JavaStation Client Software Guide

901 San Antonio Road
Palo Alto, , CA 94303-4900
USA 650 960-1300 Fax 650 969-9131

Part No: 805-5890-10
September 1998, Revision A

Adobe PostScript

Please Recycle

Contents

Preface

JavaStation™ computers rely on the services of Solaris™ operating environments in their network for initial boot information and JavaStation client software. The required Solaris services can be configured using the Netra™ j web-based administration interface.

For experienced system administrators, it is also possible to configure the required services using Solaris commands. This guide explains the concepts behind command line administration of a JavaStation network.

For information on Netra j, go to the Netra j web site at `http://www.sun.com/netra-j`, or peruse the `Netra j 3.0 Administrator's Guide`.

How This Book Is Organized

Chapter 1 describes how to plan a JavaStation network.

Chapter 2 describes the JavaStation boot sequence in detail.

Chapter 3 describes the boot progress icon that appears on the JavaStation screen.

Chapter 4 lists the properties that control JavaOS™ software behavior and explains how to set these properties.

Chapter 5 describes how to deliver the JavaStation user application dynamically during the boot sequence.

Chapter 6 describes how to deliver the JavaStation user application as part of the JavaOS binary.

Chapter 7 describes the default user applications included with JavaOS: HotJava™ Browser and HotJava™ Views™.

Chapter 8 describes how to boot a JavaStation computer over a PPP/modem connection.

Chapter 9 explains how to set up JavaStation peripheral devices.

Chapter 10 describes how to enable JavaStation computers to be used in a language other than U.S. English.

Appendix A contains forms that show users how to set up their JavaStation computers.

Appendix B describes how to troubleshoot the JavaStation boot sequence.

Using UNIX Commands

This document may not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.

See one or more of the following for this information:

- *Solaris 2.x Handbook for SMCC Peripherals*
- AnswerBook™ online documentation for the Solaris 2.x software environment
- Other software documentation that you received with your system

Typographic Conventions

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output.	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output.	`% su` `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized. Command-line variable; replace with a real name or value.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be `root` to do this. To delete a file, type `rm` *filename*.

Shell Prompts

TABLE P–2 Shell Prompts

Shell	Prompt
C shell	*machine_name*%
C shell superuser	*machine_name*#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Related Documentation

TABLE P–3 Related Documentation

Application	Title	Part Number
JavaStation setup (brick model	*JavaStation Hardware Setup Instructions*	802-7450-10
JavaStation setup (tower model)	*JavaStation Hardware Setup Instructions*	805-1249-10
Netra™ j administration	*Netra j 3.0 Administrator's Guide*	805-5363-10

Sun Documentation on the Web

The `docs.sun.com` web site enables you to access Sun™ technical documentation on the Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject at:

`http://docs.sun.com`

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

smcc-docs@sun.com

Please include the part number of your document in the subject line of your email.

CHAPTER 1

Planning the JavaStation Environment

This chapter introduces the JavaOS operating system, describes JavaStation hardware, and explains the requirements you should understand before configuring your network to administer JavaStation computers.

JavaStation Overview

The JavaStation network computer is a new type of computing device that provides application processing power but does not store software or data. The JavaStation computer relies on servers throughout the network for its boot information and software. The JavaStation environment is composed of several JavaStation clients and the server(s) that administer them. This figure shows a simple network configuration in which all the services required by JavaStation computers reside on a single server.

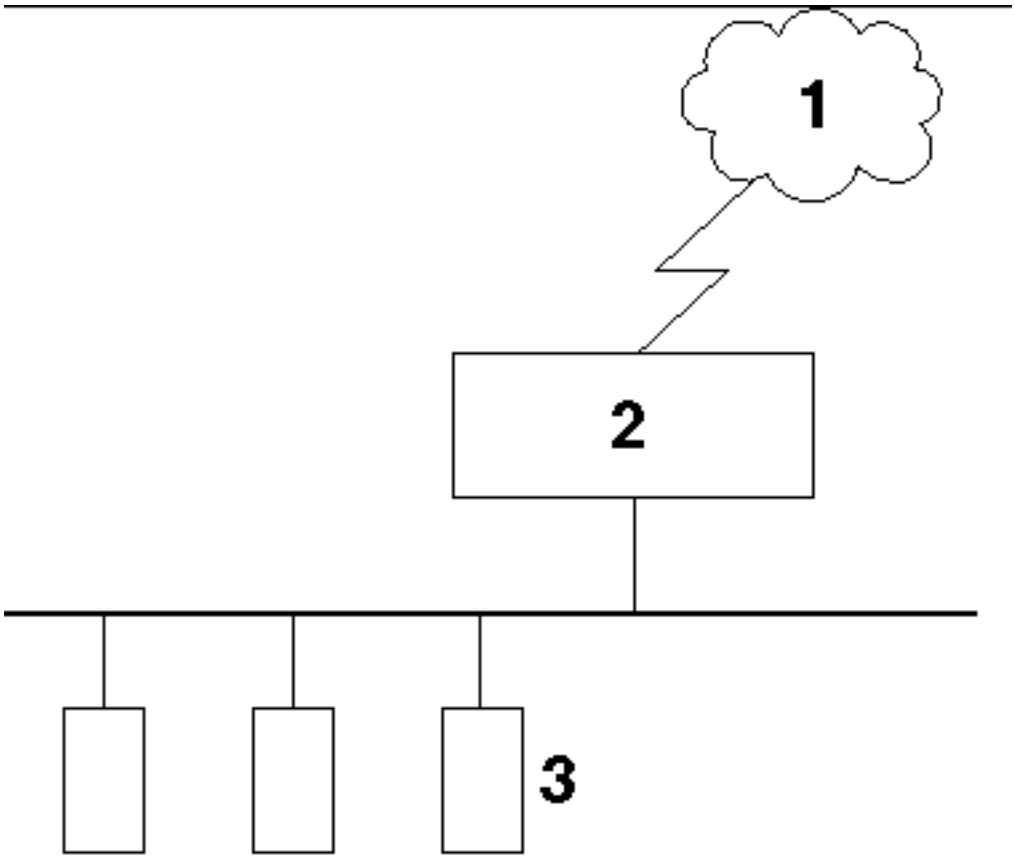


Figure 1–1 Sample Configuration of a JavaStation Network

Legend:

1. Internet or intranet
2. Sun server with home directories, boot, NIS, DNS, web, and DHCP services
3. JavaStation computers

The JavaStation computer's uniquely open software platform enables it to replace existing stateless devices, such as dumb terminals, and to bring high-performance computing to environments where it was not used before.

This chapter gives an overview of setting up a JavaStation network, addressing the following key topics:

- *JavaOS Operating System* – The compact, efficient operating system that drives JavaStation computers.

- *JavaStation Hardware* – Sun's two JavaStation hardware models, which offer scalability of memory, processing power, and device access.
- *Network Services* – The network services required to administer JavaStation computers.

JavaOS Operating System

The JavaOS operating system is specifically designed to run network computers such as the JavaStation computer. The compact architecture of the JavaOS software provides the following advantages:

- *Minimum JavaStation memory requirements* - JavaOS software has a small runtime footprint that conserves memory space and reduces memory costs.
- *Short download times* – The compact size of the JavaOS software reduces the time needed to download software to the JavaStation computers.
- *Ease of training* – The JavaOS feature set reduces user training on core OS functionality, enabling trainers to spend more time on application functionality.
- *Customization* – The JavaOS software supports customization of the JavaStation user's desktop environment through *static linking* or *dynamic loading* (see the following descriptions).

Statically Linking the User Application to JavaOS

The JavaOS operating system supports linking of application files to the JavaOS binary via a special build you can execute at the command line. The JavaOS image resulting from the new build can be downloaded to JavaStation computers in exactly the same way as the old image, with exactly the same core functionality. However, in addition, the user application will be launched automatically when the JavaOS software boots.

Static linking is useful when the JavaStation computer is a public kiosk or other fixed-function device running a single, dedicated application. For instructions on using the static link procedure, see "Using SLK" on page 65.

Dynamically Loading the User Application in JavaOS

Dynamic loading is similar to static linking, in that it enables the user application to be launched immediately when the JavaOS software boots. However, in this case the

application is not physically linked to the JavaOS image; rather, the JavaOS software finds and downloads the application from a network server via HyperText Transfer Protocol (HTTP) immediately after it boots. Because HTTP is used to download the application, the application can reside anywhere on the network that is visible to the JavaStation web server.

The dynamic loading scenario is not limited to a single application. If you like, JavaOS can open a dialog window with a list of applications for the user to choose from. When the user selects an application, JavaOS locates and launches it.

Dynamic loading works in any situation where the JavaStation computer has web access. For instructions on using the dynamic loading procedure, see “Setting Up Dynamic Delivery of an Application ” on page 58.

JavaStation Hardware

JavaStation systems are available in two distinct models as shown in the figure below. The first-generation “brick” model (left) is recognized by its rectangular shape. The newer “tower” (right) sits vertically on the desktop and features a striking, swept profile.

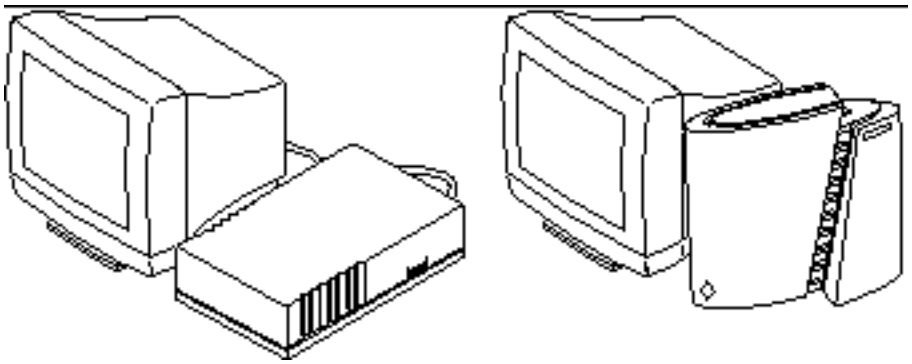


Figure 1–2 JavaStation Systems from Sun

Brick Model

The first-generation brick model JavaStation computer includes the following features:

- *microSPARC-II* – The brick model JavaStation computer is equipped with a 100 MHz microSPARC-II processor.

- *Scalable memory* – The brick model includes 8–64 Mbytes DRAM (64-bit memory bus) and a PC-compatible memory system comprising four SIMM slots (2 logical banks, 2 SIMMs per bank). Memory size can be increased by installing 4-Mbyte or 16-Mbyte SIMMs in the slots.
- *Device connectors* – Connectors for a PS2 mouse, a PS2 keyboard, and a 14-inch or 17-inch monitor are included.
- *Serial port* – A serial port enables local printing to a PostScript™ or PCL5 printer.
- *Power switch* – The brick model includes a continuous contact, long life industrial grade rocker switch for power cycling. The power switch is located at the rear of the unit.

Tower Model

The second-generation tower model JavaStation computer includes the following enhancements to the original model:

- *microSPARC-IIep processor* – The microSPARC-IIep uses less power and generates less heat than the microSPARC-II, eliminating the need for an internal fan.
- *Flash RAM and PPP support* – Tower JavaStation computers come optionally equipped with onboard nonvolatile flash random-access memory (RAM). This flash memory can be used to store the JavaStation computer's operating system (JavaOS), enabling faster booting and wide-area network (WAN) applications for the JavaStation. In a typical scenario, a JavaStation computer would boot JavaOS from its flash RAM and then access the network over a low-bandwidth dialup Point-to-Point Protocol (PPP) connection.
- *Autosensing 10/100Base-T networking* – Tower JavaStation computers also integrate full autosensing 10/100Base-T Ethernet.
- *Standby switch* – A soft touch press-on-press-off standby switch cycles JavaStation power and provides audio and tactile feedback.

JavaStation Model Comparison

As shown in the table below, the first-generation "brick" JavaStation computer must download JavaOS from a boot server over Ethernet. These computers are appropriate for local-area network (LAN) deployment where 10Base-T Ethernet networking is available.

The second generation "tower" JavaStation computers are ideal for remote or Wide Area Network (WAN) environments in addition to local 10Base-T and 100Base-T LAN environments. The integrated flash RAM enables these JavaStation computers to be installed remotely and deployed across a WAN or corporate extranet.

TABLE 1–1 JavaStation Comparison for LAN and WAN Environments

Category	Brick Model JavaStation	Tower Model JavaStation
Target Environment	LAN	■ LAN ■ WAN
Booting	Network boot	■ Network boot ■ Flash RAM
Networking	10Base-T	■ 10/100Base-T ■ PPP over telephone line

Network Services

This section gives an overview of the network services required to administer JavaStation computers. Although all of these services can be configured at the Solaris command line, the Netra j web-based administration interface provides a simpler way to configure them. For information on the Netra j software, visit `http://www.sun.com/netra-j` or peruse the `Netra j 3.0 Administrator's Guide`.

The JavaStation computer requires the following network services for boot information and software.

TABLE 1–2 JavaStation Network Services

Service	What It Does
DHCP (Dynamic Host Configuration Protocol)	Delivers configuration information to the JavaStation computer, including the JavaStation computer's IP address and the addresses of the other servers it requires.
TFTP (Trivial File Transfer Protocol)	Delivers the boot file to the JavaStation computer. May deliver JavaOS to the JavaStation computer.
NFS™	Provides access to the user's home directory on a server. May deliver JavaOS to the JavaStation computer.
DNS (Domain Name System)	Provides address resolution for systems throughout the Internet.

TABLE 1–2 JavaStation Network Services *(continued)*

Service	What It Does
NIS (Name Information Service)	Provides lookup for user names, home directory location, and print servers.
RAP (Remote Authentication Protocol)	Provides lookup for user names and home directory location.
HTTP (HyperText Transfer Protocol)	May deliver the user application to the JavaStation computer. Delivers web pages and Java applets to the JavaStation computer.

The figure at the beginning of this chapter shows a simple network configuration, whereby all the services required by the JavaStation clients reside on a single server. In contrast, a JavaStation network with distributed services might look something like the figure below.

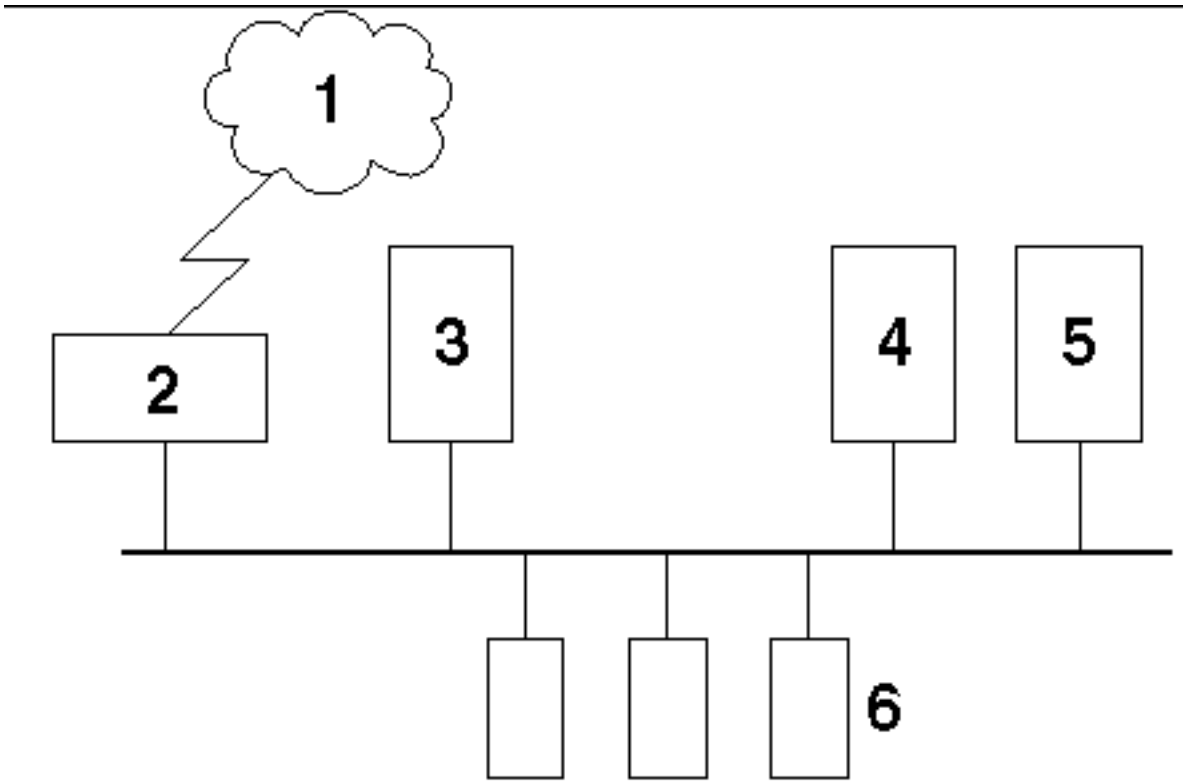


Figure 1–3 JavaStation Network With Distributed Services

Legend:

1. Internet or intranet
2. HTTP (web) server
3. NFS server
4. DNS/NIS server
5. DHCP/TFTP server
6. JavaStation computers

The following sections give an overview of each service required to administer JavaStation computers. For detailed descriptions of these services, see "JavaStation Boot Sequence" on page 13.

DHCP

DHCP is used to pass IP addresses and configuration information to diverse host systems on a TCP/IP network. When a JavaStation computer is powered on, it

receives initial boot information from a DHCP server that is configured to deliver specific parameters to JavaStation computers. The same DHCP server may also be configured to deliver different information to other systems on the subnet.

One server on the JavaStation subnet must be configured with DHCP services or configured to relay these services. If all JavaStation clients reside on the same subnet, and if there are no other DHCP servers on the subnet, no other configuration is needed.

TFTP

TFTP is used along with NFS during the network boot sequence to download a "booter" and (optionally) JavaOS over the network to the JavaStation computer. There is no need to guard against conflict with other TFTP servers on the network.

NFS

Strictly speaking, NFS is only *required* by the JavaStation computer to access and save user data and preferences information. Once a user's home directory has been determined via the NIS automounter map (`auto.home`), it is mounted using NFS. The default user applications provided with the JavaOS software (HotJava Browser, HotJava Views) only use NFS to read and update preferences information. However, many commercial applets may need to use NFS to access and save data files in the user's home directory.

NFS is also the default mechanism used to provide the JavaOS network boot download for the JavaStation computer. It is important to note that the network boot download can alternately be achieved with TFTP through DHCP directives, which may be useful for JavaStation clients used as fixed-function devices (such as kiosks or point-of-sale devices). However, NFS is much faster than TFTP for this purpose.

DNS

The JavaStation computer uses DNS to provide address resolution for host names. Using DNS ensures that JavaStation applications and applets can access URLs either on the corporate intranet or across the Internet.

DNS lookups from the JavaStation computer are rather infrequent, so the load on the DNS server will be relatively small. The JavaStation boot server can be configured as a DNS slave server if a central DNS server for the domain is busy or remotely located.

NIS

The JavaStation computer uses NIS for user authentication, file mapping, and printer access as follows:

- `passwd.byname` – The JavaStation computer uses the `passwd.byname` map to validate the user's user name and password.
- `auto.home` – The `auto.home` NIS map is used to provide NFS access to a user's home directory.
- `printers.conf.byname` – Access to network printers is supplied via the `printers.conf.byname` NIS map if one has been created.

