Oracle® OpenBoot 4.x Administration Guide

Part No: E63649-02
June 2020

ORACLE

Oracle OpenBoot 4.x Administration Guide

Part No: E63649-02

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Référence: E63649-02

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Describes how to use and administer the OpenBoot firmware from Oracle.
- **Audience** – Technicians, system administrators, and authorized service providers.
- **Required knowledge** – Firmware configuration experience on Oracle hardware.

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/goto/openboot/docs.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

Understanding OpenBoot

Use these topics to understand OpenBoot as defined by the *IEEE Standard 1275-1994 for Boot (Initialization Configuration) Firmware: Core Requirements and Practices*.

OpenBoot Firmware Overview

The OpenBoot architecture provides a significant increase in functionality and portability when compared to proprietary systems of the past. Although this architecture was first implemented by Sun MicroSystems as OpenBoot on SPARC systems, its design is processor-independent. OpenBoot is based on the IEEE Std 1275-1994 Standard for Boot (the standard is available from IEEE, which might require membership or a fee for access).

For recent SPARC systems, OpenBoot firmware is executed after Virtual Machine (VM) is started.

The primary tasks of OpenBoot firmware are to:

- Determine the hardware configuration and initialize I/O devices by running the FCode driver for each I/O device.
- Boot the OS from either a storage device or from a network.

The primary tasks are described in "Booting and Resetting a Virtual Machine" on page 37.

The OpenBoot CLI is based on an interactive command interpreter that gives you access to an extensive set of functions for hardware and software development, fault isolation, and debugging. For details, see “Accessing the OpenBoot CLI and Getting Help” on page 21 and “Using the OpenBoot CLI” on page 29.

A number of OpenBoot operating characteristics are controlled by configuration variables that are stored in nonvolatile memory. If needed, you can change the configuration variables default values to tailor operating characteristics to your environment. For details, see “Setting Configuration Variables” on page 55 and “Setting Security Variables” on page 71.

OpenBoot provides commands that you can use to gather information about the system hardware. Some commands also perform a low level sanity tests of the hardware without a running OS. For details, see “Interrogating the System With OpenBoot Commands” on page 77.

You can use the OpenBoot `nvramrc` configuration variable to store user-defined FORTH commands that are executed during start-up. For details, see “NVRAMRC Overview” on page 87.

Related Documentation

- *IEEE Std 1275-1994. Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* available from: `https://www.openfirmware.info/IEEE_1275-1994`
- “Built-In and Plug-In Device Drivers” on page 12
- “FCode Interpreter” on page 13
- “Device Tree” on page 14
- “Device Path Names, Addresses, and Arguments” on page 16
- “Device Aliases” on page 18

Built-In and Plug-In Device Drivers

The devices that are used by OpenBoot can be either built-in devices or plug-in devices. The firmware device drivers for built-in devices are usually included as permanent parts of the system’s OpenBoot implementation. The drivers for plug-in devices, such as PCI cards, are typically stored on the device itself, and are automatically installed when the device is installed. For PCI cards, FCode goes into the Expansion ROM.

Device drivers can be used to boot the OS from a device or to display text on the device before the OS has activated its own drivers. This feature enables the input and output devices supported by a particular system to evolve without changing the system.

Probing is the process of determining the presence and characteristics of physical and virtual devices. For a device that uses the OpenBoot identification methods, the bulk of the probing process consists of the execution of an FCode program associated with the device. Probing involves selecting a bus device and using it to test for the presence of devices attached to that bus.

It is possible for a plug-in device itself to be a bus device. For example, a PCIe plug-in device might be an adapter for fibre channel. Before the children of a bus device can be probed, the device itself must already exist in the device tree. For the preceding example, the PCIe would have to be probed to locate the fibre channel adapter and install its device node before the fibre channel could be probed for its children.

The device tree is thus constructed incrementally, beginning from the permanent part representing built-in devices and proceeding outward toward the leaves of the tree. The default probing sequence is automatically executed during the start-up sequence, unless overridden. Each bus device that is capable of accepting plug-in devices defines a device method for probing its subordinate devices. In addition, bus devices can define device-dependent user interface commands for probing.

Related Information

- “OpenBoot Firmware Overview” on page 11
- “FCode Interpreter” on page 13
- “Device Tree” on page 14
- “Device Path Names, Addresses, and Arguments” on page 16
- “Device Aliases” on page 18

FCode Interpreter

Plug-in drivers are written in a byte-coded, machine-independent, interpreted language called *FCode*. FCode is based on FORTH semantics. Because FCode is machine-independent, the same device and driver can be used on machines with different CPU instruction sets. Each OpenBoot image contains an FCode interpreter.

Related Information

- “OpenBoot Firmware Overview” on page 11
- “Built-In and Plug-In Device Drivers” on page 12

Device Tree

The set of devices attached to a virtual machine, including permanently installed devices and plug-in devices, is described by an OpenBoot data structure known as the device tree.

The OS can inspect the device tree to determine the hardware configuration of the virtual machine. Each device in the device tree is described by a property list. The set of properties describing a device is arbitrarily extensible so that any type of device and any kind of information that needs to be reported about the device can be accommodated.

The device tree is a hierarchical data structure representing the physical configuration of the virtual machine. Most OpenBoot elements (for example, devices, buses, and libraries of software procedures) are named and located by the device tree.

The device tree also describes user configuration choices, contains firmware device drivers for hardware devices, and contains support routines for use by those drivers. The device tree's structure mimics the organization of the hardware, viewed as a hierarchy of interconnected buses and their attached devices.

The device tree consists of a set of device nodes that are interconnected to form a tree. An individual device node represents either a hardware bus, a hardware device, or a set of interrelated software procedures. The root of the device tree is a node representing the machine’s main physical address bus.

Each device node can have these items:

- **Properties** – Data structures describing the node and its associated device.
- **Methods** – Software procedures used to access the device.
- **Data** – Initial values of the private data used by the methods.
- **Children** – Other device nodes *attached* to a given node and that lie directly below it in the device tree.
- **Parent** – Node that lies directly above a given node in the device tree.

Device nodes with children are called *hierarchical nodes*. A node's *parent* is the node to which it is attached in the device tree. The root node has no parent. Device nodes without children are called *leaf nodes*.

A node with children usually represents a bus and its associated hardware. Each bus is assumed to define a physical address space; each device connected to that bus has a distinct physical address within that space, uniquely distinguishing the particular device from other devices on that bus. The form of a physical address is bus-specific. The children of a bus node are distinguished from one another with software representations of the same physical addresses that the bus device uses to distinguish attached devices. OpenBoot uses several different representations of addresses with similar meanings but different forms:

- **Text representation** – The human-readable form of a physical address. The format is bus-dependent. For example, some buses use a comma-separated list of numbers represented as ASCII text in hexadecimal notation.
- **Stack representation** – Used to pass arguments to and results from FORTH words. This form usually consists of one or more binary numbers on the data stack.
- **Property-encoded representation** – Used to communicate with client programs through property values. This form usually consists of a sequence of binary numbers stored within an array of bytes.

The forms of these representations differ, but their meanings are the same. They represent physical addresses within a bus's physical address space. The details of these different representations differ from bus to bus, depending on the addressing characteristics of the individual bus. Specifications of OpenBoot address representations for several standard buses are specified in supplements to this document (refer to the IEEE P1275.x documents in 2.1).

Device tree nodes are added by the probing process. Some nodes in the device tree do not represent physical devices. These nodes are used instead for various general purposes in OpenBoot. These nodes do not have physical addresses. Their node names have a device name field but not a unit address field.

The physical address generally represents a physical characteristic unique to the device (such as the bus address or the slot number where the device is installed). The use of physical addresses to identify devices prevents device addresses from changing when other devices are installed or removed.

Each node in the device tree is identified by a node name using the following notation:

device-name@*unit-address*:*device-arguments*

Where:

- *device-name* – Is a sequence of between one and 31 letters, digits, and punctuation characters from the set “, . _ + -”. Uppercase and lowercase characters are distinct. By convention, this name includes the name of the device's manufacturer and the device's model name separated by a , (comma). Refer to the definition of name in annex A. Inclusion of the manufacturer name within the device name is especially important for devices intended to plug into standard buses, because this minimizes the risk of accidental name collisions

If the manufacturer name component is omitted (there is no , within the device name), the convention is to assume that the manufacturer name is the same as that of the nearest ancestor node (parent node or grandparent node) that has an explicit manufacturer name component.

- *unit-address* – Is the text representation of the physical address of the device within the address space defined by its parent node. The form of the text representation is bus-dependent.
- *device-arguments* – is a sequence of printable characters other than "/", ":", and "@". Uppercase and lowercase characters are distinct. The length is arbitrary. The device arguments field is interpreted by the driver and typically represents additional device information, such as partition name or protocol. The device arguments field and its preceding ":" can be omitted when specifying a node name, as it does not serve to identify the device node. Instead, it is passed to that node's open method if that driver is opened. By convention, a , (comma) is used to separate subfields within the device arguments field.

Related Information

- "OpenBoot Firmware Overview" on page 11
- "Built-In and Plug-In Device Drivers" on page 12
- "FCode Interpreter" on page 13
- "Device Path Names, Addresses, and Arguments" on page 16
- "Device Aliases" on page 18

Device Path Names, Addresses, and Arguments

OpenBoot deals directly with hardware devices in the virtual machine. Each device has a unique name representing the type of device and where that device is located in the addressing structure. The textual representation of a such a path is called a *device path*. Device paths are composed as follows:

```
/node-name0/node-name1/ ... /node-nameN
```

This example shows a full device path name:

```
/pci@301/pci@2/scsi@0/disk@w5000cca123456789,0:a
```

A full device path name is a series of node names separated by slashes (/). The root of the tree is the machine node, which is not named explicitly but is indicated by a leading slash (/). Each node name has the form:

device-name@*unit-address*:*device-arguments*

When OpenBoot is searching for a particular node, and either the device name or @*unit-address* portion of the node name is not given, OpenBoot arbitrarily chooses a node matching the portion that is present.

The OpenBoot CLI uses pathnames to identify particular device nodes.

Name	Description
device-name	A human-readable string consisting of one to 31 letters, digits and punctuation characters from the set ", . _ + -" that, ideally, has some mnemonic value. Uppercase and lowercase characters are distinct. In some cases, this name includes the name of the device's manufacturer and the device's model name, separated by a comma. Typically, the manufacturer's upper-case, publicly-listed stock symbol is used as the manufacturer's name (for example, `SUNW,sd`). For built-in devices, the manufacturer's name is usually omitted.
`@`	Must precede the address parameter.
unit-address	A text string representing the physical address of the device in its parent's address space. The format of the text is bus dependent.
`:`	Must precede the arguments parameter.
device-arguments	A text string whose format depends on the particular device. It can be used to pass additional information to the device's software.

The full device path name mimics the hardware addressing used by the virtual machine to distinguish between different devices. Thus, you can specify a particular device without ambiguity.

In general, the *unit-address* part of a node name represents an address in the physical address space of its parent. The exact meaning of a particular address depends on the bus to which the device is attached. Consider this example:

```
/pci@301/pci@2/scsi@0/disk@w5000cca123456789,0:a
```

Where:

- `/` – Is the root of the tree.
- `pci@301` – Is the PCIe bus with a unit address of `301`.
- `pci@2` – Is the PCIe controller with a PCI device ID of `2`.
- `scsi@0` – Is the SCSI controller with a PCI device ID of `0`.
- `disk@w5000cca123456789,0:a` – Is the SCSI drive LUN with a unit address of `5000cca0566d3b21` (WWN of the drive in this example) , and a device argument of `a` (disk slice in this example).

When specifying a path name, either the @*unit-address* or *device-name* part of a node name is optional, in which case the firmware tries to pick the device that best matches the given name. If there are several matching nodes, the firmware chooses one (but it may not be the one you want).

Related Information

- “OpenBoot Firmware Overview” on page 11
- “Built-In and Plug-In Device Drivers” on page 12
- “FCode Interpreter” on page 13
- “Device Tree” on page 14
- “Device Aliases” on page 18

Device Aliases

A device alias, is a shorthand representation of a device path. An alias is a text name identifying a device node by showing its position in the device tree. An alias represents an entire device path, but that path need not refer to a leaf node. Each implementation can have a number of predefined aliases for devices commonly installed on that machine.

For example, the alias `disk` might represent the complete device path name:

```
/pci@301/pci@2/scsi@0/disk@w5000cca123456789,0:a
```

Virtual machines have predefined device aliases for the most commonly used devices, however, you can create, modify, and examine aliases with the `devalias` command. User-defined aliases are lost after a reset or power cycle, but you can create a persistent alias by storing the `devalias` command in an NVRAMRC script. For more details, see “Create a Device Alias” on page 61.

Related Information

- “OpenBoot Firmware Overview” on page 11
- “Built-In and Plug-In Device Drivers” on page 12
- “FCode Interpreter” on page 13
- “Device Tree” on page 14
- “Device Path Names, Addresses, and Arguments” on page 16

Additional Resources

This table lists additional resources that provide information about OpenBoot and the products that use OpenBoot.

Resource	Description
OpenBoot Documentation Library http://www.oracle.com/goto/openboot/docs	This library contains OpenBoot documentation, including this document.
IEEE 1275-1994 Documentation for Open Firmware web site https://www.openfirmware.info/ IEEE_1275-1994	This web site provides information on how to obtain the *IEEE Std 1275-1994. Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* specification.
Oracle ILOM documentation libraries https://docs.oracle.com/cd/F24624_01/index.html#ilom	Entry page that enables you to access Oracle ILOM libraries for all versions.
Oracle Solaris OS documentation libraries http://docs.oracle.com/en/operating-systems/	Entry page that enables you to access Oracle Solaris OS libraries for all versions.
Oracle SPARC server documentation http://docs.oracle.com/en/servers	Entry page that enables you to access all the Oracle server libraries.
Oracle Solaris 11.3 documentation library http://www.oracle.com/goto/solaris11/docs	Provides access to all of the Oracle Solaris 11.3 documentation.
Securing System and Attached Devices in Oracle Solaris 11.3, Verified Boot section https://docs.oracle.com/cd/E53394_01/html/E54828/sysauth-vb.html	Provides details on how to set up Verified Boot and other related settings.
Installing Oracle Solaris 11.3 Systems https://docs.oracle.com/cd/E53394_01/html/E54756	Provides details on how to prepare an Oracle Solaris 11.3 installation server, including information about configuring OpenBoot keys on an installation client.
Oracle Solaris 10 1/13 Installation Guide: Network-Based Installations http://docs.oracle.com/cd/E26505_01/html/E28037/index.html	Provides details on how to prepare an Oracle Solaris 10 installation server, including information about configuring OpenBoot keys on an installation client. Refer to the Installing Over a Wide Area Network section.

Accessing the OpenBoot CLI and Getting Help

When the system or virtual machine is powered on but the OS is not booted, you communicate with the OpenBoot firmware. OpenBoot firmware displays `ok` as its prompt.

These topics describe how to reach the OpenBoot ok prompt, and how to display OpenBoot help.

