Managing kernel configurations in HP-UX 11i version 2

Introduction

With each successive release of HP-UX, system administrators have increased ability to make changes to the configuration of the HP-UX kernel without experiencing costly and inconvenient downtime. Innovations such as Dynamic Kernel Tunables and Dynamically Loadable Kernel Modules allow critical maintenance tasks to be performed without sacrificing application availability.

With these innovations comes the need for a simpler and more comprehensive mechanism to manage kernel configurations. HP-UX 11i version 2.0 introduces a new suite of kernel configuration management commands and a new Web-based graphical interface that provide unified kernel configuration management. This paper describes the use of these new tools. It is intended for use by HP-UX system administrators.

Kernel configuration features

The new suite of kernel configuration tools provides the following key features for system administrators:

- All kernel configuration tasks can be performed in a single graphical interface.
- All kernel configuration tasks can also be performed using a cohesive set of commands with the same user interface and same behavior.
- Kernel configurations can be saved and restored and moved between systems.
- Administrators can save any number of kernel configurations, and can switch between them at will—often without a reboot.
- The running kernel configuration is automatically backed up before each configuration change (if desired).
- The system automatically maintains a detailed log file of all kernel configuration changes.
- Kernel modules and kernel tunable parameters now have descriptions associated with them. Kernel tunable parameters have online documentation and descriptions of the relationships between them.
- All kernel configuration commands can produce output in both user-friendly and script-friendly formats. HP supports release-to-release compatibility for the script-friendly formats.

What is a kernel configuration?

Logically, a kernel configuration is a collection of all of the administrator choices and settings needed to determine the behavior and capabilities of the HP-UX kernel. In this implementation, the collection includes:

- A set of kernel modules, each with a desired state
- A set of kernel tunable parameter value assignments
- A primary swap device specification
- A set of dump device specifications
- A set of bindings of devices to device drivers
- A name and optional description of the kernel configuration

Physically, a kernel configuration is a directory under `/stand` that contains the files needed to realize the specified behavior. The directory includes:

- An HP-UX kernel executable
- A set of HP-UX kernel module files
- A kernel registry database, containing all of the above settings
- A system file, describing the above settings in human-readable form
- Various other implementation-specific files

In addition to the configuration of the running kernel, HP-UX systems can have any number of saved kernel configurations, limited only by the disk space available in `/stand`.

Overview of kernel configuration commands

There are three primary commands used to manage kernel configurations: *kconfig, kcmodule*, and *kctune*.

kconfig is used to manage whole kernel configurations. It allows configurations to be saved, loaded, copied, renamed, deleted, exported, imported, etc. It can also list existing saved configurations and give details about them. For more information, see the section “Managing saved configurations using *kconfig*” on page 23, or the *kconfig*(1M) man page.

kcmodule is used to manage kernel modules. Kernel modules can be device drivers, kernel subsystems, or other bodies of kernel code. Each module can be unused, statically bound into the main kernel executable, or dynamically loaded. *kcmodule* will display or change the state of any module in the currently running configuration or any saved configuration. For more information, see the section "Managing kernel modules with *kcmodule*" on page 8, or the *kcmodule*(1M) man page.

kctune is used to manage kernel tunable parameters. These are variables that control the behavior of the kernel. They have many uses—common ones include controlling the allocation of system resources and tuning aspects of kernel performance. *kctune* will display or change the value of any tunable parameter in the currently running configuration or any saved configuration. For more information, see the section "Managing kernel tunable parameters with *kctune*" on page 14, or the *kctune*(1M) man page.

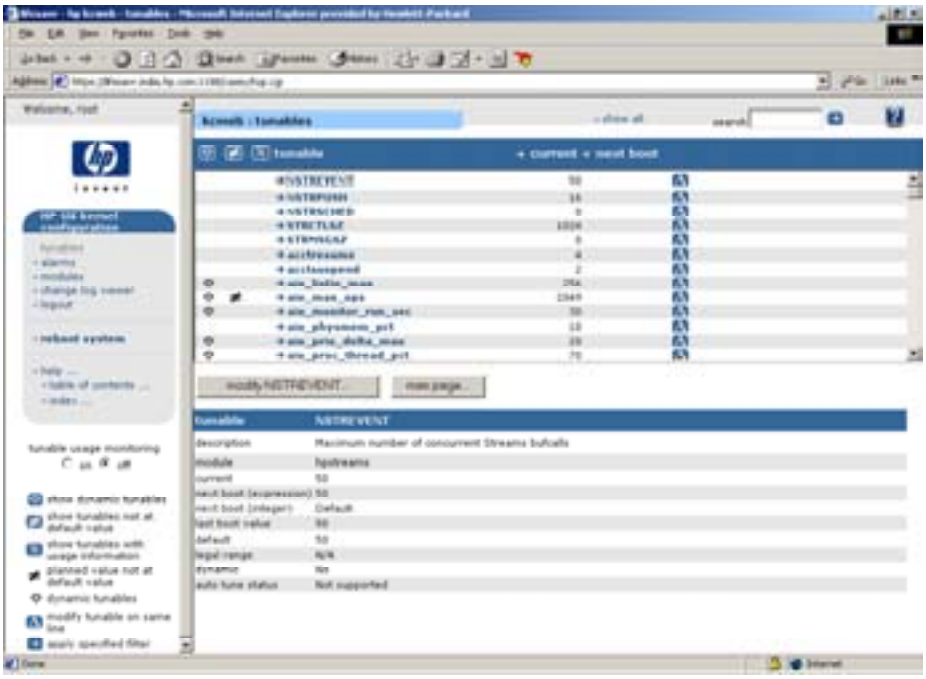
In addition to these three primary commands, there are two other kernel configuration commands. The *kcpath* command prints information about the location of the currently running kernel—it is intended for use by scripts and applications that need this information. (See the *kcpath*(1M) man page for details.) The *kclog* command searches the kernel configuration log file; for details see the section "The kernel configuration log file"on page 28, or the *kclog*(1M) man page.

Finally, users of the *mk_kernel*(1M), *kmpath*(1M), and *kmtune*(1M) commands present in previous HP-UX releases should be aware that these commands can still be used. They have been re-implemented as small shell scripts that invoke the commands listed above. These commands will be removed in a future release.

Overview of the *kcweb* tool

Administrators can configure and manage the kernel without remembering the syntax of the kernel configuration commands and the exact names of modules and tunables by using the Web-based user-friendly HP-UX kernel configuration tool. *kcweb* is a Web-based HP-UX kernel configuration tool used to configure and manage the kernel of your system. The various features of *kcweb* are as follows:

- Web-based and user-friendly graphical user interface (GUI)
- Monitor and modify kernel tunables
- Add, modify, and remove alarms
- Modify kernel module state
- Access the manpage for tunable
- Command preview—when a tunable, module, or alarm is modified, you can use the command preview feature by choosing the **?** button; this will show the kernel configuration command invocation that will perform the requested task



kcweb can be accessed in any of the following ways:

- From the command-line using the *kcweb* command
- From the HP Service Control Manager (SCM)
- From the Kernel Config (*kcweb*) area of SAM
- From a Web browser, using the URL of a *kcweb* server that has already been started

By default, *kcweb* invokes the Mozilla Web browser. If you want to invoke *kcweb* on any other browser, set the **BROWSER** environment variable to the pathname of the browser you wish to use. For more details, see the *kcweb*(1M) man page.

Other kernel configuration operations

This paper contains sections describing some special kernel configuration operations and special uses of the kernel configuration commands.

The usage of some kernel resources can be monitored, and alarms can be delivered when usage rises above a set threshold. These alarms can be configured and reviewed using the *kcalarm* command or the *kcweb* tool. The resource usages can be reviewed using the *kcusage* command or the *kcweb* tool. For more information, see the section "Monitoring kernel resource usage" on page 20.