NIS lookups are infrequent, and the performance impact is not likely to be significant. When binding to NIS, a broadcast is sent to the local network. This implies that there must be a NIS server connected to that network. If necessary, the JavaStation boot server can be set up as a NIS slave server or master server. You can also specify the NIS server's IP address in the DHCP parameters delivered to JavaStation computers.

RAP

The JavaStation computer can use RAP instead of NIS for user authentication if it has been configured to do so and if a RAP server exists on the JavaStation network. In return for the user name and password, RAP provides the following:

- User's UNIX ID
- Group ID
- Name of the user's home directory

With this information, JavaOS can mount the user's home directory via NFS.

HTTP

HTTP can be used to deliver the main user application to the JavaStation clients. HTTP is also used by the JavaStation clients to browse the corporate intranet and sites on the Internet.

Proxy Cache

Most corporate intranets implement a secure HTTP proxy system comprising several HTTP servers. The way the proxies are set up can have a significant impact on JavaStation browser performance. If a single proxy server is used for a large number of users, it can become a bottleneck and a single point of failure. If it is located at another site or across a busy backbone network, it will greatly increase the response

time for network requests from the JavaStation computer. One solution to this problem is to make the JavaStation boot server into a proxy cache for its clients.

Understanding the Boot Sequence

Most of the remaining chapters in this guide explain concepts and procedures related to booting the JavaStation computer. Chapter 2 gives a basic overview of the entire boot process, while the other chapters provide supporting information or specific details on one or two procedures. Use the list below as a reference to all the booting information in this guide.

- Chapter 2 explains the fundamentals of the boot process and describes how JavaOS properties, static linking, and dynamic loading can be used.
- Chapter 3 describes the boot progress icon that appears on the JavaStation user's screen.
- Chapter 4 describes JavaOS properties, which can be used to configure the behavior of the JavaOS operating system on the JavaStation computer.
- Chapter 5 explains how to set up the user application to be delivered to the JavaStation computer after JavaOS boots.
- Chapter 6 explains how to link the user application to the JavaOS image so that it is delivered to the JavaStation computers with JavaOS.
- Appendix B explains concepts and procedures for troubleshooting the boot process.

CHAPTER 2

JavaStation Boot Process: Theory of Operations

This chapter describes the JavaStation boot sequence and the nature of JavaStation boot services in the Solaris operating environment.

- "JavaStation Boot Sequence" on page 13
- "JavaStation Boot Services" on page 19

Note - Only experienced Solaris system administrators should set up boot services for JavaStation computers using Solaris commands. Inexperienced Solaris users should use the Netra j software. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

JavaStation Boot Sequence

When the JavaStation computer is powered on, it initiates communication with several different network services to obtain the information and software it requires to operate on the network. The JavaStation boot sequence is accomplished by the following network services:

- DHCP
- TFTP
- NFS™
- HTTP

These services need not reside on the same machine. A DNS service must also be provided on the network for the JavaStation systems to operate correctly, but it is not directly involved in the boot sequence.

The boot sequence can occur in a variety of ways. Essentially, it must accomplish the following tasks:

1. *Lease an IP address to the JavaStation computer.* Each JavaStation system must lease an IP address from an address pool managed by a DHCP server. The DHCP server should assign addresses dynamically. The Solaris DHCP server can also assign IP addresses permanently, but this method creates greater administration overhead if many JavaStation computers reside on the network.
2. *Deliver and boot the JavaOS software on the JavaStation computer.* The JavaOS image is booted from flash memory if the JavaStation computer has flash memory. Otherwise, the JavaOS image is delivered to the JavaStation computer from a network server via TFTP, NFS, or a combination of TFTP and NFS.
3. *Deliver the main user application to the JavaStation computer.* The main user application is either linked directly to the JavaOS boot image or delivered separately to the JavaStation computer via HTTP.
4. *Update the JavaOS image stored in flash memory.* This task applies only to JavaStation clients with flash memory ("tower" models). After the JavaStation computer boots, the JavaOS image stored in flash memory is updated if a newer copy of the JavaOS image exists on the network and if flash updating is enabled. If the flash memory is updated, the JavaStation computer immediately reboots with the newer copy of JavaOS.

The following sections describe the JavaStation computer's power-on and boot sequence tasks in detail.

Power On

The JavaStation programmable read-only memory (PROM) includes TFTP, NFS, and DHCP client implementations and is thus able to carry out the boot sequence. At power on, the PROM checks to see if flash memory is present and valid. If so, the PROM loads it into memory and transfers control to it. JavaOS initializes itself and then completes Task 1, described below.

If the above tests fail, the JavaOS image is not loaded from flash into memory, and the PROM proceeds to complete Task 1 itself.

Task 1: Lease an IP Address

Either the PROM or the JavaOS software leases an IP address from the DHCP server by completing the following "handshake" with the JavaStation DHCP server. For

simplicity, the description below refers to the PROM or the JavaOS software as “the DHCP client.”

1. The DHCP client broadcasts DHCPDISCOVER packets until a DHCPOFFER is received. The DHCPDISCOVER packet includes the Client Class Identifier option, which identifies the DHCP client as a JavaStation (see “Vendor-Specific Options” on page 24). If the broadcast takes an unusually long time, a notice is displayed on the JavaStation screen approximately every minute so the user knows if the server is responding or not.
2. One or more DHCP servers respond with DHCPOFFER packets. The DHCPOFFER(s) are examined by the DHCP client to determine whether they contain the options required to boot JavaStation computers (see “DHCP” on page 19).
3. The DHCP client chooses the best DHCPOFFER it receives that contains the required options. (The best offer is the DHCPOFFER with the most additional options that are applicable to JavaStation computers.) The DHCP client remembers the IP address of the server that sent the offer.
4. The DHCP client broadcasts a DHCPREQUEST packet. This packet contains the IP address of the server chosen in the previous step. All other DHCP servers that responded in the second step are thus informed that they have not been selected.
5. The selected server sends a DHCPACK packet back to the DHCP client.

At this point, the JavaStation has received all of its configuration information from the DHCP server. The figure below illustrates the initial handshake.

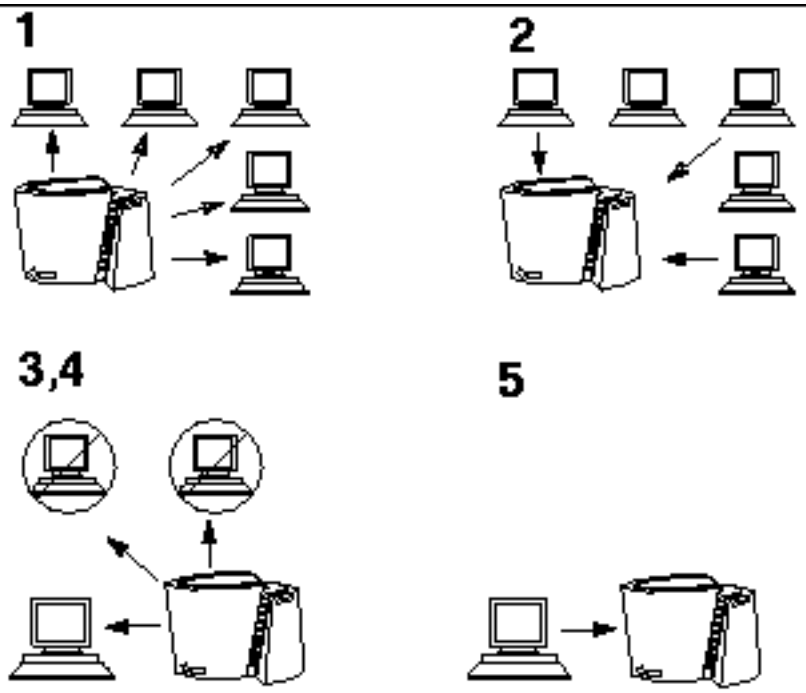


Figure 2–1 JavaStation Client – DHCP Server Handshake

Task 2: Deliver and Boot the JavaOS Software

Task 2 is to pass control of the JavaStation to the JavaOS software. If the JavaOS image has not been loaded from flash memory, it must now be downloaded from the network. The information required for the download was passed to the JavaStation computer in the initial DHCP handshake. Specifically, this information consists of:

- The name of the boot protocol to be used for the download (NFS or TFTP)
- The IP address of the server where the initial boot file (a “booter” or JavaOS itself) resides

The following table shows JavaOS boot information from the DHCP server and the resulting actions taken by the PROM.

TABLE 2–1 DHCP Information for Booting JavaOS

If the boot protocol is...	And the DHCP server also delivers...	The JavaOS software is downloaded and booted as follows
Not set or TFTP	The IP address of a server that contains an initial boot program (the "booter") in its TFTP root directory	The JavaStation PROM downloads the booter from the server using TFTP, then transfers control to it. When the booter begins execution, it checks the PROM's device tree for the name and location of the JavaOS image and the name of a server that contains this image in an NFS-exported directory. If the booter does not find this information in the PROM, it obtains it from the DHCP server. The booter mounts this directory using NFS, copies the JavaOS image, and transfers control to it. Note that this method of delivery is deprecated in favor of downloading JavaOS directly over NFS.
	The IP address of a server that contains the JavaOS image in its TFTP root directory	The JavaStation PROM downloads JavaOS from the server using TFTP, then transfers control to it. Note that this method of delivery is much slower than downloading JavaOS over NFS.
NFS	The IP address of a server that contains the JavaOS image in an exported NFS directory	The JavaStation PROM downloads JavaOS from the server using NFS, then transfers control to it.

The DHCP options used to deliver the above information are described in "The DHCP Service Configuration Table " on page 21.

Task 3: Deliver Main Application

The main application can be linked to the JavaOS binary into a single executable that is stored in flash or delivered to the JavaStation computer using either method described in the table above. The method of binding an application to the JavaOS binary is called *static linking* and is described in Chapter 6.

Alternatively, a main application is delivered to the JavaStation by a web server (HTTP server) on the network. The URL of the application zip file and other JavaOS application-loading properties are passed to the DHCP client in the initial handshake

through the DHCP Vendor-Specific Options, as described in "Vendor-Specific Options" on page 24. For descriptions of the JavaOS application loading properties, see Chapter 5.

Task 4: Update Flash Memory With New JavaOS Image

Tower model computers may include flash memory to store the JavaOS image. After the JavaOS software finishes its initialization, it determines whether it needs to update the flash memory with a new JavaOS image based on the following criteria:

- The value of the JavaOS property `javaos.alwaysUpdate`
- Whether the JavaOS checksum value is different from the checksum of the JavaOS image already in flash

Both the property and the checksum are delivered by DHCP as Vendor-Specific Options. See "Vendor-Specific Options" on page 24.

The JavaOS checksum identifies the revision level of the JavaOS image. For more information, see "Managing the JavaOS Checksum" on page 26.

The results of each possible set of conditions are shown in the table below.

TABLE 2–2 Conditions for Updating JavaStation Flash Memory

Conditions		**Results**
`javaos.alwaysUpdate`	**Checksum**	
Not set	different	JavaOS software opens a Flash Update dialog box on the JavaStation screen. The user has the option of updating flash memory with the new JavaOS binary.
Not set	same	JavaOS software does not update the flash memory.
`true`	different	JavaOS software updates flash memory without querying the user.
`true`	same	JavaOS software does not update the flash memory.
`false`	different	JavaOS software does not update the flash memory.
`false`	same	JavaOS software does not update the flash memory.

If the flash memory is updated, the JavaStation computer immediately reboots with the newly updated JavaOS image.

Note that if any of the following conditions exist, flash memory is *never* updated:

- `javaos.alwaysUpdate` is set to `false`
- The checksum is set to zero
- The checksum is not delivered by the DHCP server
- There is no flash memory on the JavaStation computer

JavaStation Boot Services

This section describes how DHCP, TFTP, NFS, and HTTP services are used to administer JavaStation computers on your network.

- "DHCP" on page 19
- "TFTP" on page 27
- "NFS" on page 29
- "HTTP" on page 29

DHCP

This section provides a description of DHCP configuration (in files) for JavaStation clients. Complete instructions for setting up a Solaris DHCP server are provided in Chapter 4 of the *TCP/IP Data Communication Administration Guide,* available at `http://docs.sun.com`, and in the following Solaris man pages:

- `dhcp(4)`
- `dhcptab(4)`
- `in.dhcpd(1M)`
- `dhtadm(1M)`
- `dhcp_network(4)`
- `dhcpconfig(1M)`
- `pntadm(1M)`

Note - Only experienced Solaris system administrators should set up boot services for JavaStation computers using Solaris commands. Inexperienced Solaris users should use the Netra j software. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

The DHCP server manages a pool of IP addresses for a variety of systems on the subnet, including JavaStation computers. During the boot sequence, the DHCP server delivers to the JavaStation its IP address along with other options that enable the JavaStation computer to operate on the network.

There are three types of configuration information for DHCP in the Solaris operating environment:

- `/etc/default/dhcp`. A file that defines where the next two tables of information are stored. They can be stored either in files or in NIS+ tables.
- *DHCP network table.* A table that maps IP addresses to all the clients on the subnet. The table is updated dynamically as IP addresses are leased and then returned to the pool. If the table is stored in a file, it is generally stored in `/var/dhcp`. Its name is the IP address of the (sub)net being served by the DHCP server (for example, `192_9_100_0`).
- *DHCP service configuration table (*`dhcptab`). A table containing the DHCP options, symbol definitions, and macros used in composing replies to clients. If this information is stored in a file, the file name is `dhcptab`, and it is generally stored in `/var/dhcp`.

The `/etc/default/dhcp` File

The `/etc/default/dhcp` file defines where DHCP configuration files are located. The following is a sample `dhcp` file.

```
# This file controls the defaults for datastore type and location
# for the DHCP service. Two directives are currently supported,
# RESOURCE and PATH. RESOURCE can be either ''files'' or ''nisplus.''
# PATH can be a UNIX pathname for ''files'' resources, or a legal
# NIS+ directory for ''nisplus'' resources.
#
RESOURCE=files
PATH=/var/dhcp
```

The DHCP Network Table

The DHCP network table contains a single entry for each client. Each entry contains the client identifier, the client IP address, the lease time, and other information. The

DHCP server updates the network table dynamically as IP addresses are leased or relinquished by the clients.

A full description of the DHCP network table is provided in the `dhcp_network(4)` man page. The following is an example network table contained in a file named `/var/dhcp/192_9_100_0`.

```
# /var/dhcp/192_9_100_0
#
# Client ID|Flags|Client IP Addr|Server IP Addr|Lease Time|Macro
#
010800208E2091 00 192.9.100.10 129.144.205.69 876425811 elvis
010060972CF0D6 00 192.9.100.11 129.144.205.69 876432501 elvis
0108002087EC88 00 192.9.100.12 129.144.205.69 876507437 elvis
00 00 192.9.100.13 129.144.205.69 0 elvis
00 00 192.9.100.14 129.144.205.69 0 elvis
```

The DHCP Service Configuration Table

The DHCP Service Configuration Table contains groups of DHCP options that are delivered to DHCP clients. For a complete description of this table, refer to the `dhcptab(4)` man page. Throughout this section this table will be called the `dhcptab` file.

The DHCP options that can be delivered to JavaStation computers are a subset of all the options supported by the DHCP specification. Some DHCP options are required for a successful JavaStation boot; others are optional.

Required DHCP Options

The following table lists the DHCP options required in `dhcptab` to boot JavaStation computers. The left column gives the option number as defined in the DHCP specification (RFC 2132), if it exists. The middle column gives the symbol name for the option in the Solaris implementation of DHCP (the name used in `dhcptab`). See “Sample `dhcptab` File ” on page 25 to see how these names are used.

TABLE 2–3 DHCP Options Required to Boot JavaStation Computers

DHCP Option Name (Number)	Symbol Name in Solaris `dhcptab` File	Definition
N/A (This symbol is a DHCP field, not an option.)	`BootSrvA`	The IP address of the server with the initial boot file (either the booter or the JavaOS image). See Table 2–6 for more information.
IP Address Lease Time (51)	`LeaseTim`	The duration (in seconds) of the IP address lease. A value of `-1` indicates an infinite lease. After this period of time has expired without being renewed, the JavaOS software shuts down the networking port.
Subnet Mask (1)	`Subnet`	The subnet mask.
Domain Name Server (6)	`DNSserv`	The IP addresses of one or more DNS servers. The JavaOS software queries additional DNS servers if the primary server fails to respond.
DNS Domain Name (40)	`DNSdmain`	The DNS domain name.
Root Path (17)	`Rootpath`	The NFS-exported directory containing the JavaOS image. This option is used by the booter to locate the JavaOS binary and by the JavaOS software to locate a newer image to update the flash. This option is required if you are using the booter method or if flash update is enabled. See Table 2–6 for more information.

The options DHCP Message Type (53) and Server Identifier (54) are also required in every DHCP packet. However, these options are provided automatically by the DHCP server and do not need to be specified in `dhcptab`.

Optional DHCP Options

The following table lists the DHCP options that can be interpreted by the DHCP client during the JavaStation boot sequence but are not necessarily required for the boot sequence.

TABLE 2–4 Optional DHCP Options for Booting JavaStation Computers

DHCP Option Name (Number)	Symbol Name in Solaris `dhcptab` File	Definition
Bootfile Name (67)	`Bootfile`	The path name of the initial boot file, which can be a "booter" file or the JavaOS software. The path name is assumed to be relative to the TFTP root directory. If this option is not provided, the name of the booter is assumed to be the same as the Client Class Identifier (`SUNW.JDM1` for a brick model; `SUNW.JSIIep` for a tower model) and is assumed to be in the TFTP root directory. See Table 2–6 for more information.
Broadcast Address (28)	`Broadcst`	The network broadcast address.
Router (3)	`Router`	The IP address of the router to be used by the JavaStation clients. If not given, the JavaOS software uses router discovery to locate a router.
Time Server (4)	`Timeserv`	The IP address of a server supporting the RFC 868 time protocol.
N/A (Solaris-specific flag)	`LeaseNeg`	A boolean flag which by its presence tells the DHCP server to renew the leases of clients requesting IP address lease renewal.
NIS Servers (41)	`NISservs`	If the IP address of an NIS server is not given, the JavaOS software broadcasts looking for NIS servers. (This works only if the NIS server is on the same subnet.)
NIS Domain Name (40)	`NISdmain`	The NIS domain name.
Vendor-Specific Options (43)	Symbol names are defined by the user. By convention, use `JOScmd1` through `JOScmd4`, `JOSchksm`, and `JSBproto`	A list of vendor-specific options. See "Vendor-Specific Options" on page 24.

See “Sample `dhcptab` File ” on page 25 for examples of how these options are set in `dhcptab`.

Vendor-Specific Options

The DHCP specification enables hardware and software vendors to create their own DHCP options. These options are delivered through the use of the Client Class Identifier option and the Vendor-Specific Options option. If a DHCP client identifies itself as being of a certain class of client, and the DHCP server has been configured to serve that class of clients, then the DHCP server can respond with a set of options specific to that client type.

Vendor-Specific Options can be used to deliver the JavaOS checksum, JavaOS property settings, and the JavaOS boot protocol to the JavaStation computer during the boot sequence.