Task	Description
Determine which method to use to access the OpenBoot CLI.	"Identify a Method to Get to the OpenBoot CLI" on page 21
Access the OpenBoot CLI using one of these procedures.	"Access the OpenBoot CLI (OpenBoot Running)" on page 22 "Access the OpenBoot CLI (Solaris Running)" on page 23 "Access the OpenBoot CLI (Powered Off)" on page 24 "Access the OpenBoot CLI (Hung System)" on page 25
View `help` options and use the `help` command.	"Use the `help` Command" on page 26

▼ Identify a Method to Get to the OpenBoot CLI

There are many ways to access the OpenBoot CLI, and the method you use is based on the state of the virtual machine. This procedure guides you through the process of determining the state, and directs you to the most appropriate method to reach the OpenBoot `ok` prompt.

Note - The Oracle ILOM CLI examples in this section are for a single PDomain system. For multiple domain systems, replace `/HOST` with `/HOST`*x*, where *x* is the PDomain number.

1. **Open an SSH session and connect to Oracle ILOM.**

Note - For more information about Oracle ILOM, refer to the documentation library that corresponds with your version of Oracle ILOM: `https://docs.oracle.com/cd/F24624_01/index.html#ilom`

For example:

```
% ssh root@SP_IP_Address_or_Hostname
Password: password (nothing displayed as you type)
...
Oracle(R) Integrated Lights Out Manager
Version 3.3.x.x
Copyright (c) 2017, Oracle and/or its affiliates. All rights reserved.
...
->
```

2. **Get the status of the host.**

   The status of the host determines the method used to get to OpenBoot.

   For example:

   ```
   -> show /HOST status
   ```

3. **Based on the host status, perform one of these tasks:**

   - **OpenBoot Running**, go to "Access the OpenBoot CLI (OpenBoot Running)" on page 22
   - **Solaris Running**, go to "Access the OpenBoot CLI (Solaris Running)" on page 23
   - **Powered Off**, go to "Access the OpenBoot CLI (Powered Off)" on page 24
   - **If the host status is reported as being in any running state, but the host is not responding, it might be hung**, go to "Access the OpenBoot CLI (Hung System)" on page 25

▼ Access the OpenBoot CLI (OpenBoot Running)

Use this procedure when the host status is OpenBoot Running.

This procedure assumes that you performed "Identify a Method to Get to the OpenBoot CLI" on page 21, and you are still logged into Oracle ILOM.

Note - The Oracle ILOM CLI examples in this section are for a single PDomain system. For multiple domain systems, replace `/HOST` with `/HOSTx`, where *x* is the PDomain number.

---

1. **Switch communication to the host console.**

   ```
   -> start /HOST/console
   Are you sure you want to start /HOST/console (y/n)? y
   ```

```
Serial console started. To stop, type #.
```

2. **Press Return and perform one of these actions:**

   - If the OpenBoot security mode is enabled, choose login and enter the OpenBoot password. The `ok` prompt is displayed indicating that you are accessing the OpenBoot CLI.
   - If the OpenBoot security mode is not enabled, the `ok` prompt is displayed indicating that you are accessing the OpenBoot CLI.
   - If you do not see the OpenBoot prompt, perform the following commands to access the OpenBoot CLI:

   a. **Enter `#.` to return to the Oracle ILOM CLI.**

   b. **Log into Oracle ILOM.**

   c. **Send a break:**

```
#.
Login
-> set /HOST send_break_action=break
```

Related Information

- "Identify a Method to Get to the OpenBoot CLI" on page 21
- "Use the `help` Command" on page 26
- "Using the OpenBoot CLI" on page 29

▼ Access the OpenBoot CLI (Solaris Running)

Use this procedure when the host status is Solaris Running.

This procedure assumes that you performed "Identify a Method to Get to the OpenBoot CLI" on page 21, and you are still logged into Oracle ILOM.

This procedure can be used on systems with a single instance of the Oracle Solaris OS, and on an individual virtual instance of the OS that is running on a physical domain, logical domain, global zone, or kernel zone. For additional details, refer to the Oracle Solaris documentation library for the version of Oracle Solaris running on your system at `http://docs.oracle.com/en/operating-systems/`.

Note - The Oracle ILOM CLI examples in this section are for a single PDomain system. For multiple domain systems, replace `/HOST` with `/HOST`*x*, where *x* is the PDomain number.

---

1. **Switch communication to the host console and press Return.**

```
-> start /HOST/console
Are you sure you want to start /HOST/console (y/n)? y
Serial console started. To stop, type #.
```

2. **Login to Oracle Solaris as a user with the superuser role, and shutdown the OS:**

```
# init 0
```

Once the OS completes the shutdown, the `ok` prompt is displayed indicating that you are accessing the OpenBoot CLI.

Related Information

- "Identify a Method to Get to the OpenBoot CLI" on page 21
- "Use the `help` Command" on page 26
- "Using the OpenBoot CLI" on page 29

▼ Access the OpenBoot CLI (Powered Off)

Use this procedure when the host status is Powered Off.

This procedure assumes that you performed "Identify a Method to Get to the OpenBoot CLI" on page 21, and you are still logged into Oracle ILOM.

Note - The Oracle ILOM CLI examples in this section are for a single PDomain system. For multiple domain systems, replace `/HOST` with `/HOST`*x*, where *x* is the PDomain number.

---

1. **Temporarily change the `auto-boot?` parameter to `false`.**

For example:

```
-> set /HOST/bootmode script="setenv auto-boot? false"
```

This command temporarily prevents the OS from booting before you reach the OpenBoot prompt. This change applies only to a single start, and expires in 10 minutes if the power is not started.

2. **Start the virtual machine.**

- Example of starting a VM with a single PDomain:

```
-> start /System
```

- Example of starting a VM with multiple PDomains:

  ```
  -> start /HOSTx
  ```

  Where *x* is the PDomain number.

3. **If you are prompted to login, select login and enter the OpenBoot password.**

   The `ok` prompt is displayed indicating that you are accessing the OpenBoot CLI.

Related Information

- "Identify a Method to Get to the OpenBoot CLI" on page 21
- "Use the `help` Command" on page 26
- "Using the OpenBoot CLI" on page 29

▼ Access the OpenBoot CLI (Hung System)

Use this procedure when the system or virtual machine is in a hung state.

This procedure assumes that you performed "Identify a Method to Get to the OpenBoot CLI" on page 21, and you are still logged into Oracle ILOM.

Note - The Oracle ILOM CLI examples in this section are for a single PDomain system. For multiple domain systems, replace `/HOST` with `/HOSTx`, where *x* is the PDomain number.

---

1. **Temporarily change the `auto-boot?` parameter to `false`.**

   For example:

   ```
   -> set /HOST/bootmode script="setenv auto-boot? false"
   ```

   This command temporarily prevents the OS from booting before you reach the OpenBoot prompt. This change applies only to a single reset and expires in 10 minutes if the power is not reset.

2. **Reset the system or virtual machine.**

   For example:

   ```
   -> reset /HOST/domain/control/
   ```

3. **If the prior step did not work, perform this command to power off the virtual machine, then proceed to "Access the OpenBoot CLI (Powered Off)" on page 24.**

- Example of stopping a VM in a single PDomain system:

```
-> stop -f /System
```

- Example of stopping a VM in a multiple PDomain system:

```
-> stop -f /HOSTx
```

  Where *x* is the PDomain number.

Related Information

- “Identify a Method to Get to the OpenBoot CLI” on page 21
- “Use the help Command” on page 26
- “Using the OpenBoot CLI” on page 29

▼ Use the help Command

Whenever you see the ok prompt on the display, you can access OpenBoot help by typing one of the help commands.

Name	Description
help	Provides information for a category or a specific command. If *name* is a specific command, lists help for that command, if available. Otherwise, displays an implementation-dependent message. Used as: ok help *command-name* If *name* is a category, lists all help messages for commands in that category or a list of subcategories. Used as: ok help *category-name* If *name* is omitted, provides general help and a list of available categories. Commands should be grouped into categories so that the help messages for a category occupy no more than twenty-three output lines. Categories can be divided into subcategories. The number and names of categories are implementation dependent.
help *category*	Shows help for all commands in the category. Use only the first word of the category description.
help *command*	Shows help for individual commands (when available).

1. **Access the OpenBoot CLI.**

See “Access the OpenBoot CLI (Solaris Running)” on page 23 or “Access the OpenBoot CLI (OpenBoot Running)” on page 22

2. **Use one of these commands:**

- **Display the help messages for all the commands in a selected category, or, possibly, a list of subcategories.**

```
{0} ok help category
```

- **Display help for a specific command.**

```
{0} ok help command
```

Examples:

```
{0} ok help diag
test <device-specifier> Run selftest method for specified device
  Examples:
    test net - test net
    test scsi - test scsi
test-all Execute test for all devices with selftest method
watch-net Monitor network broadcast packets
watch-net-all Monitor broadcast packets on all net interfaces
probe-scsi Show attached SCSI devices
probe-scsi-all Show attached SCSI devices for all host adapters

{0} ok help memory
dump ( addr length -- ) Display memory at addr for length bytes
fill ( addr length byte -- ) Fill memory starting at addr with byte
move ( src dest length -- ) Copy length bytes from src to dest address
map? ( vaddr -- ) Show memory map information for the virtual address
x? ( addr -- ) Display the 64-bit number from location addr
l? ( addr -- ) Display the 32-bit number from location addr
w? ( addr -- ) Display the 16-bit number from location addr
c? ( addr -- ) Display the 8-bit number from location addr
x@ ( addr -- n ) Place on the stack the 64-bit data at location addr
l@ ( addr -- n ) Place on the stack the 32-bit data at location addr
w@ ( addr -- n ) Place on the stack the 16-bit data at location addr
c@ ( addr -- n ) Place on the stack the 8-bit data at location addr
x! ( n addr -- ) Store the 64-bit value n at location addr
l! ( n addr -- ) Store the 32-bit value n at location addr
w! ( n addr -- ) Store the 16-bit value n at location addr
c! ( n addr -- ) Store the 8-bit value n at location addr
```

Note - In some newer systems, descriptions of additional system-specific commands are available with the help command. Help as described might not be available on all systems.

Related Information

- “Using the OpenBoot CLI” on page 29

Using the OpenBoot CLI

Use these topics to understand the OpenBoot command line interface (CLI).

OpenBoot CLI Overview

The OpenBoot CLI is based on the industry-standard interactive programming language called *FORTH*. Combining sequences of commands to form complete programs provides the capability for debugging hardware and software.

The OpenBoot CLI is based on an interactive command interpreter that gives you access to an extensive set of functions for hardware and software development, fault isolation, and debugging. Any level of user can use these functions.

The default prompt on SPARC systems is displayed as follows:

```
{0} ok
```

The number inside the braces represents the CPU identification (CPUID) of the thread of execution on which OpenBoot is running.

You can enter the OpenBoot environment by:

- Halting the OS
- Sending a break from the SP
- Power cycling the system

  If your virtual machine is not configured to boot automatically, the virtual machine stops at the OpenBoot CLI.

If automatic booting is configured, you can make the virtual machine stop at the OpenBoot CLI by sending a break from Oracle ILOM after the display console banner is displayed, but before the virtual machine starts booting the OS.

- When the hardware detects an error from which it cannot recover.

The console is used as the primary means of communication between OpenBoot and the user. The console consists of an input device, used for receiving information supplied by the user, and an output device, used for sending information to the user. Typically, the console is either the combination of a text and graphics display device and a keyboard or an ASCII terminal connected to a serial port.

For more information about accessing the OpenBoot CLI, see "Accessing the OpenBoot CLI and Getting Help" on page 21. Also refer to the administration guide for your platform. Oracle SPARC server documentation is available at `http://docs.oracle.com/en/servers/`. In the administration guide, look for sections titled *Obtain the OpenBoot Prompt* or *Getting to the ok Prompt*. For example, this URL takes you to the relevant section in the Oracle SPARC T7 Series Servers Administration Guide: `http://docs.oracle.com/cd/E54990_01/html/E55000/z40002fe1298584.html`.

You can use these variables to assign the power-on defaults. These values do not take effect until after the next power cycle or reset.

Related Information

- "Console I/O Control" on page 30
- "Command Completion Keystrokes" on page 31
- "Virtual machine Information Commands" on page 32
- "Obtain Virtual machine Information with OpenBoot Commands" on page 33

Console I/O Control

The console is the pair of input and output devices that the OpenBoot firmware uses to communicate with the user (for example, a keyboard and a bit-mapped display). The console devices are selected after probing, allowing the use of plug-in devices for the console. After probing, the drivers for devices named by `input-device` and `output-device` are opened, and console input and output is directed to those devices. The `ihandles` of the open input and output drivers are saved as the values of the `stdin` and `stdout` properties in the `/chosen` node, so that client programs can interact with the user through the console. If either of the specified devices cannot be opened, system-dependent default devices can be used instead of the specified devices.

The console activation process is performed by the `install-console` command. Normally, `install-console` is automatically executed during the OpenBoot firmware start-up sequence just after probing, but it can be executed explicitly from an NVRAMRC script if desired.

If you want to change the input and output console configuration, see "Set the Input and Output Device" on page 63.

Related Information

- "OpenBoot CLI Overview" on page 29
- "Command Completion Keystrokes" on page 31
- "Virtual machine Information Commands" on page 32
- "Obtain Virtual machine Information with OpenBoot Commands" on page 33

Command Completion Keystrokes

The command completion extension enables OpenBoot to complete long FORTH word names by searching the dictionary for one or more matches based on the already-typed portion of a word. When you type a portion of a word followed by the command completion keystroke, Control-Space, OpenBoot behaves as follows:

- If OpenBoot finds exactly one matching word, the remainder of the word is automatically displayed.
- If OpenBoot finds several possible matches, it displays all of the characters that are common to all of the possibilities.
- If OpenBoot cannot find a match for the already-typed characters, it deletes characters from the right until there is at least one match for the remaining characters.
- The system beeps if it can not determine an unambiguous match.

In addition to the keystrokes listed in this table, you can also use NVRAMRC script editor keystrokes on the command line. See "NVRAMRC Script Editor Keystroke Commands" on page 91.

Keystroke	Description
Control-Space	Completes the name of the current word.
Control-?	Displays all possible matches for the current word.
Control-/	Displays all possible matches for the current word.

Related Information

- “NVRAMRC Script Editor Keystroke Commands” on page 91
- “Virtual machine Information Commands” on page 32
- “Obtain Virtual machine Information with OpenBoot Commands” on page 33

Virtual machine Information Commands

The OpenBoot CLI provides commands to display virtual machine information. `banner` is provided by all OpenBoot implementations; the remaining commands represent extensions provided by some implementations.

Note - Additional commands and procedures are described in “Interrogating the System With OpenBoot Commands” on page 77.