Administrators of older versions of HP-UX may be accustomed to using text files ("system files" or "dfiles") to specify kernel configurations and make changes to them. The format of these files has been enhanced[1] to accommodate new kernel configuration innovations, while retaining the usefulness of a text file for configuration operations. (They are particularly useful when using the same configuration on multiple systems, since they can be easily moved between systems.) The use of system files is described in the section "Managing configurations with system files" on page 25.

Some uncommon configuration settings can be controlled only through the use of system files. These include the setting of the primary swap device, the setting of the initial dump devices, and the explicit binding of specific devices to specific device driver modules. For more information, see the section "Managing device bindings" on page 27.

All kernel configuration changes made using the kernel configuration commands are logged to the file `/var/adm/kc.log`. Details about this log file can be found in the section "The kernel configuration log file" on page 28, or the *kconfig*(5) and *kclog*(1M) man pages.

The primary kernel configuration commands support a specialized output format designed for use by scripts and applications that need to parse the output of the commands. Such scripts and applications must use this specialized output format since HP does not guarantee release-to-release compatibility for any other output format of these commands. More detail is available in the section "Parsing command output" on page 29, or the *kconfig*(5) man page.

It is possible to have an undesirable, or even unbootable, kernel configuration because of mistaken configuration changes, hardware failures, or software defects. Mechanisms exist both to prevent such problems and to help recover from them. For more details see the section "Recovering from errors" on page 30.

[1] The system file formats from previous releases of HP-UX are still accepted.

Common behavior for kernel configuration commands

Because the kernel configuration commands are part of a unified suite, they share behavior whenever possible. Shared behaviors include command line flags, output formats, exit status codes, security constraints, and persistence of changes.

Common command line flags

Command line flags are shared by the kernel configuration commands as shown in the table below.

Flag	Description	kconfig	kcmodule	kctune	kclog
-a	(all) Include information in the output that is normally omitted for brevity.	✓	✓	✓	
-B	(Backup) Back up the currently running configuration before changing it.	✓	✓	✓	
-c	(configuration) Specify the saved configuration to manage. If omitted, manage the currently running configuration.		✓	✓	✓
-C	(Comment) Include a comment in the kernel configuration log file entry associated with this command invocation.	✓	✓	✓	✓
-d	(description) Display descriptions of each item.		✓	✓	
-D	(Difference) Display only elements for which there is a change being held for next boot.	✓	✓	✓	
-h	(hold) Hold the requested changes for next boot.	✓	✓	✓	
-K	(Keep) Do not back up the currently running configuration. Keep the existing backup unmodified.	✓	✓	✓	
-P	(Parse) Use the special "parsable" output format.	✓	✓	✓	
-S	(Set) Display only elements that have been set to something other than default.	✓	✓	✓	
-v	(verbose) Display items using verbose output format.	✓	✓	✓	

Common output formats

When retrieving information, the primary kernel configuration commands produce output in three basic output formats: table, verbose, and parsable.

By default, the commands produce a short table format. This is a format that gives one line for each item being described. Only the most commonly used information is included in order to allow the output to fit on one line on most terminals.

With a -v (verbose) flag, the commands produce a verbose output format. This format gives all available information for each item being described, taking multiple lines to do so. A blank line separates the items in the output.

With a -P (Parse) flag, the commands produce an output format designed to be parsed by scripts or applications. This format is described in the section "Parsing command output" on page 29. Scripts and applications must parse this output format because HP supports release-to-release compatibility of output format only when the -P flag is used.

The kernel configuration commands all use a common format for error, warning, note, and progress messages. It is the same format used by the Software Distributor package and is, therefore, already familiar to most administrators.

```
ERROR:     This is an error message. It explains why the
           requested operation cannot complete.
WARNING:   This is a warning message. The requested
           operation completed, but not smoothly.
           A situation may exist that needs correction.
NOTE:      This is a note. It provides information about
           how the operation completed or other
           information of potential interest to the
           administrator.
           * This is a progress message. It displays the
             steps completed during the operation.
```

Common exit status codes

All of the kernel configuration commands exit with one of these status codes:

0 Operation was successful.

1 The requested changes could not be applied to the currently running system. They are being held and will be applied at next boot.

2 The operation could not complete successfully.

Common security constraints

Any user can run the kernel configuration commands to query configuration information. However, access to configuration information is subject to standard UNIX® file system permissions on the relevant files.

Superuser privileges are required to make any configuration changes.

Persistence of changes

By default, the kernel configuration tools will apply configuration changes to the currently running system, causing an immediate change in behavior. System administrators can override this default by specifying the `-h` (hold) option to any of the commands. This option causes the change(s) to be held until the system is rebooted. HP recommends that this option be used only when the next reboot is expected to happen soon. If the reboot doesn't happen for months after the change, the change could come as an unwelcome surprise to an administrator who had forgotten the request.

Some configuration changes cannot be applied without a reboot. These changes will be held until the system is rebooted even if the `-h` option is not specified. In these cases, a warning message will be printed.

If multiple configuration changes are requested in a single invocation of one of the kernel configuration commands and any one of those changes requires a reboot, all of the requested changes will be held until the system is rebooted. In particular, if a saved kernel configuration is loaded using `kconfig -l` (load) and that configuration cannot be used without a reboot, the state of the running system is not changed and the specified kernel configuration will be used at next boot instead.

Changes being held for next boot can be listed using the `-D` (Differences) option to the *kcmodule*, *kctune*, or *kconfig* commands. See below for more details on each of these commands.

Changes being held for next boot are discarded when the currently running configuration is replaced using `kconfig -i` (import), `kconfig -l` (load), or `kconfig -n` (next boot); when explicitly discarded using `kconfig -H` (unHold); or when subsequent changes are made that override them. For example, let's say you run the following commands:

```
kctune -h nproc=5000     (set to 5000, hold for next boot)
kctune nproc=6000        (set to 6000, now)
```

The value of `nproc` at next boot will be 6000; the change to 5000 is discarded. A warning will be printed in these situations.

Changes that are made to the currently running system are retained when the system is rebooted. They remain in effect until changed.

Managing kernel modules with *kcmodule*

The *kcmodule*(1M) command is used to query and change the states of kernel modules in the currently running configuration or in a saved configuration. The HP-UX kernel is built from a number of modules, each of which is a device driver, kernel subsystem, or some other body of kernel code. A typical kernel has 200–300 modules in it.

Getting information about modules

When you run *kcmodule* with no options, it shows you the modules on your system, their current state, and the state they will have at next boot. On a typical system, you will see many modules in `static` state; some modules that are `unused`, which are often device drivers for hardware your system doesn't have; and a handful of modules in `loaded` state. (The states are described below.)

When using the `-c` (configuration) option, *kcmodule* displays the module information from a saved configuration instead of from the currently running system.

The output of *kcmodule* can be varied with several options. To control which modules are listed, use the `-a` (all), `-D` (Differences), and/or `-S` (Set) flags. The `-a` option adds required modules to the output (normally they are omitted). The `-D` option restricts the output to only those modules whose state at next boot is different from their current state. The `-S` option restricts the output to modules whose state has been explicitly set (i.e., it omits required modules, unused modules, and modules added to satisfy a dependency). The output can also be restricted by listing module names on the command line.