- The checksum identifies the JavaOS image that is available from a network server and helps determine whether that image is updated in JavaStation flash memory, as described in “Task 4: Update Flash Memory With New JavaOS Image ” on page 18. Also see “Managing the JavaOS Checksum” on page 26.
- JavaOS property settings determine the resources the JavaOS software uses and other JavaOS attributes. For information on JavaOS properties, see Chapter 4.
- The JavaOS boot protocol is the protocol used to download JavaOS from a network server to the JavaStation computer. Possible settings for the boot protocol are `tftp` and `nfs`.

The delivery of Vendor-Specific Options to JavaStation clients works as follows. The DHCP client on the JavaStation (the PROM or the JavaOS software) includes the JavaStation hardware’s Client Class Identifier (DHCP Option #60) in every packet sent to the DHCP server. When the DHCP server receives the Client Class Identifier, it will deliver the JavaOS checksum, boot protocol, and/or property settings in the Vendor-Specific Options if it has been configured to do so.

The JavaStation hardware’s Client Class Identifier is specified in the PROM. The table below lists the Client Class Identifiers for each JavaStation model.

TABLE 2–5 JavaStation Client Class Identifiers

JavaStation Model	Client Class Identifier
Brick model	`SUNW.JDM1`
Tower model	`SUNW.JSIIep`

Examine the sample dhcptab file below for examples of setting the JavaOS checksum and JavaOS properties in dhcptab. Note, however, that JavaOS properties can be delivered to the JavaOS software using methods other than the DHCP Vendor-Specific Options. See Chapter 4 for more information.

Sample dhcptab *File*

The following sample dhcptab file supplies DHCP options to a variety of clients on a network. Some options are common to all clients. Other options are specific to clients attached to the server, to classes of JavaStation clients, or to specific JavaStation machines. Note that dhcptab can serve many different clients, not just JavaStation computers.

```
# /var/dhcp/dhcptab
#
# This file is a sample DHCP server configuration database for
# JavaStation clients.
#
# Refer to dhcptab(4) for details. This table is generated by
# using the dhcpconfig(1M) command in conjunction with the
# dhtadm(1M) command. It can be administered with the dhtadm
# command.

# The following are symbol definitions for the Vendor-Specific
# Options. Symbol definitions are used to create dhcptab macros.
# Refer to the dhcptab(4) man page for symbol definition syntax and
# instructions on creating macros.

# JOScmd1-4 are for JavaOS properties, JOSchksm is for the
# JavaOS checksum, and JSBproto is for the JavaOS boot protocol.

JOScmd1    s  Vendor=SUNW.JSIIep SUNW.JDM1,101,ASCII,1,0
JOScmd2    s   Vendor=SUNW.JSIIep SUNW.JDM1,102,ASCII,1,0
JOScmd3    s   Vendor=SUNW.JSIIep SUNW.JDM1,103,ASCII,1,0
JOScmd4    s   Vendor=SUNW.JSIIep SUNW.JDM1,104,ASCII,1,0

JOSchksm     s   Vendor=SUNW.JSIIep,128,NUMBER,4,1

JSBproto    s  Vendor=SUNW.JSIIep,129,ASCII,1,0


# Standard macros generated when configured with dhcpconfig(1M). # The first is the time offset from GMT (
# applies to all clients serviced by this server, and the third
# applies to all clients attached to one of the nets attached to
# the server.

Locale   m  :UTCoffst=-25200:

gibson   m  :Include=Locale:Timeserv=10.146.103.191:\
    :LeaseTim=259200:LeaseNeg:\
    :DNSserv=10.146.1.151 10.146.1.152 10.144.1.57:\
    :DNSdmain=foo.bar.com:
10.146.103.0      m   :Broadcst=10.146.103.255:\
        :Subnet=255.255.255.0:\
        :MTU=1500:Router=10.146.103.1:\
        :NISdmain=nis.foo.bar.com:\
```

```
        :NISservs=10.146.103.22:\
        :BootSrvA=10.146.103.191:

# These are macros used to configure specific classes of clients.
# In this case the JavaStation tower and brick models respectively.

SUNW.JSIIep      m   :Rootpath="/export/root/javaos/JSIIep":\
        :JOScmd1="-ihttp://gibson:8080/properties":\
        :JOSchksm=0x13d624be:

SUNW.JDM1    m    :Rootpath="/export/root/javaos/JDM1":\
       :JOScmd1="-ihttp://gibson:8080/properties.JDM1":

# The macros below contain individual DHCP options for the two
# JavaStation computers whose client ID"s (derived from the
# Ethernet address) match the keys. Here each JavaStation gets
# its boot image from a boot server other than the DHCP server.
# The first JavaStation uses NFS to directly download the
# JavaOS image. The second JavaStation computer gets a different
# properties file as well as empty vendor options that override
# any previous default definitions.

0800208E0668      m    :BootSrvA=10.146.103.11:\
         :JOSchksm=0x13b74098:JSBproto=’’nfs’’:

08002087BED4      m    :BootSrvA=10.146.103.114:\
         :JOScmd1="-ihttp://redwings/properties":\
         :JOScmd2="":JOScmd3="":JOScmd4="":\
         :JOSchksm=0x13dd4e2d:
```

Managing the JavaOS Checksum

Each time you receive a new copy of the JavaOS software, you can configure the DHCP server to deliver the new copy to JavaStation computers by following the first set of instructions below. To disable JavaOS updating on the JavaStation computers, follow the second set of instructions.

▼ To Configure the DHCP Server for a New JavaOS Binary

1. **Determine the checksum.**

   The checksum is contained in the first 4 bytes of the last 12 bytes of the JavaOS binary file.

   ```
   % tail -12c  javaos | od -X | nawk ’{print "0x" $2}’
   14eb02a1
   ```

2. **Use** `dhtadm` **to add the new checksum to the DHCP configuration.**

```
% dhtadm -M -m SUNW.JSIIep -e JOSchksm=0xchecksum
```

For example,

```
% dhtadm -M -m SUNW.JSIIep -e JOSchksm=0x14eb02a1
```

When each JavaStation computer boots, its flash memory is updated with the new JavaOS binary and then is rebooted using the new JavaOS binary. This action may be automatic or require user confirmation, depending on other options.

▼ To Disable JavaOS Updating

1. **Set the checksum to zero**:

```
% dhtadm -M -m SUNW.JSIIep -e JOSchksm=0
```

or

1. **Delete the checksum entirely from the DHCP configuration**:

```
% dhtadm -M -m SUNW.JSIIep -e JOSchksm=
```

Note that there is nothing after the equal sign.

TFTP

If a TFTP server will be involved in the boot sequence, set it up using the instructions below. For information on selecting TFTP to be used in the boot sequence, see Table 2–1.

Note - Only experienced Solaris system administrators should set up boot services for JavaStation computers using Solaris commands. Inexperienced Solaris users should use the Netra j software. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

▼ To Set Up a JavaStation TFTP Server

1. **Follow the instructions in the *TCP/IP and Data Communications Administration Guide* to configure a server on the network as a TFTP server.**

   This guide is available on the Web at `http://docs.sun.com`.

   Also, refer to the `inetd(1M)` and `in.tftpd(1M)` man pages.

2. **Use `dhtadm` to set the JavaStation boot protocol to TFTP.**

   Set the `JSBproto` vendor-specific option to `tftp`. See "Vendor-Specific Options" on page 24 for more information.

3. **Use `dhtadm` to add other TFTP boot information to the `dhcptab` file, as described in the following table:**

TABLE 2–6 DHCP Option Settings for TFTP

If the TFTP server delivers..	Set the following in `dhcptab`...	Relative Performance
The booter	■ Set `Bootfile` to the path name of the booter file. The path name is assumed to be relative to the TFTP root directory. (If `Bootfile` is not set, the name of the booter file is assumed to be the Client Class Identifier – `SUNW.JDM1` for a brick model; `SUNW.JSIIep` for a tower model.) ■ Set `BootSrvA` to the IP address of the TFTP server. ■ Set `Rootpath` to the directory on the NFS server where the JavaOS image is located.	Fast
The JavaOS image	■ Set `Bootfile` to the path name of the JavaOS image. The path name is assumed to be relative to the TFTP root directory. ■ Set `BootSrvA` to the IP address of the TFTP server. ■ To enable flash update after the boot, set `Rootpath` to the directory on the NFS server where the JavaOS image is located. (A case using all three of the above settings would be unusual. Typically, the JavaStation computer boots straight from flash, not the network, before a flash update.)	Slow

NFS

If an NFS server will be involved in the boot sequence, set it up using the instructions below. For information on selecting NFS to be used in the boot sequence, see Table 2–1. Using NFS as the boot protocol is the fastest method for delivering JavaOS software to the JavaStation computers.

Note - Only experienced Solaris system administrators should set up boot services for JavaStation computers using Solaris commands. Inexperienced Solaris users should use the Netra j software. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

▼ To Set Up a JavaStation NFS Server

1. **Follow the instructions in the *NFS Administration Guide* to configure a server on the network as an NFS server.**

   This guide is available on the Web at `http://docs.sun.com`.

2. **If a booter file will *not* be used in the boot process, use `dhtadm` to set the JavaStation boot protocol to NFS.**

   Setting the `JSBproto` vendor-specific option to `nfs`. See "Vendor-Specific Options" on page 24 for more information.

3. **Use `dhtadm` to set `Rootpath` to the directory on the NFS server where the JavaOS image is located.**

4. **Use dhtadm to set `BootSrvA` to the directory on the NFS server where the JavaOS image is located.**

HTTP

The HTTP (web) server can deliver the user's main application to the JavaStation.

Note - The user's main application can also be linked to the JavaOS image. For instructions, see Chapter 6.

Note - Only experienced Solaris system administrators should set up boot services for JavaStation computers using Solaris commands. Inexperienced Solaris users should use the Netra j software. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

▼ To Set Up a JavaStation Web Server

1. **Follow the vendor's instructions for setting up the web server.**

2. **Prepare the application to be delivered to the JavaStation computers.**
   See Chapter 5 for instructions.

3. **Set JavaOS application loading properties.**
   See Chapter 5 for instructions.

CHAPTER 3

Boot Progress Indicators

At power on, a series of steps is executed to boot the JavaStation computer, as described in Chapter 2. During this process, the JavaStation screen displays graphics and text to indicate the progress of the boot sequence. Your JavaStation computer will use one of the two progress indication methods described in this chapter.

- "Text and Logo Progress Indicators" on page 31
- "Java™ Coffee Cup Progress Indicator" on page 33

Text and Logo Progress Indicators

Some JavaStation computers display a combination of text and logos to indicate the progress of the boot sequence, as follows:

1. *Initialize boot devices* – At power on, the JavaStation PROM initializes boot devices on the JavaStation computer. On the JavaStation screen, hardware and firmware information is displayed next to the Java Coffee Cup logo, as shown in the following figure.

OpenBoot 3.11.10, 48 MB memory installed, Serial #9314419
Ethernet address 8:0:20:83:20:73, Host ID: 808e2073

Figure 3–1 Initializing Boot Devices

If any boot device initialization fails, an image similar to the following appears.

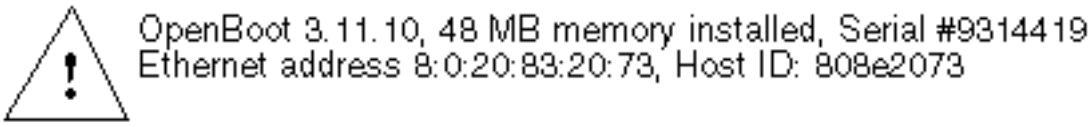


Figure 3–2 Boot Device Initialization Failure

2. *Perform network communication* – If a copy of JavaOS is stored in flash memory (available only on the tower model) and is valid, this copy of JavaOS initializes itself and executes the network communication required for the boot. If not, the JavaStation PROM executes the network communication. In either case, a new copy of JavaOS is downloaded from a network server. During this process, the JavaStation screen displays the logo in the following figure, the Sun logo, and the Java Coffee Cup logo.

Figure 3–3 Network Communication

3. *Boot JavaOS* – JavaOS takes 10-15 seconds to boot. If the booted copy of JavaOS was obtained from flash (and was not updated by the network), the JavaStation screen displays the background shown in the preceding figure. If the booted copy of JavaOS was downloaded from the network, the JavaStation screen displays a Coffee Cup "wallpaper" background.
4. *Log In* – Once JavaOS has booted, the user login window is displayed.

Figure 3–4 JavaStation Login Window

Java™ Coffee Cup Progress Indicator

Some JavaStation computers use the Java Coffee Cup icon as a progress indicator of the boot process, as follows:

1. *Power on* – After basic hardware initialization, the JavaStation logo and its shadow image appear on the JavaStation screen, as shown in the following figure.

Figure 3–5 Boot Screen at Power-On

2. *Initialize devices* – To start the boot process, the JavaStation PROM initializes a boot device. First it checks for on-board flash memory. If a copy of JavaOS is stored in the flash and is valid, control is passed to this copy of JavaOS. If not, control stays with the PROM. Next, the JavaStation Ethernet port is tried. If the port is connected, the PROM (or JavaOS) will proceed with the network boot sequence. During this initialization of boot devices, the saucer of the Coffee Cup icon appears next to the JavaStation logo.

Figure 3–6 Initializing Boot Devices

3. *Locate network servers* – The PROM or JavaOS locates the DHCP server and the boot server on the network. During this step, a cup is added to the saucer.
4. *Download JavaOS* – If the boot server is located, a copy of JavaOS is downloaded from the network, and steam appears in the cup.

Figure 3–7 Downloading JavaOS From Network Server

After JavaOS is downloaded, the shadow image of the JavaStation logo disappears, as in the following figure.

Figure 3–8 JavaOS Is Downloaded

5. *Boot JavaOS* – JavaOS takes 10-15 seconds to boot. If the booted copy of JavaOS was obtained from flash (and was not updated by the network), the JavaStation screen displays the background shown above. If the booted copy of JavaOS was downloaded from the network, the JavaStation screen displays a Coffee Cup "wallpaper" background.
6. *Log In* – Once JavaOS has booted, the user login window is displayed.

Figure 3–9 JavaStation Login Window

If at any time an error occurs, an image similar to the following appears.

Figure 3–10 Boot Error Message

The list of boot devices below the above image shows how far the boot progressed. If there was a problem with:

- the flash

- or Ethernet

the last device tried is represented by the last icon on the boot source line.

A question mark (?) in the error image indicates that the problem is recoverable: if you correct the problem, the boot sequence will continue. For example, if the Ethernet cable is disconnected, you can connect it and the boot sequence will proceed from that point as if nothing had happened. An exclamation point (!) in the error image indicates an unrecoverable error.

Troubleshooting Key Chords

JavaStation computers that boot using the Java Coffee Cup progress indicator also support key chords that can be pressed at the JavaStation keyboard to diagnose boot problems. You must press the chord within four seconds after power on. It may be helpful to press and hold the chord while powering on the JavaStation computer.

TABLE 3–1 Troubleshooting Key Chords

Key Chord	Purpose	Effect When Pressed At Power-On
Alt+b	Debug network boot process	Displays debugging information on booting over the network.
Alt+w	Display system information	Displays total memory, Ethernet address, and firmware information (the display is similar to the error image above).
Alt+d	Run diagnostics	Runs PROM-resident diagnostics and POST.
Alt+t	Try flash booting last	Re-orders the default boot order to try booting in the following sequence: Ethernet, flash.
Alt+h	Show all key chords	Displays all available key chord combinations.

CHAPTER 4

JavaOS Properties

JavaOS properties are settings that control the behavior of the JavaOS operating system on the JavaStation computer. JavaOS properties are delivered to the JavaStation computer at bootup; they can be delivered by the DHCP server or in a file referenced by the DHCP server.

Two kinds of properties control JavaOS behavior: system properties and JavaOS properties. Generally, system properties control the operation of applets or of any Java applications built into the JavaOS image (such as a browser). JavaOS properties determine the resources used by the JavaOS software, such as the language and fonts of the interface and the printers available to the JavaStation.

This chapter lists the JavaOS and system properties by functional group and explains how to set properties.

- "JavaOS and System Properties" on page 39
- "Setting Properties" on page 52

JavaOS and System Properties

The tables below list all JavaOS and system properties. The properties are grouped by function. Each table includes the name of the property, its default value, and a description of what the property controls. All properties are JavaOS properties unless marked as system properties.

Property setting examples use the JavaOS property flags, such as `-d`. These property flags are described in Table 4–7.

General Properties

The following table lists general properties for setting up the JavaOS environment.

TABLE 4–1 JavaOS General Properties

Property Name	Default Value	Description
`javaos.mountlist`	null	A semicolon-separated list of pairs of network paths and local paths. After the user logs into the JavaStation, each remote file system is mounted onto the local file system. The syntax of each pair is: *server*`:`*remote_path*`|`*local_path* For example, to enable localized keyboards, mount `/REMOTE` from the server and file system that has the JavaOS executable image, as follows: `-djavaos.mountlist=myserver:/export/root/javaos/classes|/REMOTE` See "Setting Mount Directories " on page 84 for more details.
`javaos.snmpSysContact`	null	This property sets the value that will be returned in the system Management Information Base (MIB) for the system contact field. It can be set with a string of your choice, which should be enclosed in quotes. Example: `-djavaos.snmpSysContact="Bob"`
`javaos.snmpSysLocation`	null	This property sets the value that will be returned in the system MIB for the system Location field. It can be set with a string of your choice, which should be enclosed in quotes. Example: `-djavaos.snmpSysLocation="Sun MPK14"`

TABLE 4–1 JavaOS General Properties *(continued)*

Property Name	Default Value	Description
`javaos.consoleHotKey`	`VK_PRINTSCREEN`	This property sets the keyboard hotkey that activates the JavaStation console, which displays debugging information. The value of the property is the JDK™ virtual keycode name for the hotkey. The following codes are valid: ■ `VK_F1` ■ `VK_F2` ■ `VK_F3` ■ `VK_F4` ■ `VK_F5` ■ `VK_F6` ■ `VK_F7` ■ `VK_F8` ■ `VK_F9` ■ `VK_F10` ■ `VK_F11` ■ `VK_F12` ■ `VK_PRINTSCREEN` ■ `VK_UNDEFINED` (to disable the console) The value of this property is not case-sensitive; `VK_PRINTSCREEN` and `vk_PrInTSCreen` are equivalent.
`javaos.login`	`true`	If `true`, the JavaOS software displays a login screen after booting and before starting the initial application. If `false`, the JavaOS software runs the main application as soon as it boots, without displaying a login screen. This means there is no user home directory, and no system properties are read from a `properties` file.

TABLE 4–1 JavaOS General Properties *(continued)*

Property Name	Default Value	Description
`javaos.homedir`	null	This property specifies the NFS path JavaOS should mount if NIS is not used to find the path based on the user name. The NFS path is specified as *hostname*`:/`*path*. This property is most often used to determine the directory to use for the properties file that is read by HotJava at startup. If the `javaos.login` property is set to false, `javaos.homedir` is not used.
`javaos.alwaysUpdate`	null	This property specifies that JavaOS is always or never updated in the JavaStation computer's flash memory, regardless of the value of the JavaOS checksum. It is useful for public kiosks or other systems where user input is not expected. For more information, see Table 2–2.
`javaos.allowGuest`	`false`	If true, the login screen (if displayed at all) will contain a guest login button. Guest login requires no username or password, and no user directories are mounted.
`javaos.dns`	`true`	When set to true, host name-to-address and address-to-host name resolution are performed using the DNS protocol. See also `javaos.nis`. If lookup using NIS is enabled also, NIS is attempted first, and DNS is attempted only if NIS lookup fails. See also `javaos.hostaddrmap` and `javaos.hostnamemap`.
`javaos.hostnamemap`	`host. byname`	The name of the NIS map used to perform host name-to-address resolution.