Command	Description
`banner`	Displays the power-on banner.
`.enet-addr`	Displays current Ethernet address.
`fcode-revision`	Returns the revision level of FCode interface. The human-readable representation of the revision level is a string of the form *major.minor*, where *major* and *minor* are decimal numbers. The `fcode-revision` returns a single number representation whose value is given by the formula (major + minor). For example, if the release number were 2.12, the return value would be `0x0002.000C`. The revision level of the device interface described by this standard is `3.0`. Therefore, `fcode-revision` returns `0x0003.0000`. This FCode returns the revision level of the FCode device interface (that is, which FCodes are supported). Virtual machines which support OpenBoot return a value of `(hex) 0003.0000` (that is, 3.0), or possibly greater, as might be required by future editions of this specification. OpenBoot version 2.x systems return a similar encoding, `(hex) 0002.00xx`. OpenBoot version 1.x systems return a value of `(hex) 0000.xxxx`.
`.idprom`	Displays the property name to specify the IDPROM contents. *prop-encoded-array*: Byte array, encoded with `encode-bytes`. The 32-byte value of this property is the verbatim contents of the IDPROM structure, which contains the machine’s serial number, Ethernet address, and other information.
`.traps`	Displays a list of SPARC trap types.
`show-devs`	Shows all devices beneath the indicated node.

Command	Description
	Skips leading space delimiters. Parses *device-specifier* delimited by a space. Discards the remainder of the command line. Shows the full device path for each device in the subtree of the device tree underneath the specified node. The search process by which the specified node is located is as defined in 4.3, using the rules given for `find-device`. If *device-specifier* is the empty string (that is, there is nothing on the command line following `show-devs`), shows all devices. Syntax:show-devs *device-specifier* For further details and examples, see "Display the Device Tree" on page 82.
`.version`	Displays the version of the OpenBoot firmware. For example: `{0} ok` **`.version`** `Release 4.40.4.build_03 created 2016/08/17 14:05`

Related Information

- "Obtain Virtual machine Information with OpenBoot Commands" on page 33
- "OpenBoot CLI Overview" on page 29

▼ Obtain Virtual machine Information with OpenBoot Commands

This procedure provides examples of OpenBoot commands that provide information about the virtual machine.

1. **Access the OpenBoot CLI.**

   See "Accessing the OpenBoot CLI and Getting Help" on page 21.

2. **Run a display command.**

   For a list of commands, see "Virtual machine Information Commands" on page 32.

   Examples:

   - Display the banner.

     ```
     {0} ok banner

     SPARC T7-1, No Keyboard
     Copyright (c) 1998, 2016, Oracle and/or its affiliates. All rights reserved.
     OpenBoot 4.41.0, 31.0000 GB memory installed, Serial #104812436.
     [bp-4.41]
     ```

```
Ethernet address 0:10:e0:3f:4f:94, Host ID: 863f4f94.
```

- Display the Ethernet address.

```
{0} ok .enet-addr
0:10:e0:xx:xx:xx
```

- Display the IDPROM contents.

```
{0} ok .idprom
Format/Type: 1 86 Ethernet: 0 10 e0 3f 4f 94 Date: 0 0 0 0
Serial: 3f 4f 94 Checksum: 77
```

- Display the revision of the OpenBoot image.

```
{0} ok .version
Release 4.41 created 2016/11/15 18:59
```

- Display a list of devices.

  Also see “Display the Device Tree” on page 82.

```
{0} ok show-devs
/pci@345
/pci@344
/pci@343
/pci@342
/pci@341
/cpu@d40
/cpu@d3f
/cpu@d3e
/cpu@d3d
/cpu@d3c
/pci@345/pci@1
/pci@344/pci@1
/pci@343/pci@2
/pci@343/pci@2/usb@0
/pci@343/pci@2/usb@0/storage@1
/pci@343/pci@2/usb@0/storage@1/disk
/pci@342/pci@1
```

- Display disks.

```
{0} ok show-disks
a) /reboot-memory@0
b) /pci@311/pci@1/usb@0/storage@1/disk
c) /pci@312/pci@1/usb@0/storage@1/disk
d) /pci@300/pci@2/scsi@0/disk
e) /pci@302/pci@1/scsi@0/disk
```

```
f) /iscsi-hba/disk
q) NO SELECTION
Enter Selection, q to quit
: q
{0} ok
```

- Display networks.

  In this example, the output displays the first ten networks, and is terminated by enter `q` for quit. To see the full list, enter `m` to see more networks.

```
{0} ok show-nets
a) /pci@310/pci@1/network@0,1
b) /pci@310/pci@1/network@0
c) /pci@30e/pci@1/network@0,1
d) /pci@30e/pci@1/network@0
e) /pci@300/pci@1/network@0,1
f) /pci@300/pci@1/network@0
g) /pci@30b/pci@1/network@0,1
h) /pci@30b/pci@1/network@0
i) /pci@301/pci@1/network@0,1
j) /pci@301/pci@1/network@0
m) MORE SELECTIONS
q) NO SELECTION
Enter Selection, q to quit
: q
{0} ok
```

Related Information

- “Virtual machine Information Commands” on page 32
- “Setting Configuration Variables” on page 55
- “OpenBoot CLI Overview” on page 29
- “Command Completion Keystrokes” on page 31

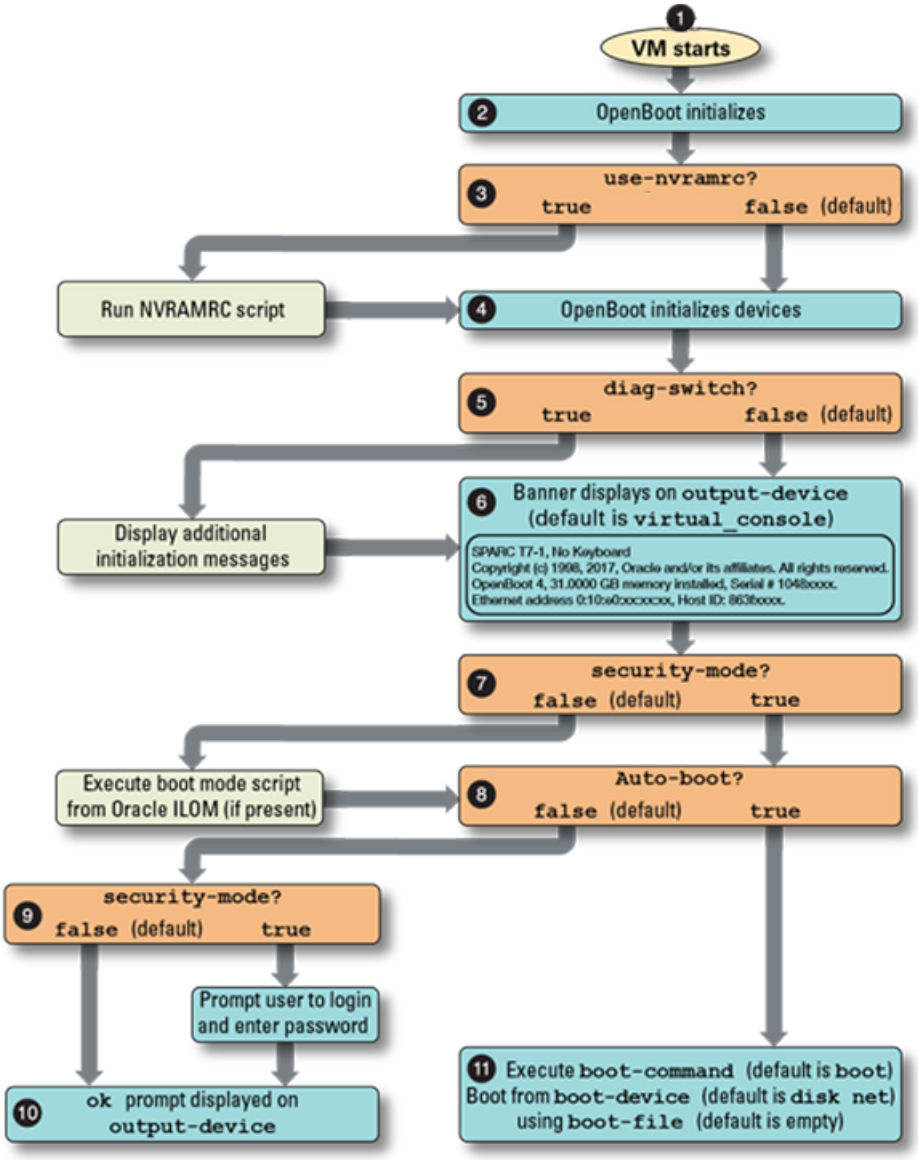
Booting and Resetting a Virtual Machine

These topics explain how to boot and reset virtual machines using OpenBoot.

Description	Links
Learn about start-up and booting concepts.	“Start-up Sequence” on page 37 “Boot Sequence” on page 40 “Boot Pools and Fallback Boot Images” on page 43 “`boot` Command Overview” on page 46
Boot a system or VM.	“Perform a Default Boot” on page 48 “Boot a Virtual machine From a Specific Device” on page 49
Learn about and boot a virtual machine over the network.	“Booting Over the Network” on page 50
Reset a virtual machine.	“Reset a Virtual machine” on page 54

Start-up Sequence

This flowchart provides a summary of a typical OpenBoot start-up sequence after a power-on or reset.



1. When the system or VM is started, OpenBoot is initialized.
2. Initialization – OpenBoot initializes its own data structures and those devices necessary for its own execution. The precise details of this initialization depend on the implementation and the computer system, but the these steps are often included (in no particular order):

- Determine the memory configuration.
- Select and prepare the memory to be used for FORTH and OpenBoot data structures like stacks, memory allocation pools, and device tree internal structures.
- Initialize various devices (for example, MMUs, interrupt controllers, and timers) that are required for the basic functioning of FORTH and OpenBoot.
- Reset the configuration variables contained therein to their default values.

3. The `use-nvramrc?` parameter is evaluated.
   - If `true`, a custom start-up script is executed (see "Customizing Start-Up with NVRAMRC" on page 87).
   - If `false`, OpenBoot does not seek to execute a custom start-up script.
4. Open boot runs `probe-all` to identify and initialize built-in and plug-in devices.
5. The `diag-switch?` parameter is evaluated.
   - If `true`, OpenBoot provides visibility into the PCIE probe sequence by displaying nodes as they are found. It also displays additional information about file system identification and the files that are loaded.
   - If false, OpenBoot does not display additional initialization messages.
6. The banner is displayed on the device specified for the `output-device` parameter. By default, the banner provides this information:
   - System type
   - OpenBoot version
   - Amount of installed memory
   - System serial number
   - Ethernet address
   - Host ID number

   The banner can be manually displayed with the banner command. See "Obtain Virtual machine Information with OpenBoot Commands" on page 33.The banner can be customized. See "Change the Power-On Banner" on page 63
7. The `security-mode?` parameter is evaluated.
   - If `false`, OpenBoot checks to see if a boot mode script is specified in Oracle ILOM. If a boot mode script is found, OpenBoot executes the script.
   - If `true`, OpenBoot does not check for a boot mode script.
8. The `auto-boot?` parameter is evaluated.
   - If `true`, OpenBoot attempts to boot according to OpenBoot parameters.

Note - To enable automatic booting, the OpenBoot `auto-boot?` variable must be set to `true` *and* the Oracle ILOM `auto-boot` property must be enabled.

- If `false`, OpenBoot does not attempt to boot.

9. The `security-mode?` parameter is evaluated.
   - If `false`, OpenBoot does not prompt the user to login and provide a password.
   - If `true`, OpenBoot prompts the user to login and provide the password. See "Setting Security Variables" on page 71.
10. OpenBoot displays the `ok` prompt on the device specified for the `output-device` parameter.
11. OpenBoot attempts to boot according to the values specified for `boot-command`, `boot-device`, and `boot-file`. See "Setting Configuration Variables" on page 55.

Related Information

- "Boot Sequence" on page 40
- "Boot Pools and Fallback Boot Images" on page 43
- "`boot` Command Overview" on page 46
- "Setting Configuration Variables" on page 55
- "Customizing Start-Up with NVRAMRC" on page 87

Boot Sequence

The most important function of OpenBoot firmware is to boot the virtual machine. Booting is the process of loading and executing a stand-alone program such as an OS.

Booting can either be initiated automatically after the start-up sequence, or by typing a `boot` command at the OpenBoot CLI. The boot process is controlled by a number of configuration variables. For information on how to set the variables, see "Setting Configuration Variables" on page 55.

These are the boot related configuration variables:

- `auto-boot?` – As described in "Start-up Sequence" on page 37, if true, attempt to boot according to OpenBoot parameters. If `false`, stop and invoke the OpenBoot CLI.

Note - To enable automatic booting, the OpenBoot `auto-boot?` variable must be set to `true` *and* the Oracle ILOM `auto-boot` property must be enabled.

- `boot-command` – The command specified for this variable is executed if `auto-boot?` is `true`. The default is `boot` with no arguments.
- `diag-switch?` – `false` by default, when set to `true`, it sets OpenBoot to diagnostic mode which provides additional displayed messages.
- `boot-device` –Specifies the device from which to boot. The value can be a device alias or a device path. The default is usually `disk net`, which attempts to boot from the device specified for the `disk` alias. If that fails, OpenBoot then attempts to boot from the device specified for the `net` alias.
- `boot-file` – Specifies the default arguments for `boot`, if `diagnostic-mode?` is `false`.

The behavior of that program can be controlled by an argument string that is made available to the program. Often, this program is a secondary boot-loader, whose purpose is to load yet another program. The secondary boot-loader might be capable of using additional protocols other than the protocol that OpenBoot used to load the first program. For example, OpenBoot can use the Trivial File Transfer Protocol (TFTP) to load the secondary boot-loader, which then uses the Network File System (NFS) protocol to load the OS from another file.

For disk booting, OpenBoot might load the secondary boot loader by reading the first few sectors from the disk, and that secondary boot loader might understand the OS's native file system structure, loading the OS from such a file.

Typical secondary boot loaders accept arguments in this form:

filename -flags ...

Where *filename* is the name of a file containing the OS, and *-flags* is a list of options controlling the details of the start-up phase of either the secondary boot loader, the OS, or both. However, from OpenBoot's point of view, the boot arguments are an opaque string that is passed uninterpreted to the boot program.

The device path of the boot device is also available to client programs, so they can determine the device from which they were booted.

The `boot` command parses the command line as follows:

- If the word following `boot` begins with a "`/`", the word is a device path and, thus, a device specifier.
- Otherwise, if there is a device alias matching that word, the word is a device specifier.
- If that word is neither a device path nor a known alias, the default boot device is used, and the word is included in arguments.

Assuming that `disk0` is predefined as a device alias for a valid device path, these are examples of valid `boot` commands:

```
ok boot (performs a default boot using values specified in configuration variables)
ok boot disk0 (boots from disk0 and uses boot program default arguments)
ok boot disk0 vmunix -asw (boots from disk0 and passes boot program vmunix -asw)
ok boot vmunix -asw (boots from default device and passes boot program vmunix -asw)
```

The `boot` command might restart the virtual machine in certain conditions and continue with the boot process when the virtual machine is restarted.