To control the output format, use the `-d` (description), `-v` (verbose), or `-P` (Parse) flags. Without these flags, the output looks like this:

```
Module                  State       Cause
fcms                    static      depend
krs                     static      required
```

Using the `-d` flag adds the description of each module.

```
Module                  State       Cause
        Description
fcms                    static      depend
        Fibre Channel Mass Storage Driver
krs                     static      required
        Kernel Registry Service
```

Using the `-v` flag gives verbose, multi-line information about each module:

```
Name                    fcms [3E4741A9]
Description             Fibre Channel Mass Storage Driver
State                   static (to resolve dependencies)
Capable                 unused static
Depends On              module libfcms
                        interface HPUX_11_23 1.0.0

Name                    krs [3E47419F]
Description             Kernel Registry Service
State                   static (required)
Capable                 static
Depends On              module libkrs
                        module libkrs_pdk
                        interface HPUX_11_23 1.0.0
```

The `-P` flag, which is designed for use by scripts or programs, gives complete control over what information is printed:

```
# kcmodule -P name,desc fcms krs
name                fcms
desc                Fibre Channel Mass Storage Driver

name                krs
desc                Kernel Registry Service
```

For more information on the `-P` (Parse) flag and its use by scripts or programs, see the section "Parsing command output" on page 29, or the *kconfig*(5) man page.

Interpreting module information

Looking at the sample output above, you can see that each module has a name and a text description. Each module also has a version, which typically looks like either `[3E36E5FA]` or `0.1.0`, depending on the age of the module. Older modules use the first form and newer modules use the second form.

A kernel configuration can only use one version of any given module. However, multiple versions may be listed if, for example, your currently running system is using a different version of a module from the one that will be used at next boot. Version numbers are normally omitted from the short listing, but will be included if there's more than one version of a module.

Each kernel module in the currently running configuration has a state, which describes how the module is being used. The possible states are:

`unused`	The module is installed on the system but not in use.
`static`	The module is statically bound into the kernel executable. This is the most common state. Moving a module into or out of this state requires relinking the kernel executable and rebooting.
`loaded`	The module is dynamically loaded into the kernel. Newer modules support this state. Such modules may be added to the kernel configuration or removed from it without rebooting.
`auto`	The module will be dynamically loaded into the kernel when it is first needed, but it hasn't been needed yet.

When *kcmodule* is giving information about the currently running system, and there are configuration changes being held for next boot, *kcmodule* will list both the current state and the state at next boot. For next boot, the same states are used, with complementary meanings:

`unused`	The module will not be used.
`static`	The module will be statically bound into the kernel executable.
`loaded`	The module will be dynamically loaded into the kernel during the boot process.
`auto`	The module will be dynamically loaded into the kernel when it is first needed after each boot.

When *kcmodule* is giving information about a saved configuration, the same states are used.

Next to each module state is a "Cause", which tells why the module is (or will be) in that state. The causes are:

`explicit`	The system administrator explicitly chose the state.
`best`	The system administrator chose to use the module, but didn't choose a specific state, so the module is in its "best" state as determined by the module developer.
`auto`	The module was in `auto` state, and was automatically loaded when something tried to use it.
`required`	The module was marked required by its developer.
`depend`	The module is in use because some other module in the configuration depends on it.

Different modules can support different states. Nearly all modules can be in `static` state, but only a few support `loaded` or `auto` states. Many modules can be in `unused` state, but required modules cannot. The "Capable" line in the output shows which states a module supports. (Hint: to see if a module is required, look to see whether `unused` appears on the "Capable" line. If it does, the module is not required.)

Modules often have dependencies between them. For example, device drivers typically cannot be configured into the kernel unless the driver support modules are also configured. Dependencies like this are shown on the "Depends On" lines in the output. A module can be dependent on a particular other module, specified by name and version. A module can also be dependent on an interface that must be supplied by some other module, without specifically saying which module(s) supply that interface. Modules that supply such interfaces have an "Exports" line in the output listing the interfaces they export.

Changing module states

To change the state of a module, put module state assignments on the *kcmodule* command line. (Or see the section, "Managing configurations with system files" on page 25.) For example, to load the CD File System module, named `cdfs`:

```
# kcmodule cdfs=loaded
```

In fact, `loaded` is the developer-chosen "best" state for `cdfs`, so this command is the same as:

```
# kcmodule cdfs=best
```

To `unload` it:

```
# kcmodule cdfs=unused
```

See the *kcmodule*(1M) man page for details.

When you change a module state using a command similar to the above examples, the change will be made immediately to the currently running system, if possible. Sometimes it's not possible to make the change immediately; for example, there might be a CD file system mounted, in which case `cdfs` can't be unloaded. In those cases, *kcmodule* will hold the change and apply it at next boot. A change that moves a module into or out of `static` state can never be applied immediately, and will always be held for next boot. If any change on the *kcmodule* command line has to be held for next boot, they all will be.

When modules are moved into or out of `static` state, the *kcmodule* command will run for quite a while. This is because such changes require that the kernel executable be relinked. If you have multiple changes to make, it is best that you list them all on the same *kcmodule* command line or make the changes in a system file and import it. (See the section "Managing configurations with system files" on page 25.) Either of these techniques will ensure that the kernel executable is only relinked once.

Sometimes you may want to force a change to be held for next boot, instead of applying it immediately. In these cases you can give the `-h` (hold) flag to *kcmodule* to force that behavior. HP recommends that this flag be used only when the next boot is expected to be soon. If the next boot doesn't happen for months after making such a change, the system administrator could be unpleasantly surprised at the effect of a pending change that had been forgotten.

Changes to saved kernel configurations can be made using the `-c` (configuration) flag. Such changes are made to the saved configuration immediately, but they won't affect the running system until that saved configuration is either loaded or booted. See the section "Managing saved configurations using *kconfig*" on page 23 for more information.

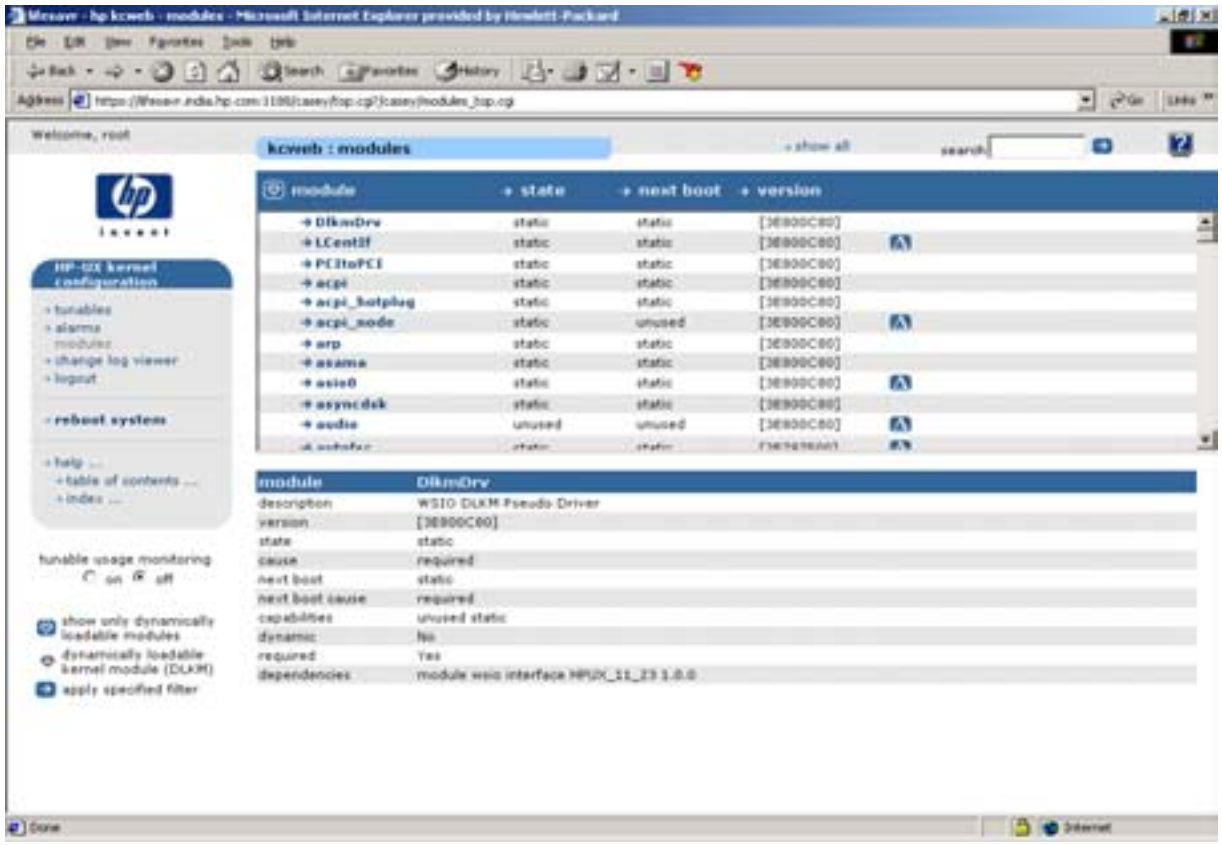
When changing module states, *kcmodule* supports the `-B` and `-K` flags to specify backup behavior, and the `-C` flag to specify a log file comment. See the sections "Recovering from errors" on page 30 and "The kernel configuration log file" on page 28 for details.