TABLE 4–1 JavaOS General Properties *(continued)*

Property Name	Default Value	Description
`javaos.hostaddrmap`	`host. byaddr`	The name of the NIS map used to perform address-to-host name resolution.
`javaos.homedirmap`	`auto.home`(sought first), `auto_home`	If NIS is enabled, this property is used to set the name of the NIS map used by the JavaOS software to determine a user"s home directory.
`javaos.nis`	`true`	When set to `true`, login authentication, host name-to-address resolution and address-to-host name resolution are performed using the NIS protocol. See also `javaos.dns`. If lookup using DNS is also enabled, NIS is attempted first, and DNS is attempted only if NIS fails.
`javaos.rap`	false	If set to `true`, the JavaOS software uses Remote Authentication Protocol (RAP) instead of NIS for login authentication.
`javaos.rap.server`	null	Set to the IP address of the RAP server. This property is ignored unless `javaos.rap` is set to `true`.

Application Loading Properties

The properties listed below control selection and loading of the main application on the JavaStation after the user logs in. For instructions on using these properties, see Chapter 5.

TABLE 4–2 JavaOS Application Loading Properties

Property Name	Default Value	Description
`javaos.apps`	null	If defined, the JavaOS software launches a simple point-and-click Application Launcher window. This property should be set to the URL of an HTML document that lists the applications to display.
`javaos.mainProgram`	`sun.applet.AppletViewer`	Set to the name of the application's `main` class.
`javaos.mainHome`	`appletviewer`	Set to the name of the property specifying the application's root directory. For example, HotJava Views uses the `hotjava.home` property to specify its root directory. Other applications may have different property names. When the virtual file system is created, this property is set to enable the application to find its files. See "To Deliver a Single Application " on page 60 for more information.
`javaos.mainZip`	null	Set to the name of the archive containing the application files.

Video Resolution Properties

The table below lists properties that control video resolution on the JavaStation user's screen.

TABLE 4–3 JavaOS Video Resolution Properties

Property Name	Default Value	Description
`javaos.fbDimensions`	null	Specifies a new frame buffer resolution to be set at boot time. The syntax of resolution parameters is *widthxheightxdepth@vfreq*, where *depth* is optional. Currently, only 8-bit depth is supported. For example, `-djavaos.fbDimensions=800x600x8@60`
`javaos.fbDimensionsPrompt`	`true`	If `javaos.fbDimensions` is specified, the user will be prompted by a Video Mode Confirmation dialog. If the user selects OK, the video mode is set to the newly specified mode. The user must then accept the new video mode by selecting YES. If they fail to select YES within a 10-second period or if they select NO, the video mode reverts to its original mode. If this property is set to `false`, the confirmation window is disabled.

User Properties

The table below lists miscellaneous user properties that affect JavaOS operation. Most are related to localization. Certain settings cause other settings to be assumed. For example, setting `user.region` to `ja` causes the system to assume the JST time zone, even if `user.timezone` is not defined.

TABLE 4–4 JavaOS User Properties

Property Name	Default Value	Description
`user.timezone`	This	This system property tells the Java date and time API the time zone in which the JavaStation system is located. Example: `-Duser.timezone=PST`. Valid time zones are as follows (this list is from `java.util.TimeZone`.) The default time zone is GMT. ■ `GMT` – Greenwich Mean Time ■ `ECT` – European Central Time ■ `EET` – Eastern European Time ■ `ART` – (Arabic) Egypt Standard Time ■ `EAT` – Eastern African Time ■ `MET` – Middle East Time ■ `NET` – Near East Time ■ `PLT` – Pakistan Lahore Time ■ `IST` – India Standard Time ■ `BST` – Bangladesh Standard Time ■ `VST` – Vietnam Standard Time ■ `CTT` – China Taiwan Time ■ `JST` – Japan Standard Time ■ `ACT` – Australia Central Time ■ `AET` – Australia Eastern Time ■ `SST` – Solomon Standard Time ■ `NST` – New Zealand Standard Time ■ `MIT` – Midway Islands Time ■ `HST` – Hawaii Standard Time ■ `AST` – Alaska Standard Time ■ `PST` – Pacific Standard Time ■ `PNT` – Phoenix Standard Time ■ `MST` – Mountain Standard Time ■ `CST` – Central Standard Time ■ `EST` – Eastern Standard Time ■ `IET` – Indiana Eastern Standard Time ■ `PRT` – Puerto Rico and US Virgin Islands Time ■ `CNT` – Canada Newfoundland Time ■ `AGT` – Argentina Standard Time ■ `BET` – Brazil Eastern Time ■ `CAT` – Central African Time
`user.language`	`en`	This system property must be set to a valid, lowercase, ISO-639 Language Code. Exceptions to this rule are to use `cs` for Czech and `iw` for Hebrew, as the current Language Codes havebeen updated since the JDK1.1 was implemented. Valid codes are described in Table 10–2. Example: `-Duser.language=en`
`user.country`	This	This system property must be set to a valid ISO-3166 Country Code. Valid codes are described in Table 10–3. Example: `-Duser.country=US`

Printing Properties

The table below lists properties related to printing. Instructions for using these properties are in Chapter 9.

Note - Due to space constraints, property names may take up two or more lines in the table below. However, they should be typed on one line only. For example, the second property below is `javaos.printservice.lpd.printers`.

TABLE 4–5 JavaOS Printing Properties

Property Name	Default Value	Description
`javaos.printservice.NIS.mapname`	`printers. conf. byname`	The name of the NIS map used to locate network printers.
`javaos.printservice.lpd. printers`	null	A semicolon-separated list of printers available for use by the lpd printing client. The format of each entry is *printer@server*.
`javaos.printdialog. alwaysShowPrinters`	null	A semicolon-separated list of all the printers available from this host. This property enables administrators to add access to the printer nearest a given JavaStation. The syntax of the printer name is *print_service*:*printer@server*. Example: `-djavaos.printdialog.alwaysShowPrinters =lpd:raw@kona;lpd:ps@kona; NIS:droid@fred`
`javaos.printers.selected`	null	A semicolon-separated list of the printers the user has selected to appear in print dialogs. The format is the same as for `alwaysShowPrinters`. This is a system property. Example: `-Djavaos.printers.selected=raw@kona; ps@kona;droid@fred`

TABLE 4–5 JavaOS Printing Properties *(continued)*

Property Name	Default Value	Description
`javaos.printservice.local.params.serial-`*port*	null	This property sets the communications parameters for the serial port. The *port* portion of this property is the name of a serial port, which can be: ■ `SerialA` or `SerialB` for an on-board JavaStation serial port (`Serial` B is not available on any JavaStation model at this writing) ■ One of `SerialP1` through `SerialP8` for a virtual serial port enabled by the multiport serial card (MPSC), which is not available on any JavaStation model at this writing The syntax of the communication parameters is *baud_rate*:*data_bits*:*stop_bit*: *parity*: *flow_control*. For example: `-djavaos.printservice.local.params.serial-SerialA=57600:8:1:none:hh` Valid values for each parameter are as follows: ■ *baud_rate*: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 ■ *data_bits*: 5, 6, 7, 8 ■ *stop_bits*: 1, 1.5, 2 ■ *parity*: none, odd, even ■ *flow_control*: none, {s,h}{s,h} You can disable flow control by specifying none. Otherwise, specify an {input}{output} pair by selecting from Xon/Xoff (s) or RTS/CTS (h) for input and output.
`javaos.printservice.local.params.parallel-`*port*	false	The *port* portion of this property is the name of a parallel port. A parallel port is not available on any JavaStation model at this writing. When set to true, this property enables the parallel port. For example: -djavaos.printservice.local.params.parallel-LPT1=true

TABLE 4–5 JavaOS Printing Properties *(continued)*

Localization Properties

The table below lists properties strictly related to localization.

Note - The User Properties in Table 4–4 and `javaos.mountlist` in Table 4–1 are also required for localization.

TABLE 4–6 JavaOS Localization Properties

Property Name	Default Value	Description
`javaos.font.properties.home`	`/FONTS`	The local path name of a directory that contains a `lib` subdirectory. The `font.properties` files are read from this `lib` directory. The `javaos.mountlist` property is typically used to associate some server path with the `/FONTS` directory to enable the JavaOS software to load and use fonts from a server. For more information, see "Setting Mount Directories " on page 84.
`javaos.im.compose.deadkeys`	`false`	This property changes the following keys into accent keys: "(single quote) "(double quote)`(grave accent) and ^(circumflex). Use this property if your keyboard is a U.S. keyboard, you are not setting the `javaos.kbd` property, and you want to produce accented characters for ISO Latin locales. If set to `true`, the above keys do not produce a value of their own, but cause the next key pressed to be an accented character. For example, pressing ‘ plus "a" produces á. If this property is false, these keys generate their expected values. For more information on this and the following three properties, see "Enabling Special Characters on the U.S. Keyboard" on page 94.

TABLE 4–6 JavaOS Localization Properties *(continued)*

Property Name	Default Value	Description
`javaos.im.compose_ar`	`ISO8859_6`	Enables the U.S. keyboard to produce Arabic input. The Ctrl-t key sequence toggles the keyboard state between Arabic and U.S. ASCII modes. A status window displays the current mode.
`javaos.im.compose_iw`	`ISO8859_8`	Enables the U.S. keyboard to produce Hebrew input. The Ctrl-t key sequence toggles the keyboard state between Hebrew and U.S. ASCII modes. A status window displays the current mode.
`javaos.im.compose_th`	`TIS620`	Enables the U.S. keyboard to produce Thai input. The Ctrl-t key sequence toggles the keyboard state between Thai and U.S. ASCII modes. A status window displays the current mode.
`javaos.im.url`	null	A semicolon-delimited list of the Solaris machines running the language engines to be used by JavaStation computers. A Microsoft Windows95 or Microsoft NT system can also be used if an Internet/Intranet Input Method Protocol (IIIMP) server from Sun for the PC server is installed on it. Set as follows: `iiimp://`*hostname*`:`*port*`;`*hostname*`:`*port*... where: ■ *hostname* is a system running a language engine ■ *port* is the port at which the engine is located. By default, JavaStation computers use port `9010`. For more information, see "Setting the Input Method " on page 96.

TABLE 4–6 JavaOS Localization Properties *(continued)*

Property Name	Default Value	Description
`javaos.loginLocaleList`	`en_US`	A semicolon-separated list of the locales to be presented as choices to the user at login. A locale is defined using one of the ISO-639 standard two-letter codes that define `user.language` followed by an underscore character "_" and one of the ISO-3166 standard two-letter codes that define `user.country`. For example: `-djavaos.loginLocaleList=\ en_US;fr_FR;ja_JP;zh_CN;zh_TW` For more information, see "Modifying the Languages Displayed at Login" on page 87.
`javaos.kbd`	null	By default, the JavaOS software assumes the keyboard is a U.S. keyboard. To enable a different keyboard, use the syntax `javaos.kbd=`*keyboard*. Possible values for *keyboard* are provided in "Adding a Keyboard" on page 92.
`javaos.im.lookup.button`	`false`	This property controls how the JavaStation user selects characters when using a Korean, Japanese, or Chinese input method. If `false`, when the list of candidate characters is displayed, letters are used to indicate each choice, and the user selects a choice by typing the letter. If `true`, letters are replaced with buttons so that the user clicks on a button to pick a choice. Note that enabling this option will negatively impact user input performance. For more information, see "Setting the Input Method " on page 96.
`javaos.im.status.fixpopup`	false	Setting this property to `true` enables a pop-up window with input method status information. For more information, see "Setting the Input Method " on page 96.

TABLE 4–6 JavaOS Localization Properties *(continued)*

Property Name	Default Value	Description
`file.encoding`	`8859_1`	This system property sets the character set to be used when files are saved. By default it is set to the 8859-1 character set (Roman alphabet). Supported character sets are listed in Table 10–5.
`doc.url`	null	This system property enables JavaStation users to access HotJava Browser documentation. It should be set as follows: `-Ddoc.url=file:/REMOTE/hotjava` HotJava Browser automatically locates the document translation for the current locale. This property is not applicable for any JavaStation application other than HotJava Browser.

Setting Properties

Note - Only experienced Solaris system administrators should set JavaOS properties using the following methods. Inexperienced Solaris users should use the Netra j software. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

You can set the properties that control JavaOS operation on the JavaStation in two places:

- In the DHCP Vendor Options (see Chapter 2).
- In a text file referenced in the DHCP Vendor Options.

The second option is preferable because the total length of DHCP Vendor Options is limited to 255 characters. If you exceed this length, the remaining text is lost and the JavaStation computer will not receive all of the options. No errors are reported by the `dhcp` daemon, but incorrect behavior is viewed on the JavaStation computer.

In contrast, the text file can have any length.

If you set a property in the DHCP options *and* in a text file, the value set in the text file will be used because it was the last value delivered to the JavaStation computer.

Syntax

Each property setting must use one of the flags in the table below. There is no space between each flag and its value. Note that certain flags are used only for JavaOS properties, while others are used only for system properties.

TABLE 4–7 Property Flags

Flag Syntax	Definition
`-d`*JavaOS_property=value*	Used to define a JavaOS property. The property name and value are stored in the JavaOS properties object. The names of properties set with this option always begin with `javaos`. Example: `-djavaos.kbd=UKPS2`
`-D`*system_property=value*	Used to define a system property. The system property and value are stored in the global system properties object. Example: `-Duser.timezone=PST`.
`-a`*JavaOS_property=value*	The JavaOS property is set to the given value if it has never been set before. If the property already has a value, then its current value is appended with a semicolon and the value. Example: `-ajavaos.printservice.lpd.printer=printer2` This example sets the property `javaos.printservice.lpd.printer` to `printer2` if the property has no current value. If the property already has a value (say, `printer1`), the new value is appended (`printer1;printer2`). This flag is useful if you are building the value of a property from multiple places using the `-i` option below. Note that this property can itself be a semicolon-separated list of values; the entire list is added or appended as described above.
`-A`*system_property=value*	This works just like the `-a` option, except that it sets system properties. Note that `-a` and `-A` are equivalent to `-d` and `-D`, except for the append behavior.
`-u`*JavaOS_property*	Used to undefine a JavaOS property.

TABLE 4–7 Property Flags *(continued)*

Flag Syntax	Definition
`-U`*system_property* `U`	Used to undefine a system property.
`-i`*HTTP_URL* `i`	The HTTP URL is expanded and the file referenced by the URL is itself evaluated as if it were a JavaOS command line. Properties are stored one per line in this file. One file can use `-i` to include another file. There is currently no check on infinite recursion.

Properties will be interpreted as the *JavaOS command line*, which is a formatted text string of any length that is interpreted when the JavaOS software boots. The command line can include:

- JavaOS and system properties
- Arguments to be used by the JavaOS software or an application launched by the JavaOS software

The syntax of the command line is as follows:

prop_setting1 prop_setting2... prop_settingn – arg1 arg2... argn

where *prop_setting* is a JavaOS property setting and *arg* is an argument. Property settings and arguments are separated by two contiguous hyphens.

Referencing a Properties File in the DHCP Vendor Options

This is the preferred method for setting JavaOS properties because `dhcptab` entries are limited to 255 characters.

Use the `-i` flag described in the preceding table in a `dhcptab` macro (see Chapter 2). The relevant line of the macro should have the following syntax:

```
JOScmd1=''-ihttp://hostname[:port_number]/filename''
```

The text file should be located in the web server root directory. It should contain property settings using the flags described in the preceding table. It can also include arguments. Each property setting and argument should be typed on a separate line, with the separator hyphens mentioned above on their own line, as follows:

```
-Duser.timezone=PST
-djavaos.login=false
--
```

http://www.sun.com

The following is a sample properties text file.

```
# Sample text file of JavaOS properties
##############################################################
# General setup
#############################################################

# Define the key that brings up the screen console (PRINTSCREEN
# is the default)
-djavaos.consoleHotKey=VK_PRINTSCREEN

# Set up values that will be returned by SNMP in the System MIB
-djavaos.snmpSysContact=JavaOS group, js_team@ignacio
-djavaos.snmpSysLocation=JavaSoft

##############################################################
# Locale-specific settings
#############################################################

# Set Server for /REMOTE file system
-ajavaos.mountlist=fred:/export/root/javaos/classes|/REMOTE
# We"re all on the US West Coast...
-Duser.timezone=PST

# Set list of countries we support, English first
# (The order doesn"t matter, but the Login window shows them
# in the order in which they appear in the property list.)
-djavaos.loginLocaleList=en_US

# Add the European Locales
-ajavaos.loginLocaleList=fr_FR;de_DE;it_IT;sv_SE;es_ES

# Add the Asian Locales
-ajavaos.loginLocaleList=ja_JP;ko_KR;zh_CN;zh_TW
##############################################################
# FONT-specific settings
##################################################################

# Set the server location for where to find fonts for the
# /FONTS directory
-ajavaos.mountlist=fred:/export/root/javaos/fonts|/FONTS


################################################################
# Printing properties
##################################################################

# Always show these printers
-djavaos.printdialog.alwaysShowPrinters=NIS:dirk@fred;
NIS:rita@scorpio

# The name of the NIS map for printers (this is the default, so it"s
# not really necessary)

-djavaos.printservice.NIS.mapname=printers.conf
```

```
# Set up the hardware parameters for locally connected printers
-djavaos.printservice.local.params.serial-Serial=57600:8:1:none:hh

################################################################
# set doc.url
# This allows us to read the HotJava Browser User's Guide from
# a server.
#################################################################
-Ddoc.url=http://fred.eng/JavaOS/LunaApps/
```

CHAPTER 5

Dynamically Loading Applications

The JavaOS software supports dynamic delivery of the user's main application to the JavaStation computer by a web (HTTP) server after the initial boot of the JavaOS software. This chapter describes how to set up an application to be dynamically delivered.

- "Overview" on page 57
- "Setting Up Dynamic Delivery of an Application " on page 58

For more information on the JavaStation boot sequence, see Chapter 2.

Note - Dynamic delivery of a single application can be set up quickly using Netra j. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

Overview

Once the JavaOS software is invoked on the JavaStation computer, it can download the user application from an HTTP server on the network. This method of application delivery, called "dynamic loading," is distinct from static linking (see Chapter 6) in that the application is not bound to the JavaOS binary but is accessed by the JavaOS software after it boots.

An application that will be dynamically delivered must be bundled as an "application archive," which can be a Java archive (JAR) or zip file. The JavaOS software references the application archive by its URL, which enables the application archive to reside anywhere on the network accessible to the web server.

JavaOS Properties

To set up an application to be delivered dynamically, you must set one or more of the following JavaOS properties:

- `javaos.apps`
- `javaos.mainProgram`
- `javaos.mainHomeprop`
- `javaos.mainZip`

This chapter describes each property in detail and explains their possible settings. For complete instructions on setting JavaOS properties, see Chapter 4.