Booting usually happens automatically based on the values contained in the configuration variables described above. However, you can also initiate booting from the OpenBoot CLI. For specific booting instructions, refer to these procedures:

- "Perform a Default Boot" on page 48
- "Boot a Virtual machine From a Specific Device" on page 49
- "Boot Over the Network" on page 52

OpenBoot performs the following steps during the boot process:

- The firmware might reset the virtual machine if a client program has been executed since the last reset. (The execution of a reset is implementation dependent.)
- A device is selected by parsing the `boot` command line to determine the boot device and the boot arguments to use. Depending on the form of the `boot` command, the boot device and argument values might be taken from configuration variables.
- The `bootpath` and `bootargs` properties in the `/chosen` node of the device tree are set with the selected values.
- The selected program is loaded into memory using a protocol that depends on the type of the selected device. For example, a disk boot might read a fixed number of blocks from the beginning of the disk, while a tape boot might read a particular tape file.
- If Verified Boot is enabled, then signatures on the selected program are verified before the program is executed. If signature verification fails, the virtual machine continues to boot with warning messages displayed if verified boot policy is set to warning, or the virtual machine halts with FATAL messages if the policy is set to enforce. Details on how to set up Verified Boot and other related settings are available at: `https://docs.oracle.com/cd/E53394_01/html/E54828/sysauth-vb.html`.
- The loaded program is executed. The behavior of the program might be further controlled by any argument string that was either contained in the selected configuration variable or was passed to the `boot` command on the command line.

  The program loaded and executed by the boot process can be a *secondary boot loader*, whose purpose is to load yet another program. This secondary boot loader might use a protocol different from that used by OpenBoot. For example, OpenBoot might use the TFTP

to load the secondary boot loader while the secondary boot loader might then use the NFS protocol to load the OS.

Typical secondary boot loaders accept arguments of the form:

filename -flags

Where *filename* is the name of the file containing the OS and where *-flags* is a list of options controlling the details of the start-up phase of either the secondary boot loader, the OS or both. As shown in the `boot` command template, OpenBoot treats all such text as a single, opaque *arguments* string that has no special meaning to OpenBoot itself. The arguments string is passed unaltered to the specified program.

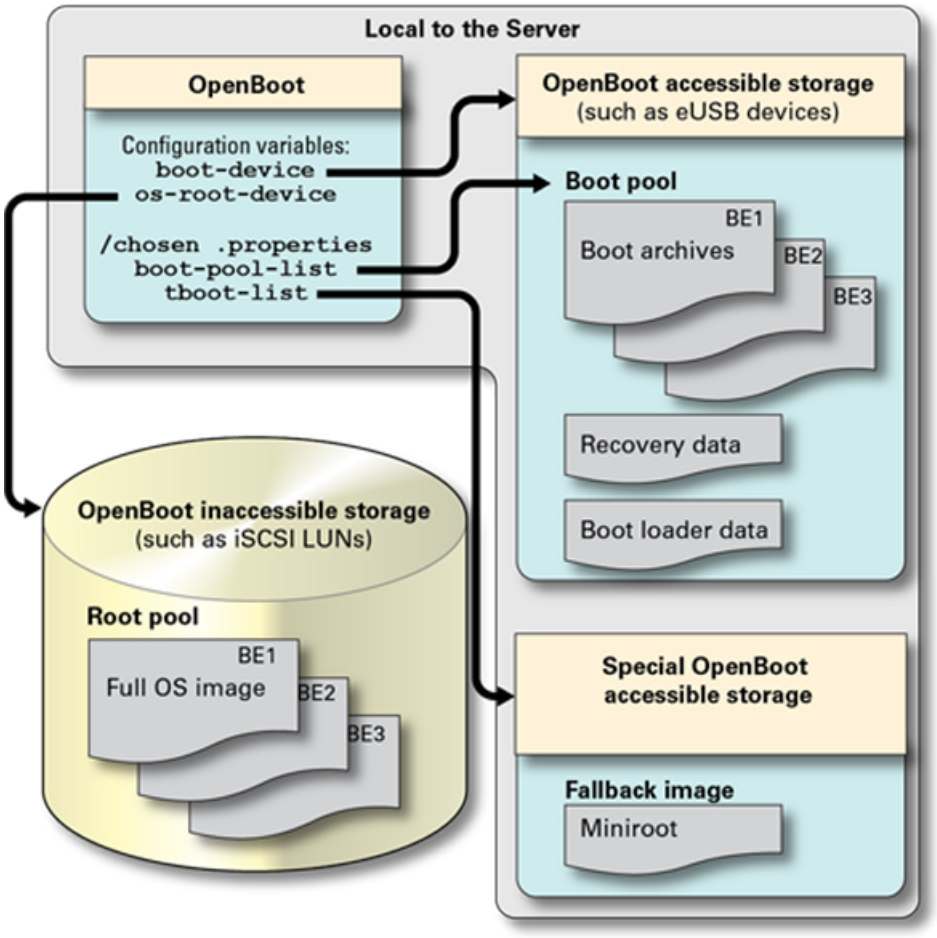
Related Information

- “Start-up Sequence” on page 37
- “Boot Pools and Fallback Boot Images” on page 43
- “`boot` Command Overview” on page 46
- “Setting Configuration Variables” on page 55

Boot Pools and Fallback Boot Images

The boot process has been enhanced to enable booting from devices that are not directly accessible to OpenBoot. This feature requires OS support (Oracle Solaris 11.3 includes this support) and support in OpenBoot (available in OpenBoot version 4.37.3 and later. To identify the OpenBoot version,see “Obtain Virtual machine Information with OpenBoot Commands” on page 33).

The systems that support this feature boot from an OpenBoot accessible boot pool that is located in the system, for example, from an eUSB device in each physical domain. Then the boot process uses certain OpenBoot configuration variables and properties (shown in this illustration) to enable the virtual machine to locate an OS that is not directly accessible to OpenBoot.



The OpenBoot parameters perform these functions:

Note - Except for the `boot-device` variable, the variables and properties listed here are usually automatically maintained and do not require intervention.

- `boot-device` – Is an OpenBoot configuration variable that specifies boot devices. In this diagram, the `boot-device` variable is set to the path of a local device, such as an eUSB device, where a boot pool is available. The setting is viewable using the OpenBoot `printenv` command or by using the Oracle Solaris `eeprom` command.

- `os-root-device` – Is an OpenBoot configuration variable that specifies properties that describe to the OS how to access the root file system. This setting is viewable using the OpenBoot `printenv` command or by using the Oracle Solaris `eeprom` command.
- `boot-pool-list` – Is an OpenBoot property for the `/chosen` node. This property specifies platform-defined devices that can be used to comprise a boot pool. Note - the list does not indicate if a device provides a boot pool or not. You can view this read-only property with the `.properties` command under the `/chosen` node at the OpenBoot prompt.
- `tboot-list` – Is an OpenBoot property for the `/chosen` node. This property contains a list of device paths that might allow access to fallback boot images (only if the platform has installed or flashed the image in non-volatile storage or if bootable media is inserted in the devices referenced by the property). You can view this read-only property with the `.properties` command under the `/chosen` node at the OpenBoot prompt.

The other items shown in the illustrations serve these purposes:

- **Boot pool** – A collection of boot archives that are stored on firmware-assessable devices. Each dataset in the boot pool is linked to a specific boot environment on a specific root pool.
- **Boot archive** – Is a file that contains the OS kernel and modules necessary to locate and mount the root file system to complete the second stage of booting the OS.
- **Recovery data and Boot loader data**– Provide a means to reconstitute the `os-root-device` property in the event that the property is reset, corrupted, or erased.
- **Root pool** – Is a ZFS pool that contains a set of boot environments (datasets of the form *pool_name*`/ROOT/`*BE_name*), each of which constitutes a root file system for an instance of the Oracle Solaris OS. A root pool can reside on a device such as an iSCSI device accessed through IP over Infiniband (IPoIB). This storage might not be directly accessible from OpenBoot. In such cases, when a root pool is created, a new boot pool is also automatically created on OpenBoot-accessible devices.
- **Fallback boot image** – (Also known as the fallback miniroot) is a boot image that can be booted if no devices in the boot pool are accessible from OpenBoot. This image is managed differently depending on the platform. For example, on some systems the boot image is directly accessible to root domains to which the USB controller for the RKVMS is associated. In other systems, no `tboot-list` may exist and so the fallback image must be manually managed and added to the `boot-device list`, This is because the Oracle Solaris installer does not know which device references a fallback miniroot image. For further details, refer to your platform documentation.

For detailed instructions on how you can manage and configure boot pools, refer to the Oracle Solaris 11.3 documentation at `http://www.oracle.com/goto/solaris11/docs`. Refer to the document titled *Booting and Shutting Down Oracle Solaris 11.3 Systems*.

Also refer to the documentation for your server platform to determine if these features are supported. Server documentation is available at: `http://docs.oracle.com/en/servers/`

Related Information

- “Start-up Sequence” on page 37
- “Boot Sequence” on page 40
- “boot Command Overview” on page 46
- “Setting Configuration Variables” on page 55

boot Command Overview

Note - When booting a virtual machine that is configured to boot by default, you can usually type `boot` and you do not need to specify the boot device nor any arguments that are described in this section. For such situations, instead see “Perform a Default Boot” on page 48.

This is the `boot` command syntax:

```
{0} ok boot [device-specifier] [arguments]
```

Note - Most commands (such as `boot` and `test`) that require a device name accept either a full device path name or a device alias. In this document, the term *device-specifier* indicates that either an appropriate device path name or a device alias is acceptable for such commands.

Name	Description
device-specifier	The name (full path name or alias) of the boot device. Typical values include: ■ `cdrom` (CD-ROM drive) ■ `disk` (hard disk) ■ `net` (Ethernet) If *device-specifier* is not specified and if `diagnostic-mode?` returns `false`, `boot` uses the device specified by the `boot-device` configuration variable. The term *device-specifier* denotes a string that is either a device path, or an alias that refers to a non-leaf node followed by additional *node-name* components.
arguments	The name of the program to be booted (for example, *stand/diag*) and any program arguments. If *arguments* is not specified, `boot` uses the file specified by the `boot-file` configuration variable.
filename	The name of the program to be booted (for example, `stand/diag`). *filename* is relative to the root of the selected device and partition (if specified). If *filename* is not specified, the boot program uses the value specified in the `boot-file` parameter.

Name	Description
options	These options are specific to the OS, and might differ from virtual machine to virtual machine.

Because a device alias cannot be syntactically distinguished from the *arguments*, OpenBoot resolves this ambiguity as follows:

- If the space-delimited word following `boot` on the command line begins with `/`, the word is a device-path and, thus, a *device-specifier*. Any text to the right of this *device-specifier* is included in *arguments*.
- If the space-delimited word matches an existing device alias, the word is a *device-specifier*. Any text to the right of this *device-specifier* is included in *arguments*.
- Otherwise, the appropriate default boot device is used, and any text to the right of `boot` is included in *arguments*.

Consequently, `boot` command lines have the following possible forms.

`{0} ok` **`boot`**

With this form, `boot` loads and executes the program specified by the default boot arguments from the default boot device.

`{0} ok` **`boot`** *device-specifier*

If `boot` has a single argument that either begins with the character `/` or is the name of a defined `devalias`, `boot` uses the argument as a device specifier. `boot` loads and executes the program specified by the default boot arguments from the specified device.

If `boot` has a single argument that neither begins with the character `/` nor is the name of a defined `devalias`, `boot` uses all of the remaining text as its arguments.

`{0} ok` **`boot`** *arguments*

`boot` loads and executes the program specified by the arguments from the default boot device.

`{0} ok` **`boot`** *device-specifier arguments*

If there are at least two space-delimited arguments, and if the first such argument begins with the character `/` or if it is the name of a defined `devalias`, `boot` uses the first argument as a device specifier and uses all of the remaining text as its arguments. `boot` loads and executes the program specified by the arguments from the specified device.

For all of the above cases, `boot` records the device that it uses in the `bootpath` property of the `/chosen` node. `boot` also records the arguments that it uses in the `bootargs` property of the `/chosen` node.

Device alias definitions vary from virtual machine to virtual machine. Use the `devalias` command (described in “Create a Device Alias” on page 61) to obtain the definitions of your virtual machine's aliases.

For a description of the boot sequence changes and instructions on how you can manage the boot pool by using the `bootadm boot-pool`, refer to the Oracle Solaris 11.3 documentation at `https://docs.oracle.com/en/operating-systems/solaris.html`

.

Caution - The Verified Boot policy setting of `enforce` does not allow the boot process to proceed if the OpenBoot `use-nvramrc?` variable is set to `true`. You can directly set the `use-nvramrc?` variable with the `setenv` command, or the variable is automatically set to `true` when you use the `nvalias` command. If you set the `use-nvramrc?` variable to `false`, you will not be able to create device aliases with the `nvalias` command. For further details, refer to `https://docs.oracle.com/cd/E37444_01/html/E37446/z40001291613819.html`.

Related Information

- “Start-up Sequence” on page 37
- “Boot Sequence” on page 40
- “Boot Pools and Fallback Boot Images” on page 43
- “Perform a Default Boot” on page 48
- “Setting Configuration Variables” on page 55

▼ Perform a Default Boot

This procedure assumes that the virtual machine's boot configuration variables (see “Boot Sequence” on page 40) are set to boot from an established boot device.

Based on the values of the configuration variables, the boot process can proceed in a number of different ways. For example:

- If `auto-boot?` is `true` (the default), the virtual machine will automatically boot from the specified boot device. Note – The Oracle ILOM `auto-boot` property must also be set to `enabled`.
- If `auto-boot?` is `false`, automatic booting does not occur, the `ok` prompt is displayed, and you can run OpenBoot commands such as `boot`.

1. **Access the OpenBoot CLI.**

See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **Type:**

```
{0} ok boot
```

Related Information

- “Boot Sequence” on page 40
- “`boot` Command Overview” on page 46
- “Setting Configuration Variables” on page 55

▼ Boot a Virtual machine From a Specific Device

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **(Optional) Display device aliases.**

   If you need to identify a device alias for a boot device, type:

Note - A device alias does not indicate whether a device is bootable, but most virtual machines have a device alias configured for the boot device.

```
{0} ok devalias
screen                   /pci@300/pci@4/display@0
fallback-miniroot        /pci@300/pci@2/usb@0/hub@3/storage@1/disk@0
rcdrom                   /pci@300/pci@2/usb@0/hub@3/storage@1/disk@0
primary-vds0             /virtual-devices@100/channel-devices@200/virtual-disk-server@0
primary-vsw0             /virtual-devices@100/channel-devices@200/virtual-network-switch@0
primary-vcc0             /virtual-devices@100/channel-devices@200/virtual-console-concentrator@0
net3                     /pci@300/pci@3/network@0,1
net2                     /pci@300/pci@3/network@0
cdrom                    /pci@300/pci@2/usb@0/hub@8/device@1/storage@0/disk@0
net1                     /pci@300/pci@1/network@0,1
net                      /pci@300/pci@1/network@0
net0                     /pci@300/pci@1/network@0
scsi                     /pci@301/pci@2/scsi@0
scsi0                    /pci@301/pci@2/scsi@0
disk7                    /pci@301/pci@2/scsi@0/disk@p7
disk6                    /pci@301/pci@2/scsi@0/disk@p6
disk5                    /pci@301/pci@2/scsi@0/disk@p5
disk4                    /pci@301/pci@2/scsi@0/disk@p4
disk3                    /pci@301/pci@2/scsi@0/disk@p3
```

```
disk2                   /pci@301/pci@2/scsi@0/disk@p2
disk1                   /pci@301/pci@2/scsi@0/disk@p1
```

Also see “Create a Device Alias” on page 61.

