# Linux on the Sun JavaStation NC HOWTO


## Robert S. Dubinski


2000−Nov−15


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

# Table of Contents

# Table of Contents

# Table of Contents

# 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@robert-dubinski.yi.org>`, System and Network Administrator for TAOS Mountain of Santa Clara, CA. Robert is the former computer technician and UNIX systems administrator for Marquette University's Math, Statistics and Computer Science Department, where there are 125 JavaStations running Linux. These machines were 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.

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>` (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

## 1.3. 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)
- Magdalena Wodzinska `<magda@magdalena.wodzinska.yi.org>` (Proofreading and document layout suggestions)
- 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)
- David Giller`<dave@pdx.net>` (Dover Info)
- Robert Thornburrow`<robert@tsac.fsnet.co.uk>` (non−SPARC piggyback info)

If you contributed a tidbit of info and are not listed, please email the document author to get yourself listed.

## 1.4. History of this document

**Revision History**

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

## 1.5. Document Copyright and Licenses

## 1.6. Location of the Latest Version and Source

The latest online version of this document can be found at: http://javastation−howto.homeip.net .

The pre−processed SGML source to this document, written to the Docbook DTD, is available from: http://javastation−howto.homeip.net/Files/JavaStation−HOWTO.sgml

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

# 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.

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 or the Linux OS.

## 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

## 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://javastation−howto.homeip.net/Files/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 folks 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 archived for read at: http://www.idsa.org/whatis/seewhat/idea98/winners/javastation.htm"

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://javastation−howto.homeip.net/Files/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 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 working.

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 ethenet card, demonstrating the expandability of this unit.

See the JavaStation−E at: http://javastation−howto.homeip.net/Files/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://javastation−howto.homeip.net/Files/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, folks 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 anothe 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 out there to go off of.

See the Dover at: http://javastation−howto.homeip.net/Files/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 offerred, 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 word is the performance was not as good as expected, and the 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 of bodoman.com, 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://javastation−howto.homeip.net/Files/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 model was the JavaStation development box, used by the developers of early JavaStation software. Basically it was a SS5/110 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, lance ethernet network interface, onboard PCMCIA, 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.

See the JavaStation/Fox at: http://javastation−howto.homeip.net/Files/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 has killed the line in favor of the Sun Ray. While the performance of the Sun Ray is good, keep in mind it is not a dedicated computing device and is little more than a graphics display hanging off your Sun server, which can give you some unexpected 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.

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 PROM's 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.

As the Sun Ray is not an NC in the traditional sense (it is merely a framebuffer, and not a computing device itself), there is no explanation why the two do 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, Professor at Marquette University `<alexr@mscs.mu.edu>`

> 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 <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.

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. 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 diskarrays, 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.

−−Robert Dubinski, former Computer Systems Technician at Marquette University
`<rsd@robert-dubinski.yi.org>`

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

## 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. It should be possible to order any remaining JavaStation stock from the Sun Spares site at http://www.sunspares.com.

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. Search around the auction sites like Ebay and Yahoo Auctions, and you should be able to turn some up.

Lastly, a great resource for JavaStations is "Bodoman's JavaStation site" at: http://www.bodoman.com/javastation/javastation.html . Here you can find Mr. Coffee and Krups models. As of November 15th 2000, Bodoman was still selling Mr. Coffee models for $50 and Krups models now at the premium price of $365. If you want a JavaStation from BodoMan, contact him now!

Mr. Coffee is the most widespread JavaStation model, and has tended to sell around $50 consistently for the last year or so. Krups models are getting rare and sell at higher prices (the stylish case still stands out today), and all other models are near impossible to find on the open−market.

You might also get lucky and stumble on someone who wants to get rid of JavaStations cheap. One reader reported finding a 32−MB Krups for $75 in a pristine unopened box.

# 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, the Sun SparcLinux server is needed 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.

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 example kernels and filesystems for you to use, eliminating your need of 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.

## 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 the ZLS website at: http://members.home.com/zaitcev/linux.

The current version at the time of this writing is "13".

