional Software Requirements: Replacement Firmware (PROLL)....19
3.5. Decide on your Filesystem−type: NFS−Root, or Embedded?....19
3.5.1. "NFS−Root" Filesystem....20
3.5.2. "Embedded−Root" Filesystem....20
3.6. Support Sites to Check Out: Zaitcev's Linux Site....20

4. Build Your Kernel....21
4.1. Before you begin....21
4.2. Make sure you use 32−bit mode....21
4.3. Supported Linux Kernel Versions....21
4.4. Required Kernel Configuration Options....22
4.5. Necessary Patch for "Embedded−Root" FS Configurations....23
4.6. Build the JavaStation−Ready Kernel....23
4.7. Convert Kernel from ELF to a.out format....24

Table of Contents

4.8. JavaStation−Ready Kernel Images, System.map and .config File Samples....24
4.8.1. 2.3.99pre3_embedded_RSD....24
4.8.2. 2.3.99pre3_nfsroot_RSD....25
4.8.3. 2.4.2_embedded_RSD....25
4.8.4. 2.4.2_nfsroot_RSD....26
4.8.5. Outside kernel mirrors....26

5. Build A JavaStation−Ready FileSystem....27
5.1. Preparing Yourself to Build Your Own Filesystem....27
5.2. Contents of the "/etc/fstab" File....27
5.2.1. "NFS−Root" Filesystem fstab....27
5.2.2. "Embedded−Root" Filesystem fstab....28
5.3. The "Embedded−Root" Image Creation Procedure....28
5.4. Sample FileSystems....30
5.4.1. jsroot_varol_19991221....30
5.5. Sample X Servers....30
5.5.1. XF86_FBDev_3.3.3.1_19990104....30
5.6. Outside Sample Filesystems....31

6. "Out of the Box" JavaStation Boot File Solutions....32
6.1. Simple Solution #1....32

7. Set up Your Server....33
7.1. Preface....33
7.2. Setting up the RARP service....33
7.3. Setting up the DHCP service....34
7.4. Set up NFS service ("NFS−Root Options" Only)....34
7.5. Setting up for Boot with TFTP....34

8. Booting Your JavaStation....36
8.1. What to See When Booting Linux....36
8.1.1. Stage 1: White Screen....36
8.1.2. Stage 2: Coffee Cup Logo....36
8.1.3. Stage 3: A Window to PROLL....36
8.1.4. Stage 4: Kernel Boot....37

9. Questions and Troubleshooting....38
9.1. When booting, the message "The file just loaded does not appear to be executable." Why?....38
9.2. When booting, the message "no a.out magic" appears and halts the boot. Why?....38
9.3. I tried booting a Krups but JavaOS comes up. I don't even have JavaOS!....38
9.4. Cannot Boot an "Embedded−Root" image > 10 MB on my JavaStation. Why?....38
9.5. After Booting, Typing Anything Yields Garbage Characters. Why?....38
9.6. In X Sessions to a Solaris server, the font server "xfs" crashes. Why?....39
9.7. Performing Indirect XDMCP to a Solaris Server Results in Session Login Failures. Why?....39
9.8. TFTPd config doesn't work on SUSE 6.3. Why?....39
9.9. Regarding RARP: Is it Needed or Not?....40
9.10. Can One Use the Smart Card Reader on the Espresso models?....40
9.11. Can One Use the Solaris DHCP server instead of ISC?....40

Table of Contents

9.12. Can One Pass Arguments to "/sbin/init" in a Diskless Boot like This?....41
9.13. Enabling X on the JavaStation....41
9.14. Is There Mailing List Help?....41
9.15. Can One Boot a JavaStation from Onboard Flash Memory?....42
9.16. Does "Piggyback" work for the x86 too?....42
9.17. I put new memory in, but now it doesn't boot. Why?....42
9.18. Now that JavaStations work with Linux, what about other Free OSs?....42
9.19. Do the Linux 2.4 kernels work? What's the latest that works?....42
9.20. Can I compile the kernel on a non−SPARC machine?....43
9.21. Can I get an ok> prompt like other Sun equipment?....43
9.22. My keyboard isn't recognized. What can I do?....43
9.23. Proll reports "TFTP: ARP Timeout". Why?....43
9.24. Why Can't I Get TrueColor on Krups?....43
9.25. I followed this HOWTO, but my Dover doesn't work. Why?....44
9.26. Can framebuffer be loaded following a serial console initialization?....44
9.27. I really need a complete out−of−the−box solution, pronto!....44
9.28. You Didn't Answer My Question....44

10. Reference Docs....45
10.1. Mr. Coffee Jumper Info....45
10.2. Krups Jumper Info....45
10.3. JavaStation Press Release....45
10.4. JavaOS 1.0 Download....45
10.5. Espresso IDE circuit....46
10.6. JavaStation Boot Monitoring Key Combinations....47
10.7. JavaStation Photo Gallery....47

A. GNU Free Documentation License....49

0. Preamble....50

1. Applicability and Definitions....51

2. Verbatim Copying....52

3. Copying in Quantity....53

4. Modifications....54

5. Combining Documents....56

6. Collections of Documents....57

7. Aggregation with Independent Works....58

8. Translation....59

Table of Contents

9. Termination.....60

10. Future Revisions of this License.....61

How to use this License for your documents.....62

Linux on the Sun JavaStation NC HOWTO

Robert S. Dubinski

Copyright © 1999, 2000, 2001 by Robert S. Dubinski

2001−Oct−31

This is a HOWTO document describing how to enable the GNU/Linux OS on the Sun JavaStation NC.

1. META Information

This section lists the meta−information of this document. The hows, whys, location and changes to the structure of the document are documented here. The main content begins in the next chapter.

1.1. The Purpose of this Document

This document is to serve as a comprehensive HOWTO and FAQ collection regarding the Sun JavaStation NC and enabling the GNU/Linux OS on it.

The intended audience of this document is anyone who has an interest in enabling Linux on the Sun JavaStations. The document structure is laid out to serve as either a top−to−bottom read for a newcomer, or as quick reference on a single topic for advanced users. Pointers to sample files submitted by users are included for extremely hurried readers.

The author of this document is Robert Dubinski <rsd@dubinski−family.org>. Robert is the former computer technician and UNIX systems administrator for Marquette University's Math, Statistics and Computer Science Department, where he had 125 JavaStations running Linux. These machines were all configured using the information, techniques and files presented in this document.

In early 1999, Eric Brower <ebrower@usa.net> wrote the first informal HOWTO for the JavaStation. Parts of this document are inspired by his work, and all unique information presented there have since been merged into this document. Eric's original mini−HOWTO is saved for posterity at: http://dubinski−family.org/~jshowto/Files/texts/eric_brower_js_howto_19980218.txt

This HOWTO also aims to serve as a member document of the Linux Documentation Project. The LDP can be reached at: http://www.linuxdoc.org

1.2. Acknowledgments

Enabling Linux on the JavaStations , and allowing this HOWTO to come to be would never have been possible without the fine work of the following people:

- Pete Zaitcev <zaitcev@yahoo.com> (Primary JavaStation kernel mod author)
- Eric Brower <ebrower@usa.net> (XFree mods and author of the original embedded−build HOWTO)
- Varol Kaptan <varol@ulakbim.gov.tr> (made available his Krups images and patches. Backported kernel support to 2.2.x series)
- David Miller <davem@redhat.com> (the original Linux/SPARC kernel porter)
- The Linux/SPARC kernel porters and mailing list
- The thousands of contributors to the Linux kernel

The HOWTO author wishes to give a second thank−you to Pete and Eric for their work:

> Pete got me going with Linux on the JavaStation in December 1998, has been the main kernel programmer adding in support for the JavaStation line, and despite his busy work schedule was nice enough to find time to answer all my email queries for help over the last 15 months.

> Eric worked on bringing X support to the JavaStation when it had none. He had been working on a dedicated server for the JavaStation in early 1999, and kept me informed of his progress. In mid−1999, he switched tactics and sent a working framebuffer example to test out. He also wrote the first comprehensive mini−HOWTO for the JavaStations, answered my email questions, and got me interested in the embedded solution which I employ here at Marquette.
>
> Thank−you Pete and Eric!
>
> −−Robert Dubinski, March 2000

1.2.1. Document Contributors

The following people have contributed to this specific document:

- Pete Zaitcev <zaitcev@yahoo.com> (Proofreading and factual corrections of initial drafts)
- Eric Brower <ebrower@usa.net> (Proofreading and factual corrections of initial drafts)
- Richard Tomlinson <Richard.Tomlinson@one2one.co.uk> (Document reader, Krups tester, feedback)
- Michael R. Eckhoff <foobar@null.net> (feedback on sample kernel)
- John Bodo <sales_nospam@bodoman.com> (JavaStation prototype info)
- Simon Whiting <Simon.Whiting@mysun.com> (Typo Fix in DHCP config)
- Alex Cellarius<alexc@mail.systems104.co.za> ("Dover" info and pic)
- Matt Lowry<mclowry@cs.adelaide.edu.au> (Suggestion of Boot Sequence Visuals Section)
- David Tinker<david@hemtech.co.za> (Dover model info)
- David O'Brien<obrien@NUXI.com> (Fox prototype info)
- Olaf Pueschel<olf@olmos.de> (OBP info, true color info)
- Richard Tomlinson<richard@sysgen.co.uk> (Boot monitoring key combinations)
- Zachary Drew<zach@math.umn.edu> (Troubleshooting Suggestions)
- Robert Thornburrow<robert@tsac.fsnet.co.uk> (non−SPARC piggyback info)
- Bill Childers<bill@nulldevice.net> (Assorted Dover Info)
- Simon Kuhn<address_lost!> (Donated a Sun4 for kernel builds)
- Nate Carlson<address_lost!> (supplied pre−compiled kernels, but the HOWTO author lost the info during disk failures.)

If you contributed a tidbit of info and are not listed, please email the document author to get yourself listed. Everyone deserves recognition helping this document evolve.

1.3. History

Revision History

Revision 1.30

31 Oct 2001

Many major changes: restructured for better layout, new chapters added, updated files and file pointers, new master distribution location, source broken into parts, new sample files, md5sums on all sample files, overall update and proofread of materials. Thank−you very much to Simon Kuhn for donating an old Sun4 to enable more sample files be made. Thank−you also to Nate Carlson for donating sample kernels (unfortunately that info was lost during a disk crash...Nate, please contact me.). If anyone contributed items between May 31 and Oct 31 and it does not show up in this revision, please resubmit it.

Revision 1.25

30 Oct 2001

This is called the "@#!$?" release. It is called such because there were some small−mid size changes to the

document which were lost in a disk failure, prior to me re−uploading to the LDP site. To give you an idea of how bad the situation was, the last version I had on backup was in Docbook SGML, while I had switched to Docbook XML many months prior. Simply put, any contributed changes or email contacts I had with contributors were lost, and are hopefully on their way back in with this release. If you contributed something, and it disappeared, please contact me immediately, and it'll get back in. −−RSD

Revision 1.22 31 May 2001
Changed file links, some sample file formats, and clarified info relating to the sample files, following requests on the sparclinux mailing list.

Revision 1.20 08 May 2001
Information Refresh up to the current date, and change to GNU Free Documentation License 1.1

Revision 1.15 01 May 2001
Migrate source to DocBook XML 4.12

Revision 1.13 02 Feb 2001
Minimal Bugfixes

Revision 1.12 29 Dec 2000
Additional info on the "Fox" model

Revision 1.11 23 Nov 2000
Krups truecolor blurb, removed one acknowlegement per email request

Revision 1.1 15 Nov 2000
Numerous updates and additions revisions

Revision 1.05 16 Jun 2000
Requested Format Changes and Fixes

Revision 1.04 13 Jun 2000
Suggested Fixes and Added Requests

Revision 1.03 04 May 2000
Minor Fixes, Requests

Revision 1.02 28 Apr 2000
Small fixes.

Revision 1.01 25 Apr 2000
"Brown Paper Bag" Revision.

Revision 1.0 24 Apr 2000
First submission to the LDP.

Revision 0.9 18 Apr 2000
Continued reorganization and final merges.

Revision 0.7 15 Apr 2000
Migration from LinuxDoc DTD to Docbook DTD.

Revision 0.71 14 Apr 2000
Received word doc was forwarded inside Sun.

Revision 0.7 14 Apr 2000

Linked on Metabyte Website.	
Revision 0.6 First semi−public release.	9 Apr 2000
Revision 0.4 First move to comprehensive HOWTO.	24 Mar 2000
Revision 0.2 More notes collected and merged.	15 Oct 1999
Revision 0.1 Initial scraps put together.	24 Jun 1999

1.4. Document Copyright and Licenses

Copyright (c) 1999−2001 Robert S. Dubinski. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being: "Why JavaStations are No Longer Produced", with one Front−Cover Text: "Linux on JavaStation HOWTO", and with one Back−Cover Text: "This document was written by Robert S. Dubinski in the hope that more people can put their JavaStation hardware to good use. Thank−you to the Linux kernel hackers who made this happen, and thank−you to Sun for a rock−solid piece of hardware." A copy of the license is included in the section entitled "GNU Free Documentation License".

The document author makes *no* warranties that all the information presented here is completely accurate, and cannot be held liable to any loss you experience as a result of the information you use from here.