3. **Boot the virtual machine by specifying boot parameters.**

   Type the `boot` command with an explicit boot device. The virtual machine boots from the specified boot device using the default boot arguments.

   - To explicitly boot from the device that is defined by the `disk` device alias, type:

     ```
     {0} ok boot disk
     ```

   - To explicitly boot from the network that is defined for the `net` device alias, type:

     ```
     {0} ok boot net
     ```

   - To boot from a specific device, type:

     ```
     {0} ok boot /pci@301/pci@2/scsi@0/disk@w5000cca123456789,0:a
     ```

Related Information

- “Boot Sequence” on page 40
- “`boot` Command Overview” on page 46
- “Setting Configuration Variables” on page 55

Booting Over the Network

Use these topics to understand how to boot over the network using OpenBoot.

Network Booting Process

Network devices are packet-oriented devices capable of sending and receiving packets (frames) that are addressed according to Local Area Network (LAN) specifications for Media Access Control (MAC) addresses administered by the IEEE Registration Authority. OpenBoot supports booting network devices.

The network boot process involves:

1. Obtaining the IP address of the booting client.

   The client knows its Ethernet address and system type, but needs its IP address to transfer the files it needs.
2. Downloading the standalone boot program and executing it.

   The client uses specified network protocol to download the standalone boot program and executes it.

A client booting over a network can use RARP to obtain its IP address. When booting Oracle Solaris software, the loaded standalone program is `inetboot`, which uses the RPC protocol `bootparams` to obtain boot parameters, and loads the kernel and executes it. To boot with RARP and `bootparams`, use this command:

```
{0} ok boot network-device-specifier
```

Clients that are DHCP aware can use DHCP to obtain the IP address, boot parameters, and network configuration information with more efficiency and flexibility than the combination of the RARP and `bootparams` services. In addition, using DHCP removes the requirement for a boot server on every subnet. To boot with DHCP, use this command:

```
{0} ok boot device-specifier:dhcp
```

DHCP aware PROM clients support interoperability with BOOTP servers. The client prefers DHCP configurations over BOOTP, but accepts BOOTP configurations if no DHCP configuration is offered.

The default protocol used (that is, RARP or DHCP) when the command `boot net` is executed depends on how the `net` device alias is specified. If the net devalias specifies only the path to the network device, RARP is used as the default address discovery protocol. If the device alias includes `dhcp` as an argument, DHCP is used.

To boot using RARP (the default):

Note - The *device-specifier* string might be different on your system.

```
{0} ok boot /pci@300/pci@1/network@0
```

To boot using DHCP:

```
{0} ok boot /pci@300/pci@1/network@0:dhcp
```

You can set the desired device alias by using the `nvalias` command. See “NVRAMRC Editor Commands” on page 89.

Related Information

- “Boot Over the Network” on page 52
- “Arguments Supported by Network Boot” on page 52

Boot Over the Network

1. **On the virtual machine you plan to boot, Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **Boot the virtual machine over the network.**

   For a detailed description of the boot syntax and valid boot arguments, see “Arguments Supported by Network Boot” on page 52.

   Network booting examples:

   - Using RARP (the default) and booting from the device that is specified for the `net` parameter.

     ```
     {0} ok boot net
     ```

   - Using RARP (the default) and specifying a specific *device-specifier*:

     ```
     {0} ok boot /pci@300/pci@1/network@0
     ```

   - Using DHCP:and booting from the device specified for the `net` parameter.:

     ```
     {0} ok boot net:dhcp
     ```

   - Using DHCP:and specifying a specific *device-specifier*:

     ```
     {0} ok boot /pci@300/pci@1/network@0:dhcp
     ```

Related Information

- “Network Booting Process” on page 50
- “Arguments Supported by Network Boot” on page 52

Arguments Supported by Network Boot

This is the syntax for booting over the network:

```
boot <network-device>:[dhcp|bootp],[server-IP],
[boot-filename],[client-IP],[router-IP],[boot-retries],
[tftp-retries],[subnet-mask][boot-arguments]
```

All arguments are optional. Commas are required for missing positional parameters unless they are at the end of the list.

server-IP, *client-IP*, *router-IP*, and *subnet-mask* are specified in Internet standard *dotted-decimal* notation. If any of *server-IP*, *boot-filename*, *client-ip*, *router-IP*, and *subnet-mask* are specified, the ROM client uses these values instead of any values which are (or might be) obtained by the normal configuration process.

The network boot arguments can also be specified by setting the `network-boot-arguments` configuration variable (see "Setting Configuration Variables" on page 55).

For example:

```
{0} ok setenv network-boot-arguments host-ip=client-IP,router-ip=router-ip,subnet-
mask=mask-value,hostname=client-name,http-proxy=proxy-ip:port,file=wanbootCGI-URL
```

This table lists the valid arguments for the network `boot` command.

Argument	Description
boot-filename	Is the name of the standalone program to be loaded by TFTP from the server. The default file name is constructed from the IP address if the boot protocol is RARP, or from the client class identifier if using DHCP or BOOTP.
boot-retries	Is the maximum number of retries attempted before the boot process is determined to have failed.
client-IP	Is the IP address of the client (that is, the virtual machine being booted).
`dhcp`	Specifies the use of DHCP as the address discovery protocol to be used. `bootp` is treated as a synonym of DHCP (that is, the client still uses DHCP format messages). The client accepts a BOOTP configuration only if no DHCP configurations are offered.
host-ip	Is the IP address of the network interface on the virtual machine being booted.
router-IP	Is the IP address of router to be used.
server-IP	Is the IP address of the TFTP server from which the standalone boot program is to be downloaded.
subnet-mask	Is the *subnet mask* on the local network.
tftp-retries	Is the maximum number of retries attempted before the TFTP process is determined to have failed.

Related Information

- "Network Booting Process" on page 50
- "Boot Over the Network" on page 52

Reset a Virtual machine

Occasionally, you might need to reset your virtual machine. The `reset-all` command resets the virtual machine.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **At the OpenBoot prompt, type:**

   ```
   {0} ok reset-all
   ```

Related Information

- “Accessing the OpenBoot CLI and Getting Help” on page 21
- “Using the OpenBoot CLI” on page 29

Setting Configuration Variables

Use these topics to understand what configuration variables are available, and how to use OpenBoot to set the configuration variables.

Configuration Variables Overview

A number of OpenBoot operating characteristics are controlled by configuration variables that are stored in nonvolatile memory. The list of configuration variables varies from system to system.

These variables determine the start-up configuration and related communication characteristics. You can modify the values of the configuration variables, and any changes you make remain in effect even after a power cycle.

Most configuration variables have both a current value and a default value, with the default value stored in ROM. OpenBoot implementations can maintain a checksum of the nonvolatile memory used for configuration variable storage. If that memory becomes corrupted, the OpenBoot implementation can restore some of the configuration variables to their default values, leaving untouched those configuration variables without default values.

The nonvolatile memory locations where particular parameter values are stored can change from machine to machine and from revision to revision. Thus, they cannot be accessed at fixed locations, but instead must be accessed by name. You can access them by name using `printenv` and `setenv`. Client programs can access them by name through client interface operations on the `/options` device node.

The properties of the `/options` node are the virtual machine's configuration variables. The property names are the names of those configuration variables, and the property values are the output text representations of those configuration variables. Client programs can examine and change the values of these properties with `getprop` and `setprop`, thus examining and changing the values of the corresponding configuration variables. Similarly, you can examine and change them with `printenv`, `setenv`, and `$setenv` commands.

Use these topics to understand the standard configuration variables and the commands that you can use to modify the configuration variables.

- "Standard Configuration Variables" on page 56
- "Configuration Variable Commands" on page 58

Standard Configuration Variables

This section provides a list of the configuration variables that are typically included in SPARC systems.

Note - Different OpenBoot implementations might use different defaults or provide different configuration variables. Use the OpenBoot `printenv` command to display the variables on your virtual machine. See "Display the Current Variable Settings" on page 59.

The value of a configuration variable can be a number, a string, a true or false flag, a selection from a set of choices, or one of several other data types, depending on the particular variable. The stack comment shows the way that information is presented on the FORTH stack.

Name	Default Value	Stack Comment	Description
`auto-boot?`	`true`	`( -- auto? )`	If `true`, boots automatically after power on or reset. The Oracle ILOM `auto-boot` property must also be enabled to allow automatic booting to occur.
`boot-command`	`boot`	`( -- addr len )`	Command that is executed if `auto-boot?` is `true`.
`boot-device`	`disk net`	`( -- dev-str dev-len )`	Device from which to boot.
`boot-file`	empty string	`( -- arg-str arg-len )`	Default arguments for `boot`.

Name	Default Value	Stack Comment	Description
			Configuration variable type: `string[32]`. Suggested default value: an empty string.
`diag-switch?`	`false`	`( -- diag? )`	If `true`, run in diagnostic mode. Diagnostic mode provides visibility into the PCIE probe sequence by printing nodes as they are found. It also provides some visibility into the boot sequence by displaying file system identification and files loaded.
`fcode-debug?`	`false`	`( -- names? )`	If `true`, includes name fields for plug-in device FCodes.
`input-device`	`virtual-console`	`( -- dev-str dev-len )`	Console input device.
`network-boot-arguments`	empty string	`( -- arg-str arg-len )`	Arguments to the `obp-tftp` package, if no arguments are specified to the `obp-tftp` 'open' method.
`nvramrc`	empty	`( -- data-addr data-len )`	Contents of `nvramrc`.
`oem-banner`	empty string	`( -- text-str text-len )`	Custom OEM banner (enabled by `oem-banner? true`).
`oem-banner?`	`false`	`( -- custom? )`	If `true`, use custom OEM banner.
`oem-logo`	No default.	`( -- logo-addr logo-len )`	Byte array custom OEM logo (enabled by `oem-logo? true`). Displayed in hexadecimal.
`oem-logo?`	`false`	`( -- custom? )`	If `true`, uses custom OEM logo (else, uses the Oracle logo).
`output-device`	`virtual-console`	`( -- dev-str dev-len )`	Console output device.
`os-root-device`	empty string	`( -- dev-str dev-len )`	A semicolon-separated, technology-specific list of key value pairs. Defines devices and root file systems for root pools. See "Boot Pools and Fallback Boot Images" on page 43.
`screen-#columns`	`80`	`( -- n )`	Number of on-screen columns (characters/line).
`screen-#rows`	`34`	`( -- n )`	Number of on-screen rows (lines).
`security-#badlogins`	No default.	`( -- n )`	Number of incorrect security password attempts.
`security-mode`	`none`	`( -- n )`	Firmware security level (options: `none`, `command`, or `full`).
`security-password`	No default.	`( -- password-str password-len )`	Firmware security password (never displayed).
`use-nvramrc?`	`false`	`( -- enabled? )`	If `true`, execute commands in `nvramrc` during start-up.
`local-mac-address?`	`true`	`( -- enabled? )`	If `true`, network devices use their own MAC addresses.
`error-reset-recovery`	`boot`	`( -- n )`	Recovery action after an error reset CPU trap (options: `none`, `sync`, or `boot`).

Related Information

- "Configuration Variables Overview" on page 55
- "Configuration Variable Commands" on page 58

- “Display the Current Variable Settings” on page 59
- “Change a Variable Setting” on page 60

Configuration Variable Commands

You can modify the values of the system configuration variables that are stored in persistent storage using the OpenBoot commands listed in this section.

Alternatively, you can use the Oracle Solaris `eeprom` command for modifying OpenBoot configuration variables when the virtual machine is running the OS.

All configuration variable modifications persist across a power cycle as long as the configuration of the virtual machine does not change.

Caution - Use extreme care when changing a configuration variable. An incorrect setting or typo can prevent the virtual machine from booting.

Name	Description	Procedure
`printenv`	Displays all current parameters and current default values. Numbers are usually shown as decimal values. `printenv` *parameter* shows the current value of the named parameter.	“Display the Current Variable Settings” on page 59
`setenv` *parameter value*	Sets parameter to the specified decimal or text value. (Changes are permanent, but usually only take effect after a reset.)	“Change a Variable Setting” on page 60
`set-default` *parameter*	Resets the value of the named parameter to the factory default.	“Reset a Variable to Its Default Value” on page 66 **Note -** To reset all the variables, see “Set Up the Security Password” on page 71.
`set-defaults`	Resets all variables to default values.	“Reset All Variables to Default Values (Using OpenBoot CLI)” on page 67

Related Information

- “Configuration Variables Overview” on page 55
- “Standard Configuration Variables” on page 56

Display the Current Variable Settings

Use this procedure to display all the OpenBoot configuration variables, their settings, and default values.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **Use the `printenv` command to display all the variables.**

   In the formatted list of the current settings, numeric variables are often shown in the decimal format.

```
{0} ok printenv
Variable Name           Value                     Default Value

reboot-command
security-mode           none                      No default
security-password                                 No default
security-#badlogins     0                         No default
verbosity               min                       min
diag-switch?            true                      false
local-mac-address?      true                      true
fcode-debug?            true                      false
scsi-initiator-id       7                         7
oem-logo                                          No default
oem-logo?               false                     false
oem-banner                                        No default
oem-banner?             false                     false
ansi-terminal?          true                      true
screen-#columns         80                        80
screen-#rows            34                        34
ttya-mode               9600,8,n,1,-              9600,8,n,1,-
output-device           virtual-console           virtual-console
input-device            virtual-console           virtual-console
auto-boot-on-error?     false                     false
load-base               16384                     16384
auto-boot?              false                     true
os-root-device
network-boot-arguments
boot-command            boot                      boot
```

```
boot-file
boot-device             vdisk                       disk net
multipath-boot?         false                       false
boot-device-index       0                           0
use-nvramrc?            false                       false
nvramrc
error-reset-recovery    boot                        boot
```

Related Information

- “Configuration Variables Overview” on page 55
- “Standard Configuration Variables” on page 56
- “Configuration Variable Commands” on page 58
- “Change a Variable Setting” on page 60
- “Reset a Variable to Its Default Value” on page 66
- “Reset All Variables to Default Values (Using Oracle ILOM CLI)” on page 68

▼ Change a Variable Setting

Use this procedure to change an OpenBoot configuration variable setting.

Caution - Use extreme care when changing a configuration variable. An incorrect setting or typo can prevent the virtual machine from booting.

Note - Many variable changes do not affect the operation of the firmware until the next power cycle or reset, at which time the firmware uses the new value of the variable.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **At the OpenBoot prompt, use this syntax:**

   ```
   {0} ok setenv variable-name value
   ```

   Where *variable-name* is the name of the variable and *value* is a numeric value or text string appropriate for the named variable. A numeric value is interpreted as a decimal number, unless preceded by `0x`, which is the qualifier for a hexadecimal number.