Managing kernel modules with *kcweb*

kcweb can be used to query and change the states of kernel modules in the currently running configuration. Using *kcweb*, you can:

- Determine which modules are currently running in the kernel
- View details about a module
- Modify the state of a module

You can view the modules pane by choosing the **modules** menu item from the navigation column in *kcweb*.



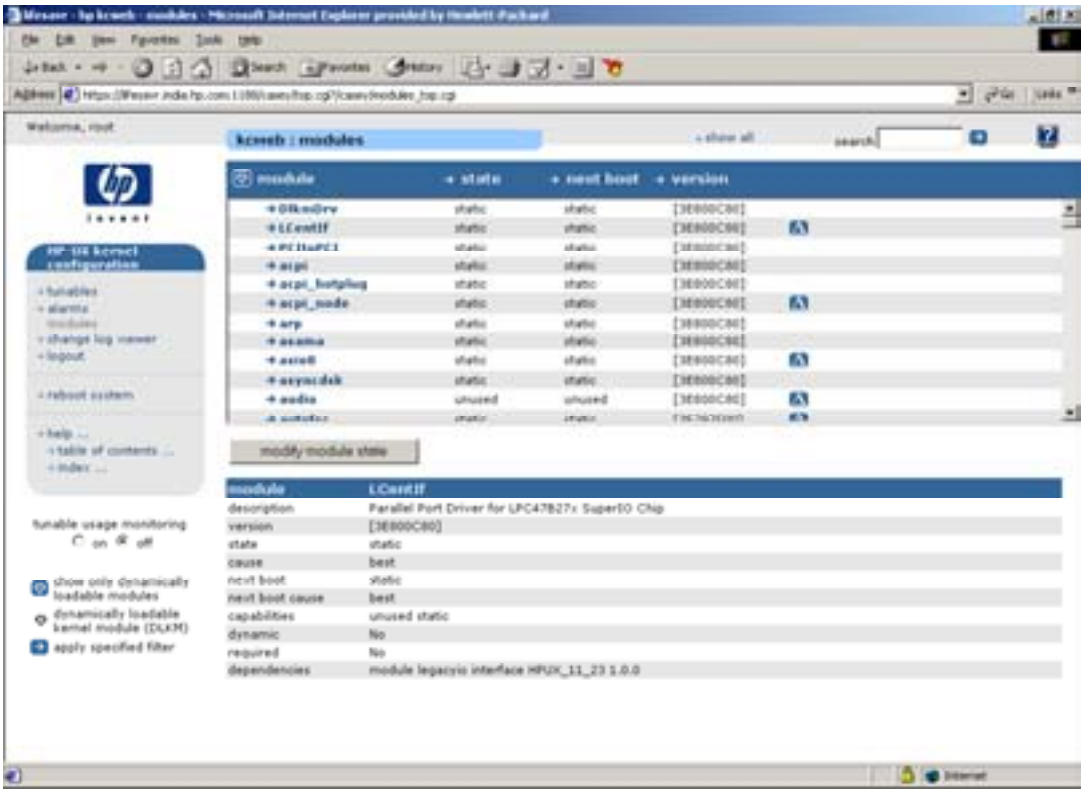
Getting information about modules

To get more detailed information about a particular module, execute the following two steps:

- Select the **modules** menu item in the navigation column. The modules pane is displayed, listing all the modules that are currently configured on your system.
- Select a module to view the details about a particular module in the details pane.

Interpreting module information

If you choose a module, the module details pane is displayed.



The module details pane contains the following information:

Field name	Description
module	Indicates the name of the module.
description	Indicates a brief description of the module.
version	Indicates the version of the module.
state	Indicates the state of the module in the kernel that is currently running (`unused`, `static`, `loaded`, `auto`).
cause	Indicates the reason why the module is in its current state (`explicit`, `auto`, `depend`, `required`, `default`).
next boot	Indicates the state of the module after the system is restarted.
next boot cause	Indicates the reason why the module is in its next boot state.
capabilities	Indicates all the states that the module is capable of supporting.
dynamic	Indicates that it is a dynamically loadable kernel module.
required	Indicates whether or not the kernel requires the module.
dependencies	Indicates the modules required by this module.
exports	Lists all the interfaces exported by this module.

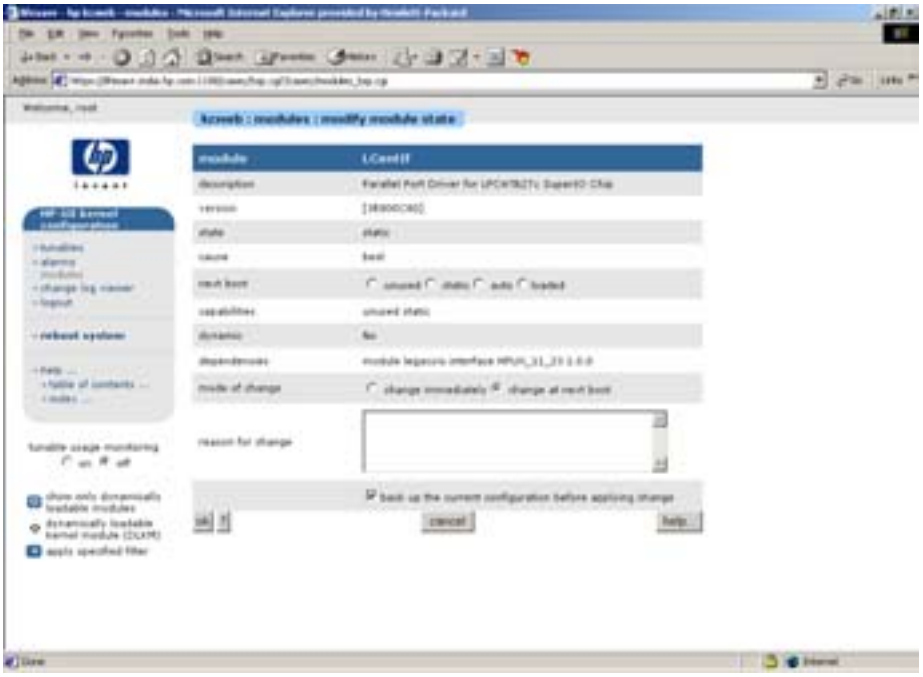
Changing module states

To change the state of a module, execute the following steps:

- Select the **modules** menu item in the navigation column. The modules pane is displayed, listing all the modules that are currently configured on your system.
- Select a module that you wish to modify by choosing the  icon or the **modify module state** button.

The modify module state page is displayed.

Note: If the cause is dependent/required, the **modify module state** button will not appear since *kcweb* does not allow modifications to the state of a required module or a module on which other modules are dependent.