PROLL can also be found mirrored on "VGER", and also on this HOWTO's distribution site at: http://javastation−howto.homeip.net/Files/proll_13.tar.bz2 (HOWTO website mirror − version 13)

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

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

### 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).

### 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.

First time users will want to set up an "NFS−Root" configuration. When you have things stabilized, move to "Embedded−Root" and make 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 Pete Zaitcev's website at: http://members.home.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.

# 4. Build Your Kernel

## 4.1. Before you begin

This chapter assumes you wish to compile your own Linux kernel for the JavaStation. It assumes you already know how to compile Linux kernels in general, perhaps on 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, this HOWTO assumes you do it on a Linux/Sparc based server running in 32−bit mode.

## 4.2. Make sure you use 32−bit mode

When compiling your own JavaStation−capable kernel, you need to make sure the Sun server you are working on is set to 32−bit mode. So, if you're on an Ultra−class machine, be sure you 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 on which model of JavaStation you have.

Mr. Coffee support has worked 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. Pete Zaitcev `<zaitcev@yahoo.com>` added Krups support in the early 2.3.x sequence, but the MMU changes to the 32−bit SPARC kernel kept it from compiling cleanly until later on. The kernel is known to compile cleanly with the Mar. 17 CVS kernel, and should compile cleanly with any 2.3.99pre3+ version kernel.
Krups support has been backported by Varol Kapton `<varol@ulakbim.gov.tr>`, and it is fully supported in the 2.2.15−prepatch versions.

By the time this document gets widespread exposure, it is hoped that the 2.4.x stable kernel cycle will be ready, at which time any 2.4.x kernel should compile cleanly with support for the entire JavaStation line.

If you can not get a kernel to compile, you should 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:

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
```

You can get a working ".config" file which has these 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 RAMdisk 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 change by 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://javastation−howto.homeip.net/Files/ramdisk_patch

It should 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 "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.

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.

## 4.7. JavaStation−Ready Kernel Images, System.map and ".config" File Samples

Here are some sample ".config" and JavaStation−ready kernel images. They have been donated by Linux−running JavaStation users.

### 4.7.1. Sample ".config" Files

http://javastation−howto.homeip.net/Files/kernel_embedded_config_2_3_99pre3_mar_17

This is a ".config" file donated by Robert Dubinski <rsd@robert−dubinski.yi.org>. It was used at Marquette University to build an embedded boot image from the Mar. 17, 2000 CVS kernel version. This includes support for both Mr. Coffee and Krups in an "Embedded−Root" filesystem configuration. These options should be valid for newer kernels as well; Perform a **make oldconfig** when using with latter kernels.

http://javastation−howto.homeip.net/Files/kernel_nfsroot_config_2_3_99pre3_mar_17

This is an nfs−root capable version of the above ".config" file.

### 4.7.2. Sample JavaStation−Ready Kernel Files

http://javastation−howto.homeip.net/Files/vmlinux_embedded_2_3_99pre3_mar_17

This is a kernel file donated by Robert Dubinski <rsd@robert−dubinski.yi.org >. It was built for Marquette University and is based off the Mar. 17, 2000 CVS kernel version.

This kernel image includes support for both Mr. Coffee and Krups models in an "Embedded−Root" filesystem configuration.

This boot kernel image has already been converted to the required AOUT executable format.

http://javastation−howto.homeip.net/Files/vmlinux_nfsroot_2_3_99pre3_mar_17

This is the nfs−root version of the above kernel.

http://javastation−howto.homeip.net/Files/system.map_embedded_2_3_99pre3_mar_17

The System.map for the embedded kernel image.

http://javastation−howto.homeip.net/Files/system.map_nfsroot_2_3_99pre3_mar_17

The System.map for the nfsroot kernel image.

# 5. Build A JavaStation−Ready FileSystem

This chapter details 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.

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.

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 and 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. |

## 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.

A filesystem image contributed by Varol Kapton <varol@ulakbim.gov.tr> is at: http://javastation−howto.homeip.net/Files/jsroot_varol.tar.gz

# 6. Set up Your Server

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

## 6.1. 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".

## 6.2. 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 at start−up (Linux/SPARC has no RARP daemon, per se):

```
### 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
###
```

## 6.3. 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
```

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.