   For example, to set the `auto-boot?` variable to `false`, type:

   ```
   {0} ok setenv auto-boot? false
   ```

Related Information

- “Configuration Variables Overview” on page 55
- “Standard Configuration Variables” on page 56
- “Configuration Variable Commands” on page 58
- “Display the Current Variable Settings” on page 59
- “Reset a Variable to Its Default Value” on page 66
- “Reset All Variables to Default Values (Using Oracle ILOM CLI)” on page 68

▼ Create a Device Alias

Virtual machines have predefined device aliases for the most commonly used devices, however, you can create, modify, and examine aliases with the `devalias` command. User-defined aliases are lost after a reset or power cycle, but you can create a persistent alias by storing the `devalias` command in the NVRAMRC script. You can enter the `devalias` command in the script either manually, or by using the `nvalias` command. For more details, see “Customizing Start-Up with NVRAMRC” on page 87.

This procedure uses the `nvalias` command.

Name	Description
`devalias`	Displays current aliases.
`devalias` *alias*	Displays the device path name that corresponds to the alias.
`devalias` *alias device-path*	Defines an alias representing *device-path*. If an alias with the same name already exists, the new value supersedes the old. **Note -** The alias is lost upon the next reset or power cycle. Use the `nvalias` command to create a persistent device alias.
`nvalias` *alias device-path*	Creates a nonvolatile device alias by adding the `devalias` command line into the NVRAMRC script. For details, see “NVRAMRC Editor Commands” on page 89. **Note -** The `use-nvramrc?` variable is automatically set to `true` when the `nvalias` command is used.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **View the current device aliases.**

```
{0} ok devalias
screen                   /pci@300/pci@4/display@0
fallback-miniroot        /pci@300/pci@2/usb@0/hub@3/storage@1/disk@0
```

```
rcdrom                  /pci@300/pci@2/usb@0/hub@3/storage@1/disk@0
primary-vds0            /virtual-devices@100/channel-devices@200/virtual-disk-server@0
primary-vsw0            /virtual-devices@100/channel-devices@200/virtual-network-switch@0
primary-vcc0            /virtual-devices@100/channel-devices@200/virtual-console-concentrator@0
net3                    /pci@300/pci@3/network@0,1
net2                    /pci@300/pci@3/network@0
cdrom                   /pci@300/pci@2/usb@0/hub@8/device@1/storage@0/disk@0
net1                    /pci@300/pci@1/network@0,1
net                     /pci@300/pci@1/network@0
net0                    /pci@300/pci@1/network@0
scsi                    /pci@301/pci@2/scsi@0
scsi0                   /pci@301/pci@2/scsi@0
disk7                   /pci@301/pci@2/scsi@0/disk@p7
disk6                   /pci@301/pci@2/scsi@0/disk@p6
disk5                   /pci@301/pci@2/scsi@0/disk@p5
disk4                   /pci@301/pci@2/scsi@0/disk@p4
disk3                   /pci@301/pci@2/scsi@0/disk@p3
disk2                   /pci@301/pci@2/scsi@0/disk@p2
disk1                   /pci@301/pci@2/scsi@0/disk@p1
```

3. **Create the device alias.**
   This example uses the `nvalias` command to create a persistent device alias.

   ```
   {0} ok nvalias my_device /pci@301/pci@2/scsi@0/disk@w5000cca123456789,0:a
   ```

   If the NVRAMRC script was successfully modified, but `use-nvramrc?` is `false`, the command sets `use-nvramrc?` to `true`.

4. **Verify the device alias.**
   For example:

   ```
   {0} ok devalias my_device
   my_device               /pci@301/pci@2/scsi@0/disk@w5000cca123456789,0:a
   ```

5. **Verify that the `use-nvramrc?` variable is set to `true`.**
   For example:

   ```
   {0} ok printenv use-nvramrc?
   use-nvramrc? =          true
   ```

Related Information

- “Configuration Variables Overview” on page 55
- “Standard Configuration Variables” on page 56
- “Configuration Variable Commands” on page 58
- “Display the Current Variable Settings” on page 59

Set the Input and Output Device

The `input-device` and `output-device` variables control the firmware's selection of input and output devices after a power-on reset. These are the default values:

- `input-device` – `virtual-console`
- `output-device` – `virtual-console`

The values of must be device specifiers.

When the virtual machine is reset, the named device becomes the initial firmware console input or output device. (If you want to temporarily change the input or output device, use the `input` or `output` commands.

1. **Specify the console input device.**

   `{0} ok` **`setenv input-device`** *device-specifier*

2. **Specify the console output device.**

   `{0} ok` **`setenv output-device`** *device-specifier*

Related Information

- “Configuration Variables Overview” on page 55
- “Standard Configuration Variables” on page 56
- “Configuration Variable Commands” on page 58
- “Display the Current Variable Settings” on page 59

Change the Power-On Banner

The banner is displayed at a system-dependent screen location (usually either at the top of the screen or at the current cursor position). If the current output device has a `device_type` property whose value is `display`, displays a logo by executing the current output device’s `draw-logo` method with the following arguments:

- The *line#* argument, at the system’s discretion, is either `0` or the line number corresponding to the current cursor position.
- If `oem-logo?` is `true`, the *addr* argument is the address returned by `oem-logo`. Otherwise, it is the address of the system-dependent default logo. The width and height arguments are both 64. Note – `oem-logo?` is only effective for graphics-based console output devices.

In any case, display additional information as follows:

- If `oem-banner?` is `true`, display the text given by the value of `oem-banner`.
- Otherwise, display implementation-dependent information about the system (for example, the machine type, serial number, firmware revision, network address, and hardware configuration).

If executed within an NVRAMRC script, it suppresses automatic execution of the following OpenBoot start-up sequence:

- `probe-all`, `install-console`, `banner`

  See also: `suppress-banner`.
- `oem-banner` `( --` *text-str text-len* `)`.
- `oem-banner?` `( -- custom? )`.

  If `true`, `banner` displays custom message in `oem-banner`. If `false`, `banner` displays the normal system-dependent messages.

Although it is not a direct preventative or detective control, a banner can be used for these reasons:

- Convey ownership.
- Warn users of the acceptable use.
- Indicate that access or modifications to OpenBoot variables is restricted to authorized personnel.

The banner configuration variables are:

- `oem-banner`
- `oem-banner?`
- `oem-logo`
- `oem-logo?`

The banner consists of two parts: the text field and the logo (over serial ports, only the text field is displayed). You can replace the existing text field with a custom text message using the `oem-banner` and `oem-banner?` configuration variables. The banner can be up to 68 characters. All printable characters are accepted.

The graphic logo is handled differently. The `oem-logo` variable is a 512-byte array, containing a total of 4096 bits arranged in a 64 x 64 array. Each bit controls one pixel. The most significant bit (MSB) of the first byte controls the upper-left corner pixel. The next bit controls the pixel to the right of it, and so on.

To restore the original power-on banner, set the `oem-logo?` and `oem-banner?` variables to `false`, as in these examples:

```
{0} ok setenv oem-logo? false
{0} ok setenv oem-banner? false
```

Because the `oem-logo` array is so large, `printenv` displays approximately the first 8 bytes (in hexadecimal). To display the entire array, type `oem-logo dump`. The `oem-logo` array is not erased by `set-defaults`, since it might be difficult to restore the data. However, `oem-logo?` is set to false when `set-defaults` executes, so the custom logo is no longer displayed.

Note - Some systems do not support the `oem-logo` feature.

---

1. View the current power-on banner.

```
{0} ok banner
SPARC T5-8, No Keyboard
Copyright (c) 1998, 2015, Oracle and/or its affiliates. All rights reserved.
OpenBoot 4.37.4.build_xx, 315.5000 GB memory available, Serial #136621172.
Ethernet address x:xx:xx:xx:xx:xx, Host ID: 1234ac12.
{0} ok
```

The banner for your system will be different.

2. If desired, insert a custom text field.

```
{0} ok setenv oem-banner Hello World
oem-banner = Hello World
{0} ok setenv oem-banner? true
oem-banner? = true
{0} ok banner


Hello World



{0} ok
```

3. If desired, change the logo.

This example fills the top half of the `oem-logo` variable with an ascending pattern.

a. Create a FORTH array containing the correct data.

```
{0} ok create logoarray d# 512 allot
```

b. Copy the array into the `oem-logo` variable.

```
{0} ok logoarray d# 256 0 do i over i + c! loop drop
```

c. Install the array in the `oem-logo?` variable with the `$setenv` command.

```
{0} ok logoarray d# 256 " oem-logo" $setenv
```

d. Change the `oem-logo` variable setting.

```
{0} ok setenv oem-logo? true
```

e. Verify the output.

```
{0} ok banner
```

Related Information

- “Configuration Variables Overview” on page 55
- “Standard Configuration Variables” on page 56
- “Configuration Variable Commands” on page 58
- “Display the Current Variable Settings” on page 59

▼ Reset a Variable to Its Default Value

Use this procedure to reset an individual an OpenBoot configuration variable to its default value.

Caution - Use extreme care when changing a configuration variable. An incorrect setting or typo can prevent the virtual machine from booting.

Note - Many variable changes do not affect the operation of the firmware until the next power cycle or reset, at which time the firmware uses the new value of the variable.

1. Access the OpenBoot CLI.

See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. At the OpenBoot prompt, use this syntax:

```
{0} ok set-default variable-name
```

Where *variable-name* is the name of the variable.

For example, to set the `auto-boot?` variable to its default value (`true`), type:

```
{0} ok set-default auto-boot?
```

Related Information

- “Standard Configuration Variables” on page 56
- “Display the Current Variable Settings” on page 59
- “Reset All Variables to Default Values (Using OpenBoot CLI)” on page 67
- “Reset All Variables to Default Values (Using Oracle ILOM CLI)” on page 68

Reset All Variables to Default Values (Using OpenBoot CLI)

Use this procedure to reset All OpenBoot configuration variables to default values.

Caution - Use extreme care when changing a configuration variable. An incorrect setting or typo can prevent the virtual machine from booting.

Note - Many variable changes do not affect the operation of the firmware until the next power cycle or reset, at which time the firmware uses the new value of the variable.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **At the OpenBoot prompt, type:**

   ```
   {0} ok set-defaults
   ```

Related Information

- “Standard Configuration Variables” on page 56
- “Display the Current Variable Settings” on page 59
- “Reset a Variable to Its Default Value” on page 66
- “Reset All Variables to Default Values (Using Oracle ILOM CLI)” on page 68

Reset All Variables to Default Values (Using Oracle ILOM CLI)

Oracle ILOM provides a set of host boot mode properties that enables you to override the default method for booting the virtual machine OS on a SPARC server.

The host boot mode properties in Oracle ILOM are intended to help resolve corrupt boot mode settings with OpenBoot or LDoms. The boot mode properties, when set in Oracle ILOM, apply only to a single boot and expire within 10 minutes if the power on the host SPARC server is not reset.

For additional details about this Oracle ILOM feature, refer to the section called *Overriding SPARC Host Boot Mode* in the *Oracle ILOM Administration Guide for Configuration and Maintenance* document for the version of Oracle ILOM running on your system. This URL goes to that section for version 3.2.x `http://docs.oracle.com/cd/E37444_01/html/E37446/z40000061582150.html`.

Note - The Oracle ILOM CLI examples in this section are for a single PDomain system. For multiple domain systems, replace `/HOST` with `/HOST`*x*, where *x* is the PDomain number.

---

1. **Access Oracle ILOM.**

   You can use the Oracle ILOM web interface or CLI to set the host boot mode properties. This procedure uses the Oracle ILOM CLI.

2. **At the Oracle ILOM CLI prompt, enter one of these command lines:**

   For a single-domain server, type:

   ```
   -> set /HOST/bootmode state=reset_nvram
   ```

3. **Reset the virtual machine.**

   - **For a single-domain server, type:**

     ```
     -> reset /System
     ```

   - **For a multi-domain server, type:**

     ```
     -> reset /HOSTx
     ```

     Where *x* is the PDomain number.

Related Information

- “Standard Configuration Variables” on page 56
- “Display the Current Variable Settings” on page 59
- “Reset a Variable to Its Default Value” on page 66
- “Reset All Variables to Default Values (Using OpenBoot CLI)” on page 67

Setting Security Variables

The security variables enable the virtual machine to be configured so that a password is required to access most commands from the OpenBoot `ok` prompt. Several levels of security (including `none`) can be configured.

These topics describe how to configure OpenBoot security variables:

▼ Set Up the Security Password

Use this procedure to create or change and existing OpenBoot password.

Caution - It is important to remember your security password and to configure the security password *before* setting the security mode. If you forget this password, you cannot use your virtual machine; you must call your vendor's customer support service to make your virtual machine bootable again.

The security password you assign must be between zero and eight characters. Any characters after the eighth are ignored. You do not have to reset the virtual machine; the security feature takes effect immediately. All printable characters are accepted. Control characters are not accepted.

Note - Setting the password to zero characters turns off security and treats the `security-mode` parameter as if it were set to `none`. However, it does not change the setting.

Once security mode is enabled, if you enter an incorrect security password, there will be a delay of about 10 seconds before the next boot prompt appears. The number of times that an incorrect security password is typed is stored in the security-#badlogins variable.

1. **Access the OpenBoot CLI.**
   See "Accessing the OpenBoot CLI and Getting Help" on page 21.

2. **At the OpenBoot prompt, type:**

```
{0} ok password
New password (8 characters max): password
Retype new password: password
{0} ok
```

Related Information

- "Set the security-mode Variable" on page 72
- "Disable the security-mode Variable" on page 74
- "Check for Failed Log-Ins" on page 74
- "Configure OpenBoot Keys on an Installation Client" on page 75

▼ Set the security-mode Variable

The security-mode variable can restrict the set of operations that users are allowed to perform from the OpenBoot CLI.

Caution - It is important to remember your security password and to configure the security password *before* setting the security mode. If you forget this password, you cannot use your virtual machine; you must call your vendor's customer support service to make your virtual machine bootable again.

1. **Access the OpenBoot CLI.**
   See "Accessing the OpenBoot CLI and Getting Help" on page 21.

2. **Set the OpenBoot password before setting the security-mode variable.**
   See "Set Up the Security Password" on page 71.

3. **At the OpenBoot prompt, type:**

```
{0} ok setenv security-mode name
```

Where *name* is one of the three security mode values that are listed in this table (from most to least secure).

For business continuity reasons, consider setting the security-mode parameter to command.

Name	Description
full	A password is required to perform any action, including the boot and go commands.
command	Applies these policies: ■ A password is not required if you type the boot command by itself or the go command. However, if you use the boot command with an argument, a password is required. ■ A password is required to execute any other command.
none	No password is required (default).

4. Obtain the security mode prompt.

After setting the security mode, there are two ways to obtain the security mode prompt.

- **Use the `logout` and `login` words.**

```
{0} ok logout
Type boot , go (continue), or login (command mode)
>
> login
Firmware Password: password
Type help for more information
{0} ok
```

To exit the security mode, use the logout and login names, as shown in the example.

- **Use the `reset-all` command.**

```
{0} ok reset-all
```

This command resets the virtual machine. When the virtual machine comes back up, OpenBoot goes to the security mode prompt. To log back in to the command prompt (or log out of the security mode), use the logout and login names, and then enter the password, as described above.

