Device Configuration

This chapter discusses how IRIX establishes the inventory of available hardware, and how devices are represented to software.

This information is essential when your work involves attaching a new device or a new class of devices to IRIX. The information is helpful background material when you intend to control a device from a user-level process.

The following primary topics are covered in this chapter.

- "Hardware Inventory" on page 29 describes the hardware inventory table displayed by the *hinv* command and how the inventory is initialized.
- "Device Special Files" on page 32 describes the system of filenames in */dev* and how they are created.
- "Configuration Files" on page 38 summarizes the files used for system generation and kernel configuration.

Hardware Inventory

During IRIX bootstrap, each device driver probes the hardware attachments for which it is responsible, and adds information to a hardware inventory table. The hardware inventory table is kept in kernel virtual memory. It is available to users and to programs.

Using the Hardware Inventory

The hardware inventory is used by users, administrators, and programmers.

Contents of the Inventory

Using database terminology, the hardware inventory consists of a single table with the following columns:

Class	A code for the class of device; for example, audio, disk, processor, or network.
Type	A code for the type of device within its class; for example, FPU and CPU types within the processor class.
Controller	When applicable, the number of the controller, board, or attachment.
Unit	When applicable, the logical unit or device within a Controller number.
State	A descriptive number, such as the CPU model number.

Displaying the Inventory with hinv

The *hinv* command formats all or selected rows of the inventory table for display (see the hinv(1) reference page), translating the numbers to readable form. The user or system administrator can use command options to select a class of entries or certain specific device types by name. The class or type can be qualified with a unit number and a controller number. For example,

```
hinv -c disk -b 1 -u 4
```

displays information about disk 4 on controller 1.

You can use *hinv* to check the result of installing new hardware. The new hardware should show up in the report after the system is booted following installation, provided that the associated device driver was called and was written correctly.

A full inventory report (*hinv -v*) is almost mandatory documentation for a software problem report, either submitted by your user to you, or by you to Silicon Graphics.

Testing the Inventory In Software

Within a shell script, you can test the output of *hinv* most conveniently in the command exit status. The command sets exit status of 0 when it finds or reports any items. It sets status of 1 when it finds no items. The code in Example 2-1 could be used in a shell script to test the existence of a disk controller.

Example 2-1 Testing the Hardware Inventory in a Shell Script

```
if hinv -s -c disk -b 1;
   then ;
   else echo No second disk controller;
fi ;
```

You can access the inventory table in a C program using the functions documented in the getinvent(3) reference page. The only access method supported is a sequential scan over the table, viewing all entries. Three functions permit access:

- **setinvent()** initializes or reinitializes the scan to the first row.
- **getinvent()** returns the next table row in sequence.
- **endinvent()** releases storage allocated by **setinvent()**.

These functions use static variables and should only be used by a single process within an address space. Reentrant forms of the same functions, which can safely be used in a multithreaded process, are also available (see getinvent(3)). Example 2-2 demonstrates the use of these functions.

The format of one inventory table row is declared as type *inventory_t* in the *sys/invent.h* header file. This header file also supplies symbolic names for all the class and type numbers that can appear in the table, as well as containing commentary explaining the meanings of some of the numbers.

Example 2-2 Function Returning Type Code for CPU Module

```
#include <stddef.h> /* for NULL */
#include <invent.h> /* includes sys/invent.h */
int getIPtypeCode()
{
   inv_state_t * pstate = NULL;
   inventory_t * work;
   int ret = 0;
   setinvent_r(&pstate);
   do {
      work = getinvent_r(pstate);
      if ( (INV_PROCESSOR == work->inv_class)
      &&   (INV_CPUBOARD == work->inv_type) )
         ret = work->inv_state;
   } while (!ret)
   endinvent_r(pstate); /* releases pstate-> */
   return ret;
}
```

Creating an Inventory Entry

Device drivers supplied by Silicon Graphics add information to the hardware inventory table when they are called at their *pfx***init()** or *pfx***edtinit()** entry points. One of these entry points is called by the IRIX kernel during bootstrap. (The small distinction between the two entry points is discussed in “Initialization Entry Points” on page 143.)

The function that adds a row to the inventory table is **add_to_inventory()**. Its prototype is declared in the include file *sys/invent.h*. The function takes arguments that are scalar values corresponding to the fields of the *inventory_t* structure.

Note: In IRIX 6.2, the only valid inventory types and classes are those declared in *sys/invent.h*. Only those numbers can be decoded and displayed by the *hinv* command, which prints an error message if it finds an unknown device class, and which prints nothing at all for an unknown device type within a known class. There is no provision for adding new device-class or device-type values for third-party devices.

Warning: The driver interface to the hardware inventory will change in a release following IRIX 6.2. The add_to_inventory() function of IRIX 6.2 is not formally part of the device driver API, and a different function or functions will be used in a future release.

Device Special Files

Devices are represented within IRIX as objects in the file system, specifically device special file nodes in the */dev* directory. These special file nodes are, in some cases, created automatically during the bootstrap process, and in some cases created manually by the system administrator.