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

## 6.4. 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.

## 6.5. 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
```

## 6.6. The Last Configuration Step

The last step to configuring your Linux−running JavaStation: boot it and cross your fingers!

> **Tip:** Report of success are also heard of where one or more of these configuration steps have been used: knocking on a wooden surface, booting during a full moon, walking under ladders, breaking of mirrors, throwing salt over one's shoulder, hunting black cats and sacrificing chickens (KFC will suffice).

## 6.7. What to See When Booting Linux

When you boot things properly, your JavaStation will start up with the normal white background screen with the PROM banner at the top, and you will get the black "exclamation mark in triangle" logo, signalling the system doesn't yet know who it is. When contact is made with the DHCP server, the logo goes away and changes to the Java coffee cup logo. After this, a black background window opens. This is the PROLL window. It'll show status of the TFTP download in progress, and give stats on the size of the file downloaded. Next, the whole screen should go black, you should see a picture of Tux the penguin in the upper left hand of the screen, and have the normal Linux kernel messages printed before you. Any mistakes from this point are due to the filesystem you are using, the filesystem mounting, or missing kernel drivers which should have been compiled in.

# 7. Troubleshooting

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

## 7.1. 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.

## 7.2. 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.

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

This likely means you have a copy of JavaOS loaded on your flash SIMM. Remove the SIMM and the problem should go away.

## 7.4. 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.

This limit can be fixed if needed by someone, as the source to PROLL has been released under 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 by "luck"!

## 7.5. 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. A "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.

---

## 7.6. 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 architecture and point your JavaStations there, or to upgrade your Solaris to the 2.7 11/99 maintenance release or Solaris 8 which both have fixes to this problem.

---

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

Congratulations! You must 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.

> Here's a little rant:
>
> I 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 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.
>
> −−Robert Dubinski, former Computer Systems Technician at Marquette University
> `<rsd@robert-dubinski.yi.org>`

---

## 7.8. TFTPd config doesn't work on SUSE. 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'

# 8. Answers to Miscellaneous Questions

This chapter aims to answer some miscellaneous questions about Linux and the JavaStations.

## 8.1. 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 document explains how to set up kernel−level RARP for the remaining models. In 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://members.home.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.

## 8.2. 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.

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

Yes, this is possible. Earlier ISC daemons had problems, while the Solaris server was more robust. 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:Router=192.168.128.101:NISdmain=my.own.net:NISservs=192.168.128.100:
#
#  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=192.168.128.100:TFTPsrvN=lnxserv:
SUNW.Linux.Krups m :Include=www:Rootpath=/tftpboot:BootFile=proll.krups:BootSrvA=192.168.128.100:TFTPsrvN=lnxserv:
#
#
#  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:
#
#
#
###
```

## 8.4. 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)

## 8.5. Enabling X on the JavaStation

Enabling X on the JavaStation is possible.

First, be sure you have enabled the appropriate framebuffer device in your kernel's configuration (as described elsewhere in this document).

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, like the samples pointed to below.

As of this time, XFree 4.0 does not work on the SPARC line. You'll need to use an XFree 3.3.x variant in the meantime. The new driver model of 4.0 will provide the path necessary to provide a dedicated accellerated X server for the JavaStations.

Sample XFree Sun Frambuffer X Server File is at: http://javastation−howto.homeip.net/Files/XF86_FBDev

Sample XFree JavaStation−Ready XF86Config File is at:
http://javastation−howto.homeip.net/Files/XF86Config

## 8.6. Is There Mailing List Help?

There is a mailing devoted exclusively to running Linux on SPARC processor based machines like the JavaStations.

The mailing list address is "sparclinux@vger.rutgers.edu". You should first subscribe to it by sending a message to "majordomo@vger.rutgers.edu" 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 Linux/Sparc mailing list are kept at: http://www.progressive−comp.com/Lists/?l=linux−sparc&r=1&w=2"

---

## 8.7. 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.

---

## 8.8. 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://javastation−howto.homeip.net/Files/piggyback_nonsparc.tar.gz

---

## 8.9. 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.

---

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

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

---

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

As of this date (Nov. 15, 2000), we are still in Linux 2.4 pre kernel phase. So we can't say right now JavaStations will work out of the box with the Linux 2.4 kernel.