Related Information

- "Set Up the Security Password" on page 71
- "Disable the security-mode Variable" on page 74
- "Check for Failed Log-Ins" on page 74

- “Configure OpenBoot Keys on an Installation Client” on page 75

Disable the security-mode Variable

Caution - When security-mode is set to none, the OpenBoot password is not automatically cleared. If security-mode is later enabled, the existing password takes effect. If the original password is forgotten, you cannot use your virtual machine and you must call your vendor's customer support service to make your virtual machine bootable again. To avoid this scenario, always clear the password when you disable the security-mode, as described in this procedure.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **Set the security-mode variable to none.**

```
{0} ok setenv security-mode none
```

3. **Set the password to zero length by typing Return after both password prompts.**

```
{0} ok password
New password (8 characters max):
Retype new password:
{0} ok
```

Related Information

- “Set Up the Security Password” on page 71
- “Set the security-mode Variable” on page 72
- “Check for Failed Log-Ins” on page 74
- “Configure OpenBoot Keys on an Installation Client” on page 75

Check for Failed Log-Ins

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **Display the value for security-#badlogins.**

   You can determine if someone has attempted and failed to access the OpenBoot environment by displaying the security-#badlogins variable. For example:

```
{0} ok printenv security-#badlogins
```

If this command returns any value greater than `0`, a failed attempt to access the OpenBoot environment was recorded.

3. **Reset the variable.**

```
{0} ok setenv security-#badlogins 0
```

Related Information

- "Set Up the Security Password" on page 71
- "Set the `security-mode` Variable" on page 72
- "Disable the `security-mode` Variable" on page 74
- "Configure OpenBoot Keys on an Installation Client" on page 75

▼ Configure OpenBoot Keys on an Installation Client

To enable security for SPARC clients, you must generate an OpenBoot HMAC key and encryption key for each client . These keys also secure the download of the initial network boot files.

Note - Check the product documentation for availability of HMAC keys for a secured installation.

This procedure is based on an installation server that is running the Oracle Solaris 11 OS, and a SPARC-based installation client. For details on how to prepare the installation server, refer to the section called *Installing Using an Install Server* in the *Installing Oracle Solaris 11.3 Systems* document at: `https://docs.oracle.com/cd/E53394_01/html/E54756`.

For Oracle Solaris 10 OS instructions, refer to the *Solaris 10 1/13 Installation Guide: Network-Based Installations* guide at: `http://docs.oracle.com/cd/E26505_01/html/E28037/index.html` The instructions are in Installing Over a Wide Area Network chapter, and in the Installing Keys on the Client section.

1. **Access the OpenBoot CLI.**

   See "Accessing the OpenBoot CLI and Getting Help" on page 21.

2. **On the installation client, set the OpenBoot keys.**

   This example sets the OpenBoot AES encryption key on a SPARC installation client.

```
{0} ok set-security-key wanboot-aes 030fd11c98afb3e434576e886a094c1c
```

This example sets the OpenBoot hashing (HMAC) key on a SPARC installation client.

```
{0} ok set-security-key wanboot-hmac-sha1 e729a742ae4ba977254a2cf89c2060491e7d86eb
```

Note - To unset a key on the client, use the same command that you used to set the key, but do not provide any key value. For example: `set-security-key wanboot-hmac-sha1`.

Once the installation server and client are set up, boot the client from the network. See "Boot Over the Network" on page 52.

Related Information

- "Set Up the Security Password" on page 71
- "Set the `security-mode` Variable" on page 72
- "Disable the `security-mode` Variable" on page 74
- "Check for Failed Log-Ins" on page 74

Interrogating the System With OpenBoot Commands

OpenBoot provides commands that you can use to gather information about the system hardware. Some commands also perform a low level sanity tests of the hardware without a running OS.

These topics are provided in this section:

▼ Probe All SCSI Devices

You can use the OpenBoot `probe-scsi-all` command to probe the SCSI buses and to display information about the attached devices. The command displays the device path, device type, target, unit numbers, and the unique WWN.

1. **Access the OpenBoot CLI.**

   See "Accessing the OpenBoot CLI and Getting Help" on page 21.

2. **Run the `probe-scsi-all` command.**

   Example:

```
{0} ok probe-scsi-all
/pci@300/pci@2/usb@0/hub@4/storage@2
  Unit 0   Disk     smiMICRON  eUSB DISK       1111

/pci@300/pci@2/usb@0/hub@3/storage@1
  Unit 0   Removable Read Only device    SUNRemote ISO CDROM1.01
```

```
/pci@300/pci@2/usb@0/hub@8/device@1/storage@0
  Unit 0   Removable Read Only device    TEAC    DV-W28S-A 9.2A

/pci@301/pci@2/scsi@0

FCode Version 1.00.64, MPT Version 2.05, Firmware Version 5.00.00.00

Target 9
  Unit 0   Disk   HITACHI  H109060SESUN600G A606    1172123568 Blocks,
600 GB
  SASDeviceName 5000cca043d8bdf0  SASAddress 5000cca043d8bdf1 PhyNum 0
Target a
  Unit 0   Disk   HITACHI  H109060SESUN600G A606    1172123568 Blocks,
600 GB
  SASDeviceName 5000cca043daf4d8  SASAddress 5000cca043daf4d9 PhyNum 4
```

Related Information

- “Monitor Network Interfaces” on page 78
- “List All NVMe Devices” on page 80
- “Browsing the Device Tree” on page 80

▼ Monitor Network Interfaces

You can use the `watch-net` and `watch-net-all` OpenBoot commands to check the status of network interfaces. The commands send packets addressed to itself, and watches the packets return.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **Identify the network specified for the `net` device alias.**

   When you run the `watch-net` command, the network specified by the `net` device alias is the network that is tested. The `devalias` command displays that network.

   ```
   {0} ok devalias net
   net   /pci@300/pci@1/network@0
   ```

3. **Run the `watch-net` command to test the network interface that is specified by the `net` device alias.**

   The command receives Ethernet multicast packets , and reports the status with a . (period) for each error-free packet received. or with an `X` for each packet received with a error.

Example:

```
{0} ok watch-net
1000 Mbps full duplex Link up
Looking for Ethernet Packets.
'.' is a Good Packet.  'X' is a Bad Packet.
Type any key to stop.
.........................X.............................XX.....................
```

4. **Press any key to stop the activity.**

5. **Run the `watch-net-all` command to list all network interfaces and send and receive test packets.**

   The command reports the status with a . (period) for each error-free packet received. or with an `X` for each packet received with a error.

   Example:

```
{0} ok watch-net-all
Link down
Waiting for link
Link Down

/pci@300/pci@3/network@0
Link down
Waiting for link
Link Down

/pci@300/pci@1/network@0,1
Link down
Waiting for link
Link Down

/pci@300/pci@1/network@0
1000 Mbps full duplex Link up
Looking for Ethernet Packets.
'.' is a Good Packet.  'X' is a Bad Packet.
Type any key to stop.
................................................................................................
```

6. **Press any key to stop the activity.**

Related Information

- “Probe All SCSI Devices” on page 77
- “List All NVMe Devices” on page 80

- “Browsing the Device Tree” on page 80

List All NVMe Devices

You can use the OpenBoot `probe-nvme-all` command to list all the flash accelerator devices (sometimes referred to as NVMe) in a system. The information displayed can provide the information you need to set one of the devices as the boot device.

Note - Not all systems have flash accelerators.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **Run the OpenBoot `probe-nvme-all` command.**

   In this example, a server with one physical domain has two factory installed flash accelerator cards. The lowest numbered card contains the preinstalled Oracle Solaris OS software. The lowest-numbered card has the /`pci@301/pci@1/nvme@0` device path.

```
{0} ok probe-nvme-all
/pci@315/pci@1/nvme@0
     NVME Controller VID: 8086 SSVID: 108e SN:CVMD512100AA1P6N MN:  INTEL SSDPEDME016T4S
 FR: 8DV1RA13 NN: 1
         Namespace ID:1 Size: 1.600 TB
/pci@301/pci@1/nvme@0
     NVME Controller VID: 8086 SSVID: 108e SN:CVMD512100F81P6N MN:  INTEL SSDPEDME016T4S
 FR: 8DV1RA13 NN: 1
         Namespace ID:1 Size: 1.600 TB
```

Related Information

- “Probe All SCSI Devices” on page 77
- “Monitor Network Interfaces” on page 78
- “Browsing the Device Tree” on page 80

Browsing the Device Tree

The set of devices attached to the system, including permanently installed devices and plug-in devices, is described by an OpenBoot data structure known as the device tree.

You can inspect the device tree to determine the hardware configuration of the system. Each device in the device tree is described by a property list that you can view from the OpenBoot CLI.

A conceptual description of the device tree is covered in “Device Tree” on page 14.

Use these topics to browse the device tree:

- “Commands for Browsing the Device Tree” on page 81
- “Display the Device Tree” on page 82

Commands for Browsing the Device Tree

You can browse the device tree to examine individual device tree nodes. The device tree browsing commands are similar to the Oracle Solaris commands for changing, displaying, and listing the current directory in the Oracle Solaris directory tree. Selecting a device node makes it the current node.

For example:

Name	Description
`.properties`	Displays the names and values of properties of the active package.
`dev` *device-path*	Makes the specified device node the active package. Parses *device-specifier* delimited by `end-of-line`. Performs the equivalent of `find-device` with *device-specifier* as its argument.
`dev` *node-name*	Searches for a node with the given name in the subtree below the current node, and choose the first such node found.
`dev ..`	Chooses the device node that is the parent of the current node.
`dev /`	Chooses the root machine node.
`device-end`	Unselects the active package, leaving `none` selected.
`find-device`	Makes the device node *dev-string* the active package.
`ls`	Displays the names of the active package’s children.
`open-dev`	Opens device (and parents) named by given device specifier. Leaves instance handle (`ihandle`) on stack if device is opened.
`close-dev`	Closes device and all of its parents.
`pwd`	Displays the device path that names the active package.
`see`	Decompiles the FORTH command *old-name*.
`show-devs` *device-path*	Shows all devices beneath the indicated node.

Name	Description
	Skips leading space delimiters. Parses *device-specifier* delimited by a space. Discards the remainder of the command line. Shows the full device path for each device in the subtree of the device tree underneath the specified node. The search process by which the specified node is located is as defined in 4.3, using the rules given for `find-device`. If *device-specifier* is the empty string (that is, there is nothing on the command line following `show-devs`), shows all system devices.
`show-props` *device-path*	Displays the names and values of properties of the device-specifier.
`select-dev`	Opens the device (and parents) named by given device specifier. Sets `my-self` to instance handle (`ihandle`) of the device. Returns nothing on stack.
`unselect-dev`	Closes device and all of its parents, and undoes select-dev.
`words`	Displays the names of methods or commands. If there is an active package, displays the names of its methods. Otherwise, displays an implementation-dependent subset (preferably the entire set) of the globally visible FORTH commands. In either case, the order of display is to display more recently defined names before less recently defined names.

Related Information

- “Device Tree” on page 14
- “Display the Device Tree” on page 82

▼ Display the Device Tree

Device tree browsing enables you to examine and modify individual device tree nodes. The active package is the device node that you can examine and modify with subsequent node examination commands. Selecting a device node makes it the active package. When a node is the active package, user-created FORTH commands are added to the list of methods for that device. You can add new properties to the list of properties for that device. Dictionary search commands will operate on the node's list of methods (followed by system FORTH words), and that node's methods can be executed as FORTH words by typing their names.

For a list of commands, see “Commands for Browsing the Device Tree” on page 81.

1. **Access the OpenBoot CLI.**

   See “Accessing the OpenBoot CLI and Getting Help” on page 21.

2. **Use the `show-devs` command to see the all of the full device path names in the device tree.**

   For example:

```
{0} ok show-devs
/pci@345
/pci@344
/pci@343
/pci@342
/pci@341
/cpu@d40
/cpu@d3f
/cpu@d3e
/cpu@d3d
/cpu@d3c
/pci@345/pci@1
/pci@344/pci@1
/pci@343/pci@2
/pci@343/pci@2/usb@0
/pci@343/pci@2/usb@0/storage@1
/pci@343/pci@2/usb@0/storage@1/disk
/pci@342/pci@1
```

3. **Display the properties of a node.**

   For example:

```
{0} ok show-props /memory
reg                     00000000 50000000 0000001f 60000000
                        00004000 00000000 0000001f d0000000
                        00008000 00000000 0000003f d0000000
                        0000c000 00000000 0000003f d0000000
                        00010000 00000000 0000003f d0000000
                        00014000 00000000 0000003f d0000000
                        00018000 00000000 0000003f d0000000
                        0001c000 00000000 0000001f d0000000
available               0001c01f ce790000 00000000 00002000
                        0001c000 00000000 0000001f ce780000
                        00018000 00000000 0000003f d0000000
                        00014000 00000000 0000003f d0000000
                        00010000 00000000 0000003f d0000000
                        0000c000 00000000 0000003f d0000000
                        00008000 00000000 0000003f d0000000
                        00004000 00000000 0000001f d0000000
                        00000000 90000000 0000001f 20000000
                        00000000 8fe00000 00000000 0002a000
                        00000000 50400000 00000000 3e9c0000
name                    memory
```

```
{0} ok
```

4. **Select a node and display its properties.**

   The dev *device-path* command makes the node the active package.

   For example:

```
{0} ok dev /pci@301/pci@2/scsi@0
{0} ok .properties
firmware-version         5.00.00.00
mpt-version              2.05
local-wwid               50 80 02 00 01 aa 37 58
vf-assigned-addresses    83020000 00000001 00010000 00000000 00010000
assigned-addresses       81020010 00000000 00000000 00000000 00000100
                         83020014 00000001 00000000 00000000 00010000
                         82020030 00000000 00100000 00000000 00100000
vf-reg                   03020000 00000000 00000000 00000000 00010000
compatible               pciex1000,97
                         pci1000,97
model                    LSI,3008
reg                      00020000 00000000 00000000 00000000 00000000
                         01020010 00000000 00000000 00000000 00000100
                         03020014 00000000 00000000 00000000 00010000
                         02020030 00000000 00000000 00000000 00100000
version                  1.00.64
wide                     00000010
device_type              scsi-sas
name                     scsi
#address-cells           00000004
vf-stride                00000001
first-vf-offset          00000001
total-vfs                00000010
initial-vfs              00000010
#vfs                     00000010
port-type                PCIE-Endpoint
interrupts               00000001
cache-line-size          00000010
class-code               00010700
subsystem-id             00000090
subsystem-vendor-id      00001000
revision-id              00000002
device-id                00000097
vendor-id                00001000
{0} ok
```

5. **Display the names of the methods of the current node.**

```
{0} ok dev /pci@300/pci@2/disk@w5000cca123456789,0:a
```

```
{0} ok words
size          load          write         read          seek
close         open          write-blocks64
read-blocks64 write-blocks  read-blocks   max-transfer
r/w-blocks64  r/w-blocks    4c!           2c!
block-size    #blocks64     #blocks       read-block-size
read-block-extent           read-capacity-16-cmd
read-capacity-cmd           mode-sense-bd /block
timed-spin    sstart-cmd    init-label-package
report-failure              label-package offset-high
offset-low    set-timeout   init-deblocker
deblocker     eject         device-present?
eject-cmd     cb!           cmdbuf        /cmdbuf       4c@
3c@           -c@           3c!           +c!
short-data-command          no-data-command
retry-command parent-set-address          parent-max-transfer
dma-free      dma-alloc
{0} ok
```

6. Find a device.

`find-device` is similar to `dev`, differing only in the way the input path name is passed.

```
{0} ok "/usb@0/ACME,widget" find-device
```

Note - After choosing a device node with `dev` or `find-device`, you can not execute that node's methods because `dev` does not establish the current instance. For a detailed explanation of this issue, refer to *Writing FCode 3.x Programs*.