Best efforts have been made to ensure everything included is accurate as of the publication date listed at the beginning of this document, but there is always a possibility something may be wrong. In this case, doublecheck with alternative sources first before considering implementing anything at a production−level. If you find something wrong, drop the author a line at `<rsd@dubinski−family.org>` or send a patch to the document source, and corrections will be made immediately.

This document is an official member document of the Linux Documentation Project.

1.5. Location of the Latest Version and Source

The latest online version of this document can be found at: http://dubinski−family.org/~jshowto .

The pre−processed XML source to this document, written to the Docbook DTD, version 4.1.2, is available from: http://dubinski−family.org/~jshowto/doc/src/

The pre−processed XML source to the GNU Free Documentation License, written to the Docbook DTD, and which this document is licensed under, is available from:
http://dubinski−family.org/~jshowto/doc/src/gfdl.xml

Copies of this document are also available from the Linux Documentation Project at:
http://www.linuxdoc.org/HOWTO/JavaStation−HOWTO.

This project used to be available at the URL 'http://javastation−howto.homeip.net'. In Spring 2001, homeip.net discontinued their free service and moved to a fee based scheme. Given the hundreds of mirrors of LDP documents, I do not find the fees justifiable. I have changed all references in this document back to my home server. Between my server's address, http://dubinski−family.org/~jshowto, the LDP website, and its hundreds of mirrors, you should be able to always find the JavaStation−HOWTO. If this is not the case, email me immediately at either <rsd@dubinski−family.org> or <dubinski@mscs.mu.edu>.

1.6. Reporting Bugs Found In or Additions to the HOWTO

Any problems or concerns about the HOWTO should be reported via email to the author, Robert Dubinski, at <rsd@dubinski−family.org>. Do *NOT* send document bug reports to the SparcLinux mailing list, the debian−sparc mailing list, or the Linux Documentation Project. The folks on there really do not care about my typos or server misconfigurations, so please don't trouble them.

1.7. TODO List for this HOWTO

1. As NetBSD now supports JavaStations as well, it would be good to talk about support and sample files for it too.

2. What is a JavaStation?

This chapter explains to the reader what the JavaStation line is, its components, NC concepts, how to get one, and why one would choose the Linux OS for it.

2.1. What is a JavaStation NC?

The JavaStation NC is a model line of network computers built and sold by Sun Microsystems between November 1996 and March 2000. The JavaStation line was Sun's low−cost terminal option during that timeframe. It was the marketed successor to the Xterminal 1 and is succeeeded by the SunRay, although all three machines are fundamentally different.

The JavaStation hardware ran Sun's own JavaOS and either Sun's Hotjava web browser, Sun's HotJava Views task−manager software, or custom Java applications of the customer's choice.

The JavaStation was originally billed in November 1996 sneak previews as a low−cost desktop terminal, providing customers access to hot new Java applications, "legacy" X applications, and "legacy" MS Windows apps. During its lifetime, The JavaStation's marketed functionality was changed twice from "desktop terminal" to "single−app desktop device" to finally a "browser−based kiosk device".

At no time did Sun market the JavaStation as capable of running its flagship Solaris operating system the Linux OS, or any other OS than Sun's JavaOS.

2.2. Definition of an NC including the Differentiation from PC's

A network computer, or NC, was hailed as "the next big thing" in computing from late 1995 to early 1998. Conventional PC's, called "fat clients", were expected to be minimized in businesses by thin−client NC's.

Thin−clients get their OS, applications, and data files entirely through the network. They are different from dumb−terminals; they run full−scale graphical applications. Thin−clients are also different than graphical X−terminals. X−terminals typically run an X server and display the client programs of a remote server. Thin clients generally run full−scale graphical programs locally, such as a web browser, a Java application, or a "legacy−connectivity program", which enables the thin−client to display X apps or MS Windows apps which run on more powerful servers.

Advantages of NC's include:

- "Zero−Administration". (Add a new NC and it will get *everything* it needs off the network, without an admin ever needing to visit it.)
- Lower Total−Cost−of−Ownership (TCO) (No internal hard drives, floppy drives or CD players reduces form−factor, repair expenses, selling price and thus total−cost−of−ownership.)
- Access to all web−based apps as well as "legacy" X and MS Windows apps.
- Quick upgrades (just upgrade your server and the changes propogate throughout)
- Longer lifespan (just upgrade the software, growing hard disk and memory requirements is not an issue)
- Smaller OS footprint (when running brower−based apps)

Disadvantages of NC's:

- No local access to data files (all your files stored on a remote server)
- Requires fast, stable networks
- NC's generally have a low maximum amount of memory. Though not as bad as with fat−clients, this does eventually become a liability for the thin−client.

2.3. Description of the JavaStation Model Line including Hardware Specs

Depending on who you talk to, the number of JavaStation models that were created is anywhere from one to six. The descriptions below will explain why.

2.3.1. JavaStation−1 ["Mr. Coffee"] ["the brick"] [Sun Option No. JJ−xx]

This model is the most prevalent JavaStation model you are likely to find, although it wasn't the one and only *JavaStation* model Sun wished to sell to the public. The JavaStation−1 was the first generation JavaStation, released in November 1996 to pilot deployments as Sun's "proof of concept" of the Java NC design.

Hardware−wise, the JavaStation−1 is a Sun4M architecture machine. It is based on the SPARCStation−4 design, with some deletions and PC−like modifications. It is powered by a 110 Mhz MicroSPARC IIe CPU and has no SCSI, internal disks , floppy, CD or expansion slots. The Mr. Coffee motherboard is Sun Part No. 501−3141.

Instead of using the Sun−type keyboard and mice, JavaStation−1 uses PC−like PS2 parts instead. One of the original marketing highlights of the JavaStation was that it would use standard PC parts wherever possible to keep overall price down.

The "brick" has four PC−like SIMM slots. The SIMMs taken are industry−standard 60ns, 32−bit, 72−pin, 5V fast page SIMMs, installed in pairs. Each slot is capable of holding up to a 16MB SIMM, bringing the maximum total capacity of the unit to 64MB. The "xx" in the Sun Option# of the unit indicated how much memory the unit shipped with.

For video display, the JavaStation−1 utilizes the Sun TCX framebuffer, capable of 1024x768@70Hz in 8−bit color. The port connector however, is a standard VGA jack , enabling the user to use standard PC monitors if desired (again, low cost in mind). The on−board audio is a Crystal CS4231 chip , and the network interface is the Sun Lance 10Mbps interface. In addition, the "brick" also came with a 9−pin serial port and 1/8" audio out jack on its back.

The JavaStation−1 was fitted into the Sun "unidisk" form factor case, and has been seen in a number of color schemes. JavaStations have been fitted with casings in the white with light blue trim scheme used in Sun workstations, as well as the dark blue−grey "new desktop" scheme. Some say "JavaStation" and have the Java coffee cup logo written on it, others do not. Collectors may wish to collect all case variations.

The JavaStation−1 was used in early Sun demos, and sold to pilot sites. When first brought out, the cost to pilot sites was $699US. This was at a time when PC 's were still higher than $1000US. By the end of the pilot run, Sun was selling any remaining or used units for $299−$399US, in anticipation for its "real" JavaStation model.

See the JavaStation−1 at: http://dubinski−family.org/~jshowto/Files/photos/mr_coffee_front_view.jpg

2.3.2. JavaStation−NC [" JavaStation−10"] [" Krups"] ["the tower"] ["the percolator"] [Sun Option No. JK−xx]

This model is the second most prevalent JavaStation model you are likely to find. When you talk to industry people about the "JavaStation", this is typically the model remembered first. Delayed numerous times, the Krups model officially went on sale to the general public Mar. 26, 1998 at the annual JavaOne conference.

Though generation two of the JavaStation line, the Krups model was *the JavaStation* . Sporting a completely different board design than JavaStation−1 , Krups establishes what was to be the characteristic JavaStation architecture.

Krups is powered by a 100Mhz MicroSPARC IIep chip, (note the 'p'). Its mainboard had the internal addition of a PCI bus, about a year before this standard bus made its well−publicized appearance on the Sun Ultra workstation line. The Krups motherboard is Sun Part no. 501−4267.

Krups keeps the PS2 keyboard and PS2 mouse ports from JavaStation−1 , keeping in mind the low−cost, interoperable goal of generation 1.

With the new board design, came new memory chip sockets. Instead of SIMMs, the "tower" moved to 168−pin DIMMs. DIMMs had begun to make their way from the workstation realm to PC's in the time between generations one and two of the JavaStation line, so it was fitting for Sun to switch to it in anticipation of their status low−cost commodity memory chips. The DIMMs accepted by the "tower" are 168pin, 3.3V unbuffered EDO DIMMs (not SDRAM). With two sockets capable of holding a 32MB DIMM each, the Krups has a maximum capacity of 64MB RAM. As with the JavaStation−1 , the number "xx" in the Sun option number refers to the amount of memory shipped with the unit.

For video display, the JavaStation−NC utilizes the PCI−based IGS C1682 framebuffer, capable of 1280x1024@80Hz in 24−bit "true color". This is a step up from the 8−bit display on JavaStation−1 . The port connector remained a standard VGA jack like JavaStation−1, enabling the user to use standard PC monitors if desired. The on−board audio remains a Crystal CS4231 chip like JavaStation−1. The network interface on Krups is the Sun HappyMeal 10/100 Mbps interface, another step up from the original offering of JavaStation−1.

The "tower" came with the 9−pin serial port and 1/8" audio out jack as JavaStation−1, but it also added a 1/8" audio−in jack, to do sound recording with.

Another addition in the JavaStation−NC is a flash memory SIMM. This allows one to load the current revision of the OS onboard, increasing boot−speed tremendously.

Perhaps the thing most memorable about the JavaStation−NC is its case design. The Krups comes in an aesthetically appealing casing. The mainboard is mounted vertically, and the shell entraps it, giving it the "tower " or "percolator" shape referred to. With the streamlined case, the power supply is moved outside to small transformer. The Krups unit gives off so little heat that there are no onboard cooling fans, making the Krups a *dead−silent* machine. Imagine the difference in noise when replacing a lab of traditional desktops with the Krups! This case design earned Krups a" 1998 Industrial Design Excellence Award" from the Industrial Designers Society of America. This award announcement is still available for read at: http://www.idsa.org/whatis/seewhat/idea98/winners/javastation.htm" . It is also archived locally via "fair use" for future readers at: http://dubinski−family.org/~jshowto/Files/texts/krups_idsa_award.txt"

The Krups had an initial base price of $599US, $100US cheaper than Mr. Coffee's rollout price. Due to it being the only model formally sold by Sun to the general public, this is how Krups is sometimes referred to as the only JavaStation, and not one model of a product line.

See the JavaStation−NC at: http://dubinski−family.org/~jshowto/Files/photos/krups_front_view.jpg

2.3.3. JavaStation−E ["Espresso"] [Sun Option No. JE−xx]

This model is extremely rare to find. It was never available for sale in quantities to either the general public or the initial JavaStation deployments, limiting the model's production quantity. To call this "Generation Three" of the JavaStation may be improper, as Espresso is nothing like the generation three JavaStation written about in early Sun marketing literature.

The Espresso was designed as an extension of the Krups. It was geared to sites that wanted a little bit more functionality and expansion capability from their JavaStations: a cross between an NC and a workstation.

Espresso is powered by the same 110Mhz MicroSPARC IIep chip as Krups . It's mainboard is similar to Krups, with the addition of PCI slots and an IDE channel for local hard disks. The IDE on Espresso was not enabled in the demo units. Those who have tried to make it work have concluded the wiring is incorrect, and it requires a hardware rework to get going.

Espresso continues with the PS2 keyboard and PS2 mouse ports from Mr. Coffee and Krups.

Espresso uses the same 168−pin, 3.3V unbuffered EDO DIMMs as Krups. The maximum amount of memory for Espresso is reported to be 96MB. As with the Mr. Coffee and Krups , the number "xx" in the Sun option number refers to the amount of memory shipped with the unit.

For video display, the Espresso uses the PCI−based IGS C2000 framebuffer, along with the same standard VGA port connector as Krups and Mr. Coffee. The on−board audio remains a Crystal CS4231 chip like Krups, and the network interface remains a Sun HappyMeal 10/100 Mbps interface like Krups as well.

Espresso came with the 9−pin serial port and 1/8" audio out and 1/8" audio in jacks of Krups, and a new addition of a parallel port, and a second 9−pin serial port. Espresso also comes with the flash memory to load your OS on and bypass the network boot cycle.

One new addition to the Espresso is a smart card slot.

The Espresso comes in a "pizza box" style case like the old Sun SparcStations, only a little taller, and not quite as wide.

The Espresso was never sold to the public. There was an internal testing period at Sun, but the units never went into mass−production.

One Espresso user mentioned he now uses his unit as both a server and router, with the addition of an IDE disk and 3C905 ethernet card, demonstrating the expandability of this unit.

See the JavaStation−E at: http://dubinski−family.org/~jshowto/Files/photos/espresso_front_view.jpg

2.3.4. JavaEngine–1 ["JE–1"]

Like the Espresso, this unit is also an extremely rare find.