Device Representation

The IRIX record of a file's existence is sometimes called an *inode*. The device special files consist of inodes only, with no associated data. The fields of the inode are used to encode the following critical information about a device:

Filename	Programs use the name of a device file to open the device using **open()**.
Permissions, Owner ID, Group ID	The file access permissions, owner ID, and group ID of a device file establish which users can read and which can write to that device.
Block or Character	A device file belongs to one of two classes, block or character, visible as the first letter of an *ls -l* display.
Major device number	A code for the device driver that controls this device.
Minor device number	A code specifying the unit or position of this device under its controller.

All this information is visible in a display produced by *ls -l*. The major and minor numbers are shown in the column used for file size for regular files. Examine the output of a command such as

```
ls -l /dev/* | more
```

A device special file can be used the same as a regular file in most IRIX commands; for example, a device file can be the target of a symbolic link, the destination of redirected input or output, and so on.

Block Versus Character

IRIX supports two classes of device. A *block device* such as a disk drive transfers data in fixed size blocks between the device and memory, and usually has some ability to reposition the medium so as to read or write the same data again. The driver for a block device typically has to manage buffering, and may schedule I/O operations in a different sequence than they are requested.

A *character device* such as a printer accepts or returns data as a stream of bytes, and usually acts as a sink or source of data—the medium cannot be repositioned and read again. The driver for a character device typically transfers data as soon as it is requested and completes one operation before accepting another request. Character devices are also called *raw* devices, because their input is not buffered.

Major Device Number

The *major device number* recorded in the device special inode selects the device driver to service this device. When a device is opened, IRIX selects the driver to handle the device based on the major device number. Each device driver supports one or more specific major numbers. There are two unrelated ranges of major numbers, one for character device drivers and one for block device drivers.

The possible major numbers are declared and given names in the file *sys/major.h*. When you create a new kernel-level device driver you must choose a major number for it—a number not used by any other driver. Numbers 60-79 are not used by Silicon Graphics. (See “Selecting a Major Number” on page 226.)

In IRIX releases through 5.2 (and 6.0.x, which is based on 5.2), major numbers were limited to the range 0 through 254. Beginning with releases 5.3 and 6.1, the IRIX inode structure permits major numbers to have up to 14 bits of precision. However, major numbers are currently restricted to at most 9 bits to conserve kernel data space.

In order to use this limit symbolically, use the name L_MAXMAJ defined in *sys/sysmacros.h*. When you declare a variable for a major device number in a program, use type *major_t* declared in *sys/types.h*.

Normally a device driver services only one major number. However, it is possible to designate the same device driver to service more than one major number. In this case, the driver may need to discover the major number at execution time. The **getemajor()** function returns the number in use for a given request (see the getemajor(D3) reference page).

Minor Device Number

The *minor device number* is passed to the device driver as an argument when the driver is called. (The major and minor numbers are passed together in a long integer called a *dev_t*.) The minor device number is interpreted only by the device driver, so it can be a simple logical unit number, or it can contain multiple, encoded bit fields. For example:

- The IRIX tape device driver uses the minor device number to encode the options for rewind or no-rewind, byte-swap or nonswap, and fixed or variable blocking, along with the logical unit number.
- The IRIX disk device drivers encode the disk partion number into the minor device number along with a disk unit number. Both disk and tape devices encode the SCSI adapter number in the minor number.
- The IRIX generic SCSI driver encodes the adapter (bus) number, target (control unit) number, and logical unit number into the minor number (see "Generic SCSI Device Special Files" on page 78).

The IRIX inode structure permits minor numbers to have up to 18 bits of precision. In order to use this limit symbolically, use the name L_MAXMIN defined in *sys/sysmacros.h*. When you declare a variable for a minor device number in a program, use type *minor_t* declared in *sys/types.h*.

With STREAMS drivers, the minor device number can be chosen arbitrarily during a CLONE open—see "Support for CLONE Drivers" on page 552.

Defining Device Names

The device special files related to Silicon Graphics device drivers are created by execution of the script */dev/MAKEDEV*. Additional device special files can be created with administrator commands.

IRIX Conventional Device Names

The device drivers distributed with IRIX depend on certain conventions for device names. These conventions are spelled out in the following reference pages: intro(7), dks(7), dsreq(7), and tps(7). For example, the components of a disk device name in */dev/dsk* include

dks*c*	Constant prefix "dks" followed by bus adapter number *c*.
d*u*	Constant letter "d" followed by disk SCSI ID number *u*.
l*n*	Optionally, letter "l" (ell) and logical unit number *n* (used only when disk u controls multiple drives).
s*p* or **vh** or **vol**	Constant letter "s" and partition number *p*, or else "vh" for volume header, or "vol" for (entire) volume.

Programs throughout the system rely on the conventions for these device names. In addition, by convention the associated major and minor numbers agree with the names. For example, the logical unit and partition numbers that appear in a disk name are also encoded into the minor number.

The Script MAKEDEV

The conventions for all the IRIX device special names are written into the script */dev/MAKEDEV*. This is a make file, but unlike most make files, it is not used to compile executable programs. It contains the logic to prepare device special names and their associated major and minor numbers and file permissions.