We've had reports of the success with the following recent 2.3.x and 2.4−pre kernels: 2.3.99−pre9

## 8.12. 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.

## 8.13. 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.

## 8.14. 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.

## 8.15. 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. Unanswered Questions

This chapter lists questions which have been asked by the author or others, but as of now have no answers to.

## 9.1. Where Can One Find Espressos for Sale?

Enquiring minds want to know.

## 9.2. Do Tools Exist to Configure Net Boot Entries Quickly?

Enquiring minds want to know.

## 9.3. What can one use the Krups Flash memory for?

Though it is not supported without some experimental patches from ZLS, the question arises as to what uses one might put the flash to use for, aside from booting?

## 9.4. Why Can't I Get TrueColor on Krups?

Enquiring minds want to know.

# 10. Appendix

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 Nov−14−2000) 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 Download

Surprisingly, Sun's JavaOS 1.0 environment for the JavaStations is still mirrored about on the Internet even today (Nov. 14, 2000). JSE 1.0.1 can be found at: http://sunsite.tut.fi/javastation Many thanks to Gary <gary@spiritwars.com> for pointing this out.

---

## 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 38; 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 38; 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 38; 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.13;
```

## 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
```

## 10.7. JavaStation Photo Gallery

This section contains links to pictures of the JavaStation line.

Front view of Mr. Coffee is at: http://javastation−howto.homeip.net/Files/mr_coffee_front_view.jpg

Top view of Mr. Coffee is at: http://javastation−howto.homeip.net/Files/mr_coffee_top_view.jpg

Inside view of Mr. Coffee is at: http://javastation−howto.homeip.net/Files/mr_coffee_inside_view.jpg

Mr. Coffee white case variation #1 at: http://javastation−howto.homeip.net/Files/mr_coffee_white_case_1.jpg

Mr. Coffee white case variation #2 at: http://javastation−howto.homeip.net/Files/mr_coffee_white_case_2.jpg

Front view of krups is at: http://javastation−howto.homeip.net/Files/krups_front_view.jpg

Side view of krups is at: http://javastation−howto.homeip.net/Files/krups_side_view.jpg

Top view of krups is at: http://javastation−howto.homeip.net/Files/krups_top_view.jpg

Front view of Espresso is at: http://javastation−howto.homeip.net/Files/espresso_front_view.jpg

Side view of Espresso is at: http://javastation−howto.homeip.net/Files/espresso_side_view.jpg

Rear view of Espresso is at: http://javastation−howto.homeip.net/Files/espresso_rear_view.jpg

Inside view of Espresso is at: http://javastation−howto.homeip.net/Files/espresso_inside_view.jpg

See the JavaEngine−1 at: http://javastation−howto.homeip.net/Files/je1_overhead_view.jpg

View of the JavaStation mousepad is at: http://javastation−howto.homeip.net/Files/javastation_mousepad.jpg

View of a Lab of JavaStations running Linux is at:
http://javastation−howto.homeip.net/Files/lab_of_javastations.jpg

JavaStation Prototype at: http://javastation−howto.homeip.net/Files/pre_js_1.jpg

JavaStation Prototype Pic 2 at: http://javastation−howto.homeip.net/Files/pre_js_2.jpg

JavaStation Prototype Pic 3 at: http://javastation−howto.homeip.net/Files/pre_js_3.jpg

"Dover" JavaStation Internal Pic at: http://javastation−howto.homeip.net/Files/dover_inside.jpg

JavaStation Cluster of Eric Brower running a parallel POVRay calculation at:
http://javastation−howto.homeip.net/Files/cluster.jpg

JavaStation/Fox front view at: http://javastation−howto.homeip.net/Files/fox_front.jpg

JavaStation/Fox back view at: http://javastation−howto.homeip.net/Files/fox_back.jpg

JavaStation/Fox facing view at: http://javastation−howto.homeip.net/Files/fox_face.jpg

JavaStation/Fox internal left view at: http://javastation−howto.homeip.net/Files/fox_internal_left.jpg

JavaStation/Fox internal right view at: http://javastation−howto.homeip.net/Files/fox_internal_right.jpg