This unit is supposed to be of similar board design to the Krups, but in an ATX form factor, with soldered onboard flash memory, and with a regular SVGA video chipset.

Gleb Raiko <raiko@niisi.msk.ru)> with the help of Vladimir Roganov <roganov@niisi.msk.ru> did initial the Linux kernel support on "JE–1". Pete Zaitcev <zaitcev@yahoo.com> later obtained a "JE–1" unit and restored full support in Linux kernel 2.3.x+ .

As the author of this document has never seen a "JE–1", submissions from the public are welcome.

See the JavaEngine–1 at: http://dubinski–family.org/~jshowto/Files/photos/je1_overhead_view.jpg

2.3.5. The "Dover" JavaStation model

This is another box which does not exist officially outside of Sun. Little was known of it at the first revision of this HOWTO. Since then, proud owners have stepped forward. Basically, the Dover takes the Espresso theme and moves it to stock X86 parts.

Dover comes in a case similar to the Espresso, but there's nothing where the 'JavaStation–E' tag would be. Dover can be situated in a vertical position by removable feet. All that is printed on the case is "Sun MicroSystems 1998", and typically a serial number sticker of '12345678' and 'Made in Taiwan'.

The motherboard is 'baby ATX' in configuration, but not quite totally. Near the the front of the case is a fan that points at the CPU heat sink. The CPU heat sink has another fan on top of it. The motherboard has a Socket 7 CPU socket that houses a Cyrix MediaGCm–266GP CPU. There are typical PC motherboard jumpers with silk–screened legends for setting both clock speed and multiplier. The motherboard accepts a PC100 DIMM (max. size unknown) and a powersupply with AT–type power connectors. Included among them are two floppy and regular hard drive type plug. There are two small jumpers going to the motherboard, JPSB1 and JAUTO1, possibly for power management.

Expansion in Dover is via a two–card riser, with one PCI and one shared PCI/ISA slot. As mentioned earlier, the motherboard deviates slightly from standard ATX. Along the back edge under the cards are connectors for audio out, audio in, mic, HD15F video, two USB ports, D25F parallel printer, stacked PS/2 keyboard/mouse ports, and four 9–pin serial ports, marked A through D. Unlike other JavaStation models, there is no on–board ethernet. Instead, it typically is provided by a supplied 3COM 3C905B–TX Fast Etherlink XL PCI card (with a wake–on–LAN cable going to the motherboard). There is a standard Sun MAC address label on the back of the case.

Video is via a Cyrix CX5530 chip, but with the MediaGX chip, may be just an auxilliary chip. There exist both a FDD and HDD headers on the motherboard, but nowhere to mount a FDD in the case and no provision for an HDD bracket either. There is a simple piezo buzzer mounted to the motherboard and additionally a speaker with a cable leading back near the audio out jacks. Like the Espresso, there is a smart–card reader as well, and what looks like a compact–flash socket inside.

When booting it up, you get a blue JS screen. Under the exclamation point, are two memory card icons and a <...> icon. It reads:

```
Boot device: /ethernet   Arguments:
MAC Address: 08:00:20:95:5b:49
Open Boot 3.0, Built February 16, 1999 17:38:37
NIC: 10b7,9055 ethernet in PCI1                                          64MB SDRAM
Non-Volatile Device Memory Module Not Installed
SmartCard Reader Found
CPU Speed:  266 MHz


Can't open boot device

ok
```

The Dover model, since it is based on an x86 chip, is supported by Linux. This HOWTO however focuses on the SPARC–based JavaStations, so the procedures presented here *will not work* with it. However, there's plenty of x86 documentation at large to work from.

See the Dover at: http://dubinski–family.org/~jshowto/Files/photos/dover_inside.jpg

2.3.6. The Generation 3 "Super JavaStation"

Sun originally envisioned three generation models of the JavaStation: Mr. Coffee, the Krups, and the "Super JavaStation". Generation Three was billed in early literature as going to be the fastest JavaStation offered, with a high–speed CPU and a JavaChip co–processor to translate Java–bytecode in hardware.

All indications are that it never got beyond the mental stage, and was more of a marketing myth than anything else.

First, consider that the cost of higher performance CPU as a factor. If Sun packaged a high–performance CPU into a JavaStation, the low–cost advantage of an NC goes away.

Next, Sun did have their PicoJava chip available to decode Java bytecode, but rumor is the performance was not as good as expected, and the complete JavaChip project was shelved in the Summer of 1998, not long after Krups was formally released.

The "Dover" project was being worked on, but the "Corona " project, which would go on to become the Sun Ray , was the final nail in the JavaStation 's coffin.

So all indications are that this model is a piece of "vaporware". It is included here though, for the sake of completeness.

2.3.7. The Pre–Mr. Coffee JavaStation Prototype

After the original publishing of this HOWTO, word of one more "JavaStation" model surfaced. John Bodo, a reseller of JavaStation equipment, chimed in that he has a motherboard of a pre–JavaStation machine. It was made by Diba Corporation, which was later bought out by Sun. The unit was released as an early embededded Java platform that developers could use to build embedded Java machines. It has a Motorola 68030 CPU, 14.4k bps modem, ethernet interface, standard VGA interface and even a TV output. The prototype's date is circa 1996.

See the JavaStation Prototype at: http://dubinski–family.org/~jshowto/Files/photos/pre_js_1.jpg

2.3.8. The Pre−Mr. Coffee JavaStation/Fox

After receiving word of the JavaStation prototype from Diba, yet more information has come regarding another pre−Mr. Coffee model. This one though, has a greater known history we can share here.

This model was the JavaStation development box used by the developers of early JavaStation software. Basically it was a SS4/110 in a smaller, custom case similar to the Mr. Coffee enclosure, with more squarish profile.

The case has an off−white color with lateral stripe in Sun gray. It sits like a Mr. Coffee would on its side. The front was a 1/2 cyclinder i design in Sun gray, has the Sun Logo, the word "Sun" under that, and the Java cup logo at the bottom.

When booting up it claims to be a "JavaStation/Fox". The motherboard does not have a normal Sun part number. The CPU is a microSPARC−II running at 110MHz. The box has an onboard external SCSI connector, dual A and B serial ports, audio in and out sound ports (Crystal Semiconductor 4231, lance ethernet network interface, onboard PCMCIA (stp4020), one SBUS expansion slot, one AFXbus expansion slot, 2 72−pin SIMM slots (double−banked SIMMs only), and no on−board video. One would then add their own S−Bus frame buffer, or the 24−bit frame buffer from a ss5. Also, an optional internal SCSI laptop hard drive could be put in.

The motherboard's part number is 501−2785. The CPU is dated 1995 while the NCR chips are dated 1994, establishing the time frame of the Fox.

The NetBSD/SPARC FAQ has a few more words on the Fox at:
http://www.netbsd.org/Ports/sparc/faq.html#fox

See the JavaStation/Fox at: http://dubinski−family.org/~jshowto/Files/photos/fox_face.jpg

2.4. Reasons for Running Linux and NC Myths Dispelled

It turns out that Linux makes the JavaStations perform more than adequately on the desktop. Thanks to the dedicated work of the Linux developer community, the JavaStations offer users the low−cost, zero−admin, versatile desktop NC's they were originally billed to be, but with the added freedom granted by the Linux OS.

While low−cost PC's now eclipse the JavaStation in terms of default CPU speed and RAM size, the JavaStations running Linux are still well−suited for a number of tasks:

- Diskless X−Terminal. (Gives the JavaStations the capability of the Sun Xterminal 1 hardware that they replaced).
- The NC solution, Linux−style: local X + a java−capable browser can make the JavaStations perform like they did with JavaOS/HotJava, only *many* times faster.
- A beowulf node, or a dedicated RC5/SETI@HOME client. The JavaStation running Linux makes a stable, long−lasting number cruncher.
- A small, standalone machine. While a task more suited on today's low−cost machines, there's not much that prevents the JavaStation from performing as a full−fleged standalone UNIX machine by itself. Just remember to set your expectations appropriately when doing so; they were "low−budget" clients when they were sold, and should not be directly compared to today's workstation offerings.
- A small router and server, particularly with the Espresso model decked out with added IDE disks and NIC.

In all of the above scenarios, there is little to no maintenance of the machine once configured properly. Such is the advantage of the NC hardware.

JavaStations run so much better with Linux than JavaOS, one would think that even Sun should have offered it as an option. Unfortunately, Sun had killed the line in favor of the Sun Ray . While the performance of the Sun Ray is good, keep in mind it is not intended as a dedicated computing device, and due to its firmware is little more than a graphics display hanging off your Sun server, which can give you some unexpected bonus features (translation: "brand−name product lock"). The performance on the JavaStations with Linux will be similar to what you can get with a Sun Ray, but if ever you want to do something different with your machines, you have the flexibility to do so with the JavaStations. There was rumor of work to try and override the default behavior of the SunRay firmware, and make it into an adjustable computing device, but until that happens, running another OS on a SunRay is just a pipe−dream.

Lastly, if you're thinking of switching to diskless Xterminals on your network, you might consider the JavaStations over stripped down PC's. The hardware is standardized, smaller, and you do not need to worry about burning boot PROMs and the like.

2.5. Why JavaStations are No Longer Produced

Sun's official stance is that the JavaStation line was terminated in favor of the new Sun Ray line. A trip to the former JavaStation section of Sun's website at http://www.sun.com/javastation verifies this formal positioning. (fair use archival copy at: http://dubinski−family.org/~jshowto/Files/texts/sun_js_site_death.txt)

As the Sun Ray is not an NC in the traditional sense (it has a MicroSparc IIep CPU, but the firmware on the device prevents anyone from grasping it), there is no explanation why the two products could not co−exist.

In talking to the users of the JavaStations in the pre−Linux era, you will find strong opinions as to why the JavaStations are no more. The common thread in almost all opinions collected is that the software provided by Sun was inadequete for a production environment. Here are collected opinions from users of the Sun−provided software, included with their permission:

> I only used the Java Stations last summer while teaching 51 and 55/154. GoJoe was incredibly slow and I seem to remember having to login to several different screens and browsers just to be able to start anything.
>
> I had to apologize to my students for the slow and inconvenient machines −−− I remember making some jokes about technological progress.
>
> −−Dr. Alex Ryba, Former Professor at Marquette University (Quoted March 2000)

> Well, of course the old JavaStations were practically unusable. It's not a matter of just my opinion; we used to have CU 310 full of students using the Xterms all the time. As soon as the JavaStations appeared there were NO STUDENTS in there at all. The JavaStations killed CU 310. Now that the JavaStations are (thanks to you) back up to speed, students are beginning to come back, but they've gotten out of the habit of working in our lab, and are used to working on their own in the dorms. I think this is a big loss −− they don't learn anything from talking to each other in the labs anymore.

Ghostview was slow, etc, but even vi was too slow. I am used to typing quickly, and when the cursor can't keep up with me, I can't handle it. I would also have worked at home if I didn't have to be here. And there were those annoying red squares left all over the Xterm window when you were in vi. I had to type ^L every few lines to get rid of them to see what I was typing... The pits. The whole setup made me lose a lot of respect for Sun (although I try to separate the different product lines as much as possible); I also think Sun will not get respect for hyping a product like the JavaStation so strongly, and then just dumping it. I would wonder why anyone would not just dump Sun...

BTW, the JavaStations, now that they are fast, are quite fine. I really like mine, and don't see why they aren't a viable product.

−−Dr. Mark Barnard, Professor at Marquette University (Quoted March 2000) <markb@mscs.mu.edu>

I believe that it was the triple combination of Sun's JavaOS, the Hotjava software, and GraphOn's GoJoe X−connectivity software which ultimately doomed the JavaStation line.

JavaOS was always sluggish in performance for us. It was rated as having one of the slowest Java VMs by a ZDNet Online Magazane review at http://www.zdnet.com/pcmag/features/javaguide/hfgr10.htm . I speculate this was the the main cause of delaying the JavaStation's formal public release to April 1998.

(fair use archive copy of the PC mag review at: http://dubinski−family.org/~jshowto/Files/texts/pcmag_js_jvm_review.txt)

JavaOS also always lagged behind the current Java developer spec (ie running Java 1.0 when Java 1.1 was prevalent, and Java 1.1 when Java 1.2 was issued). It was tough explaining to students why the books they were buying were all using the new event−model of Java 1.1, but they could not program to it and have it run on "the Java machine ". There were also some implementation problems with some of the AWT peers which sometimes made programming across platforms difficult.

These performance and implementation problems were never addressed in subsequent build of JavaOS for the duration we ran it. I believe the last edition we had used a Java 1.1.4 runtime, when we had a Java 1.2 development kit on the server.

The HotJava browser software suffered from not being able to handle web standards HTML4, cascading style−sheets, or the ECMA javascript. All of these standards were employed in commercial sites at the time, resulting in many sites that weren't viewable by the JavaStations. The Hotjava Browser engine also had serious printing problems with certain webpages, some of which appeared on Sun's own website!

The HotJava Views task selector software also was rough. Users could have multiple apps running, but only one displayed at a time. Manipulation of multiple window panes was difficult (no minimization, no quick list to all apps, resizing not always possible). Flexibility users had grown accustomed to was tossed out in favor of this task−selector approach. On Sun's Java website there was a page boasting of a committee formed that decided this was the "right way" to make a desktop. Tell that to our users.