The modify module state page contains the following fields:

Field name	Description
module	Name of the module that will be modified.
description	A description of the module.
version	Version number of the module
state	The current value of the module.
cause	How the module got into its current state.
next boot	The state that the module will be changed to if you click the **OK** button.
capabilities	All the states that the module can support.
dynamic	Indicates whether the module is a dynamically loadable kernel module.
dependencies	All the modules on which this module depends.
mode of change	Contains a set of radio buttons to apply kernel configuration changes immediately or to hold kernel configuration changes till next boot. This field is displayed only for dynamic modules. By default, the **change at next boot** radio button is selected. If you do not select any radio button, kernel configuration changes will be held till next boot.
reason for change	Editable text field to enter comments for change in module state.
back up the current configuration before applying change	Backup of current configuration before applying the change; by default, this checkbox is selected.

Managing kernel tunable parameters with *kctune*

The *kctune* command is used to query and change the values of kernel tunable parameters (tunables) in the currently running configuration or in a saved configuration. Tunables are variables that govern the behavior of the HP-UX kernel. Tunables are used for a variety of different tasks—some control resource allocations, others control security policies, others enable optional kernel behavior, etc. There are 150–200 tunables in a typical kernel.

System administrators can create their own user-defined tunables if they choose. These will not affect the operation of the system directly, but they can be used in computing the values of other tunables. For example, an administrator could choose to create a `num_databases` tunable and then set several kernel tunables based on its value. A subsequent change to the value of `num_databases` would cause all of the related kernel tunable values to be changed as well.

Getting information about tunables

When you run *kctune* with no options, it shows you the tunables associated with the kernel modules on your system (as well as any user-defined tunables), their current values, and the expressions used to compute those values. If there are changes to those values being held for next boot, those will be shown as well. On a typical system, the expression for most tunables is "Default," meaning that the administrator is allowing the system to choose the tunable value.

When using the `-c` (configuration) option, *kctune* displays the tunable information from a saved configuration instead of from the currently running system.

The output of *kctune* can be varied with several options. To control which tunables are listed, use the `-D` (Differences) or `-S` (Set) flags. The `-D` option restricts the output to only those tunables whose value at next boot is different from their current value. The `-S` option restricts the output to only those tunables that are set to a non-default value. The output can also be restricted by listing tunable names on the command line.

To control the output format, use the `-d` (description), `-g` (group), `-v` (verbose), or `-P` (Parse) flags. Without these flags, the output looks like this:

```
Tunable            Current    Expression    Changes
acctresume               4    Default
maxuprc                256    Default       Immed
nproc                 4200    Default       Immed
```

Using the `-d` flag adds the description of each tunable:

```
Tunable            Current    Expression    Changes
         Description
acctresume               4    Default
         Percentage of disk space that must be free to resume accounting
maxuprc                256    Default       Immed
         Maximum number of processes for each non-root user
nproc                 4200    Default       Immed
         Maximum number of processes on the system
```

Using the `-g` flag adds the name of the module defining the tunable and sorts the output by module name. This has the effect of grouping related tunables together in the output.

```
Module         Tunable       Value    Expression    Changes
acct           acctresume        4    Default
pm             maxuprc         256    Default       Immed
pm             nproc          4200    Default       Immed
```

Using the `-v` flag gives verbose, multi-line information about each tunable:

```
Tunable                 acctresume
Description             Percentage of disk space that must be free to resume
Module                  pm
Current Value           4 [Default]
Value at Next Boot      4 [Default]
Value at Last Boot      4
Default Value           4
Constraints             acctresume >= -100
                        acctresume <= 101
                        acctresume > acctsuspend
Can Change              At Next Boot Only

Tunable                 nproc
Description             Maximum number of processes on the system
Module                  pm
Current Value           4200 [Default]
Value at Next Boot      4200 [Default]
Value at Last Boot      4200
Default Value           4200
Constraints             nproc >= 100
                        nproc <= 30000
                        nproc >= maxuprc + 5
                        nproc <= nkthread - 100
                        nproc >= semmnu + 4
Can Change              Immediately or at Next Boot
```

The `-P` flag, which is designed for use by scripts or programs, gives complete control over what information is printed:

```
# kctune -P name,current acctresume nproc
name            acctresume
current         4

name            nproc
current         4200
```

For more information on the `-P` flag and its use by scripts or programs, see the section "Parsing command output" on page 29, or the *kconfig*(5) man page.

Interpreting tunable information

Looking at the sample output above, you can see that each tunable has a name and a text description. Each tunable is associated with a kernel module whose name is listed in the verbose output (or in the table output if `-g` is specified). Tunables can be seen and changed only if they are associated with a module that is installed on the system (or are user-defined). The module does not have to be in use.

When displaying tunable information for the currently running system, *kctune* includes the current tunable value and the expression used to compute it. If changes to the tunable's value are being held for next boot, the next boot value and expression are also shown. Verbose listings also show the value the tunable had when the system was last booted. When displaying tunable information for a saved configuration, *kctune* displays only a current value.

Tunable values are computed integer expressions, which can refer to other tunable values. (Circular references are not permitted.) The value of a tunable could be `4200`, or `0x400`, or `12*1024`, or `4*nproc+20`. Values and expressions use the syntax of the C programming language. Therefore, numbers can be written in decimal (256), octal (01000), or hexadecimal (0x100). Expressions can use the following operators and symbols:

```
( ) ~ ! - + * / % << >> < <= > >= & ^ | == != && || ?:
```

Whitespace is not permitted in any tunable expression. For backward compatibility, tunable names used in expressions can appear in all capitals, but this usage is discouraged and support for it will be removed in a future release.

All kernel tunables have a default value, which is chosen by the developer and is shown in the verbose output. For some tunables, the default value is fixed and never changes. For other tunables a new default value is chosen by the system at boot time. Still others can be automatically tuned, meaning that the default value can change periodically while the system is running, in response to changing system resources and needs. When a tunable is set to default, its expression is reported as "Default," as seen in the examples above. In these cases, the system is free to choose the value it thinks optimal, and to change it as needed. HP recommends that tunables be left set to default unless the default is known to be unsatisfactory.

Note: setting a tunable to "Default" is not the same thing as setting it explicitly to the default value reported by *kctune*. Using the example above, if you set `nproc` to 4200, its value will remain 4200 until you change it. However, if you set `nproc` to "Default," its value will be kept up to date with any improvements HP makes to the default value for `nproc`.

Some tunables have constraints on their values, which are shown in the verbose output. Sometimes these are minimum and/or maximum values, as shown for `nproc` above. Other times these are fixed relationships between tunables (for example, `acctresume` must be greater than `acctsuspend`) or restrictions on the allowed values (for example, `dnlc_hash_locks` must be a power of two). These constraints are enforced whenever changing tunable values. There are other constraints, not shown by *kctune*, that are based on the current state of the system and can change over time (for example, `nproc` cannot be set to less than the number of processes currently running). These constraints are enforced only when changing the currently running system, and not when making changes held for use at next boot or changes to a saved configuration.

Some tunables have restrictions on when their values can be changed. These restrictions are noted in the *kctune* output. Tunables whose values can be changed immediately are marked `Immed`. Tunables whose values can be automatically tuned by the system are marked `Auto`. Tunables without either marking can only be changed with a reboot.

All HP-UX tunables have man pages. To obtain information about the behavior, allowed values, and side effects of a tunable, consult the man page for that tunable, which can be found in section 5 of the online manual. An overview of all of the kernel tunables can be found in the "Tunable kernel parameters" document, available on docs.hp.com.

Changing tunable values

To change the value of a tunable, put tunable value assignments on the *kctune* command line. (Or see the section "Managing configurations with system files" on page 25.) For example, to set `nproc` to 4300:

```
# kctune nproc=4300
```

To set a tunable to "Default," either of these assignments will work. (Setting a user-defined tunable to "Default" causes it to be removed.)

```
# kctune nproc=
# kctune nproc=default
```

Assignments can be to expressions, as noted above. Note that the assignment may need to be quoted to avoid interpretation by the shell.

```
# kctune 'nkthread=nproc*2+100'
```

To create a user-defined tunable, use the `-u` (user-defined) flag when you assign the tunable a value. The `-u` flag is not needed to change the value of an existing user-defined tunable.