Setting Up Dynamic Delivery of an Application

This section describes the procedures for setting up an application to be delivered dynamically to JavaStation computers at bootup.

You can configure dynamic delivery in two ways:

- A single, “fixed” application is launched automatically when the JavaOS software boots.
- The user is presented with a list of applications to choose from in the JavaStation AppLoader window, shown below.

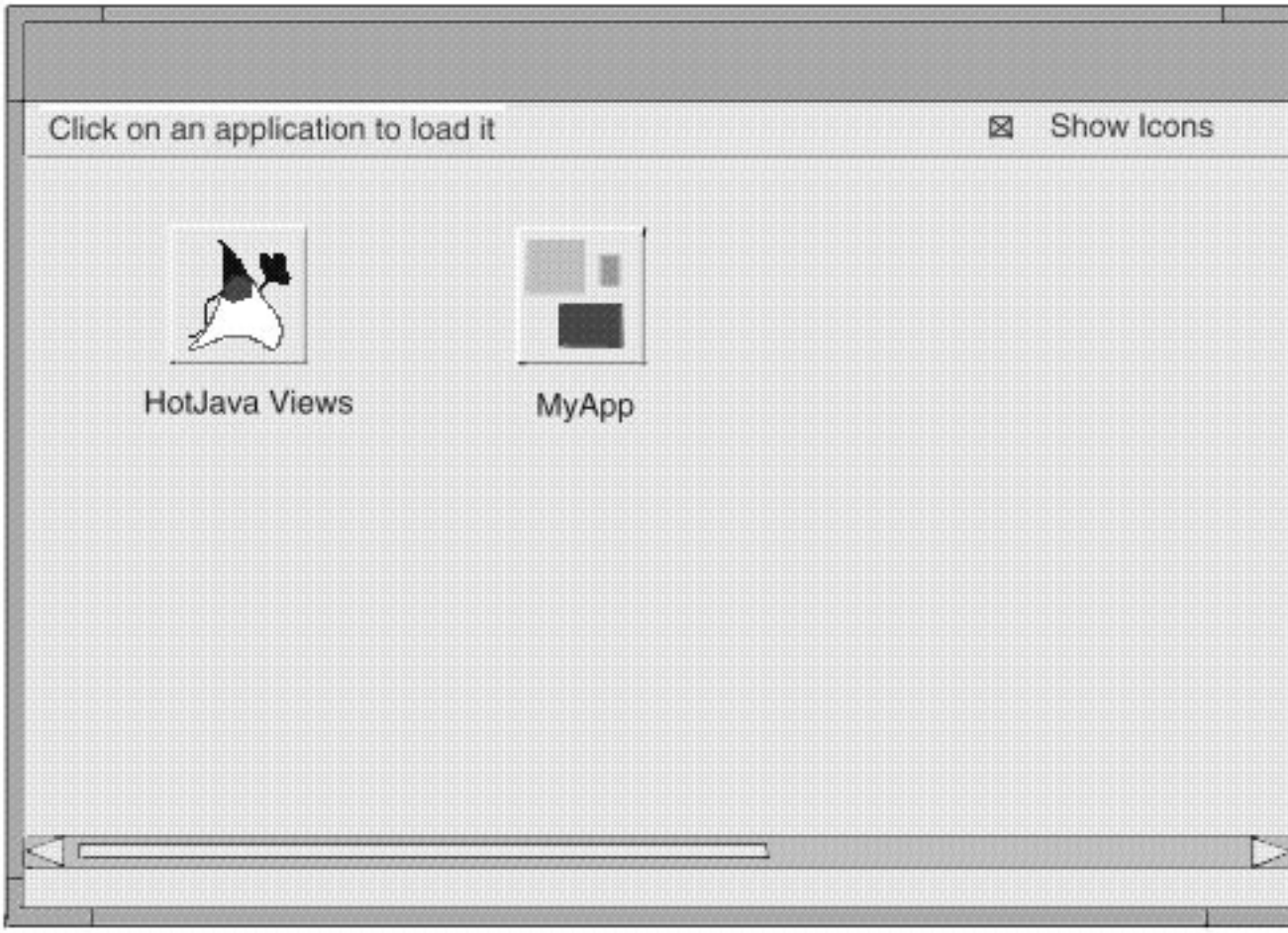


Figure 5–1 AppLoader Window on the JavaStation Screen

Note - Dynamic delivery does not provide an environment to enable multiple applications to run on a single virtual machine. Only one application can run at a time.

▼ To Create an Archive

1. **Verify that your project directory contains both a `classes` and a `lib` directory and that all application classes reside in the `classes` directory.**

   Applications typically comprise class, text, graphics, HTML, and property files. Many application hierarchies are organized so that all the class files are in the

`classes` directory and everything else is in the `lib` directory. This policy is enforced for dynamic loading to ensure the application classes can be found and added to the system `CLASSPATH`.

2. **Remove any unnecessary files from the `classes` and `lib` directories.**

   It is important to conserve memory on the JavaStation system.

3. **Create a JAR file containing the `classes` and `lib` directories.**

   The JAR utility is provided in the Java Developer's Kit™ (JDK™). Use the following syntax:

   `jar -cf` *archive_name* `classes lib`

   For example, to create a JAR file of the HotJava Views application:

   ```
   % jar -cf hotjava.jar classes lib
   ```

   Note - You can also use the zip utility (not provided in the JDK) to create archives.

4. **Place the archive in a directory that is accessible to the HTTP server.**

▼ To Set Up Dynamic Delivery

1. **Create an archive of the application.**

   An application archive can be a zip or JAR file. See "To Create an Archive" on page 59.

2. **Determine whether you will deliver a single "fixed" application or a dialog with a list from which the user can select an application.**

- For the first option, see "To Deliver a Single Application " on page 60.
- For the second option, see "To Set Up AppLoader With a List of Applications " on page 62.

▼ To Deliver a Single Application

1. **Create the application archive.**

   See "To Create an Archive" on page 59.

2. **Determine how your application will access ancillary files as it runs on the JavaStation computer, and set the `javaos.mainHomeprop` property if needed.**

   Many applications need to access ancillary files at runtime (such as images and configuration files). Typically, applications access these files relative to their installation directory. For example, when HotJava Browser runs on Solaris, it determines the environment variable `HOTJAVA_HOME` and then sets the `hotjava.home` property to that value. HotJava then uses this property to locate the additional files it needs at runtime.

   In the JavaOS environment an application is installed in the ROM file system at boot time, and there is no startup script to determine where the application resides. To prevent you from having to hard-code the location into your application, the JavaOS software sets a property of your choosing to this location. You provide the name of the property in the JavaOS property `javaos.mainHomeprop`. Thus, for HotJava Browser, the property is the `hotjava.home` property and it is specified as follows:

   ```
   -djavaos.mainHomeprop=hotjava.home
   ```

   At boot time, the JavaOS software sets this property (`hotjava.home`) to where the application is "installed" in the ROM file system. Internally your application may use this property to determine the location of a data file. For example, HotJava Browser could use the `hotjava.home` property to locate an image file (error checking removed for brevity):

   ```
   String imagePath = System.getProperty("hotjava.home") +File.separator + "image.gif";
   ```

   As you set up an application for dynamic delivery, determine whether you will need to use the `javaos.mainHomeprop` mechanism at all – maybe your application does not have ancillary files, or maybe you can use `Class.getResource()` to find resources. If you do use this mechanism, pick the property name for the JavaOS software to set, and use that property internally.

3. **Set the JavaOS properties listed in the following table.**

TABLE 5–1 JavaOS Properties Required to Load a Single Application

Property	Description
`javaos.mainProgram`	The name of the application's `main` class.
`javaos.mainZip`	The name of the application archive. The archive name is absolute. Example: `http://amber.eng/javaos/hotjava.`jar.

In the following example, the JavaOS properties listed in the preceding table and `javaos.mainHomeprop` are set to launch HotJava Views. In this example, `amber.eng/javaos` are the host and directory names.

```
-djavaos.mainProgram=sunw.hotjava.Main
-djavaos.mainZip=http://amber.eng/javaos/hotjava.zip
-djavaos.mainHomeprop=hotjava.home
```

JavaOS properties are delivered to the JavaStation in the DHCP options or in a file referenced in the DHCP options. For complete instructions on setting JavaOS properties, see Chapter 4.

▼ To Set Up AppLoader With a List of Applications

1. **Create an archive for each application.**

   See "To Create an Archive" on page 59.

2. **Create an application tag file.**

   The application tag file is an HTML file containing application "tags," which are sets of information on each application. AppLoader parses the application tag file and presents a window displaying a list of applications for the user to select from.

   Each application tag must contain the attributes listed in the following table.

TABLE 5–2 Application Tag Attributes

Attribute	Description
`code`	The name of the application's `main` class.
`name`	The name to be displayed in the Application Launcher list window. This attribute is optional. By default, AppLoader displays the path to the archived zip file.
`archive`	The name of the application archive. The archive name can be either relative (`hotjava.jar`) to the directory containing the application tag file or absolute (`http://amber.eng/javaos/hotjava.`jar).

An application tag can also have the parameters listed below.

TABLE 5–3 Application Tag Parameters (Optional)

Parameter	Description
`homeprop`	The name of the property specifying the application's root directory. This parameter sets the `javaos.mainHomeprop` property. See "To Deliver a Single Application " on page 60 for more information.
`args`	Enables you to pass an arbitrary list of arguments to the `main()` of your application. Within the list, items are separated with a space, just as on the command line.

The following is a sample application tag file.

```
<DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Applications</title>
  </head>
  <body>
      <h1>Applications</h1>

<application code="sunw.hotjava.Main" name="HotJava Browser" archive="hotjava.jar">
<param name="homeprop" value="hotjava.home">
<param name="args" value="http://java.sun.com/">
</application>

<application code="sunw.hotjava.Main" name="HotJava Views" archive=jdt/html/classes/Selector.jar>
<param name="homeprop" value="hotjava.home">
<param name="args" value="http://http_server/apps/selector.init">
</application>

<application code="sunw.applet.AppletViewer" name="AppletViewer" archive="appletviewer.zip">
<param name="args" value="http://http_server/taos/taos.html">
</application>

  </body>
</html>
```

3. **Place the application tag file in your HTTP server"s document root directory or a subdirectory under it.**

4. **Use the JavaOS property `javaos.apps` to specify the location of the application tag file.**

   For example:

   ```
   -djavaos.apps=http://amber.eng/javaos/applauncher.html
   ```

CHAPTER 6

Statically Linking an Application to the JavaOS Image

This chapter describes how to use the Static Linking Kit (SLK) to bind your application to the JavaOS image delivered to JavaStation clients at bootup. The SLK is included in the JavaStation client software.

Static Link Overview

The SLK is a special JavaOS image to which you can link your own application files. When the re-linked JavaOS is delivered to the JavaStation clients, the user application is started automatically.

Using the SLK ensures a faster, simpler boot process and trivial application configuration. The SLK is also required for booting tower model JavaStation computers over a PPP/modem connection.

Using SLK

To use the SLK, you must prepare a directory tree containing JavaOS files and application files and then run the `link_javaos` utility to create a new JavaOS version incorporating the application files. This procedure is described in the following section.

▼ To Statically Link a Custom Application to the JavaOS Image

1. **Run** `pkginfo` **to make sure the required Solaris packages are installed on your Solaris system:**

```
% pkginfo SUNWarc SUNWbtool SUNWcsu SUNWlibm SUNWsprot SUNWtoo
   system SUNWarc         Archive Libraries
   system SUNWbtool         CCS tools bundled with SunOS
   system SUNWcsu         Core Solaris, (Usr)
   system SUNWlibm         Sun WorkShop Bundled libm
   system SUNWsprot         Solaris Bundled tools
   system SUNWtoo         Programming Tools
```

   If these packages are not available, install them from your Solaris CD-ROM or contact a Sun sales representative.

2. **Untar the file** `/opt/SUNWjsos/lib/`***hardware_type***`.tar` **in a new working directory.**

   The SLK tar files are installed with the JavaStation client software. *hardware_type* refers to the JavaStation model and can be one of the following:

TABLE 6–1 JavaStation Hardware Types

Model	Hardware Type
Brick model	`JDM1`
Tower model	`JSIIep`

   The following example and the remainder of this procedure use `JSIIep` as the hardware type.

```
% mkdir work
% cd work
% tar xvf /opt/SUNWjsos/lib/lib.JSIIep.tar .
```

3. **Create the directories** `classes` **and** `lib` **under the directory** `work/lib.JSIIep.`

```
% cd work/lib.JSIIep
   % mkdir classes lib
```

4. **Copy your application class files and other required resources to the `classes` and `lib` directories.**

   Copy all `CLASSPATH`-loadable files to the `classes` directory and all non-class files to the `lib` directory.

   For example:

   ```
   % cp -r CLASSPATH_loadable_files classes/.
   % cp -r non_class_files lib/.
   ```

   Alternatively, you can create symbolic links to the application build area.

5. **Create a file called `javaos.properties` in the `lib` directory that contains JavaOS properties and other properties to be set before the application executes.**

   At minimum, this file should include the property `javaos.mainProgram`, which points to the class containing the `main` entry point of your application. This property is set as follows:

   ```
   javaos.mainProgram=main_class
   ```

   For example:

   ```
   javaos.mainProgram=Orion.main
   ```

   For a description of all JavaOS properties, refer to Chapter 4.

6. **Use `link_javaos` to link your application to the JavaOS image.**

   Usage for `link_javaos` is as follows.

   ```
   ./link_javaos [-sun | -gnu]
   ```

where:

- `-sun` (default) uses the Solaris assembler/linker tools found in `/usr/ccs/bin`
- `-gnu` uses GNU tools, which must be installed in `/opt/gnu`

In some cases this step can take a very long time, up to 20 minutes.

The `link_javaos` utility does several verifications and then builds a new `javaos` binary file that incorporates your application files. The table below shows error messages that may be displayed during each step in the process.

TABLE 6–2 Error Messages During the `link_javaos` Process

`javaos_link` **Action**	**Error Messages**
Verifies your host system is running Solaris 2.5 or 2.6	If your system is running a different version of Solaris, the following message is displayed. `Error: Linking "JavaOS" requires Solaris 2.5 or 2.6`

Verifies the specified tools have been properly installed	If `-sun` was selected (or left default) and the required tools directory is not found, the following message is displayed:
	`Error: The required tools directory (/usr/ccs) was not found`
	If `-gnu` was selected and the GNU tools directory is not found, the following message is displayed:
	`Error: The required tools directory (/opt/gnu) was not found`
	The GNU tools can be obtained from `www.gnu.org`. The minimum GNU packages required are `binutils`, `make`, and `gcc`.
	Note that the SLK build scripts will always look in `/opt/gnu` for the GNU tools. Some systems have the GNU compiler and tools installed in the directory `/usr/local`.

Verifies the available Java compiler is version 1.1 or later	If it is not available, the following message is displayed. `Error: Java version 1.1 or greater is required`
Verifies the file `javaos.properties` exists	If not found, the following message is displayed: `Error: ’./lib/javaos.properties’ was not found.` This file is required to specify the `javaos.mainProgram` property.
Creates resource ROM files	No specific printed errors.
Creates application `mclass`	No specific printed errors.
Creates ROM versions of all classes	No specific printed errors.
Adds new ROM data to archive	No specific printed errors.

TABLE 6–2 Error Messages During the `link_javaos` Process *(continued)*

`javaos_link` **Action**	**Error Messages**
Links `javaos` binary file	If the following message appears: `Error: 'javaos' was not properly created due to previous errors.` it probably indicates unresolved external references from missing libraries to missing Java classes.

Compresses `javaos` binary file	If the compression succeeds, the following message is displayed:
	`...Compressed "javaos" successfully created.`
	If the compression fails, the following message is displayed:
	`...*Uncompressed* "javaos" created.`
	`Unable to produce compressed "javaos" image.`
	`Note: While your JavaStation may be able to boot with 'javaos', it may not be suitable for loading into flash memory.`

A successful `link_javaos` build yields a single compressed `javaos` file that incorporates your application.

You can now set up `javaos` to be delivered to JavaStation computers during the boot sequence as described in Chapter 2.

Below is an example of output from a successful `link_javaos` build.

```
$ link_javaos
Static-Linking-Kit:

(1) Verifying SLK build environment  ...
Building with Solaris tools (as/ld/ar) ...
Statically linking "JavaOS" on a 2.5 system ...

(2) Creating resource ROM files ...
Creating "classes.ROM.o" ...
Creating "others.ROM.o" ...

(3) Creating application 'mclass' ...

(4) ROMizing all classes ...

(5) Assembling ROMizer output ...
```

```
(6) Adding the new ROMizer data to archive ...

(7) Linking "./javaos" binary ...

(8) Compressing "./javaos" binary ...

(9) ... Compressed "./javaos" successfully created.

$
```

CHAPTER 7

HotJava Browser and HotJava Views

This chapter describes the default JavaStation user applications included with the JavaStation client software.

- "HotJava Browser" on page 71
- "HotJava Views " on page 71

Netra j, the web-based JavaStation boot solution, includes a module for easy configuration of HotJava Views. For information on Netra j, go to `http://www.sun.com/netra-j`.

HotJava Browser

HotJava Browser is a highly customizable, modular browser written entirely in the Java programming language. HotJava Browser's small footprint makes it an ideal scalable solution for JavaStation deployment.

HotJava Browser can be deployed to the JavaStation computers via static linking (Chapter 6) or dynamic loading (Chapter 5).

HotJava Views

HotJava Views includes the browser and a set of office productivity tools. HotJava Views offers the following integrated components:

- *Selector* - An environment manager with a pushbutton interface for switching between applications.

- *MailView* - An IMAP4 mail client for composing, sending, and saving messages and handling a variety of attachments.
- *CalendarView* - A calendar client for managing personal and group calendars
- *NameView* - An enterprise name directory service client that retrieves and displays a configurable set of fields and enables contact via email, URLs, and calendar data
- *WebView* - An HTML 3.2-capable web browser (URL access can be restricted by the system administrator)

Like HotJava Browser, HotJava Views can be deployed to the JavaStation computers via static linking (Chapter 6) or dynamic loading (Chapter 5).

HotJava Views Model

HotJava Views enables the zero client-administration network computer and also attempts to minimize server-side administration. Users are organized into groups, and each group has its own profile, or set of properties.

Through HotJava Views Administration in Netra j, you can define groups of users that share client properties, specify applets to appear in the Selector, specify any sliding panels that appear from the edges of the screen, administer other properties that affect the user"s desktop, and specify properties for particular network computers.

When the JavaStation client boots, a URL is passed to the Selector. The URL points to an initial configuration file. Once the Selector locates the web server, it loads HotJava Views' set of property files.

Properties

HotJava Views is controlled by a set of eight property files at the group, user, and client levels.

- *User properties* - User properties are stored in the user's home directory. Initial group properties are overridden by the user's individual property file.
- *Group properties* - Each user is normally a member of a group and inherits the group properties. Group properties are usually the main source of the final properties. Users who are not members of a group inherit the group properties of the group currently designated as the "default" group. There are both initial and final group property files.
- *Client properties* - Client properties are specific to a given network computer. They typically control a few items, such as the default printer, that are specific to the physical location of the JavaStation.