The GraphOn Go−Joe software was by far the most damaging piece of software to the JavaStation line. This was an X−connectivity software Sun licensed from GraphOn to give users access to the Solaris servers' X apps. The connectivity worked via a daemon installed on the Solaris server, which was connected to by a Java connectivity applet on the NC side. This small applet (only about 250K) simply threw up the latest display state and sent back to the daemon the mouse and keyboard strokes of the user. Unlike Xterminals though, the actual Xserver process was spawned and communicated with on the remote server−side by the daemon. Communication between the GraphOn client applet and the server daemon was supposedly done by a patented protocol to compress communication and speed things up. However, the performance of X under Go−Joe was terribly sluggish, with horrible refresh rates (10−seconds for some page scroll refreshes). Many sites operators I spoke to elected to not run the Go−Joe software past a trial period for this reason. We had to run it though, as our users were heavily X dependant. Alternatives like Weird/X were not available at this time, and VNC proved not up to snuff given the slow JavaOS VM.

This performance in Go−Joe alone was enough to give uninformed users the impression that the JavaStation was an underpowered machine, especially when placed side−by−side with the low−cost, end−of−lifed Sun Xterminal 1 hardware it was meant to replace. Our students left labs in droves, faculty were upset, and giving demos to outsiders was downright embarrassing. In reality the hardware was solid and stable, but was hampered by this new, untested OS and new, untested applications running on a new, untested hardware architecture. This triple−threat combination, and Sun's timeline for fixing the problems is what I feel truly doomed the JavaStation.

I remember that in 1998, Sun publicized that it had rolled out 3000 of these machines in−house, including one on Scott McNealy's desk. One who has used the JavaStations with the Sun software would have to wonder whether he ever turned it on and used it solely for a day? Had he done so, I'm sure he'd demand things be done differently. (update Oct. 2001: many ex−Sun employees who've contacted me say they made great doorstops and paper weights.)

Why Sun never ported and released its tried and tested XTerminal software to the JavaStation, or even a mini−Solaris, remained a mystery to us the whole time before we switched to Linux. It was only after we moved to Linux and the JavaStation line was formally killed by Sun when we learned from some inside Sun sources that Solaris actually was ported to Mr. Coffee, but released only internally at Sun. As a heavily invested customer site who had begged for help, this was not only disheartening, but insulting to discover.

Lastly, the customer support we received at the time was horrible. We pled our case on more than a few occassions, but requests always seemed to fall on deaf ears. Calling up SunSolve for JavaStation help always resulted in a transfer to a Java *Language* engineer. If the Sun employees do not know their own products, that's a problem!

From our view, there no doubt was politics involved in this, and as customers, we were the ones to bear the results of this. We continue using Sun equipment when it comes to the proven models like the Enterprise−class servers and disk arrays, but on the latest low−cost desktop offerings, we will be forever cautious given the JavaStation history.

Linux now proves the JavaStations are adequate machines, and Sun could take this bait and go with it. If they sell the JavaStations for $250 a piece and the JavaStation running a

> proven OS like Linux (or Solaris) with proven apps (X), the JavaStation makes for a great network appliance. The recent NetPliance I–Opener Linux hack and subsequent controversy proves there certainly is a market for this type of low–cost device. (Oct. 2001 addition: After the publishing of the Linux hack, NetPliance made their new hardware unhackable, and subsequently ran out of business. The demand for cheap diskless stations still exist. Today's hackable units are set–top receivers and failed internet toasters like the 3Com Audrey)
>
> ––Robert Dubinski, former Computer Systems Technician at Marquette University (Quoted March 2000) <rsd@dubinski–family.org>

More comments and rebuttal statements by Sun employees are always welcome.

(update Oct 2001): A year and a half of this document's existance and not a single rebuttal statement by Sun. There were a couple initial requests to omit this section, but I refused. After all, imagine a new reader who never saw a JavaStation before: They'd read to this section, think "Wow, what a great little machine..let me get one!", and then ask themselves, "If it did all this, why don't they make them anymore?". The bad must be included with the good, and to leave this section out is a disservice to all the users who suffered through the poor software and support during the official lifetime of the JavaStation. This section, therefore, is a necessity, and although this document is licensed under the GNU Free Documentation License, the eagle–eyed reader will note that this section has been labeled as "invariant" to protect it from entities who may wish to bury it (which is precisely the reason why the Invariant clause of the GFDL exists).

2.6. Where to Purchase a JavaStation

Since Sun has canceled production of the JavaStation line, it no longer sells them through their official channels. Sun contacts have informed me that all internal JavaStation stock was cleaned out and dumped in 2000. Therefore, All JavaStations are now found out in the wild.

Your best bet to get JavaStations though is out on the open market. Educational institutions which received a handful from Sun as demo units are now trying to offload them any way they can (too bad they don't read this HOWTO). Search around the auction sites like Ebay and Yahoo Auctions, and you should be able to turn some up.

A great resource for JavaStations used to be "Bodoman's JavaStation site" at: http://www.bodoman.com/javastation/javastation.html. Sadly, as of October 2001, the domain bodoman.com seems to no longer resolve. Ebay may now be your best bet.

Mr. Coffee is the most widespread JavaStation model, and has tended to sell around $30–80US consistently for the last year or so.

Krups models more rare and sell at higher prices, probably because the stylish case still stands out today. Prices on Ebay are always over $100, but for Oct. 2001, their technology is definitely no longer worth that much. A good price would be $80–85US. Many reports have come from the UK telling of many Krups models getting dumped there.

The Dover models were a very hush–hush thing when this HOWTO was initially published, but the secret is out: if you want one, go to South Africa. Dovers seemed to have been dumped there en masse. Pricing is unknown, but should be comparable to a Cyrix–266 PC clone.

The Espresso and JavaEngine models are near impossible to find, so if you get one, consider yourself lucky. If you have a Fox, well, you're just too cool. Pricing for these models is likely a premium. (>$100US).

3. Background Requirements for Linux on a JavaStation

This chapter describes the base hardware and software requirements for enabling Linux on the JavaStation .

3.1. Complete Hardware Requirements

For hardware, you will need one or more JavaStation clients and a server to feed it its Linux image from, all networked on the same net segment.

This server you use can be any server which supports DHCP and TFTP, and RARP. These are the base protocols needed to perform a network boot of the JavaStations. You may also need NFS service as well, but it is not necessary in one type of configuration this HOWTO describes. Also, you can get by without RARP on both the Krups and Espresso models.

This document will describe how to set up serving the network Linux OS image to the JavaStation from a Sun server running SparcLinux. While you do not need a Sun server to serve your Linux image off of, a Sun SparcLinux server is recommended should you wish to compile a kernel of your own, or prototype a new filesystem for your JavaStations to use. Otherwise, you will need to use prepackaged kernels and filesystems somebody else has pre−built and made publicly available for use. (You might also use a cross−compiler to produce the kernel images, but prototyping a filesystem is best done on a Sun SparcLinux server.)

Reports of successful boot servers used include Sun boxes running Sparclinux, Sun boxes running Solaris, and PCs running MS Windows. It is only when you are building a new kernel or filesystem that a Sun box running Linux becomes valuable.

Your network can be a simple 10 Mbps ethernet LAN, but when you begin using more than 50 JavaStations at once, a switched 100 Mbps network becomes desirable for your server to handle multiple concurrent boot requests.

This HOWTO includes pointers to example kernels, filesystems and a complete out−of−the−box solution for you to use, eliminating your need for a Linux/SPARC server, but you still need a server of some type to feed the image to the JavaStations as they boot.

3.2. Network Service Requirements

As discussed in the last section, the JavaStation boot cycle will make use of DHCP and TFTP with possibly NFS and RARP. To understand why, read up on the JavaStation boot sequence in the next section.

3.3. Understand the JavaStation Boot Sequence

The JavaStations follow a typical diskless workstation boot sequence.

When powered on, the JavaStation sends out a broadcast request for its IP. It gets its IP info via RARP or DHCP. With a DHCP response, it gets information about the network it is on and where to go download its boot image from via TFTP.

There are subtle variations in diskless boots from one diskless machine to the next. For instance, BOOTP may sometimes be substituted where DHCP is, and RARP may be eliminated in favor of either of the two. But in general, the sequence is typically the same between the client and the server:

1. C: "Who am I?"
2. S: "You are xxx"
3. C: "Where do I go for my boot image?"
4. S: "You go here."
5. C: "Give me my image from here...Please?"
6. S: "Here's your image."

After the kernel is finished loading, your diskless client typically mounts its root filesystem from the network via NFS. Alternatively, it may load and mount it from a RAMdisk.

The original JavaOS and Hotjava Views environment, when run on a JavaStation, required the setup and maintenance of the core services above, plus also NIS, HTTP, DNS, POP, and NTP servers. If setting up a JavaStation boot server seems like a lot of work, imagine adding these extra services into the mix too.

3.4. Additional Software Requirements: Replacement Firmware (PROLL)

JavaStations came with two different PROMs installed in them. Version 2.30 shipped with the earliest Mr. Coffee models, and was updated by latter versions of the Sun Netra J software environment to 3.11. Krups and Espresso came with 3.x versions of the PROM by default.

It turns out the later 3.x series of PROMs is not conducive to booting Linux upon. Fortunately, a complete PROM replacement called PROLL now exists to get by this limitation.

PROLL becomes the first image your JavaStation grabs by TFTP. It then will load your true kernel image and boot into Linux .

No matter what PROM revision you have, get PROLL. This can make troubleshooting new installs easier.

The current, master version of PROLL is available from: http://people.redhat.com/zaitcev/linux/.

The current version at the time of this writing is "14".

PROLL can also be found mirrored on "VGER ", and also on this HOWTO's distribution site at: http://dubinski−family.org/~jshowto/Files/proll/proll_14.tar.bz2 (HOWTO website mirror − version 14)

3.5. Decide on your Filesystem−type: NFS−Root, or Embedded?

Before you begin, you must decide upon the root−filesystem type you wish to use for your diskless JavaStation. There are two possibilities.

3.5.1. "NFS−Root" Filesystem

In this setup, after the boot kernel is retrieved off the network, the running JavaStation makes an NFS connection for its root filesystem. The root directory "/" is mounted off the network for the duration of the current session.

The "NFS−Root" solution is the recommended way to go for beginners, as it is easier to troubleshoot if there are problems. It also makes it easier to prototype the proper filesystem, as any changes you make on a running system can be propogated for the next boot cycle (so long as you are in read−write mode, of course).

Drawbacks of this type of system is increased network activity as the running JavaStations locate and execute files, plus file organization in large environments.

3.5.2. "Embedded−Root" Filesystem

In this setup, the root filesystem is loaded directly into RAM and accessed from there.

The advantage of this setup is that there is no NFS traffic to worry about, resulting in a clean solution.

The disadvantage of this configuration is that you can no longer do rapid prototyping of your filesystem, as any changes you make to a running system are lost. If you have no "NFS−Root" setup available, you develop an embedded filesystem by making small tweaks and performing reboots to test. Other disadvantages include the requirement of fitting the full filesystem in available RAM; due to a limitation of PROLL, this requirement is much lower on JavaStations than expected. Still, embedded root is the way to go for the cleanest environment.

First time users will want to set up an "NFS−Root" configuration. When you have things stabilized, move to "Embedded−Root" to take use of its advantages.

3.6. Support Sites to Check Out: Zaitcev's Linux Site

One website to keep on reference when you begin thinking about putting Linux on your JavaStation is kernel hacker Pete Zaitcev's website at: http://people.redhat.com/zaitcev/linux/, referenced throughout this document as the "ZLS" site (short for "Zaitcev's Linux Site"). Here you will find the latest version of PROLL and many low−level details about dealing with the JavaStations. Many items on the ZLS have been merged into this document, but not all.

Oct. 2001 update: It is in your best interest to review all the information on Pete's site, in this document, and references pointed to, before diving in and setting up your JavaStation with Linux. Almost all questions people have had in setting up their systems are covered in the materials presented.

4. Build Your Kernel

4.1. Before you begin

This chapter assumes you wish to compile your own Linux kernel for the JavaStation. If this is something you can not do, there are sample kernels pointed to at the end of this chapter.

This chapter assumes you already know how to compile Linux kernels in general, perhaps on a PC, a SPARC server running Linux, or any of the other Linux ports. If not, read the Kernel−HOWTO and the README file of your kernel source.

Compiling a kernel for a JavaStation is not much different than compiling a Linux kernel elsewhere. You just need to know the right options to pick. In general, you're compiling for a Sun4M class architecture, and enabling JavaStation−specific options. The following sections in this chapter will take you through the steps.

While it may be possible to compile the JavaStation −enabled kernel on alternate platforms by way of a cross−compiler, this HOWTO assumes you will do it on a Linux/Sparc based server running in 32−bit mode. Cross−compiling will not be covered, and questions regarding it will not be entertained.

4.2. Make sure you use 32−bit mode

When compiling your own JavaStation−capable kernel on a Sun server, you need to make sure the machine you work on is set to 32−bit mode. So, if you're on an Ultra−class machine, be sure to first switch to 32−bit mode before you begin compiling.