Using the += symbol, you can increase the value of a tunable (by 100, in this example):

```
# kctune nproc+=100
```

Using the >= symbol, you can ensure a minimum value of a tunable. The following command will set `nproc` to 5000 if its current value is below 5000. If its current value is already 5000 or greater, it will be left unchanged. Note that the assignment was quoted to avoid interpretation by the shell:

```
# kctune 'nproc>=5000'
```

See the *kctune*(1M) man page for details.

When you change a tunable value using a command similar to the above examples, the change will be made immediately to the currently running system, if possible. Sometimes it's not possible to make the change immediately; for example, you might be trying to reduce the maximum value of some resource to below the current usage. Also, there are some tunables that cannot be changed without a reboot. In those cases, *kctune* will hold the change and apply it at next boot. If any change on the *kctune* command line has to be held for next boot, they all will be.

Sometimes you may want to force a change to be held for next boot, instead of applying it immediately. In these cases you can give the `-h` (hold) flag to *kctune* to force that behavior. HP recommends that this flag be used only when the next boot is expected to be soon. If the next boot doesn't happen for months after making such a change, the system administrator could be unpleasantly surprised at the effect of a pending change that had been forgotten.

Changes to saved kernel configurations can be made by using the `-c` (configuration) flag. Such changes are made to the saved configuration immediately, but they won't affect the running system until that saved configuration is either loaded or booted. See the section "Managing saved configurations using *kconfig*" on page 23 for more information.

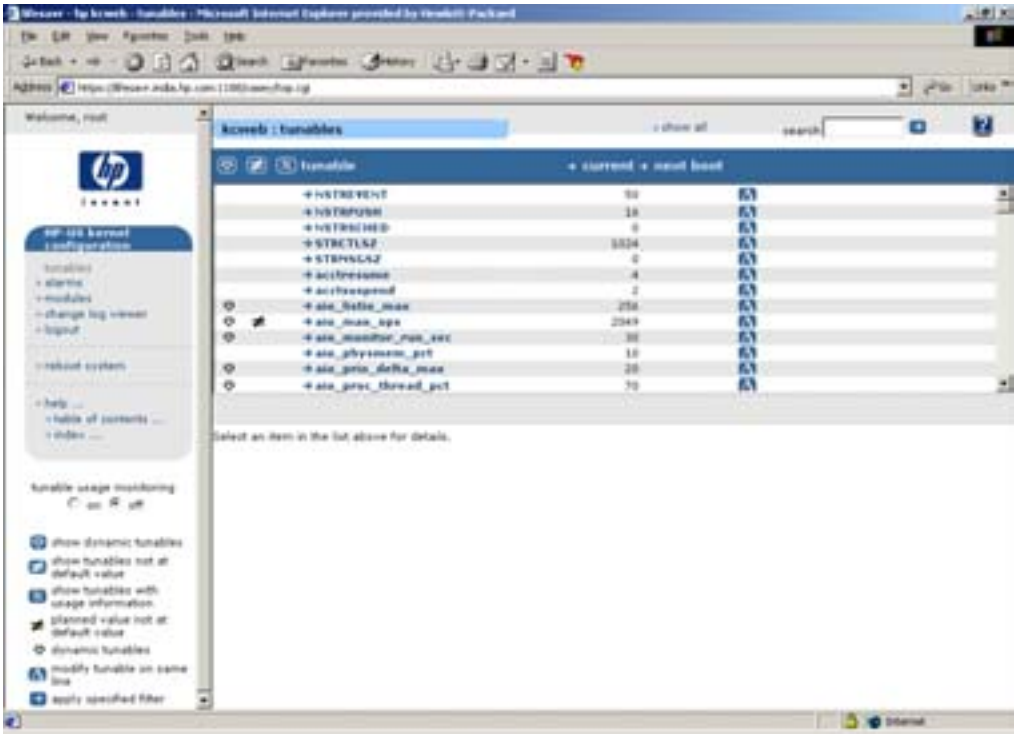
When changing tunable values, *kctune* supports the `-B` and `-K` flags to specify backup behavior, and the `-C` flag to specify a log file comment. See the sections "Recovering from errors" on page 30 and "The kernel configuration log file" on page 28 for details.

Managing kernel tunable parameters with *kcweb*

kcweb can be used to query and change the values of kernel tunable parameters ("tunables") in the currently running configuration. Using *kcweb*, you can

- Modify the value of a tunable
- View details about a tunable
- Search for a tunable
- Check the current and next boot value for a tunable
- Print details about a tunable or print a list of all tunables

You can view the tunables pane by choosing the **tunables** menu item from the navigation column in *kcweb*.



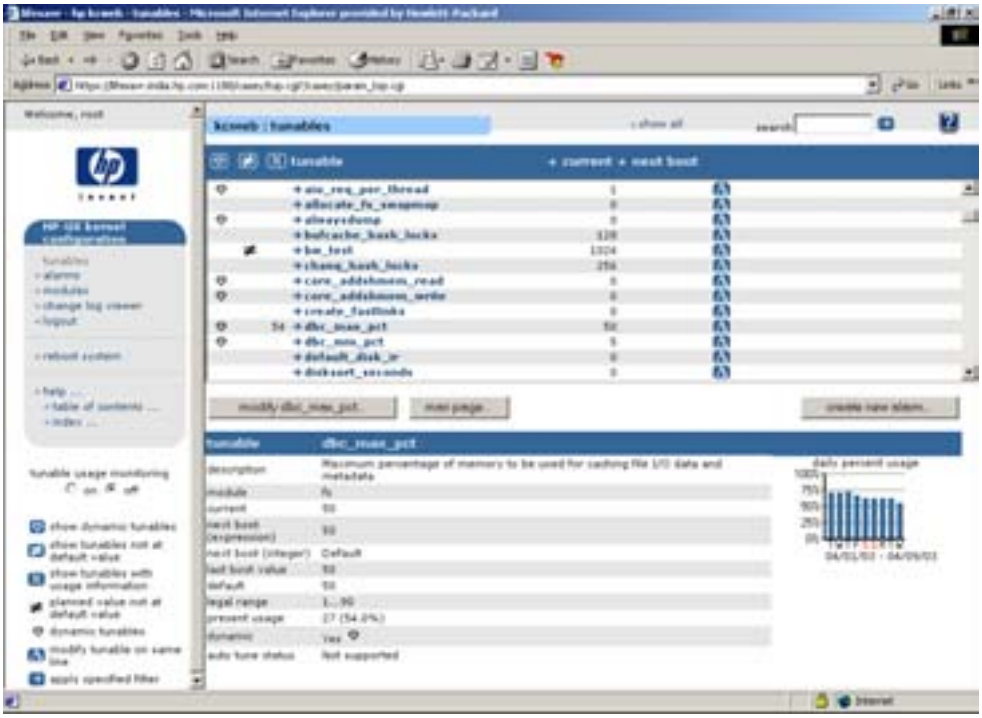
Getting information about tunables

To get more detailed information about a particular tunable, execute the following two steps:

- Select the **tunables** menu item in the navigation column. The tunables pane is displayed, listing all the tunables that are currently configured on your system.
- Select a tunable to view the details about a particular tunable in the details pane.

Interpreting tunable information

If you choose a tunable, the tunable details pane is displayed.