CHAPTER 8

JavaStation PPP-Modem Dialup

This chapter describes how to set up JavaStation computers to boot over a modem connection.

PPP-Modem Overview

Tower model computers contain enough flash random-access memory (RAM) to hold a bootable copy of the JavaOS binary file. On these JavaStation models, support for Point-to-Point Protocol (PPP) and a modem dialer is included to enable the JavaStation computers to be deployed in a wide-area network (WAN) environment. PPP is not supported on brick model JavaStation computers because they have no flash RAM and must download the JavaOS software across the network each time they boot.

PPP-Modem Requirements

In order for the JavaStation computer to boot over PPP, the following must be true:

- A valid copy of the JavaOS image must exist in flash RAM.
- The user application must be statically linked to the JavaOS image (see Chapter 6).
- The JavaStation computer's Ethernet connection must be disconnected.
- An external serial modem must be configured.

As the JavaStation computer boots, it checks to see if it has a valid copy of the JavaOS image in its flash RAM and then checks to see if an Ethernet cable is present. If the JavaOS image in flash RAM is valid and no Ethernet cable is plugged into the system, the JavaStation computer will open a dialog window enabling the user to initiate the dialup/PPP process.

Figure 8–1 JavaOS PPP Dialer Window

The JavaStation user can open several windows from the PPP Dialer Window to set up the PPP configuration. The entire set of windows is described in Appendix A.

Once a connection has been made and the JavaStation computer is configured as a PPP client, operation proceeds as if the JavaStation computer were configured on a local area network. The JavaOS software leases an IP address and (potentially) downloads a Java user application over the PPP connection.

CHAPTER 9

JavaStation Peripherals

This chapter describes how to enable peripheral devices such as printers and touch screens for the JavaStation computer.

- "Configuring Printers" on page 75
- "Configuring a Touch Screen " on page 79

Note - JavaStation printers can be set up quickly using Netra j. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

Configuring Printers

The JavaStation computer is able to access:

- Serial and parallel printers that are directly attached to the JavaStation
- Printers located on the network

Printer access is configured using JavaOS properties. The JavaOS properties used to set up each type of printer are listed below.

TABLE 9–1 JavaOS Properties Used to Set Up JavaStation Printers

Printer Type	JavaOS Properties
NIS network printer	■ `javaos.printers.selected` ■ (Optional) `javaos.printservice.NIS.mapname`
lpd network printer	■ `javaos.printers.selected` ■ `javaos.printservice.lpd.printers`
Local serial printer	javaos.printservice.local.params.serial-*port*
Local parallel printer	javaos.printservice.local.params.parallel-*port*

This chapter describes the properties in the preceding table in detail and explains their possible settings. For complete instructions on setting JavaOS properties, see "Setting Properties" on page 52.

NIS Network Printers

The JavaOS software receives NIS printer names in a JavaOS property and relies on the NIS map `printers.conf.byname` for printer addresses. To set up NIS printer access for the JavaStation computer, use the following procedure.

▼ To Set Up NIS Printer Access

1. **Use the `javaos.printers.selected` property to set printer names.**

   This property is a semicolon-separated list of entries with the syntax *print_service*:*printer@server*. For example:

   ```
   -djavaos.printers.selected=NIS:mde@mde-host;NIS:mdecolor@mde-host
   ```

   The above example specifies that `mde` and `mdecolor` are NIS printers available to the JavaStation computer, and that `mde-host` is their server.

2. **Set up an NIS printer map.**

   The `printers.conf.byname` NIS map is the default NIS map for printers for the JavaStation computer.

   - If `printers.conf.byname` already exists, you do not need to do anything.
   - If the name of the NIS printer map is something other than `printers.conf.byname`, you can configure the JavaOS software to use the

new map name by setting the `javaos.printservice.NIS.mapname` property. For example:

```
-djavaos.printservice.NIS.mapname=js.nismap
```

- Whenever you add or remove printers, follow the procedure below to update the NIS map (must be done every time).

▼ To Create an NIS Printer Map

1. **On the NIS server, use Admintool to set up the printers.**

   Refer to the chapter titled "Managing Printing Services" in the *Solaris 2.6 System Administrator Guide.*

2. **On the NIS server, become root and type the following commands to push the map out.**

```
# cd /var/yp
  # /usr/ccs/bin/make -f /var/yp/Makefile -f /usr/lib/print/Makefile.yp \
  printers.conf.byname
```

lpd Network Printers

You can use two different properties to specify lpd printer names to the JavaOS software.

▼ To Set Up lpd Printer Access

1. **Use either** `javaos.printers.selected` **or** `javaos.printservice.lpd.printers` **to list the lpd printers accessible to the JavaStation computer.**

   - The `javaos.printers.selected` property is a semicolon-separated list of entries with the syntax *print_service*:*printer@server*. It is a system property that must be set using the `-D` flag. For example:

```
-Djavaos.printers.selected=lpd:raw@konaprint
```

   The `javaos.printers.selected` property can list both NIS and lpd printers. For example:

```
-Djavaos.printers.selected=NIS:mde@mde-host;lpd:raw@konaprint
```

- Alternatively, you can use the `javaos.printservice.lpd.printers` property to specify lpd printers. The syntax for each printer is *printer@server*. For example:

```
-djavaos.printservice.lpd.printers=SPARCprinter@chaco
```

Local Printers

All JavaStation models support local printing over a serial port. Future JavaStation models will also support local printing over a parallel port. (However, this feature is not available at this writing).

▼ To Set Up a Local Serial Printer

1. **Use the `javaos.printservice.local.params.serial-`*port* property to set serial port transmission parameters.**

   The *port* portion of this property is the name of a serial port, which can be:

   - `SerialA` or `SerialB` for an on-board JavaStation serial port (`SerialB` is not available on any JavaStation model at this writing)
   - One of `SerialP1` - `SerialP8` for a serial port enabled by the multiport serial card (MPSC), which is not available on any JavaStation model at this writing

The syntax of the parameters is *baud_rate*:*data_bits*:*stop_bit*: *parity*: *flow_control*. For example:

```
-djavaos.printservice.local.params.serialSerialA=57600:8:1:none:hh
```

The serial parameters supported are listed under “Printing Properties” on page 47.

An application running on the JavaStation computer must use the CommPort API to communicate with the JavaStation serial port. To view the CommPort API, go to `http://www.sun.com/javastation`.

▼ To Set Up a Local Parallel Printer

1. **Use the `javaos.printservice.local.params.parallel-`*port* property to enable the parallel port.**

The *port* portion of this property is the name of the parallel port. At this writing, a parallel port is not available on any JavaStation model. To enable the parallel port, you would set this property to true as follows:

```
-djavaos.printservice.local.params.parallel-LPT1=true
```

An application running on the JavaStation computer must use the CommPort API to communicate with the JavaStation parallel port. To view the CommPort API, go to `http://www.sun.com/javastation`.

Configuring a Touch Screen

A touch screen can be connected to the JavaStation serial port for user input when the JavaStation computer is implemented as a kiosk.

Since the touch screen functions as a mouse, a facility outside the JavaOS software is needed to generate mouse events. A handler applet or application must be written, using the CommPort API and the MouseInput class, to move the cursor and generate button press events. To view the CommPort API and the MouseInput class, go to `http://www.sun.com/javastation`.

▼ To Set Up a Touch Screen

1. **Use the `javaos.printservice.local.params.serial-`*port* property to set serial port transmission parameters.**

   For instructions on setting this property to enable the serial port, see "To Set Up a Local Serial Printer " on page 78.

CHAPTER 10

Setting Locales and Adding Fonts

This chapter explains how to configure JavaStation computers that will be used in different languages.

Note - Most language settings can be configured quickly using Netra j. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

What You Must Configure

You must configure some or all of following features for JavaStation computers that will be operated in languages other than the default language, U.S. English.

TABLE 10–1 Localization Features

Feature	Description	Configuration Procedure
Mount Directories	Contain the resources required by the JavaOS software to support alternate locales.	"Setting Mount Directories " on page 84
Locale	Controls the language and font that appears in the user interface, help text, and error messages	"To Change the Locale Setting " on page 86
Keyboard	Controls the mechanical input of each character when it is typed by the user	"To Add a Localized Keyboard" on page 93
Font	Controls the appearance of characters typed by the user	"To Install and Configure Fonts" on page 88
Input Method	Controls how the user composes characters; must be configured for Chinese, Japanese, and Korean languages only; requires a localized version of Solaris	"To Set the Input Method" on page 97
File Encoding	Controls the character set used in files that are saved and the default character set for web pages displayed in a browser	"To Change the File Encoding Setting " on page 98
Document URL	Provides access to documentation for the HotJava Browser running on the JavaStation (not applicable if another application is used).	"Setting the HotJava Browser Document URL" on page 99

Overview and Examples

Localization configurations are set using JavaOS properties, which are delivered to the JavaStation computer when it boots. The following sections describe the JavaOS properties used for localization. For complete instructions on setting JavaOS properties, see Chapter 4.

JavaOS Properties for All Locales

The following JavaOS properties are required for all locales:

- `javaos.mountlist`
- `user.language`
- `javaos.loginLocaleList`

The following property is required for all locales except West European locales:

- `-Dfile.encoding`

The following property is required for all Asian locales:

- `-djavaos.im.url`

The following property is required if you are running HotJava Browser on the JavaStation computers:

- `-Ddoc.url=file:/REMOTE/hotjava`

Other properties may also be required for the locale you intend to support. Each feature description in this chapter indicates the associated property settings and the locales that require them.

Example Configurations

This section shows some sample locale configurations.

To enable localization for a French-speaking Swiss user:

```
-djavaos.loginLocaleList=fr;de;it
-Duser.language=fr
-Dfile.encoding=8859_1
-djavaos.kbd=SwissFrenchPS2
-djavaos.im.compose.deadkeys=false
-ajavaos.mountlist=server:/export/root/javaos/classes|/REMOTE
```

To enable localization for a Czech user:

```
-Duser.language=cs
-Dfile.encoding=8859_2
-djavaos.kbd=CzechPS2
-djavaos.im.compose.deadkeys=false
-ajavaos.mountlist=server:/export/root/javaos/classes|/REMOTE
```

To enable localization for a Korean user:

```
-Duser.language=ko
-Dfile.encoding=KSC5601
-djavaos.kbd=KoreanPS2
-djavaos.im.compose.deadkeys=false
-ajavaos.mountlist=server:/export/root/javaos/classes|/REMOTE;
server:/export/root/javaos/fonts|/FONTS
-ajavaos.im.url=iiimp://server:9010
```

To enable localization for a Thai user who enters text at a U.S. keyboard:

```
-Duser.language=th
-Dfile.encoding=TIS620
-djavaos.im.compose_th=TIS620
-djavaos.im.compose.deadkeys=false
-ajavaos.mountlist=server:/export/root/javaos/classes|/REMOTE; server:/export/root/javaos/fonts|/FONTS
```

Setting Mount Directories

You must set at least one mount directory to enable support for a locale other than U.S. English on the JavaStation computer.

When the JavaOS software boots on the JavaStation computer, it can mount some or all of the following directories on a server to obtain localization resources:

- Translations and localized property files are in `/export/root/javaos/classes`. This directory is mounted by the JavaStation computer as `/REMOTE`.
- Keyboard classes are in `/export/root/javaos/classes/sun/javaos`. If the `/REMOTE` directory above is mounted, then the JavaStation computer can access the keyboard class files also.
- Fonts for Asian localizations are in `/export/root/javaos/fonts`. This directory is mounted by the JavaStation computer as `/FONTS`.

▼ To Set Mount Directories

1. **Use the** `javaos.mountlist` **property to tell the JavaStation computer to mount the** `/REMOTE` **and** `/FONTS` **directories, as in this example:**

```
-ajavaos.mountlist=myserver:/export/root/javaos/fonts|/FONTS;
   myserver:/export/root/javaos/classes|/REMOTE
```

Note - `javaos.mountlist` should be set using the `-a` (append) flag so that no previous settings for this property will be overridden.

Setting the Locale

A locale setting is required to enable a language other than U.S. English to be used in the JavaStation user interface, help text, and error messages.

The JavaOS software supports the locales listed in the table below.

TABLE 10–2 JavaStation Locale Settings

Language	Locale Setting
Arabic	`ar`
Catalan	`ca`
Chinese	`zh`
Czech	`cs`
English	`en`
French	`fr`
German	`de`
Hebrew	`iw`
Hungarian	`hu`

TABLE 10–2 JavaStation Locale Settings *(continued)*

Language	Locale Setting
Italian	`it`
Japanese	`ja`
Korean	`ko`
Polish	`pl`
Portuguese	`pt`
Russian	`ru`
Spanish	`es`
Swedish	`sv`
Thai	`th`

The JavaOS software also supports the language variants shown in the table below.

TABLE 10–3 JavaStation Country Settings

Country	Country Setting
Israel	`IL`
Republic of China	`TW`
Thailand	`TH`
United States	`US`

▼ To Change the Locale Setting

1. Set the `user.language` **property:**

```
-Duser.language=locale
```

where *locale* is one of the JavaStation locale settings listed in the first table above.

2. **If needed, set the** `user.country` **property:**

```
-Duser.country=country
```

where *country* is one of the JavaStation country settings listed in the second table above.

For complete instructions on setting JavaOS properties, see Chapter 4.

Modifying the Languages Displayed at Login

The localized version of the JavaStation software supports all the locales described in "Setting the Locale" on page 85. At login, the user is asked to choose one of these locales.

If you want to restrict or add to the locales presented at login, especially if you will be running an application that is localized for a language not listed in "Setting the Locale" on page 85, you must modify the `javaos.loginLocaleList` property.

▼ To Modify the Languages Displayed at Login

1. **Set the following JavaOS property:**

```
-djavaos.loginLocaleList=locale-1;locale-2; ... ;locale-n
```

where each locale is defined using one of the two-letter codes that define `user.language` followed by (in some cases) an underscore character and one of the two-letter codes that define `user.country`.

The default value of this property is:

```
-djavaos.loginLocaleList=en_US
```

Thus by default U.S. English is the only language choice available at login.

Adding Fonts

Font support is automatic in JavaOS for all the languages in Table 10–2 except the following:

- Arabic

- Chinese Simplified
- Chinese Traditional
- Hebrew
- Japanese
- Korean
- Thai

Font sets for the above languages are provided in the JavaStation client software. To make any of these font sets available to JavaStation computers, you must set the `javaos.mountlist` property, as described in "To Make Fonts Available to JavaStation Computers " on page 92.

You do not need to install new fonts for any of the locales supported on the JavaStation computer, unless you would like to change the look of characters typed by the user. To install new fonts, use the procedure described in this section.

Note - The Arabic, Hebrew, and Thai locales also require keyboard support. The Chinese (Simplified and Traditional), Japanese, and Korean locales require keyboard and input method support. See "Overview and Examples" on page 83, "Adding a Keyboard" on page 92, and "Setting the Input Method " on page 96.

Overview

Font sets must reside on a network directory that is accessible to the JavaOS software via NFS. By default, this directory is `/export/root/javaos/fonts`. To install a font, you will go to the fonts directory and do the following:

- Add the font files (`.ttf` files). You can install any TrueType or TrueType-compliant font.
- Modify the file `FONTS.LST`, which maps font names recognized by the JavaOS software to the font file names on the server.
- Modify font property files in the subdirectory `lib`.

You then must make the font available to the JavaStation computers by setting the `javaos.mountlist` property to enable the JavaOS software to mount the fonts directory during boot-up.

▼ To Install and Configure Fonts

1. **Install the font files in the fonts directory.**

   By default, this directory is `/export/root/javaos/fonts` on the fonts server.

Follow the instructions that accompany the font package.

2. **Modify the `FONTS.LST` file, which maps font names recognized by the JavaOS software to the font file names you have installed.**

   `FONTS.LST` contains a list of one-line entries, each of which contains:

   font_name style `truetype` *file_name*

   where:

   font_name is the alias that the JavaOS software uses for the font. *style* is one of `PLAIN`, `BOLD`, `ITALIC`, and `BOLDITALIC`. *file_name* is the font file you have installed.

   Example:

```
hggothicb
PLAIN truetype HG-GothicB.ttf
heiseimin PLAIN truetype HeiseiMin-W3H.ttf
hgminchol PLAIN truetype HG-MinchoL.ttf
```

   In this example, three TrueType fonts have been installed. They are all of style PLAIN. The font file names in the *font_dir* directory are aliased to the names for use by the JavaOS software, as shown in the following table:

TABLE 10–4 Font Name-to-JavaOS Alias Mapping in Above Example

Font Name	Name for Use by JavaOS Software
`HG-GothicB.ttf`	`hggothicb`
`HeiseiMin-W3H.ttf`	`heiseimin`
`HG-MinchoL.ttf`	`hgminchol`

Note - *font_name* is not case-sensitive. The JavaOS software recognizes `HGGothicB`, `hggothicb`, and `HggothicB` as the same name.

---

3. **In the `lib` subdirectory, modify the `font.properties.`*locale* file.**

   locale is the locale that the font properties file is relevant for. For the English locale, the name `font.properties` (without a locale specification) is used.

   The font properties file has four sections.

- Section 1 defines where the new fonts are to be used in place of default system fonts.

  The server provides five default system fonts to JavaStation computers: serif, sansserif, monospaced, dialog, and dialoginput. Section 1 contains up to one line for each default system font, where each line uses the following syntax:

  system_font`.`*suffix*=*JavaOS_font_name*

  This line specifies that the font is available to JavaOS. If you want a new font (identified by its JavaOS font name) to override the system font, set *suffix* to 0. If you want the new font to be available in addition to the system font, set *suffix* to any other number (1 is a good choice).

  The following example makes the `hgminchol` font available in addition to the serif font:

  ```
  serif.1=hgminchol
  ```

  The serif font will be used for all English characters. The `hgminchol` font will be used for all Kana and Kanji characters.

- Section 2 makes each new font name available to the JavaOS software.

  Section 2 makes it possible for the JavaOS software to recognize the new font by its name, so that the font can be identified and used. Section 2 contains up to one line for each new font, where each line uses the following syntax:

  JavaOS_font_name`.0=`*JavaOS_font_name*

  The following example makes the `hgminchol` font name available to the JavaOS software:

  ```
  hgminchol.0=hgminchol
  ```

- Section 3 (optional) enables you to further alias the new font names.

  Section 3 contains up to one line for each new font, where each line uses the following syntax:

  `alias.`*new_name*=*JavaOS_font_name*

  The following example aliases the `hgminchol` font to the name “mincho.”

  ```
  alias.mincho=hgminchol
  ```

- Section 4 specifies the character set encoding of each font. Supported encodings are listed in the table below.

TABLE 10–5 Character Set Encodings Supported by the JavaOS Software

Encoding	Locales
8859_1	West European locales
8859_2	East European locales
8859_5	Russian
8859_6	Arabic
8859_8	Hebrew
GB2312	Chinese (PRC)
CNS11643	Chinese (Taiwan)
BIG5	Chinese (Taiwan)
Ja-EUC	Japanese
EUCJIS	Japanese
KSC5601	Korean
TIS620	Thai
Unicode	Large, universal character set

For each font, you must add a line with the following syntax:

`fontcharset.`*`font`*`.1=sun.io.CharToByte`*`encoding`*

Where *encoding* is one of the values in the table below.