To check what mode you're in, do a **`uname -a`**. If it says "sparc", you're in 32−bit mode and don't have to do anything. If it reports "sparc64", then you should perform a **`sparc32 bash`** first to switch to 32−bit mode. A subsequent **`uname -a`** should reflect the change.

4.3. Supported Linux Kernel Versions

The kernel source revision you should use depends both on which model of JavaStation you have, and which series kernel you are using. The current "stable" series of Linux kernels is 2.4.x, but as we will read in a minute, this may not be the best bet to use.

First, a few note on the 2.2.x and 2.3.x series. Mr. Coffee has had kernel support since about kernel version 2.2.5, and definitely works out of the box with the RedHat 6.0+/SPARC distribution kernels. Krups support did not work well out of the box until the latter 2.3.x kernel cycle. Krups support was added in the early 2.3.x sequence, but the MMU changes to the 32−bit SPARC kernel had kept it from compiling cleanly until later on.

Kernels for both Mr. Coffee and Krups compiled cleanly by the HOWTO author with the Mar. 17, 2000 CVS kernel, and are included in the Sample Kernels section.. Krups support was backported into the 2.2.x kernels (where x>15). The latest 2.2.x kernel "should" compile cleanly for the Mr. Coffee and Krups models, but your mileage may vary.

Now onto the 2.4.x series.

The only kernel which has been tested and compiles cleanly for Mr. Coffee and Krups is version 2.4.2. All other versions are broken or require a patch.

The reason for this is that the sparc32 branch of the kernel has not had an active maintainer for many months. Some are contributing fixes, but without an active maintainer things go slow.

There is another reason to be weary of the 2.4.x series. From 2.4.0 through 2.4.9, the VM of the kernel was found to be inadequate under heavy loads, and was subsequently replaced in 2.4.10+. This was a big change for the so−called "stable" series of kernels.

To add further insult to injury, there have been security flaws detected in all of 2.2.x kernel series and up through 2.4.12. This is patched in pre−2.2.20 and 2.4.12+. As of this writing, 2.4.12+ has not been checked by the author as functioning on the JavaStations.

So basically, it has been a crap−shoot over which kernel to choose. Try a few until you find one that suits you best.

If you can not get a kernel to compile, or wish to avoid the headache or trying, you may try the samples pointed to by this document.

4.4. Required Kernel Configuration Options

When you do your **`make config`** command to enter the kernel configuration stage, there are a few things you are required to enable. Note that the following option names are from a 2.2.x kernel, and may be slightly different on a 2.4.x series kernel. If in doubt, check the sample files later in the chapter.

For all JavaStations, you want to enable PCI support:

```
CONFIG_PCI=y
```

Don't forget your mouse:

```
CONFIG_BUSMOUSE=y
CONFIG_SUN_MOUSE=y
```

You'll want video, done with the Linux framebuffer interface:

```
CONFIG_FB_TCX=y (for Mr. Coffee)
CONFIG_FB_PCI=y
CONFIG_FB_IGA=y (for Krups/Espresso)
```

Audio is done with the Crystal Audio 4231 chipset:

```
CONFIG_SPARCAUDIO=y
CONFIG_SPARCAUDIO_CS4231=y
```

Don't forget your network interface:

```
CONFIG_SUNLANCE=y (Mr. Coffee)
CONFIG_HAPPYMEAL=y (Krups/Espresso)
```

You'll no doubt need to support a filesystem:

```
CONFIG_EXT2_FS=y
```

You'll want IP autoconfiguration, and RARP/BOOTP support:

```
CONFIG_IP_PNP=y
CONFIG_IP_PNP_BOOTP=y
CONFIG_IP_PNP_RARP=y
```

When doing the "NFS−Root" filesystem configuration, you will need both NFS and NFS−Root support:

```
CONFIG_NFS_FS=y
CONFIG_ROOT_NFS=y
```

When doing the "Embedded−Root" filesystem, configure both RAM disks and "initial ramdisk" support:

```
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_INITRD=y   (for 2.4.x, also configure size of ramdisk)
```

You can get a working ".config" file which has the required options set later in this chapter.

4.5. Necessary Patch for "Embedded−Root" FS Configurations

If you have decided to go with the "Embedded−Root" filesystem option, you will want to make a patch to the RAMdisk driver source first.

The default size of a RAM disk when using the RAM disk driver is 4 MB. Chances are that you will want an embedded filesystem of more than that size, particularly when you start thinking about running an X server, or including a Java runtime.

You can do this for 2.2.x kernels by a manual edit yourself, or by using the patch pointed to below. The change is a one−line edit in the file <LINUXROOT>/drivers/block/rd.c . Look for a line that says:

```
int rd_size = 4096; /* Size of the RAM disks */
```

and change it to the size of the RAMdisk you wish. Typically, most embedded systems are under 16 MB, so a common edit is to change the line to:

```
int rd_size = 4 * 4096; /* Size of the RAM disks */
```

If you can not do this, the patch below makes the edit for you.

4MB to 16MB kernel patch file is at: http://dubinski−family.org/~jshowto/Files/patches/ramdisk_patch.txt

Kernels in the 2.4.x series allow you to select the amount of RAM as a configuration option. The patch is no longer needed for those kernels.

It should also be noted in this section that there is currently a limit on the size of Linux boot image for all JavaStation models, due to the implementation of PROLL. This limit is technically 8 MB. This topic is mentioned again in the "Questions and TroubleShooting" section of this document.

4.6. Build the JavaStation−Ready Kernel

To build the kernel, you type **`make vmlinux`**. If you come from an x86 Linux background, you might be surprised that you do not perform a **`make bzImage`** or **`make zImage`**. Do not be alarmed: this command is correct.

When the compile is finished, you will find a file named "vmlinux " in the kernel source root directory. You are almost ready to put this kernel to use.

4.7. Convert Kernel from ELF to a.out format

You need to make one more change to your kernel before it is ready for use. You need to convert it from ELF to AOUT executable format. You can do this with the "elftoaout " utility included in most Linux/SPARC distributions.

To convert your kernel image to the AOUT executable format, you issue the command:

```
elftoaout -o vmlinux.aout vmlinux
```

You will probably now want to rename the image file to a longer name which includes the current date and kernel revision you used, so as not to get confused with when you have multiple boot kernel images down the road.

The elftoaout program should come with your SparcLinux distribution. If not, try VGER or your favorite kernel mirror.

4.8. JavaStation−Ready Kernel Images, System.map and .config File Samples

Here are some sample ".config" and JavaStation −ready kernel images. They were prepared and donated to help get you up−to−speed quickly.

Warning: Some of these kernel images are considered out of date, and should be avoided in a production environment. It is up to you to decide how much of a liability you feel running them holds. The document author and kernel contributors cannot be held liable for any damage caused by the use of these kernels. They are provided with absolutely no warranties.

If for some reason you have troubles downloading, try holding left−shift on your browser as you click the link. Kernel images are compressed with bzip2 compression. They must be uncompressed before use. Kernel images are already converted to a.out format.

If you mirror these files, or can verify they work on a machine not yet confirmed, *PLEASE* email me so I can add your information here.

4.8.1. 2.3.99pre3_embedded_RSD

.config (md5sum c59329ceb2e831f2502c1e410ece141c):
http://dubinski−family.org/~jshowto/Files/kernels/2.3.99pre3_embedded_RSD/config__2.3.99pre3_embedded_RSD.txt

kernel (md5sum 8e8d28b13961b92e3f95e4ba98f6f319):
http://dubinski−family.org/~jshowto/Files/kernels/2.3.99pre3_embedded_RSD/vmlinux__2.3.99pre3_embedded_RSD.bz2

System.map (md5sum 43205a86fcb0b16ecae7313d38fcbb2c):
http://dubinski−family.org/~jshowto/Files/kernels/2.3.99pre3_embedded_RSD/system.map__2.3.99pre3_embedded_RSD.

Description:

This kernel is donated by Robert Dubinski. It was used at Marquette University to build an embedded root boot image. This is based off of the Mar. 17, 2000 CVS kernel. It includes support for both Mr. Coffee and Krups machines.

Tested on Mr. Coffee: YES

Tested on Krups: YES

Tested on Espresso: NO

4.8.2. 2.3.99pre3_nfsroot_RSD

.config (md5sum e715370346ac298555dd7ce099c8f80a):
http://dubinski−family.org/~jshowto/Files/kernels/2.3.99pre3_nfsroot_RSD/config__2.3.99pre3_nfsroot_RSD.txt

kernel (md5sum fd141e8e8f639df67427d5ecd0ecba76):
http://dubinski−family.org/~jshowto/Files/kernels/2.3.99pre3_nfsroot_RSD/vmlinux__2.3.99pre3_nfsroot_RSD.bz2

System.map (md5sum fd141e8e8f639df67427d5ecd0ecba76):
http://dubinski−family.org/~jshowto/Files/kernels/2.3.99pre3_nfsroot_RSD/system.map__2.3.99pre3_nfsroot_RSD.txt

Description:

This kernel is donated by Robert Dubinski. It was used at Marquette University to prototype a filesystem. This is based off of the Mar. 17, 2000 CVS kernel. It includes support for both Mr. Coffee and Krups machines.

Tested on Mr. Coffee: YES

Tested on Krups: YES

Tested on Espresso: NO

4.8.3. 2.4.2_embedded_RSD

.config (md5sum dd1a9dd2e92b9b175b7ba747c94edca7):
http://dubinski−family.org/~jshowto/Files/kernels/2.4.2_embedded_RSD/config__2.4.2_embedded_RSD.txt

kernel (md5sum 5a1592b7e0a37909ae16374296a7070e):
http://dubinski−family.org/~jshowto/Files/kernels/2.4.2_embedded_RSD/vmlinux__2.4.2_embedded_RSD.bz2

System.map (md5sum 1de202e0fab7a9e661bebc80255605b7):
http://dubinski−family.org/~jshowto/Files/kernels/2.4.2_embedded_RSD/system.map__2.4.2_embedded_RSD.txt

Description:

This kernel is donated by Robert Dubinski. It is a demonstration kernel for the 2.4.x series, and has not been tested...yet. It includes support for both Mr. Coffee and Krups machines.

Tested on Mr. Coffee: NO

Tested on Krups: NO

Tested on Espresso: NO

4.8.4. 2.4.2_nfsroot_RSD

.config (md5sum cabd1d98613ad169b372666b7eaa869b):
http://dubinski−family.org/~jshowto/Files/kernels/2.4.2_nfsroot_RSD/config__2.4.2_nfsroot_RSD.txt

kernel (md5sum c24f42f72c58920c00ac7ff7aaffadde):
http://dubinski−family.org/~jshowto/Files/kernels/2.4.2_nfsroot_RSD/vmlinux__2.4.2_nfsroot_RSD.bz2

System.map (md5sum 6af2b374c7d3fc3f97d48ab71b335062):
http://dubinski−family.org/~jshowto/Files/kernels/2.4.2_nfsroot_RSD/system.map__2.4.2_nfsroot_RSD.txt

Description:

This kernel is donated by Robert Dubinski. It is a demonstration kernel for the 2.4.x series, and has not been tested...yet. It includes support for both Mr. Coffee and Krups machines.

Tested on Mr. Coffee: NO

Tested on Krups: NO

Tested on Espresso: NO

4.8.5. Outside kernel mirrors

Other sites mirror the kernels here or other kernel samples. Here are a few known sites as of Oct−31−2001:

1. ftp://atari−source.com/pub/javastation/
2. http://people.redhat.com/pjones/krups

5. Build A JavaStation−Ready FileSystem

This chapter describes how one constructs a filesystem suitable for use on the Linux−running JavaStations .

5.1. Preparing Yourself to Build Your Own Filesystem

Building a filesystem for use with the JavaStations is a time−consuming, but rewarding task for those who undertake it. You will learn more about library dependencies than you ever thought you could, all the time while trying to keep the overall image size as small as possible.

WARNING: This is not an easy task. Creating a lasting filesystem is not for novices. If you seriously consider undertaking this step, prepare to budget a bit of time to get things just right, particularly if you plan to make an embedded−root filesystem which fits in the 8MB limit. You have now been properly warned.

There are two common approaches one can take when rolling a new JavaStation−ready filesystem.

1. Start with an established distribution's filesystem and whittle down to the core.
2. Start with an established distribution's "rescue disk" filesystem and add desired functionality.

Which path you take, of course, is entirely up to you. The "rescue disk" build procedure seems to work best though, as more base commands in a rescue disk are statically linked, increasing the starting image size but causing less initial library headaches. Commands included on a rescue disk also happen to be bare−bones, with many extraneous options not compiled in.

Obviously when building a filesystem in the context of the JavaStation, you will be basing off of an existing Linux/SPARC filesystem. The filesystems that come with the RedHat, SuSE or Debian distributions are good starting points.

Warning
In the future, you will also need to make sure you base off a filesystem built with compiled 32−bit mode executables, as a 64−bit userland project is presently in progress for 64−bit SPARC Linux kernels. As of Oct. 2001, this is still a ways away, but it is being mentioned now for the future.

5.2. Contents of the "/etc/fstab" File

The configuration lines placed into "/etc/fstab" depend on whether you will be using the "NFS−Root" or "Embedded−Root" filesystem configuration.