Related Information

- "Device Tree" on page 14
- "Commands for Browsing the Device Tree" on page 81

Customizing Start-Up with NVRAMRC

These topics describe how to use the OpenBoot NVRAMRC feature.

NVRAMRC Overview

You can use the `nvramrc` configuration variable to store user-defined FORTH commands that are executed during start-up. Commands are stored in ASCII, just as you would type them at the console.

The contents of the variable are called the NVRAMRC script.

The NVRAMRC script can be used to save start-up configuration variables, to patch device driver code, or to define installation-specific device configurations. It can also be used for bug patches or for user-installed extensions.

You can use the NVRAMRC script to create an alias for a device (see “Create a Device Alias” on page 61).

The custom start-up script occupies the portion of configuration memory that is not dedicated to other purposes such as configuration variables. The size of custom start-up script is determined by an implementation and is usually up to 1024 characters long.

While it is possible to alter the contents of the NVRAMRC script with `setenv` or the `$setenv` command, use of the NVRAMRC script editor (`nvedit`) is preferred.

The contents of the NVRAMRC script are cleared by `set-defaults`. Under some circumstances cleared contents can be recovered with `nvrecover`.

If the `use-nvramrc?` configuration variable is `true`, the NVRAMRC script is evaluated during the OpenBoot start-up sequence as shown:

1. Perform virtual machine initialization.
2. Evaluate the script (if `use-nvramrc?` is `true`).
3. Execute `probe-all` (evaluate FCode).
4. Execute `install-console`.
5. Execute `banner`.
6. Execute secondary diagnostics.
7. Perform default boot (if `auto-boot?` is `true` and `auto-boot` is enabled).

It is sometimes desirable to modify the sequence of `probe-all`, `install-console`, and `banner`. For example, commands that modify the characteristics of plug-in display devices might need to be executed after the plug-in devices have been probed, but before the console device has been selected. Such commands would need to be executed between `probe-all` and `install-console`. Commands that display output on the console would need to be placed after `install-console` or `banner`.

This is accomplished by creating a custom script which contains either `banner` or `suppress-banner` because the sequence `probe-all install-console banner` is not executed if either `banner` or `suppress-banner` is executed from the NVRAMRC script. This allows the use of `probe-all`, `install-console`, and `banner` inside the NVRAMRC script, possibly interspersed with other commands, without having those commands re-executed after the NVRAMRC script finishes.

Most OpenBoot commands can be used in the NVRAMRC script *except* for these commands:

- `boot`
- `go`
- `nvedit`
- `password`
- `reset-all`
- `setenv security-mode`

Caution - The Verified Boot policy setting of `enforce` does not allow the boot process to proceed if the OpenBoot `use-nvramrc?` variable is set to `true`. You can directly set the `use-nvramrc?` variable with the `setenv` command, or the variable is automatically set to `true` when you use the `nvalias` command. If you set the `use-nvramrc?` variable to `false`, you will not be able to create device aliases with the `nvalias` command. For further details, refer to https://docs.oracle.com/cd/E37444_01/html/E37446/z40001291613819.html.

Related Information

- "NVRAMRC Editor Commands" on page 89
- "NVRAMRC Script Editor Keystroke Commands" on page 91
- "Activate the NVRAMRC Script" on page 92
- "Example NVRAMRC Script" on page 92

NVRAMRC Editor Commands

Name	Description
`nvalias` *alias device-path*	Edits the NVRAMRC script to create a persistent device alias, as follows: Creates this command line in the NVRAMRC script: `devalias` *alias-name device-specifier* If the NVRAMRC script already contains a `devalias` line with the same alias name, deletes that entry and replaces it with the new entry at the same location in the NVRAMRC script. Otherwise, places the new entry at the beginning of the script. If there is insufficient space in the NVRAMRC script for the new `devalias` command, displays a message to that effect and aborts without modifying the script. If the NVRAMRC script was successfully modified, executes the new `devalias` command immediately, creating a new memory-resident alias. If the NVRAMRC script is currently being edited (that is, if `nvedit` has been executed, but has not been completed with either `nvstore` or `nvquit`), aborts with an error message before taking any other action. If the NVRAMRC script was successfully modified, but `use-nvramrc?` is `false`, sets `use-nvramrc?` to `true`. Syntax: `ok nvalias` *alias-name /full/pathname* `<eol>`
`$nvalias`	Creates nonvolatile device alias, and edits the NVRAMRC script. Performs the same function as `nvalias`, except that the parameters are stack strings. The alias name is specified by *name string*. The device specifier is specified by *dev-string*. Used as: `ok "` *new-alias*`" "` *device-specifier*`" $nvalias`
`nvedit`	Enters the NVRAMRC script editor (exits with `^c`). `nvedit` operates on a temporary buffer. If data remains in the temporary buffer from a previous `nvedit` command, editing resumes with those previous contents. If not, `nvedit` reads the contents of the NVRAMRC script into the temporary buffer and begins editing the temporary buffer.

Name	Description
	Editing continues until `^c` is typed, at which point editing ceases and normal operation of the command interpreter is resumed. The contents of the temporary buffer are not automatically saved to the NVRAMRC script. The `nvstore` command must be executed afterwards to save the buffer into the NVRAMRC script. The intra-line editing keystrokes are used within the NVRAMRC script editor with some additions.
`nvquit`	Discards the contents of `nvedit` temporary buffer. Prompts for confirmation of the user's intent to carry out this function. If confirmation is obtained, discards the `nvedit` temporary buffer. Otherwise, takes no further action.
`nvrecover`	Attempts to recover lost script contents. Attempts to recover the contents of the NVRAMRC script if it has been lost as a result of the execution of `set-default` or `set-defaults`. Enters the script editor as with the `nvedit` command. In order for `nvrecover` to succeed, `nvedit` must not have been executed between the time that the script contents were lost and the time that `nvrecover` is executed.
`nvrun`	Executes the contents of the temporary buffer.
`nvstore`	Copies the contents of `nvedit` temporary buffer into the NVRAMRC script. The `nvedit` temporary buffer is then cleared. Used after `nvedit` to save the results of an editing session into the NVRAMRC script.
`nvunalias` *alias*	Deletes nonvolatile device alias from the NVRAMRC script. If the NVRAMRC script contains a `devalias` command line with the same name as *alias-name*, deletes that command line from the script. Otherwise, leaves the script unchanged. If the script is currently being edited (that is, `nvedit` has been executed, but has not been completed with either `nvstore` or `nvquit`), aborts with an error message before taking any other action. Used as: `ok nvunalias` *alias-name*
`$nvunalias`	Deletes nonvolatile device alias from the NVRAMRC script. Performs the same function as `nvunalias`, except that the alias name is specified by *name-string*. Used as: `ok " ` *alias-name*`" $nvunalias`

Related Information

- "NVRAMRC Overview" on page 87
- "NVRAMRC Script Editor Keystroke Commands" on page 91
- "Activate the NVRAMRC Script" on page 92
- "Example NVRAMRC Script" on page 92

NVRAMRC Script Editor Keystroke Commands

You use the keystrokes in this table on the command line, or with the NVRAMRC script editor. Use the keystrokes to re-execute previous commands without retyping them, to edit the current command line, to fix typing errors, or to recall and change previous commands.

Keystroke	Description
Control-B	Moves backward one character.
Escape B	Moves backward one word.
Control-F	Moves forward one character.
Escape F	Moves forward one word.
Control-A	Moves backward to beginning of the line.
Control-E	Moves forward to end of the line.
Control-N	Moves to the next line of the NVRAMRC script editing buffer.
Control-P	Moves to the previous line of the NVRAMRC script editing buffer.
Return (Enter)	Inserts a new line at the cursor position and advances to the next line.
Control-O	Inserts a new line at the cursor position and stays on the current line.
Control-K	Erases from the cursor position to the end of the line, storing the erased characters in a save buffer. If at the end of a line, joins the next line to the current line (that is, deletes the new line).
Delete	Erases the previous character.
Backspace	Erases the previous character.
Control-H	Erases the previous character.
Escape H	Erases from beginning of word to just before the cursor, storing erased characters in a save buffer.
Control-W	Erases from beginning of word to just before the cursor, storing erased characters in a save buffer.
Control-D	Erases the next character.
Escape D	Erases from the cursor to the end of the word, storing the erased characters in a save buffer.
Control-U	Erases the entire line, storing the erased characters in a save buffer.
Control-Y	Inserts the contents of the save buffer before the cursor.
Control-Q	Quotes the next character (i.e. allows you to insert control characters).
Control-R	Retypes the line.
Control-L	Displays the entire contents of the editing buffer.
Control-C	Exits the NVRAMRC script editor, returning to the OpenBoot command interpreter. The temporary buffer is preserved, but is not written back to the script. (Use `nvstore` afterwards to write it back.)

Related Information

- “NVRAMRC Overview” on page 87
- “NVRAMRC Editor Commands” on page 89
- “Activate the NVRAMRC Script” on page 92
- “Example NVRAMRC Script” on page 92

▼ Activate the NVRAMRC Script

1. **At the OpenBoot prompt, type `nvedit`.**
   Edit the NVRAMRC script using the editor commands described in “NVRAMRC Editor Commands” on page 89.
2. **Press Control-C to get out of the editor and back to the OpenBoot prompt.**
   If you have not yet typed `nvstore` to save your changes, you can type `nvrun` to execute the contents of the temporary edit buffer.
3. **Type `nvstore` to save your changes.**
4. **Enable the interpretation of the NVRAMRC script by typing:**

   ```
   {0} ok setenv use-nvramrc? true
   ```

5. **Type `reset-all` to reset the virtual machine and execute the NVRAMRC script, or execute the contents directly by typing:**

   ```
   {0} ok nvramrc evaluate
   ```

Related Information

- “NVRAMRC Overview” on page 87
- “NVRAMRC Editor Commands” on page 89
- “NVRAMRC Script Editor Keystroke Commands” on page 91
- “Example NVRAMRC Script” on page 92

Example NVRAMRC Script

The following example shows you how to create a simple colon definition in the NVRAMRC script.

```
{0} ok nvedit
0: : hello ( -- )
1: ." Hello, world. " cr
2: ;
3: ^C
{0} ok nvstore
{0} ok setenv use-nvramrc? true
{0} ok reset-all
...
{0} ok hello
Hello, world.
{0} ok
```

Notice the `nvedit` line number prompts (`0:`, `1:`, `2:`, and `3:`) in this example. These prompts are system-dependent.

Related Information

- “NVRAMRC Overview” on page 87
- “NVRAMRC Editor Commands” on page 89
- “NVRAMRC Script Editor Keystroke Commands” on page 91
- “Activate the NVRAMRC Script” on page 92

Glossary

B

BE Boot environment. A bootable instance of the Oracle Solaris image. A BE can contain additional installed software packages.

boot pool A special pool on firmware-accessible devices that contains the set of files required to boot the Oracle Solaris kernel for a BE. Each dataset in the boot pool is linked to a BE. See also BE and pool.

BOOTP Bootstrap Protocol.

D

device-specifier Most commands (such as `boot`) that require a device name accept either a full device path name or a device alias. In this book, the term *device-specifier* indicates that either a device path name or a device alias is acceptable for such commands.

DHCP Dynamic Host Configuration Protocol. Software that automatically assigns IP addresses to clients on a TCP/IP network.

E

EEPROM Electrically erasable programmable read-only memory.

eUSB Embedded USB. A flash-based drive designed specifically to be used as a boot device. An eUSB does not provide storage for applications or customer data.

F

FCODE FORTH code. FORTH is a programming language used in OpenBoot.

G

GB Gigabyte. 1 gigabyte = 1024 megabytes.

GbE Gigabit Ethernet.

H

HMAC Hash message authentication code. Used in cryptography.

I

ILOM See Oracle ILOM.

InfiniBand A networking communications standard that features very high throughput and very low latency.

IPoIB Internet protocol over InfiniBand.

iSCSI Internet small computer system interface. An IP-based storage networking standard that enables a system to access storage across a network. In an iSCSI network, the remote storage is called the iSCSI target.

iSCSI using IPoIB A boot process that enables a server or virtual machine to boot an iSCSI target accessible using IP over an InfiniBand network. See also IPoIB.

M

MAC address Media access control address. A unique address associated with a network interface.

MMU Memory management unit.

MSB Most significant bit. The oem-logo variable is a 512-byte array, containing a total of 4096 bits arranged in a 64 x 64 array. Each bit controls one pixel. The MSB of the first byte controls the upper-left corner pixel. The next bit controls the pixel to the right of it, and so on.

N

NVMe Non-Volatile memory express. A host controller specification.

NVRAM Non-volatile random access memory.

NVRAMRC script You can use the `nvramrc` configuration variable to store user-defined FORTH commands that are executed during start-up. The contents of the variable are called the NVRAMRC script.

O

OpenBoot Oracle firmware that enables a PDomain to boot the Oracle Solaris OS. Provides an interface for testing hardware and software interactively.

Oracle ILOM Oracle Integrated Lights Out Manager. The system management firmware that is preinstalled on the SPs.

P

pool A logical group of devices describing the layout and physical characteristics of the available storage. Storage space for datasets is allocated from a pool. ZFS uses a model where storage devices are aggregated into a storage pool. See also boot pool and root pool.

POST Power-on self-test. Diagnostic software that runs when the system is initialized from a reset or power-on.

PROM Programmable read-only memory.

R

RARP Reverse Address Resolution Protocol.

root pool A dataset containing a complete Oracle Solaris image or a BE. See also pool.

root port In a PCIe device path, the root port is always the second element (for example, `/pci@300/pci@0`).

RPC Remote Procedure Call.

S

SCSI Small computer system interface. A standard for attaching peripherals to computers.

T

TFTP Trivial File Transfer Protocol.

V

virtual machine (VM) The terms domain, guest, and virtual machine are often used interchangeably. A domain is a configurable set of resources, including memory, virtual CPUs, network devices, and disk devices, in which virtual machines run. A domain is granted virtual resources and can be started, stopped and restarted independently of other domains and of the host server itself. A guest is a virtualized operating system running within a domain. In this document, domain, guest and virtual machine are all referred to as virtual machines in which OpenBoot Firmware runs.

Z

ZFS Zettabyte file system. A file system that uses storage pools to manage physical storage. See also BE, pool, boot pool, and root pool.

Index

D

E

F

G

P

R

S

T

U

V

W