The following example is for the monospaced font that is Unicode-encoded:

```
fontcharset.monospaced.1=sun.io.CharToByteUnicode
```

The following is an example font properties file for a server that has two new Unicode-encoded Japanese fonts. The new fonts will be available to the JavaOS software in addition to the system fonts. Thus both English and Japanese can be used on the JavaStation.

```
# Copyright (c) 1994-1996 by Sun Microsystems, Inc.
#
# AWT Font Properties for handling Japanese in the JavaOS
# environment using disk-based fonts
```

```
serif.1=hgminchol
sansserif.1=hggothicb
monospaced.1=hggothicb
dialog.1=hggothicb
dialoginput.1=hggothicb

hgminchol.0=hgminchol
hggothicb.0=hggothicb

alias.mincho=hgminchol
alias.gothic=hggothicb

fontcharset.serif.1=sun.io.CharToByteUnicode
fontcharset.sansserif.1=sun.io.CharToByteUnicode
fontcharset.monospaced.1=sun.io.CharToByteUnicode
fontcharset.dialog.1=sun.io.CharToByteUnicode
fontcharset.dialoginput.1=sun.io.CharToByteUnicode
```

▼ To Make Fonts Available to JavaStation Computers

1. **Set the `javaos.mountlist` property.**

   This property setting tells the JavaOS software to mount the fonts directory at startup:

   -ajavaos.mountlist=*host:font_dir*
   |/FONTS

   The default fonts directory is `/export/root/javaos/fonts`. To mount this directory on a server called `sunroom`, you would type:

   ```
   -ajavaos.mountlist=sunroom:/export/root/javaos/fonts|/FONTS
   ```

2. **Reboot JavaStation computers that need access to the new fonts.**

   To reboot a JavaStation computer, turn it off and then on.

Adding a Keyboard

The JavaOS software supports a number of PS/2 keyboards that may not have been supplied with your JavaStation computers. To support most locales, you must configure a new keyboard to replace the default keyboard, `USPS2`. As an exception, Latin accent characters and Arabic, Hebrew, and Thai characters can be typed at any

supported keyboard if one of the `javaos.im.compose` properties has been set. This document refers to the U.S. keyboard. See "Enabling Special Characters on the U.S. Keyboard" on page 94.

Note - The Arabic, Hebrew, and Thai locales also require font support. The Chinese (Simplified and Traditional), Japanese, and Korean locales require font and input method support. See "Overview and Examples" on page 83, "Adding Fonts" on page 87, and "Setting the Input Method " on page 96.

JavaOS supports the following keyboards:

- Arabic
- Belgian
- Bulgarian
- CanadianBi
- CanadianFr
- Chinese ROC
- Czech
- Danish
- Estonian
- French
- German
- Greek
- Hebrew
- Hungarian
- Italian
- Japanese
- Korean
- Latvian
- Lithuanian
- Netherlands
- Norwegian
- Polish
- Portugese
- Russian
- Slovakian
- Spanish
- SpanishLatin
- Swedish
- Swiss
- Thai
- Turkish
- UK
- US
- USInternatl

▼ To Add a Localized Keyboard

1. **Connect the keyboard to the JavaStation.**

2. **Set the `javaos.mountlist` property.**

   This property setting tells the JavaOS software to mount the locale directory at startup.

   ```
   -ajavaos.mountlist=host:localization_top_dir
   |/REMOTE
   ```

   By default, the locale directory is `/export/root/javaos/classes`. If you set `javaos.mountlist` as follows:

   ```
   -ajavaos.mountlist=sunroom:/export/root/javaos/classes|/REMOTE
   ```

   The JavaOS software mounts the directory `/export/root/javaos/classes/sun/javaos`. Note that if you are

specifying a FONTS directory as well as a REMOTE directory, the javaos.mountlist property is a list delimited by semicolons. For example:

```
-ajavaos.mountlist=sunroom:/export/root/javaos/fonts|/FONTS;
sunroom:/export/root/javaos/classes|/REMOTE
```

3. **Set the javaos.kbd property.**

   This property setting tells the JavaOS software the name of the keyboard file, which contains the keyboard mapping table. Keyboard files for all of the countries listed on the previous page are included in the JavaStation client software.

   -djavaos.kbd=*keyboard*

   The syntax of *keyboard* is *name*PS2, where *name* is one of the countries in the preceding list. For example, to add the Swedish keyboard:

   ```
   -djavaos.kbd=SwedishPS2
   ```

Enabling Special Characters on the U.S. Keyboard

Four javaos.im.compose properties enable you to modify the characters typed at the U.S. English keyboard. You can enable input of Latin accent characters or Arabic, Hebrew, or Thai characters.

- *Latin accent characters* – When the javaos.im.compose.deadkeys property is set to true, the following keys can be typed in combination with other keys to produce Latin accent characters.

  - ‘ (single quote)
  - ‘ (back single quote)
  - “ (double quote)
  - ^ (circumflex)

  For example, pressing the ‘ key plus the “a” key produces an á. This feature is commonly used in European locales.

  Pressing one of the above keys twice produces its normal value.

- *Arabic, Hebrew, or Thai* – The following properties enable Arabic, Hebrew, or Thai characters to be produced at the U.S. English keyboard:

  - javaos.im.compose_ar=ISO8859_6
  - javaos.im.compose_iw=ISO8859_8
  - javaos.im.compose_th=TIS620

These properties also enable a status window on the JavaStation screen that shows the current direction of text input (right-to-left or left- to-right).

You can use one of these properties and also use the `javaos.kbd` property to set up a native keyboard. For example, you can set `-djavaos.kbd=ArabicPS2` to enable the Arabic keyboard and also set the `-djavaos.im.compose_ar` property to enable Arabic characters to be typed at a U.S. keyboard. If the `javaos.kbd` property is set, the Alt-Graph key sequence is used to toggle the input direction. If the compose property is set, the Ctrl-t key sequence is used to toggle the input direction. If both properties are set, both key sequences are enabled.

Note - The Arabic, Hebrew, and Thai locales also require font support. See "Overview and Examples" on page 83 and "Adding Fonts" on page 87 for more information.

▼ To Enable Latin Accent Characters on the U.S. Keyboard

1. **Set the following JavaOS property**:

```
-djavaos.im.compose.deadkeys=true
```

▼ To Enable Arabic Characters on the U.S. Keyboard

1. **Set the following JavaOS property**:

```
-djavaos.im.compose_ar=ISO8859_6
```

ISO8859_6 is the code for the Arabic character set. Once this property is set, the Ctrl-t key sequences toggles the JavaStation keyboard between U.S. and Arabic modes.

▼ To Enable Hebrew Characters on the U.S. Keyboard

1. **Set the following JavaOS property**:

```
-djavaos.im.compose_iw=ISO8859_8
```

ISO8859_8 is the code for the Hebrew character set. Once this property is set, the Ctrl-t key sequences toggles the JavaStation keyboard between U.S. and Hebrew modes.

▼ To Enable Thai Characters on the U.S. Keyboard

1. **Set the following JavaOS property:**

```
-djavaos.im.compose_th=TIS620
```

TIS620 is the code for the Thai character set. Once this property is set, the Ctrl-t key sequences toggles the JavaStation keyboard between U.S. and Thai modes.

Setting the Input Method

An input method controls how JavaStation users in Chinese (Simplified and Traditional), Japanese, and Korean locales will compose characters at the keyboard. The input method is administered by a language engine running on a Solaris system in the JavaStation network. Localized versions of Solaris offer language engines to support the input methods listed below.

Note - To fully support one of the preceding locales, you must also set up keyboard and font support. See "What You Must Configure" on page 81, "Adding Fonts" on page 87 and "Adding a Keyboard" on page 92 for more information.

TABLE 10–6 Input Methods by Language

Language	Input Methods
Chinese Simplified	■ Location ■ Double Pinyin ■ Stroke ■ Full Pinyin ■ Golden Input ■ Intelligent Pinyin ■ Chinese Symbol
Chinese Traditional	■ TsangChieh ■ ChuYin ■ I-Tien ■ Telecode ■ ChienI ■ NeiMa ■ ChuanHsing
Korean	■ 2-Bulsik ■ Hanja Input ■ Special Character ■ Code Value
Japanese	■ CS00 ■ Wnn6 (Solaris 2.6 only)

Note - Arabic, Hebrew, and Thai characters are also composed using an input method. However, support for these input methods is automatic in the JavaOS software and does not require additional configuration.

▼ To Set the Input Method

1. **Set the `javaos.im.url` property.**

   This property enables JavaStation computers to access to the Solaris machine running the language engine for the input method. To support multiple input methods for all the JavaStation computers in your network, use a semicolon-delimited list of Solaris machines. Set this property as follows:

   `-djavaos.im.url=iiimp://`*`hostname`*`:`*`port`*`;`*`hostname`*`:`*`port`*`...`

   where:

   hostname is the Solaris host running the language engine. *port* is the port at which the engine is located. By default, the JavaStation computers use port 9010.

Note - To make an input method server accessible to only a subset of the JavaStation computers in your network, you will need to create a unique macro for these computers in the `dhcptab` file. See "Sample `dhcptab` File " on page 25.

2. **(Optional) Set the `javaos.im.status.fixpopup` property.**

   Setting this property to true, as follows, enables a pop-up window with input method status information to appear on the JavaStation monitor.

   ```
   -djavaos.im.status.fixpopup=true
   ```

3. **(Optional) Set the `javaos.im.lookup.button` property.**

   This property controls how the JavaStation user selects characters when using an input method. If false, when the list of candidate characters is displayed, letters are used to indicate each choice. If true, letters are replaced with buttons so that the user must click on a button to pick a choice. Note that enabling this option degrades the performance of user input.

   ```
   -djavaos.im.lookup.button=true
   ```

4. **Reboot the JavaStation computer.**

   To reboot a JavaStation computer, turn it off and then on.

Changing the File Encoding Setting

If the locale you have chosen does not use the 8859_1 character set (Roman alphabet), you must change the `file.encoding` property to describe which character set will be used when files are saved. Supported character sets are listed in Table 10–5.

▼ To Change the File Encoding Setting

1. **Set the following JavaOS system property:**

   `-Dfile.encoding=`*`encoding`*

where encoding is one of the settings listed in Table 10–5.

Setting the HotJava Browser Document URL

If Sun's HotJava Browser will run on the JavaStation computers, you must set the `doc.url` property to make HotJava Browser documentation accessible to JavaStation users.

▼ To Set the Document URL

1. **Set the following JavaOS system property:**

```
-Ddoc.url=file:/REMOTE/hotjava
```

HotJava Browser documentation is translated for all locales. If the `doc.url` property is set exactly as above, HotJava Browser automatically locates the translation for the current locale.

APPENDIX **A**

JavaStation User Setup Forms

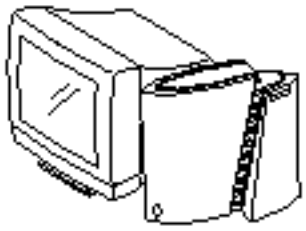
This appendix contains forms that show JavaStation users how to boot their JavaStation computers. The first form shows how to boot using the traditional boot sequence. The second form shows how to boot using Point-to-Point Protocol (PPP) modem dialup.

Copy a form for each user and complete the required information. Be sure to add contact information at the bottom of the form. Deliver the form to the user with the JavaStation system.

Welcome to Your JavaStation Computer

1. Set up your hardware.

2. Turn on the JavaStation computer and monitor.



3. Log in.

Your user name is _____

Type it and press Enter.

Your password is _____

As you type your password, asterisks appear for each letter. This enables you to keep your password a secret.

Problems?

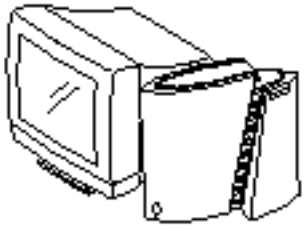
Contact:

Phone:

Welcome to Your JavaStation Computer

1. Set up your hardware.

2. Connect the modem cable to the JavaStation serial port. Disconnect the Ethernet cable from the JavaStation Ethernet port.



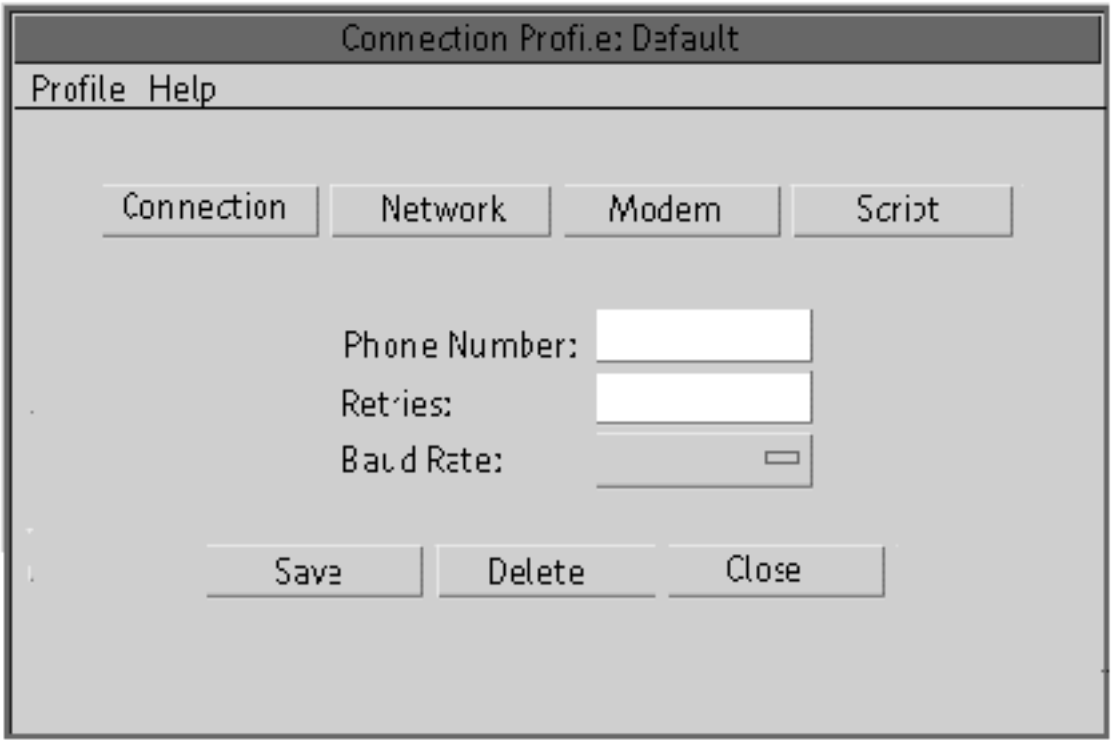
3. Turn on the JavaStation computer and monitor. Turn on the modem.

4. Dial in.

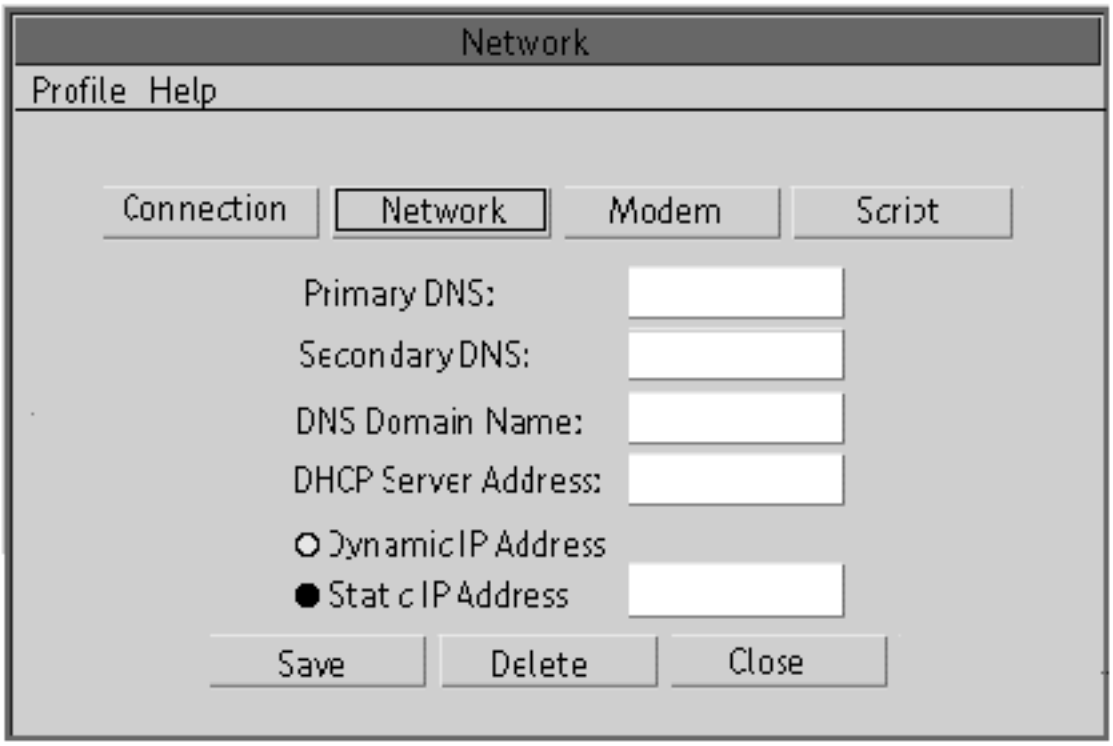
Set options if needed, including the phone number that will be dialed. Click Connect when done.

To set options:

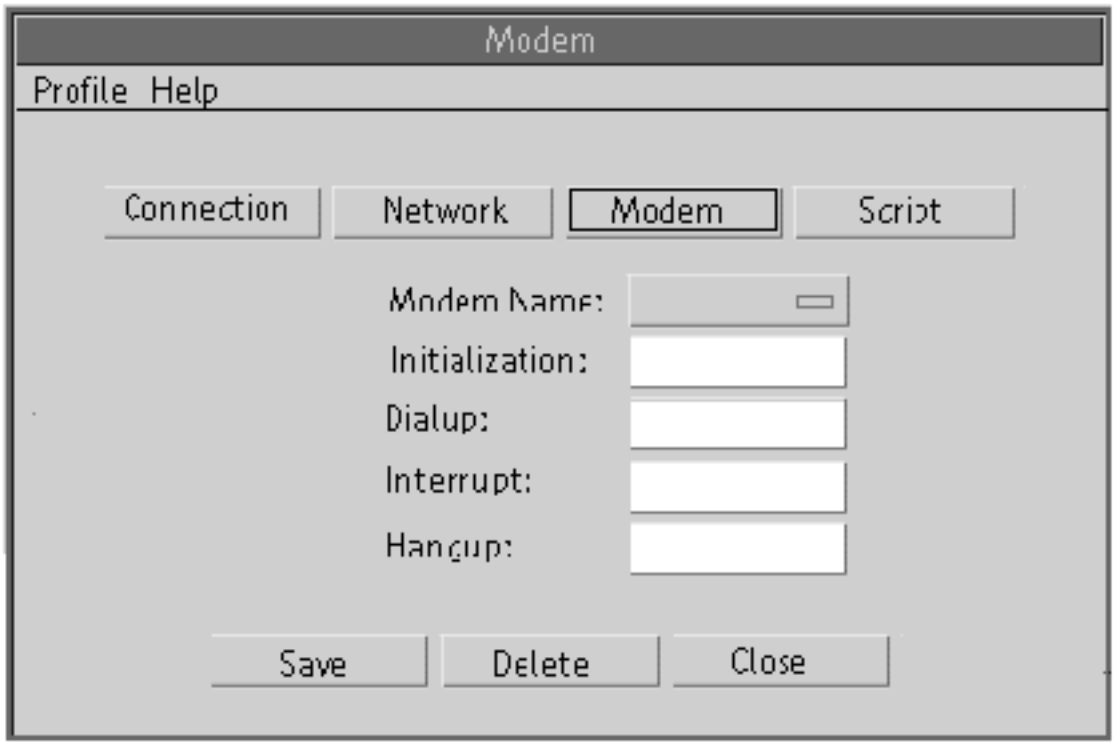
Click Options, then click an Options radio button. Select your options and click Save. Click Close to close the Options window.