5.2.1. "NFS−Root" Filesystem fstab

Here is an example of an "/etc/fstab" for an "NFS−Root" boot option.

```
###
#
your.nfs.server:/path/to/filesystem  /  nfs defaults,rsize=8192,wsize=8192 1 1
#
none                    /proc                   proc    defaults        0 0
###
```

5.2.2. "Embedded−Root" Filesystem fstab

Here is an example of an "/etc/fstab" for an "Embedded−Root" boot option.

```
###
#
/dev/ram /     ext2  defaults
#
/proc    /proc    proc  defaults
###
```

5.3. The "Embedded−Root" Image Creation Procedure

Prepping up the "Embedded−Root" boot image requires a number of extra steps. Due to these extra steps, the "NFS−Root" filesystem option is recommended for beginners to Linux on the JavaStation. You might also try the samples pointed to in this document. Should you still wish to build and embedded image on your own, this section outlines the basic instructions.

Creating the "Embedded−Root" boot image is a 5−Step Procedure:

1. *Prototype Your Filesystem*

   This whole chapter deals with rolling your own filesystem. In this step, it is assumed you create your own filesystem, perhaps by prototyping one on a working "NFS−Root" filesystem configuration.

   One thing to keep in mind is that unlike your "NFS−Root" filesystem, the "Embedded−Root" filesystem must fit within the confines of your allocated RAMdisk, generally 4−16 MB. Your maximum size is dependant on the setting of the RAMdisk driver.

2. *Create an Empty File for Your FileSystem*

   You now need to create a file−based filesystem "container". This is just a file that is the size of your RAMdisk.

   To create this, try the **dd** command:

   ```
   dd if=/dev/zero of=./fs_test.img bs=1k count=8000
   ```

   Using this example, you now should have an 8 MB file named "fs_test.img". Note: Be *sure* the count you use matches the RAMdisk size you allocated for in the kernel's RAMdisk driver!

3. *Format your Filesystem "Container"*

   Now that you have a "container" for your filesystem, it is time to format it and place a bare filesystem on it.

   In our kernel phase, we added in support for the ext2 filesystem. We'll now format our "container" with this filesystem type.

   ```
   mkfs.ext2 ./fs_test.img
   ```

   Ignore any warnings about the file not being a block device, and proceed anyway. This is an expected warning message.

4. *Mount the Filesystem "Container" and Write to It*

   Now that you have your filesystem container, you can mount it and load your prototyped filesystem on it.

   To mount the container, use the kernel loopback device. Make sure your server's kernel has loopback support enabled and issue a:

   ```
   mount -o loop ./fs_test.img /mnt
   ```

   Copy your files to the filesystem, and make sure "/etc/fstab " has the RAMdisk entries as described elsewhere in this document.

   To avoid symbolic links being changed into actual copies of files, use a copy tool like "tar" or "cpio" instead of a "cp".

5. *Unmount and Compress the Root Filesystem*

   Unmount the root filesystem you just created.

   ```
   umount /mnt
   ```

   Compress the filesystem file with maximum "gzip" compression levels.

   ```
   gzip -v9 ./fs_test.img
   ```

   You should now have "fs_test.img.gz" file.

6. *Hook the Root−Filesystem Onto the Back of Your Kernel Image*

   Now you must append the filesystem image onto your kernel.

   You do this with a utility program called "piggyback". The piggyback program takes care of the task of appending the two and letting the kernel know where both it and the filesystem begins and ends.

   The "piggyback" program is found in your kernel source tree under <LINUXROOT>/arch/sparc/boot. It might also be found on your favorite ftp.kernel.org site.

   For piggyback to work, it needs your AOUT format kernel image, the System.map file from your kernel source root directory, and the compressed root−filesystem you just created.

   We put it all together with a:

   ```
   piggyback vmlinux.aout System.map fs_test.img.gz
   ```

   Be sure to backup your kernel image first, as piggyback used the same "vmlinux.aout" filename for output. Check the filesize of your "vmlinux.aout" file after giving this command and you can verify the filesystem has indeed been appended.

Congratulations! You've created an "Embedded−Root" kernel/filesystem boot image.

5.4. Sample FileSystems

Here are some sample filesystems for you to start with. They have been contributed by various JavaStation users.

Warning: Some of these filesystem images may be considered out of date, and should be avoided in a production environment. It is up to you to decide how much of a liability you feel running them holds. The document author and filesystem contributors cannot be held liable for any damage caused by the use of these files. They are provided with absolutely no warranties.

5.4.1. jsroot_varol_19991221

filesystem (md5sum 450669bc5f3f8a4006fdc75471c0454b):
http://dubinski−family.org/~jshowto/Files/filesystems/jsroot_varol/jsroot_varol_19991221.tar.bz2

Description:

This image, created by Varol Kapton <varol@ulakbim.gov.tr>, was based on RedHat 6/SPARC. It has the Xfree 3.3.5 framebuffer server dated 19990823, but only works with Krups. If you are working with a Mr. Coffee unit, you must substitute the other X server discussed later in this HOWTO.

As the network settings included are configured for Varol's network, you must first mount this image, and edit /etc/hosts and /etc/resolv.conf accordingly.

Confirmed OK: YES

Good for Mr. Coffee: YES

Good for Krups: NO

Good for Espresso: NO

5.5. Sample X Servers

One of the most frequently asked questions users have is where to get an X server from. Here are some sample X servers for you to start with. They have been contributed by various JavaStation users.

Warning: Some of these files may be considered out of date, and should be avoided in a production environment. It is up to you to decide how much of a liability you feel running them holds. The document author and filesystem contributors cannot be held liable for any damage caused by the use of these files. They are provided with absolutely no warranties.

5.5.1. XF86_FBDev_3.3.3.1_19990104

X server (md5sum 88b49bbbfa1c36a5049b62b44c54ed81):
http://dubinski−family.org/~jshowto/Files/xfree/XF86_FBDev_3.3.3.1_19990104.bz2

XF86Config file (md5sum d9fa291efbd178812b3bd253dffb1893):
http://dubinski−family.org/~jshowto/Files/xfree/XF86Config_FBDev_3.3.3.1_19990104.txt

Description:

This is a server for XFree 3.3.3.1 with support for the framebuffers of Mr. Coffee and Krups.

Confirmed OK: YES

Good for Mr. Coffee: YES

Good for Krups: YES

Good for Espresso: NO

5.6. Outside Sample Filesystems

Of course, other filesystems and tools exist outside this document, and have been used by JavaStation users. Here are a few files that were reported on the sparclinux mailing list as having been used.

1. http://busybox.lineo.com (a single executable which has dozens of common unix tool functions built in)
2. http://www.ultralinux.org/js (Jim Mintha's filesystems)

6. "Out of the Box" JavaStation Boot File Solutions

This chapter is for administrators who have neither the time nor energy to put together files as described in the previous two chapters, or just want to see Linux on a JavaStation rapidly. In this chapter are complete kernel+embedded−root−fs solutions for to try.

The images here perform simple (or meaningless) tasks, but are useful for verifying a server configuration. Configuring the boot server is covered in the next chapter.

6.1. Simple Solution #1

This is simple solution #1, or at least it will be when it gets created. Right now this section is a stub for the future files.

7. Set up Your Server

This chapter describes the configuration steps necessary for the server machine to hand−off your JavaStation boot image.

7.1. Preface

It is now time to setup your server to deliver the OS and filesystem to the JavaStation.

In our examples here, we configure a Linux/SPARC server "lnxserv " at private IP 192.168.128.100 to deliver a boot image to JavaStation "java01" at private IP 192.168.128.1. Both are on private network 192.168.128/24. When using an "NFS−Root" Filesystem, the location on the server of the filesystem in our sample is at "/path/to/nfsroot ".

7.2. Setting up the RARP service

We first need to set up RARP service on our server, so the JavaStation can auto−configure its IP.

First, populate the "/etc/ethers" file with the mapping of the mac address of the JavaStation to its hostname:

```
### /etc/ethers
8:0:20:82:7a:21  lnxserv  # 192.168.128.100 (server is not necessary,)
#                         #                 (just for completeness)
#
#
08:00:20:81:C2:ae java01   # 192.168.128.1 (JavaStation)
#
###
```

Next, populate the "/etc/hosts" file with the IP to hostname maps:

```
### /etc/hosts
192.168.128.100 lnxserv
192.168.128.1 java01
###
```

Lastly, configure the RARP cache to fill. On 2.2.x based systems, you do this with the /sbin/rarp command, so fill the cache at startup:

```
### Part of rc.local
#
# If necessary, first load the rarp module to be able to fill the cache.
# /sbin/insmod rarp
#
# Now we fill the rarp cache.  You better have the rarp command available.
if [ −f /sbin/rarp ]; then
        /sbin/rarp −f
fi
###
```

On 2.4.x based systems, you must use the userland RARP daemon to answer RARP requests instead.

7.3. Setting up the DHCP service

You now need to configure your server to deliver DHCP service. This will help identify the JavaStation, the network it is on, and where to get its boot image from.

The following is a sample "dhcpd.conf" file for the ISC DHCP server software which ships with most Linux/SPARC distributions.

```
### Sample /etc/dhcpd.conf file for ISC DHCPD
#
deny unknown-clients;
#
subnet 192.168.128.0 netmask 255.255.255.0
{
   range 192.168.128.1 192.168.128.150;
}

group
{
   host java01
   {
      hardware ethernet 08:00:20:81:C2:ae;
      filename "C0A88003";        # "/tftpboot/xxx"
      fixed-address java01;       # 192.168.128.1
   }
}
#
### End dhcpd.conf file
```

A longer dhcpd.conf from the ZLS is mirrored here for demonstration purposes.

Note: Some early versions of ISC DHCPD are reported to not work well. It is recommended you use ISC DHCPD Version 2.0 and above. If you still find youself having problems, there is a patch to the ISC DHCP server on the ZLS website.

7.4. Set up NFS service ("NFS−Root Options" Only)

When you are serving up an "NFS−Root" filesystem, you need to share the filesystem you created to the JavaStation client. You do this with the "/etc/exports" file.

```
###/etc/exports
/path/to/nfsroot        java01(rw,no_root_squash)
###
```

Be sure your NFS server gets properly started up at boot−time.

7.5. Setting up for Boot with TFTP

Now we need to set up the last step on our server: the TFTP configuration. For this step, you will need the kernel you created (using the "NFS−Root" option) or the piggybacked kernel/fs boot image (using the "Embedded−Root" option), the appropriate PROLL, and some knowledge of hexadecimal numbering.

The first thing you need to do is verify that "TFTPd" is enabled in your "/etc/inetd.conf" file:

```
tftp    dgram   udp     wait    root    /usr/sbin/tcpd  in.tftpd
```

Now, you move your copy of proll for your JavaStation architecture, along your kernel or piggybacked kernel image to /tftpboot.

Now, you create of symbolic link from the hexidecimal version of your IP to your PROLL image, and a map from "HEXIP.PROL" to your real kernel image. If you are using "Embedded−Root" option, you point to your "Embedded−Root" Filesystem plus Kernel image. If you are using the "NFS−Root" option, you need to point to the normal "vmlinux.aout" image, plus have a separate map of IP−>nfsroot location. For sake of completeness, you might also want a "HEXIP.SUN4M" −> "HEXIP " map, as that is the custom way of dealing with net boot situations with the Sun.

Example for java01 booting from "NFS−Root":

```
$ ls -ld /tftpboot
-rw-r--r--   1 root     root        89608 Mar 20 10:15 proll.aout.krups.11
-rw-r--r--   1 root     root        52732 Mar 17 11:52 proll.aout.mrcoffee.11
lrwxrwxrwx   1 root     root           19 Mar 20 10:16 proll.krups -> proll.aout.krups.11
lrwxrwxrwx   1 root     root           22 Mar 17 11:54 proll.mrcoffee -> proll.aout.mrcoffee.11
lrwxrwxrwx   1 root     root           10 Apr  1 13:00 C0A88001.SUN4M -> COA88001
lrwxrwxrwx   1 root     root           10 Apr  1 13:00 C0A88001 -> proll.mrcoffee
lrwxrwxrwx   1 root     root           12 Apr  1 13:00 C0A88001.PROL -> vmlinux.aout
-rw-r--r--   1 root     root      1456189 May 21 12:53 vmlinux.aout
-rw-r--r--   1 root     root      6743821 Apr  1 12:53 vmlinux_embed.aout
lrwxrwxrwx   1 root     root           18 Apr  1 12:53 192.168.128.1 -> /path/to/nfsroot
```

Example for java01 booting from "Embedded−Root" boot image:

```
$ ls -ld /tftpboot
-rw-r--r--   1 root     root        89608 Mar 20 10:15 proll.aout.krups.11
-rw-r--r--   1 root     root        52732 Mar 17 11:52 proll.aout.mrcoffee.11
lrwxrwxrwx   1 root     root           19 Mar 20 10:16 proll.krups -> proll.aout.krups.11
lrwxrwxrwx   1 root     root           22 Mar 17 11:54 proll.mrcoffee -> proll.aout.mrcoffee.11
lrwxrwxrwx   1 root     root           10 Apr  1 13:00 C0A88001.SUN4M -> COA88001
lrwxrwxrwx   1 root     root           10 Apr  1 13:00 C0A88001 -> proll.mrcoffee
lrwxrwxrwx   1 root     root           12 Apr  1 13:00 C0A88001.PROL -> vmlinux_embed.aout
-rw-r--r--   1 root     root      1456189 May 21 12:53 vmlinux.aout
-rw-r--r--   1 root     root      6743821 Apr  1 12:53 vmlinux_embed.aout
```

8. Booting Your JavaStation

Once you've selected or built your boot files to use, and configured your boot server to serve them, it is time to boot your JavaStation with Linux!