The MAKEDEV script is executed during IRIX startup from a script in */etc/rc2.d*. It is executed after all device drivers have been initialized, so it can use the output of the *hinv* command to construct device names to suit the actual configuration.

The system administrator can invoke MAKEDEV to construct device special files. Administrator use of MAKEDEV is described in *IRIX Administration: System Configuration and Operation*.

Making Device Files

You or a system administrator can create device special files explicitly using the commands *mknod* or *install*. Either command can be used in a make file such as you might create as part of the installation script for a product.

For details of these commands, see the install(1) and mknod(1M) reference pages, and *IRIX Administration: System Configuration and Operation*. The following is a hypothetical example of *install*:

```
# install -m 644 -u root -g sys -root /dev -chr 62,0
```

The *-chr* option specifies a character device, and *62,0* are the major and minor device numbers, respectively.

Tip: The *mknod* command is portable, being used in most UNIX systems. The *install* command is unique to IRIX, and has a number of features and uses beyond those of *mknod*. Examples of both can be found by reading */dev/MAKEDEV*.

Multiple Names for One Device

It is possible to point to the same device with more than one device special filename. This is done in the distributed IRIX system for several reasons:

- To supply default names for devices with specific names. For example, the default device */dev/tapens* is a link to the first device file in */dev/rmt/**.
- To pass different parameters to the device driver. For example, the same tape device appears multiple times in */dev/rmt/tps**, with different combinations of *nr* (norewind), *ns* (nonswapped), and *v* (variable block) suffixes. The minor number for each name encodes these options for the same unit number.
- To supply both block and character drivers for the same device. For example, each disk device appears in */dev/dsk/** as a block device, and again in */dev/rdsk/** as a character device.

Configuration Files

IRIX uses a number of configuration files to supplement its knowledge of devices and device drivers. This is a summary of the files. The use of each file for device driver purposes is described in more detail in other chapters. (The uses of these files for other system administration tasks is covered in *IRIX Administration: System Configuration and Operation*.)

Most configuration files used by the IRIX kernel are located in the directory */var/sysgen*. Files used by the X11 display system are generally in */usr/lib/X11*. With regard to device drivers, the important files are:

*/var/sysgen/master.d.**	Descriptions of the attributes of kernel modules
*/var/sysgen/boot/**	Kernel object modules
/var/sysgen/system/.sm*	Device configuration information
*/var/sysgen/mtune/**	Values and limits of tunable parameters
/var/sysgen/stune	New values for tunable parameters
*/usr/lib/X11/input/config/**	Initialization commands for Xdm input modules

Master Configuration Database

Every configurable module of the kernel (this includes kernel-level device drivers and some other service modules) is represented by a single file in the directory */var/sysgen/master.d*.

A file in *master.d* describes the attributes of a module of the kernel which is to be loaded at boot time. The general syntax of the file is documented in detail in the master(4) reference page. Only a subset of the syntax is used to describe a device driver module. In general, the *master.d* file specifies device driver attributes such as:

- the driver's *prefix*, a name that qualifies all its entry points
- whether it is a block, character, or STREAMS driver
- the major number serviced by the driver
- whether the driver can be loaded dynamically as needed
- whether the driver is multiprocessor-aware
- which of the possible driver entry points the driver supplies

For each module described in a *master.d* file there should be a corresponding object module in */var/sysgen/boot*. The creation of device driver modules and the syntax of *master.d* files is covered in detail in Chapter 10, “Building and Installing a Driver.”

System Configuration Files

The files */var/sysgen/system/*.sm* direct the *lboot* command in loading the modules of the kernel at boot time. Although there are normally several files with the names of subsystems, all the files are treated as one single file. The contents of the files direct *lboot* in loading components that are described by files in */var/sysgen/master.d*, and in probing for devices to see if they exist.

The exact syntax of these files is documented in the system(4) reference page. The use of the VECTOR lines to probe for hardware is covered in this book in the context of each type of attachment. For example, probing for VME devices is covered under “Configuring the System Files” on page 311.

System Tuning Parameters

The IRIX kernel supports a variety of tunable parameters, some of which can be interrogated by device drivers. The current values of the parameters are recorded in files in */var/sysgen/mtune/** (one file per major subsystem).

You or the system administrator can view the current settings using the *systune* command (see the systune(1M) reference page). The system administrator can use *systune* to request changes in parameters. Some changes take effect at once; others are recorded in a modified kernel that is loaded the next time the system boots.

To retrieve certain tuning parameters from within a kernel-level device driver, include the header file *sys/var.h*.

The use of *systune* and its related files is covered in *IRIX Administration: System Configuration and Operation*.

X Display Manager Configuration

Most files related to the configuration of the X Display Manager *Xdm* are held in */var/X11*. These files are documented in reference pages such as xdm(1) and in the programming manuals related to the X Windows System™.

One set of files, in */usr/lib/X11/input/config*, controls the initialization of nonstandard input devices. These devices use STREAMS modules, and their configuration is covered in Chapter 19, “STREAMS Drivers.”