The tunable details pane contains the following information:

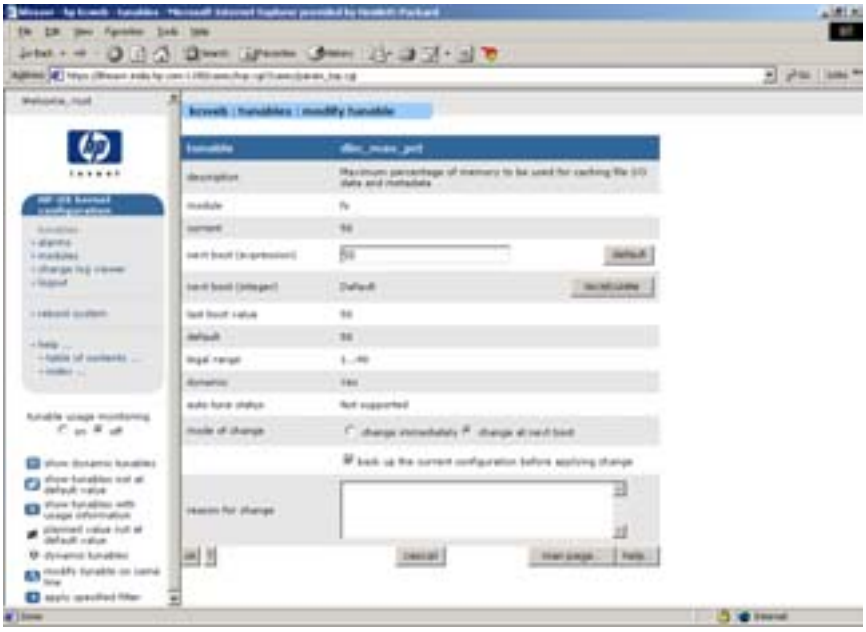
Field name	Description
tunable	Indicates the name of the tunable.
description	Indicates a brief description of the tunable.
module	Indicates the name of the module (if any) that the tunable is associated with.
current	Indicates the current maximum value for the resource.
next boot (expression)	Indicates a formula describing the next boot value (note: this can also be an integer).
next boot (integer)	Indicates the planned value, with all formulae computed.
last boot value	Indicates the value of the tunable when the system was last booted.
default	Indicates the default value for the tunable.
legal range	Indicates the range of values that are legal for the tunable.
present usage	Indicates the amount of the resource consumed at the time the pane was displayed, as an integer value followed by the percent usage of resource in parentheses.
dynamic	Indicates that a dynamic kernel tunable can be modified without rebooting the system.
auto tune status	Indicates whether the tunable is being automatically tuned.

Changing tunable values

To change the value of a tunable, execute the following steps:

- Select the **tunables** menu item in the navigation column. The tunables pane is displayed, listing all the tunables that are currently configured on your system.
- Select a tunable that you wish to modify by choosing the  icon or the ***modify tunable_name*** button.

The "modify tunable" page is displayed.



The modify tunable page contains the following fields:

Field name	Description
tunable	Indicates the name of the tunable that will be modified.
description	Indicates a description of the tunable.
module	Indicates the kernel module that the tunable is associated with.
current	Indicates the current value of the tunable.
next boot (expression)	A formula describing the next boot value (can be an integer).
next boot (integer)	Indicates the calculated value of the user input field "next boot"; may need to be refreshed by clicking the **recalculate** button.
last boot value	Indicates the value of the tunable when the system was last booted.
default	This is the default value of the tunable; pressing the **default** button will copy the default value into the planned field.
legal range	Indicates the range of acceptable values for the tunable; negative numbers are indicated by a minus sign (-), positive values have an implicit plus sign (+). "NA" means not available, and indicates that the underlying command, *kctune*, is returning neither a minimum nor a maximum value.
dynamic	Indicates whether the tunable value can be changed without rebooting the system.
auto tune status	Indicates whether the tunable is automatically tuned.
mode of change	Contains a set of radio buttons to apply kernel configuration changes immediately or to hold kernel configuration changes till next boot. This field will appear only for dynamic tunables. By default, kernel configuration changes will be held till next boot.
back up the current configuration before applying change	Implies backup of current configuration before applying the change. By default, this checkbox is selected.
reason for change	Enter a comment.

Monitoring kernel resource usage

Some tunable parameters represent kernel resources whose usage can be monitored. For these tunables, you can set alarms to notify you when the usage of the corresponding kernel resource crosses a threshold you specify.

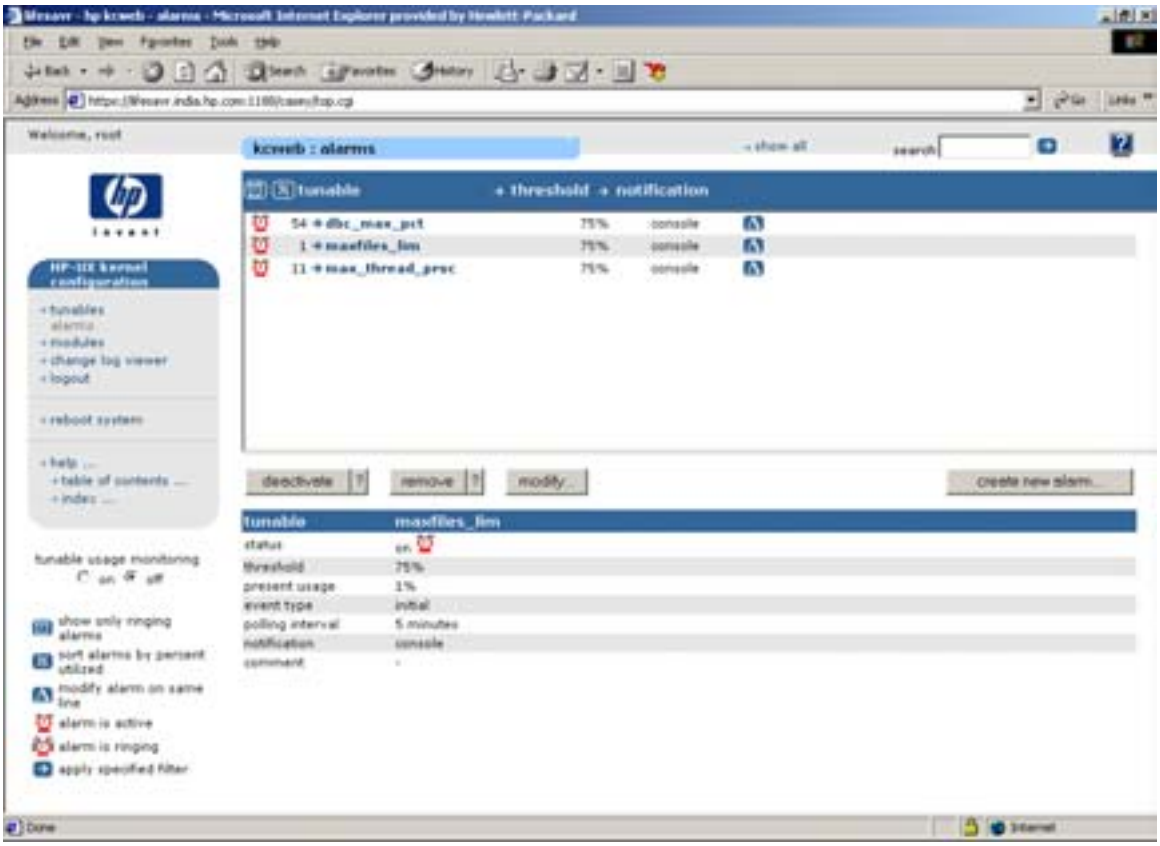
Getting information about alarms

To get more detailed information about a particular alarm using *kcweb*, execute the following two steps:

- Select the **alarms** menu item in the navigation column. The alarms pane is displayed, listing all the alarms that are currently configured on your system.
- Select an alarm to view the details about a particular alarm in the details pane.