8.1. What to See When Booting Linux

There are multiple stages to the JavaStation boot cycle. What you see on screen can give you clues as to whether things are going well or not. Therefore, it is vital you become familiar with these boot stages, so you can troubleshoot problems rapidly.

8.1.1. Stage 1: White Screen

When you first boot, your JavaStation will start up with a white background screen and black−text PROM banner on top. You will also see a black "exclamation mark in triangle" warning logo. This means the system doesn't yet know who it is, and begins sending RARP/BOOTP requests.

If you do not get this white screen, there is something wrong with your JavaStation. Check all connections, particularly your keyboard, monitor and mouse cords. If the JavaStation does not detect a keyboard or mouse, it thinks it is being booted into a serial console, and will not use the monitor (or keyboard/mouse). In exceptional cases, you may need to reset a jumper if the unit has been set to always boot into the serial console.

8.1.2. Stage 2: Coffee Cup Logo

When contact is made with the DHCP server, the logo goes away and changes to the Java coffee cup logo. The screen is still white background. The logo should be solid, and not blinking. This step lasts a second or two, as the unit should already be contacting the TFTP server and downloading the boot image.

If your coffee logo is blinking, it means there is a problem getting your IP address, usually due to a DHCP leasing problem. Check your DHCP server's logs to see that your JavaStation's mac address is being sent a proper IP address.

If your JavaStation doesn't even get a blinking coffee cup, check both your DHCP and RARP server settings. Also, if running the ISC DHCP server, you may be having a problem with 1514−byte packets and need the patch from the ZLS website.

8.1.3. Stage 3: A Window to PROLL

After the coffee cup logo is solid a few seconds, a white−text on black background window opens. This is the PROLL window. It'll show status of the TFTP download in progress, and when finished will give stats on the size of the file downloaded. The size should match your completed file. When finished, the screen should read 'Booting Linux'. Although, this goes so fast you may not see it.

If you don't see the PROLL window open, confirm your TFTP settings are correct. Also, verify you are pointing to a version of PROLL specific to your JavaStation model. In other words, PROLL for Krups is different from PROLL for Mr. Coffee, and so on.

If, at the bottom of the PROLL window, the system prints the phrase 'obio_range' and hangs for minutes without end, the boot is halted, and you are likely running an old version of PROLL. Verify your PROLL version is the most recent and try again.

8.1.4. Stage 4: Kernel Boot

After PROLL finishes its work, the whole screen should go black. You should see a picture of the Linux logo, Tux the penguin, in the upper left hand of the screen. At this point, messages relating your kernel should be spilling down before you. The color of the text is either white or grey depending on your monitor.

If the screen didn't flip, and you do not see Tux, chances are you are using a kernel compiled with framebuffer support for a different JavaStation model than you are using.

After the message, 'decompressing kernel' appears and the kernel begins spitting out its boot messages, any mistakes from this point are due either to: the filesystem you are using, the filesystem mounting, or missing kernel drivers which should have been compiled in; in other words, your own fault.

9. Questions and Troubleshooting

This chapter is intended to provide solutions to frequently and infrequently encountered problems in enabling Linux on the JavaStations.

9.1. When booting, the message "The file just loaded does not appear to be executable." Why?

On systems that have the older OpenBoot version 2.3, and are not set up to use PROLL, you will get this message when attempting to boot up a kernel image that is not in AOUT format. Be sure to run **`elftoaout`** on your kernel image, as described in the "Kernel Build" chapter.

9.2. When booting, the message "no a.out magic" appears and halts the boot. Why?

On systems that are set up to use PROLL, you will see this message when attempting to boot up a kernel image that is not in AOUT format. Be sure to run **`elftoaout`** on your kernel image, as described in the "Kernel Build" chapter.

9.3. I tried booting a Krups but JavaOS comes up. I don't even have JavaOS!

This likely means you have a flash SIMM install, and the flash SIMM has JavaOS loaded on it. Remove the SIMM and the problem should go away.

9.4. Cannot Boot an "Embedded–Root" image > 10 MB on my JavaStation. Why?

There is a known limit of 8 MB when using the "Embedded–Root" boot image option.

The cause of this is the current version of the PROLL software, which map only 8 MB of low memory. Any more and banking support would need to be added to it.

If needed, this limit can be fixed by someone, as the source to PROLL has been released under terms of the General Public License (GPL).

So in reality, the embedded image size limit is really 8 MB , not 10 MB. If 10 MB somehow works for you, it is sheerly by "luck"!

9.5. After Booting, Typing Anything Yields Garbage Characters. Why?

There are a few possibilities for this. Among them:

1. You have an incorrect device # for tty0.
2. The "keytable" loaded is incorrect. Make sure you use "sun" instead of "PC" if you use the keytable program. Look for the keytable configuration file if it exists.

9.6. In X Sessions to a Solaris server, the font server "xfs" crashes. Why?

If you do X sessions to a Solaris server, and you find that your sessions are no longer opening up new windows, chances are the font server on the Solaris host has crashed. This is a known bug in Solaris 2.6 and 2.7 when you have about 2 dozen X terminals sessions running.

The fix is to move the font server to a different OS and point your JavaStations there, or to upgrade your Solaris to the 2.7 11/99 maintenance release or Solaris 8 which both (apparently) have fixes to this problem.

9.7. Performing Indirect XDMCP to a Solaris Server Results in Session Login Failures. Why?

Congratulations! You probably have one of patch numbers 107180−12 through 107180−19 installed on a Solaris 7 server. You need to upgrade to 107180−20 or above to fix this problem.

I (your HOWTO author) reported this problem to Sun in November 1999, at which time I was told a fix was not scheduled to be made, since I was using an "unsupported configuration.". Never mind that the client was a piece of hardware made by Sun itself. Also never mind that indirect XDMCP queries is a standard itself which was broken by Sun. A call back in late January 2000, and I learn that the record of my previous call was non−existant, but a fix was now on its way. The fix finally was made available in April 2000, five months after first reporting the problem. Considering revisions to this patch during the broken XDMCP period dealt with fixing system security issues, we were forced to run the older insecure software for five months while waiting for a fix to a problem which should have been patched immediately.

The moral of the story: test your JavaStation configuration against an upgraded server that is not in production mode.

If you have XDMCP problems not related to these faulty Solaris patches, it may be a new problem, so please report it.

9.8. TFTPd config doesn't work on SUSE 6.3. Why?

This was reported by a user after this document was first released.

In SUSE 6.3, using the tftpd from the 'a' package of the netkit rpm, you must be sure your tftpd line in /etc/inetd.conf has the −s flag. Otherwise you need to specify a full path.

Also, it is not necessary to run tftpd as root, so the suggested username and group for tftpd on SUSE 6.3 is 'nobody' and 'nogroup'

It is not known whether these changes are needed for newer versions of SUSE.

9.9. Regarding RARP: Is it Needed or Not?

RARP is not needed with the Krups or Espresso models and recent PROLL software. RARP is required for Mr. Coffee, however.

This 'Server Configuration' chapter explained how to set up kernel−level RARP on 2.2.x systems.

On servers with kernel versions 2.3.x/2.4.x, kernel−level RARP support is removed. The ZLS holds a version of ANK userland RARP from Andi Klein of SuSE that will work with Linux/SPARC. It is available from: http://people.redhat.com/zaitcev/linux/rarpd−ap1.tar.bz2. The command to use then is **`rarpd−ank −e eth0`**. "−e" makes it ignore /tftpboot checking, and "eth0" is needed if you are behind a firewall.

9.10. Can One Use the Smart Card Reader on the Espresso models?

This is not currently supported, but the reader follows an ISO standard (ISO 7816−3). On Espresso, if you look into PROLL, there are definitions for the GPIO smartcard data/clock in "eeprom.c". So a programmer should technically be able to get the Smart Card slot running.

Whether the smartcard reader on Dover and Espresso are equivalent is not known.

9.11. Can One Use the Solaris DHCP server instead of ISC?

Yes, this is possible. Earlier ISC daemons had problems dealing with 1514−byte requests of the JavaStations, while the Solaris server was able to handle them without problems. Also, former users of JavaOS may already have their Solaris DHCP server active, and wish to keep things on one machine.

Here is how to configure it:

First, fill in your /var/dhcp/"networks" file, populating it with ethernet to IP info, and the appropriate leastime.

```
# This example uses "infinite" leastime
#
0108002081C2AE 03 192.168.128.1  192.168.128.100 −1 java01  # JavaStation
010800208E4CF6 03 192.168.128.2  192.168.128.100 −1 java02  # JavaStation
```

Next, fill in your /var/dhcp/dhcptab file with entries similar to:

```
##
# First, some network info
#
Locale   m  :UTCoffst=21600:
www  m  :Include=Locale:Timeserv=192.168.128.100:DNSdmain=my.own.net:DNSserv=192.168.128.100:
192.168.128.0  m  :Broadcst=192.168.128.255:Subnet=255.255.255.0:MTU=1500:BootSrvA=192.168.128.100:R
#
#  note: BootServA can point to a different TFTP server to get the kernel image
#        off of.
#
#
##
# Now we define the JavaStation TFTPboot parameters
#
```

```
SUNW.Linux m :Include=www:JOSchksm=0x155dbf97:Rootpath=/tftpboot:BootFile=proll.mrcoffee:BootSrvA=19
SUNW.Linux.Krups m :Include=www:Rootpath=/tftpboot:BootFile=proll.krups:BootSrvA=192.168.128.100:TFT
#
#
#  note: different classes are defined for the different PROLL images.
#
##
# Lastly, we list our hosts and which boot class each one gets.
java01  m  :LeaseTim=-1:Include=SUNW.Linux:
java02  m  :LeaseTim=-1:Include=SUNW.Linux.Krups:
#
#
#
###
```

9.12. Can One Pass Arguments to "/sbin/init" in a Diskless Boot like This?

PROLL ships with DHCP options disabled, but it could be changed. You would then do something like "/tftpboot/0A0A0000.ARGS" to get those parameters in.

If you boot from flash memory, PROLL picks up SILO options (where SILO is > version 0.9.6 and PROLL is >= version 11)

9.13. Enabling X on the JavaStation

This is a very frequently asked question.

Enabling X on the JavaStation is possible.

First, be sure you have enabled the appropriate framebuffer device in your kernel's configuration, as described in the "Kernel Build" chapter.

Next, you'll want to use the generic Sun Framebuffer X server and "XF86Config" file. You can build this yourself, or you can try someone's prebuilt binaries, such as the samples pointed to in the "FileSystem Build" chapter.

Recent editions of the framebuffer server coinciding with XFree 4 are reported not to work. Use the older version based on XFree 3.3, or fix the new version and be a hero to thousands.

9.14. Is There Mailing List Help?

There are two mailing devoted exclusively to running Linux on SPARC processor based machines such as the JavaStations.

The first mailing list is the sparclinux list on VGER, at <sparclinux@vger.kernel.org>. You should first subscribe to it by sending a message to <majordomo@vger.kernel.org> with a subject and body line of "subscribe sparclinux <your_email_address>". You can leave out your email address, but it is helpful to put it in if you have multiple valid addresses at your site.

Archives of the VGER sparclinux mailing list are kept at:
http://www.progressive−comp.com/Lists/?l=linux−sparc&r=1&w=2"

The second mailing list is the debian−sparc list at the Debian Project, at <debian−sparc@lists.debian.org>. You should first subscribe to it by sending a message to <debian−sparc−request@lists.debian.org> with a subject and body line of "subscribe <your_email_address>". You can leave out your email address, but it is helpful to put it in if you have multiple valid addresses at your site.

As many of the SPARC kernel hackers run Debian, it is helpful to subscribe to both lists.

Please do not report problems about this document to either list, but send them to <rsd@dubinski−family.org> instead. Also, please use the list archives. JavaStations have been supported on Linux for a while now, and chances are any questions you have not answered by this document are answered in the archives.

9.15. Can One Boot a JavaStation from Onboard Flash Memory?

It is possible to boot a JavaStation−NC from flash, but requires too much arcane knowledge at the moment to be recommended. One problem even if you do go this route is that flash can only be mounted read−only. This gets to be a problem with many things, like X, which require the writing of socket files. A hybrid ramdisk/flash solution would be required.

9.16. Does "Piggyback" work for the x86 too?

With the great embedded−root solution for the JavaStations, the question popped up whether something similar can be done for stock x86 hardware. While there are some x86 NICs that have boot roms on them, you'd also need the piggyback program to put things together. According to Eric Brower, this currently is not possible as the piggyback program looks for a header specific to the SPARC platform. (28−Apr−2000)

Robert Thornburrow<robert@tsac.fsnet.co.uk> sent a version of piggyback which runs on non−SPARCLinux architectures like Linux/x86 and Solaris. This automates the task of creating your embedded root image. You can get his updated piggyback package at:
http://dubinski−family.org/~jshowto/Files/tools/piggyback_nonsparc.tar.gz

9.17. I put new memory in, but now it doesn't boot. Why?

Are you using EDO memory by chance? Mr. Coffee uses fast−page memory only, not EDO.