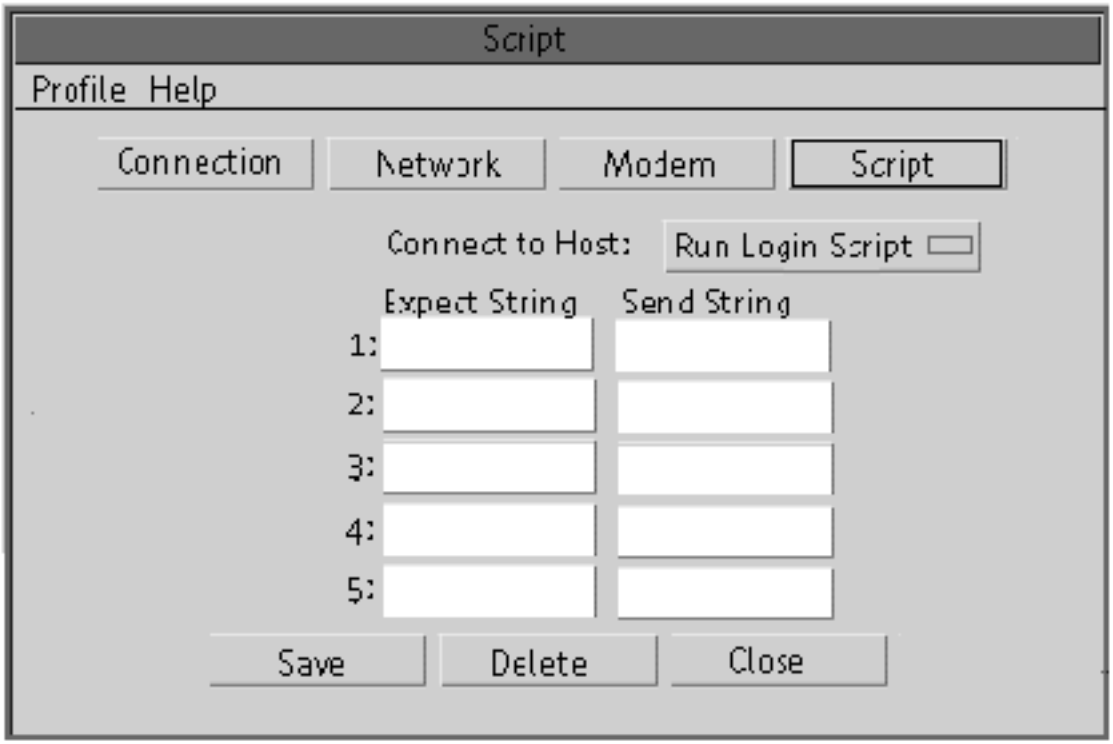
Phone number: PPP server modem's phone number
Retries: Number of retries attempted
Baud rate: JavaStation serial port baud rate



Primary DNS: Primary JavaStation DNS server
Secondary DNS: Secondary JavaStation DNS server
DNS Domain Name: Domain in which JavaStation computer resides
DHCP Server Address: JavaStation DHCP server IP address
Dynamic IP Address: Select if JavaStation IP address is assigned dynamically
Static IP Address: Select if JavaStation IP address is assigned permanently



Modem Name: Modem brand name
Initialization: Modem initialization command string (vendor-specific)
Dialup: Modem dial command string
Interrupt: Modem command string (can be set if Modem Name is set to "Other")
Hangup: Modem hangup command string (can be set if Modem Name is set to "Other")



Connect to Host: A set of expect/send pairs to dial and connect to a PPP service
Expect String: String received from server
Send String: String sent to server

Problems?

Contact:

Phone:

APPENDIX **B**

Troubleshooting the Boot Process

Troubleshooting the JavaStation software and network environment can be complex, since everything the JavaStation computer "knows" comes from somewhere on the network. This section describes how to access network information on the JavaStation boot process and shows an example network trace of a JavaStation boot.

- "Troubleshooting Process" on page 109
- "Tower JavaStation Flash RAM Boot Trace" on page 110

Note - The Netra j software provides a GUI front end for viewing network boot activity. For information on Netra j, go to `http://www.sun.com/netra-j` or refer to the `Netra j 3.0 Administrator's Guide`.

Troubleshooting Process

When troubleshooting the JavaStation computer, you need to first understand the boot process and operating environment, and then gain visibility into the network. You can retrieve JavaStation boot information in two ways:

- *At the JavaOS Console* – The JavaOS software provides a JavaOS console that prints status and debugging information in a window on the JavaStation screen. To open the JavaOS console, press the PrintScreen key anytime after the coffee cup wallpaper or plain boot background is displayed (see Chapter 3 for a description of these backgrounds). A scrollable list of status information is presented in a window and is constantly updated.
- *Using the UNIX* `snoop(1M)` *command* – The UNIX `snoop` command is highly useful for observing transactions being carried out between the JavaStation computer and the providers of its various network services.

The following section is an example snoop trace of a JavaStation tower model.

Tower JavaStation Flash RAM Boot Trace

This annotated network trace follows a flash RAM-equipped JavaStation tower model through its JavaOS 1.1 boot sequence. JavaStation computers are extremely flexible and can be booted a variety of different ways. This boot trace reflects default behavior for a typical JavaStation installation.

This information is provided to assist in understanding the JavaStation boot process and to aid with debugging problematic JavaStation deployments. Excessively redundant information has been removed for brevity.

To obtain this boot trace, the UNIX `snoop(1M)` command was used with the JavaStation computer's Ethernet address. (In the `snoop` commands below, the user typed Cntrl-c between the two commands to stop collecting `snoop` data.)

Note - `snoop` must be run as root.

```
# snoop -o /tmp/trace.snoop 08:00:20:87:be:d6
# snoop -i /tmp/trace.snoop
```

For convenience, the systems involved in this particular boot sequence are listed in the following table along with their role in the network boot process.

TABLE B–1 Systems Involved in a Sample JavaStation Trace and Their Functions

System Host Name	IP Address	Ethernet Address	Function
dilbert	129.153.59.151	8:0:20:87:be:d6	JavaStation computer
jsboot	129.153.59.150	8:0:20:1f:ad:1d	Boot, Web, DNS, and NIS Server
chaco	129.153.59.100	8:0:20:22:36:95	Home Directory Server

Boot Trace

Because the JavaStation tower model boots the JavaOS software from flash RAM, it needs to retrieve only its Java main application from the network. No network traffic is generated while the JavaOS software boots from flash RAM.

Once the JavaOS software is booted, it performs DHCP, ARP, and NIS broadcasts to locate and confirm its boot server (jsboot). Even though the JavaStation tower model boots from flash RAM, it needs to know about its boot server for application configuration information or to possibly download a new version of the JavaOS image into flash RAM.

```
1   0.00000 OLD-BROADCAST -> BROADCAST    DHCP/BOOTP BOOTREQUEST
2   2.02091       jsboot -> dilbert      DHCP/BOOTP BOOTREPLY
3   0.01412 OLD-BROADCAST -> BROADCAST    DHCP/BOOTP BOOTREQUEST
4   0.18294       jsboot -> dilbert      DHCP/BOOTP BOOTREPLY
5   0.01362 OLD-BROADCAST -> (broadcast)  ARP C Who is 129.153.59.151, dilbert ?
6   1.01707      dilbert -> BROADCAST    ARP R 129.153.59.151, dilbert is 8:0:20:87:be:d6
7   0.00079      dilbert -> BROADCAST    IP  D=255.255.255.255 S=129.153.59.151 LEN=28, ID=3
8   0.00173        chaco -> dilbert      IP  D=129.153.59.151 S=129.153.59.100 LEN=36, ID=54101
9   0.02696      dilbert -> (broadcast)  ARP C Who is 129.153.59.150, jsboot ?
10   0.00034       jsboot -> dilbert      ARP R 129.153.59.150, jsboot is 8:0:20:1f:ad:1d
11   0.00110      dilbert -> jsboot       PORTMAP C GETPORT prog=100004 (NIS) vers=2 proto=UDP
12   0.00446       jsboot -> dilbert      PORTMAP R GETPORT port=768
13   0.01016      dilbert -> jsboot       NIS C MATCH 129.153.59.151 in hosts.byaddr
14   0.00261       jsboot -> dilbert      NIS R MATCH OK
15   0.01906      dilbert -> jsboot       NIS C MATCH jsboot in hosts.byname
16   0.00250       jsboot -> dilbert      NIS R MATCH OK
```

Next, the JavaStation computer downloads its configuration file via HTTP. The configuration file is used to pass information to the JavaOS software such as locale, network printer information, and which Java main application is to be downloaded, as described in Chapter 4. The configuration file for this JavaStation computer is located in `http://jsboot/netra/dilbert`.

```
17   0.00984      dilbert -> jsboot       HTTP C port=49997
18   0.00036       jsboot -> dilbert      HTTP R port=49997
19   0.01047      dilbert -> jsboot       HTTP GET /netra/dilbert HTTP/1.0

20   0.00036       jsboot -> dilbert      HTTP R port=49997
21   0.00849       jsboot -> dilbert      HTTP HTTP/1.0 200 OK

22   0.02123      dilbert -> jsboot       HTTP C port=49997
23   0.00033       jsboot -> dilbert      HTTP R port=49997
24   0.00138       jsboot -> dilbert      HTTP R port=49997
25   0.00133      dilbert -> jsboot       HTTP C port=49997
26   0.01070      dilbert -> jsboot       HTTP C port=34276
27   0.00038       jsboot -> dilbert      HTTP R port=34276
```

The Java main application listed in the configuration file also has a configuration file. In this case, `http://jsboot/netra/browser` holds the configuration file for the HotJava Browser.

```
28   0.01025      dilbert -> jsboot       HTTP GET /netra/browser HTTP/1.0
29   0.00037       jsboot -> dilbert      HTTP R port=34276
30   0.00494       jsboot -> dilbert      HTTP HTTP/1.0 200 OK
31   0.02189      dilbert -> jsboot       HTTP C port=34276
32   0.00034       jsboot -> dilbert      HTTP R port=34276
33   0.00128       jsboot -> dilbert      HTTP R port=34276
34   0.00138      dilbert -> jsboot       HTTP C port=34276
```

The JavaStation computer then attempts (unsuccessfully) to mount a font directory and finally sets its clock by talking to its time server.

```
35   0.02130      dilbert -> jsboot       PORTMAP C GETPORT prog=100004 (NIS) vers=2 proto=UDP
36   0.00367       jsboot -> dilbert      PORTMAP R GETPORT port=768
37   0.35151      dilbert -> jsboot       PORTMAP C GETPORT prog=100005 (MOUNT) vers=1 proto=UDP
38   0.00372       jsboot -> dilbert      PORTMAP R GETPORT port=32795
39   0.00451      dilbert -> jsboot       MOUNT1 C Mount /dirpath/fonts
40   0.00674       jsboot -> dilbert      MOUNT1 R Mount No such file or directory
41   0.04961      dilbert -> jsboot       TIME C port=1025
42   0.00339       jsboot -> dilbert      TIME R port=1025
```

At this point the JavaOS software is completely booted and the user login prompt is displayed on the JavaStation screen. The next step is to authenticate the user.

```
43  16.92166      dilbert -> jsboot       NIS C MATCH eric in passwd.byname
44   0.00285       jsboot -> dilbert      NIS R MATCH OK
```

Once the user has been authenticated, the JavaStation computer uses the automounter NIS map to determine the user"s home directory. When the automounter responds with the proper directory name, the JavaStation computer performs an NFS mount to the home directory server (chaco in this case). The user"s home directory is used by the HotJava Browser to access a number of configuration and property files (`/export/opt2/eric/.javaos/properties` in this case).

```
45   0.21276      dilbert -> jsboot       NIS C MATCH eric in auto.home
46   0.00248       jsboot -> dilbert      NIS R MATCH OK
47   0.00350      dilbert -> jsboot       NIS C MATCH chaco in hosts.byname
48   0.00385       jsboot -> dilbert      NIS R MATCH OK
49   0.00531      dilbert -> (broadcast)  ARP C Who is 129.153.59.100, chaco ?
50   0.00107        chaco -> dilbert      ARP R 129.153.59.100, chaco is 8:0:20:22:36:95
51   0.00079      dilbert -> chaco        PORTMAP C GETPORT prog=100005 (MOUNT) vers=1 proto=UDP
52   0.00360        chaco -> dilbert      PORTMAP R GETPORT port=32786
53   0.00423      dilbert -> chaco        MOUNT1 C Mount /export/opt2/eric
54   0.00701        chaco -> dilbert      MOUNT1 R Mount OK FH=AC28
55   0.02975      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .
56   0.00192        chaco -> dilbert      NFS R LOOKUP2 OK FH=AC28
57   0.00213      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .javaos
58   0.00149        chaco -> dilbert      NFS R LOOKUP2 OK FH=2DF5
59   0.00308      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .javaos
60   0.00147        chaco -> dilbert      NFS R LOOKUP2 OK FH=2DF5
61   0.00211      dilbert -> chaco        NFS C LOOKUP2 FH=2DF5 properties
62   0.00154        chaco -> dilbert      NFS R LOOKUP2 OK FH=7D4B
```

```
63   0.00529      dilbert -> chaco        NFS C LOOKUP2 FH=2DF5 properties
64   0.00154        chaco -> dilbert      NFS R LOOKUP2 OK FH=7D4B
65   0.00261      dilbert -> chaco        NFS C READ2 FH=7D4B at 0 for 2048
66   0.00244        chaco -> dilbert      NFS R READ2 OK (578 bytes)
67   0.00225      dilbert -> chaco        NFS C READ2 FH=7D4B at 578 for 1470
68   0.00149        chaco -> dilbert      NFS R READ2 OK (0 bytes)
69   0.02084      dilbert -> chaco        NFS C READ2 FH=7D4B at 578 for 2048
70   0.00152        chaco -> dilbert      NFS R READ2 OK (0 bytes)
71   0.00369      dilbert -> jsboot       NIS C MATCH dilbert in hosts.byname
72   0.00132       jsboot -> dilbert      NIS R MATCH OK
```

Once the user has been authenticated, the JavaStation computer employs the AppLoader in the JavaOS software to download the Java main application, as described in Chapter 5. In this case, the JavaOS software will obtain the necessary classes for the HotJava Browser by downloading `http://netra/hjb/hotjava.zip` using HTTP (as directed by `http://jsboot/netra/browser`).

```
73   0.25295      dilbert -> jsboot       HTTP C port=57205
74   0.00037       jsboot -> dilbert      HTTP R port=57205
75   0.01046      dilbert -> jsboot       HTTP GET /netra/hjb/hotjava.zip HTTP/1.0
76   0.00036       jsboot -> dilbert      HTTP R port=57205
77   0.08306       jsboot -> dilbert      HTTP HTTP/1.0 200 OK
78   0.05637      dilbert -> jsboot       HTTP C port=57205
79   0.00042       jsboot -> dilbert      HTTP (body)
80   0.00033       jsboot -> dilbert      HTTP (body)
81   0.00455      dilbert -> jsboot       HTTP C port=57205
82   0.00042       jsboot -> dilbert      HTTP (body)
83   0.00033       jsboot -> dilbert      HTTP (body)
84   0.00033       jsboot -> dilbert      HTTP (body)
```

[Lots of HTTP traffic deleted]

Once the JavaStation computer has loaded the Java main application, it once again accesses the user"s home directory to obtain the user"s HotJava Browser preference information, including security, hotlists, cookies, and default lists of URLs.

```
2103   0.95132      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .hotjava
2104   0.00177        chaco -> dilbert      NFS R LOOKUP2 OK FH=0A5D
2105   0.00210      dilbert -> chaco        NFS C LOOKUP2 FH=0A5D properties
2106   0.00203        chaco -> dilbert      NFS R LOOKUP2 OK FH=13C1
2107   0.00264      dilbert -> chaco        NFS C READ2 FH=13C1 at 0 for 2048
2108   0.02274        chaco -> dilbert      NFS R READ2 OK (889 bytes)
2109   0.00237      dilbert -> chaco        NFS C READ2 FH=13C1 at 889 for 1159
2110   0.00165        chaco -> dilbert      NFS R READ2 OK (0 bytes)
2111   0.03816      dilbert -> chaco        NFS C READ2 FH=13C1 at 889 for 2048
2112   0.00152        chaco -> dilbert      NFS R READ2 OK (0 bytes)
2113   0.00332      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .
2114   0.00153        chaco -> dilbert      NFS R LOOKUP2 OK FH=AC28
2115   0.00211      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .hotjava
2116   0.00152        chaco -> dilbert      NFS R LOOKUP2 OK FH=0A5D
2117   2.18165      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .
2118   0.00144        chaco -> dilbert      NFS R LOOKUP2 OK FH=AC28
2119   0.00505      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .hotjava
```

```
2120   0.00150        chaco -> dilbert      NFS R LOOKUP2 OK FH=0A5D
2121   0.00215      dilbert -> chaco        NFS C LOOKUP2 FH=0A5D security1.1
2122   0.00217        chaco -> dilbert      NFS R LOOKUP2 No such file or directory
2123   0.01459      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .
2124   0.00151        chaco -> dilbert      NFS R LOOKUP2 OK FH=AC28
2125   0.00333      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .hotjava
2126   0.00154        chaco -> dilbert      NFS R LOOKUP2 OK FH=0A5D
2127   0.00213      dilbert -> chaco        NFS C LOOKUP2 FH=0A5D security1.1
2128   0.00217        chaco -> dilbert      NFS R LOOKUP2 No such file or directory
2129   3.27535      dilbert -> chaco        NFS C LOOKUP2 FH=AC28 .hotjava
2130   0.00149        chaco -> dilbert      NFS R LOOKUP2 OK FH=0A5D
2131   0.00216      dilbert -> chaco        NFS C LOOKUP2 FH=0A5D hotlist.html
2132   0.00154        chaco -> dilbert      NFS R LOOKUP2 OK FH=E884
2133   0.70818      dilbert -> chaco        NFS C LOOKUP2 FH=0A5D hotlist.html
2134   0.00150        chaco -> dilbert      NFS R LOOKUP2 OK FH=E884
2135   0.20309      dilbert -> chaco        NFS C LOOKUP2 FH=0A5D urlpool
2136   0.00155        chaco -> dilbert      NFS R LOOKUP2 OK FH=B632
2137   0.01399      dilbert -> chaco        NFS C READ2 FH=B632 at 0 for 2048
```