9.18. Now that JavaStations work with Linux, what about other Free OSs?

JavaStation support is now available with the NetBSD OS as well as Linux.

9.19. Do the Linux 2.4 kernels work? What's the latest that works?

As of this date (Oct 31, 2001), the current stable Linux kernel version is 2.4.13. Kernels in the stable 2.4 series

should work with the JavaStations, but there are a few reasons why they may not work for you. For details, check the "Kernel Build" chapter's entry on supported kernels.

9.20. Can I compile the kernel on a non–SPARC machine?

It should be technically possible to compile your kernel on a non Sun workstation, such as a PC. Currently there are no reports of anyone doing this, but if you wanted, the first place to look is the GCC CrossCompiling HOWTO.

Of course, you can also compile a new kernel on a working JavaStation, if your filesystem image supports it.

9.21. Can I get an ok> prompt like other Sun equipment?

A curious thing happens when you send a JavaStation a break: it resets, not break down to the openboot prom prompt like other Sun equipment. This can be changed on a Krups by setting jumper J1300, pins 7–8. Doing this gets a OBP ok prompt with a Ctrl–Alt–Break on a PS/2 keyboard or break through a serial terminal.

You can also get the ok prompt on the Dover unit, but it requires a hardware fix. To do so on this unit, you must solder a 220K ohm resistor in location R362 (near the FDD connector).

9.22. My keyboard isn't recognized. What can I do?

While it's unlikely, it could be possible that you have a javastation set in the wrong input device mode. To rectify this, you need to enable the openboot prom prompt as described elsewhere in this HOWTO, and then set the 'input–device' directive accordingly. Or, as one contributor did before the OBP setting was discovered, load up NetBSD on your JavaStation and run the eeprom command there. Convoluted, but it works too.

9.23. Proll reports "TFTP: ARP Timeout". Why?

This has been reported to happen when the file PROLL looks for isn't available. Doublecheck your configuration before retrying.

9.24. Why Can't I Get TrueColor on Krups?

Truecolor on Krups with Linux is a bit of a controversy. Some believe it is possible, while others do not.

First, the Krups is listed as having the IGS C1682 framebuffer, while the Espresso has the IGS C2000 chip.

According to an earlier report by one kernel hacker, the reason for Krups not supporting TrueColor is due to lack of kernel support for the Cyber2000 chip. Perhaps the C2000 for the Espresso is the 'Cyber2000'? And perhaps the C2000 is near equal to the C1682. Notes on the ZLS website seem to point to this.

Recent 2.4.x series kernels have an entry labeled 'Cyber2000'. Perhaps this works? One contributor tried and failed.

Ok, there is a userland utility called 'fbset' to change the modes of a framebuffer. Does that work? One contributor said no.

In the sparclinux archives is a report of a user using the 24−bit TCX framebuffer and having success. But TCX chip was in Mr. Coffee, not Krups, and TCX onboard Mr. Coffee had 8−bit max, not 24.

So what is the real scoop with 24−bit color on the Krups? Until others try things and speak up, we don't know.

9.25. I followed this HOWTO, but my Dover doesn't work. Why?

The Dover is not a SPARC−based JavaStation, which this HOWTO caters to. You must use x86 procedures to make it work properly. You did read the warning in the Dover introduction section, didn't you?

I am receiving multiple reports of kernel load failures with the Dover unit. As more information comes in, this HOWTO will present it.

9.26. Can framebuffer be loaded following a serial console initialization?

If you boot a JavaStation via the serial console, the framebuffer console is completely disabled. Is there any way to activate the framebuffer console after booting? (asked on Sparclinux mailing list 2001−05−11).

Not to our knowledge.

9.27. I really need a complete out−of−the−box solution, pronto!

You better get busy then.

9.28. You Didn't Answer My Question.

So ask it. Email <rsd@dubinski−family.org> and I will try to help. If I can not help, I will direct you to the mailing lists or suitable contacts.

10. Reference Docs

This section is a collection of various reference documents which do not belong in any other section.

10.1. Mr. Coffee Jumper Info

```
Mr. Coffee Jumper Assignments

J0206                   JTAG header, perhaps JSCC compatible.
J0904   1-2 shortened   Enter POST - output ttya, input ttya
        1-2 open        Skip POST - output screen, input ttya
        3-4             Unused
        5-6             Unused
        7-8             Unused
J1101   1-2 open (dflt) TPE squelch
        1-2 short       Reduced squelch threshold
J1102   1-2 open (dflt) 100 Ohm TPE termination
            short       150 Ohm TPE termination
J1602                   Manufacturing test of unknown sort
J1603   1-2             PROM select (unfortunately PROM socket is emply)
        2-3 (default)   Flash select
J1604   1-2             FPROM write disable
        2-3 (default)   FPROM write enable

J0904 block is a bit block of pullup resistors which a user may shorten.
They may be read from the keyboard controller with a command 0xDD.
```

10.2. Krups Jumper Info

```
Krups Jumper Assignments

J1202   1-2     Use Flash
        2-3     Select optional diagnostic FLASH PROM in socket J1203
                (this does not sound quite right ...)
J1300   1-2     Software debug use
        3-4     Factory use - PROM switch??
        5-6     Unused
        7-8     Flash update recovery
J0500           JTAG
```

10.3. JavaStation Press Release

Surprisingly, Sun's website still (as of Oct−31−2001) has the JavaStation press release online at http://www.sun.com/961029/JES/ http://www.sun.com/961029/JES Many thanks to Gary <gary@spiritwars.com> for pointing this out.

10.4. JavaOS 1.0 Download

Surprisingly, Sun's JavaOS 1.0 environment for the JavaStations is still mirrored about on the Internet even today (Oct−31−2001). JDSE 1.0.1 can be found at: http://sunsite.tut.fi/javastation Many thanks to Gary <gary@spiritwars.com> for pointing this out.

Download the link labeled 'jdse.tar'. JDSE 1.0.1 was one of the first demo version of JavaOS available, and was a free download. Later, downloads were restricted to paying customers. This first version is merely a boot image of JavaOS and the HotJava browser. No telnet or ssh applet, no X Windows applet, no file manager, no

email applet, nothing but the browser. Starting to feel boxed in? Welcome to the official software of the JavaStation.

Copies of latter versions of JavaOS as included in the NetraJ software bundle have not been located online yet. This is probably due to the latter versions, namely NetraJ 2 and above, was retail software, and never available for free download from Sun's site.

10.5. Espresso IDE circuit

Pete Zaitcev has written a document describing how to enable IDE on your Espresso model JavaStation. It is included here with Pete's permission.

```
By Pete Zaitcev
1999/11/01
2000/08/22

I am not responsible for any direct or indirect damages to your
equiment or yourself resulting from you reading this document.
USE THIS INFORMATION ON YOUR OWN RISK.

IDE interrupt line is connected "upside down" on the Espresso.
To have IDE working we need to insert an invertor in it.
We borrow the invertor from ISA IRQ3. If you want to use ISA
modem, set it to use COM3/IRQ4 (please realize that Linux IRQ
level would be programmed in CPU PCIC).

The following picture provides an overhead view:

    +==================== wire 1 ======================+
    #                                                   #
    #           1202                      1200        #           1201
    #        +-------+                  +-------+      #         +-------+
    #       -!  REP  !--14            1 -!  INV  !- 14  #         !       !
    #       -!       !-                 -!       !-     #         !       !
    #       -!       !-                 -!       !-     #         !       !
    #     4 -!       !-                 -!       !--11==+         !       !
    # /---5--!       !-                 -!       !--10====+       !       !
    #/    6--!--\    !-                 -!       !-  9    #       !       !
A-> *     7 -!   \---!---8              -!       !-  8    #       !       !
    !        +-------+   !               +-------+        #       +-------+
    !                    !                                #
    Z                    ! /=========== wire 2 ===========+
    Z                    !/
    Z R2026          B-> +--- ZZZZ ---*
    Z (pullup)               R1208
    !
    *

I recommend to proceed in the following way:

1. Disconnect 1202 5 & 6. Not knowing if I need them I lifted pins with a
model knife. You may just cut them with side cutters.

2. Lift pins 1200 10 & 11 but do not cut them!

3. Run wires from resistor pads "A" to pin 1200 11 and from pad "B" to
   pin 1200 10. Resistor pads are much easier targets for soldering at home
   than pads under pins 1202 5 & 6. I am a software engineer, so I did it
```

```
   the easy way.

   I did not bother to glue wires as a decent electronics hacker would do.

   You are all set. Get kernel 2.4, hack drivers/block/Config.in and enjoy!

   P.S. Let me know if you have drawings of hard drive brackets for Espresso.
```

10.6. JavaStation Boot Monitoring Key Combinations

When booting your JavaStation, there are certain key combinations you can press to enable some boot monitoring functionality.

```
Javastation Key Combinations

These are the key combinations that allow you to perform the command monitor
functions:
Press left Alt, left Ctrl key, letter; then turn the power on. You have to
have the keys pressed when you turn on the power otherwise it will not work.

Ctrl-Alt-H Help on chords
Ctrl-Alt-B Show progress banner
Ctrl-Alt-W Show Ether net address and memory size
Ctrl-Alt-D Run diagnostics
```

These key combinations do not work with the Mr. Coffee model.

10.7. JavaStation Photo Gallery

This section contains links to pictures of the JavaStation line.

Front view of Mr. Coffee is at: http://dubinski−family.org/~jshowto/Files/photos/mr_coffee_front_view.jpg

Top view of Mr. Coffee is at: http://dubinski−family.org/~jshowto/Files/photos/mr_coffee_top_view.jpg

Inside view of Mr. Coffee is at: http://dubinski−family.org/~jshowto/Files/photos/mr_coffee_inside_view.jpg

Mr. Coffee white case variation #1 at:
http://dubinski−family.org/~jshowto/Files/photos/mr_coffee_white_case_1.jpg

Mr. Coffee white case variation #2 at:
http://dubinski−family.org/~jshowto/Files/photos/mr_coffee_white_case_2.jpg

Front view of krups is at: http://dubinski−family.org/~jshowto/Files/photos/krups_front_view.jpg

Side view of krups is at: http://dubinski−family.org/~jshowto/Files/photos/krups_side_view.jpg

Top view of krups is at: http://dubinski−family.org/~jshowto/Files/photos/krups_top_view.jpg

Front view of Espresso is at: http://dubinski−family.org/~jshowto/Files/photos/espresso_front_view.jpg

Side view of Espresso is at: http://dubinski−family.org/~jshowto/Files/photos/espresso_side_view.jpg

Rear view of Espresso is at: http://dubinski−family.org/~jshowto/Files/photos/espresso_rear_view.jpg

Inside view of Espresso is at: http://dubinski−family.org/~jshowto/Files/photos/espresso_inside_view.jpg

See the JavaEngine−1 at: http://dubinski−family.org/~jshowto/Files/photos/je1_overhead_view.jpg

View of the JavaStation mousepad is at:
http://dubinski−family.org/~jshowto/Files/photos/javastation_mousepad.jpg

View of a Lab of JavaStations running Linux is at:
http://dubinski−family.org/~jshowto/Files/photos/lab_of_javastations.jpg

JavaStation Prototype at: http://dubinski−family.org/~jshowto/Files/photos/pre_js_1.jpg

JavaStation Prototype Pic 2 at: http://dubinski−family.org/~jshowto/Files/photos/pre_js_2.jpg

JavaStation Prototype Pic 3 at: http://dubinski−family.org/~jshowto/Files/photos/pre_js_3.jpg

"Dover" JavaStation Internal Pic at: http://dubinski−family.org/~jshowto/Files/photos/dover_inside.jpg

JavaStation Cluster of Eric Brower running a parallel POVRay calculation at:
http://dubinski−family.org/~jshowto/Files/photos/cluster.jpg

JavaStation/Fox front view at: http://dubinski−family.org/~jshowto/Files/photos/fox_front.jpg

JavaStation/Fox back view at: http://dubinski−family.org/~jshowto/Files/photos/fox_back.jpg

JavaStation/Fox facing view at: http://dubinski−family.org/~jshowto/Files/photos/fox_face.jpg

JavaStation/Fox internal left view at: http://dubinski−family.org/~jshowto/Files/photos/fox_internal_left.jpg

JavaStation/Fox internal right view at:
http://dubinski−family.org/~jshowto/Files/photos/fox_internal_right.jpg

A. GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front−matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front−Cover Texts or Back−Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine−readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard−conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine−generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front−Cover Texts on the front cover, and Back−Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine−readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly−accessible computer−network location containing a complete Transparent copy of the Document, free of added material, which the general network−using public has access to download anonymously at no charge using public−standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front−matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties−−for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front−Cover Text, and a passage of up to 25 words as a Back−Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front−Cover Text and one of Back−Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self−contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. Future Revisions of this License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front−Cover Texts being LIST, and with the Back−Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front−Cover Texts, write "no Front−Cover Texts" instead of "Front−Cover Texts being LIST"; likewise for Back−Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.