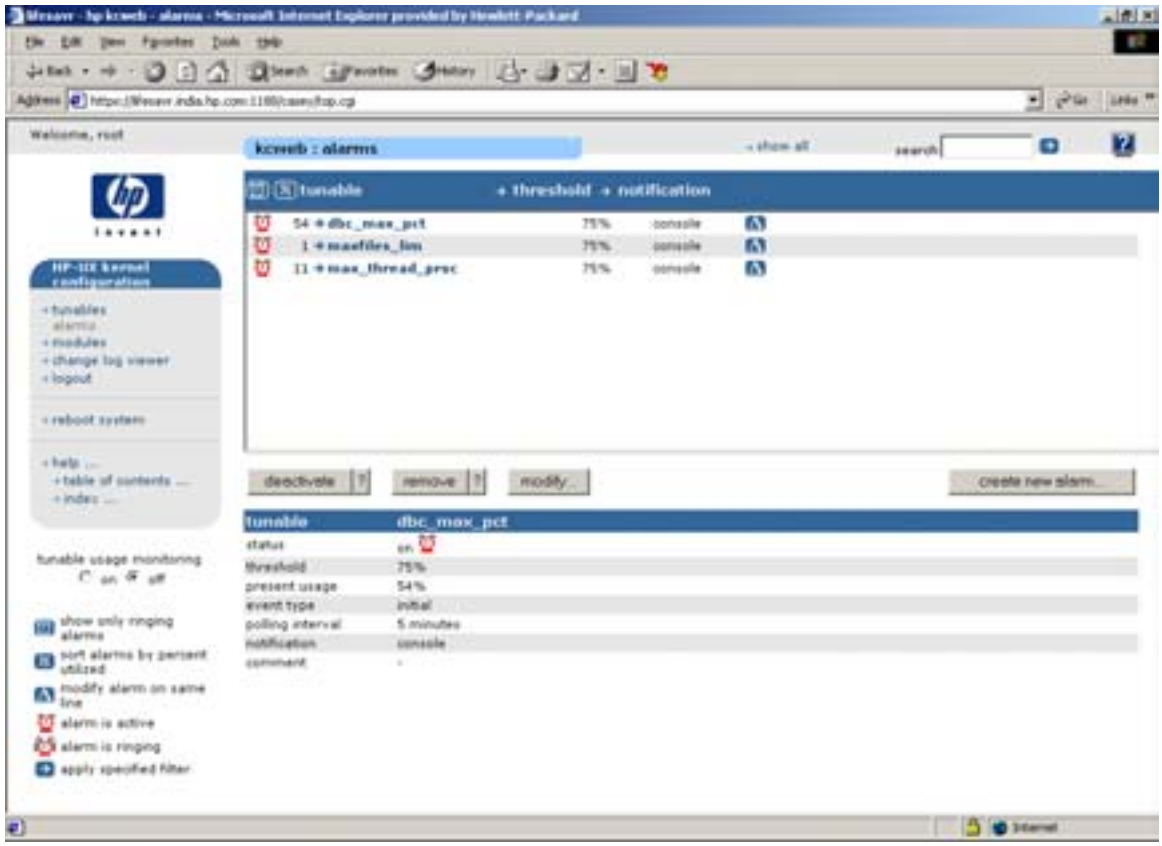
The alarms page allows you to:

- Create and remove alarms
- Activate and deactivate alarms
- Find alarms that have been triggered
- View details on alarms



Interpreting alarms information

If you choose an alarm, the alarm details pane is displayed.



The alarms details pane contains the following information:

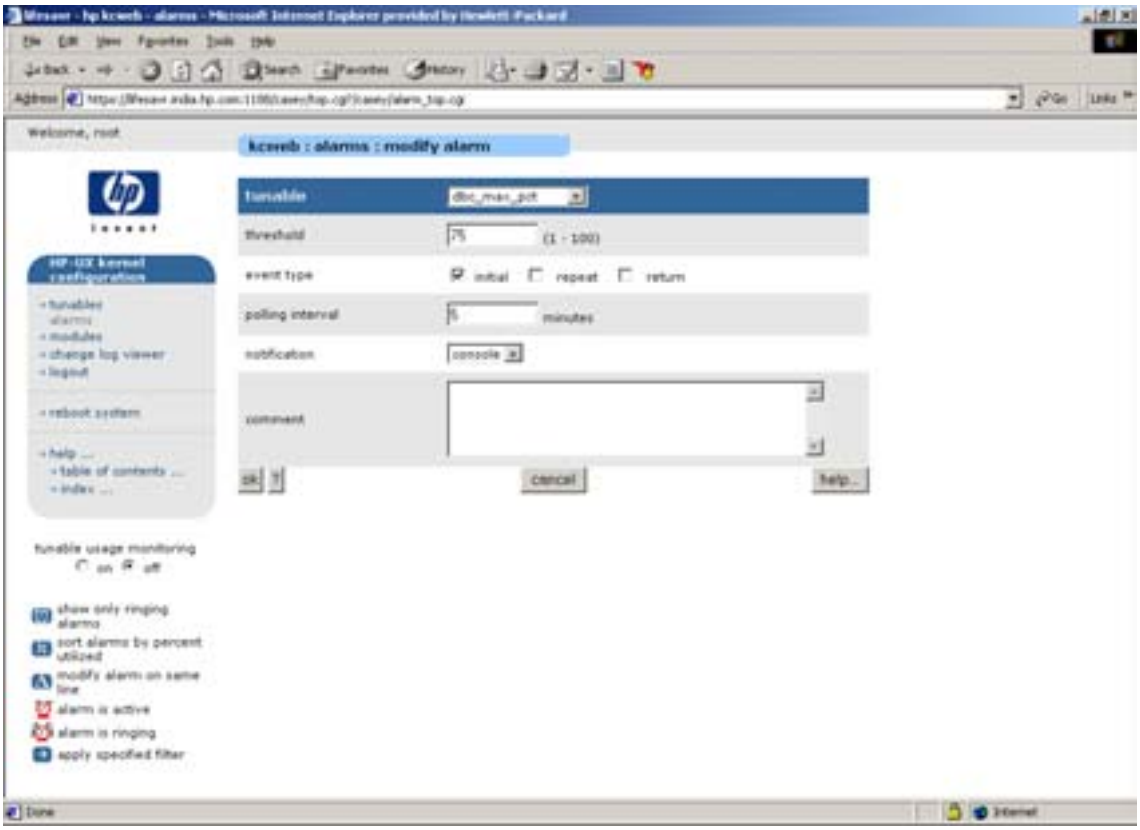
Field name	Description
tunable	Indicates the name of the tunable.
status	Indicates the status of the alarm if it is active or if the resource is currently exceeding the threshold.
threshold	Indicates the percentage at which the alarm should activate.
present usage	Indicates the percentage of resource being consumed at the previous polling.
event type	Indicates the event notification to be used.
polling interval	Indicates the time interval between polling.
notification	Indicates the method used to notify about alarm triggering.
notification data	Indicates supplementary information used by the notification method (not present if the notification method does not require it).
notification port	Indicates the port to communicate notification on (not present if not required by the notification method).
comment	Indicates the comment field; some comment data is added automatically when alarms are deactivated.

Changing alarm values

To change the value of an alarm for a tunable, execute the following steps:

- Select the **alarms** menu item in the navigation column. The alarms pane is displayed, listing all the alarms that are currently configured on your system.
- Select an alarm that you wish to modify by choosing the  icon or the **modify . . .** button.

The modify alarm page is displayed.



The modify alarm page contains the following fields:

Field name	Description
tunable	Indicates the name of the tunable for which the alarm will be modified.
threshold	Indicates the percent at which the alarm is to trigger.
event type	Displays the checkboxes that determine when notifications are to be sent: **initial** - first polling at which resource usage exceeds threshold; when an alarm is first added, activated, deactivated, or the system reboots **repeat** - each polling at which resource usage exceeds threshold (this can lead to a large number of messages if the polling interval is small) **return** - first polling at which resource usage falls below threshold If none of the checkboxes are checked, the default event type, as set by *kcalarm* will be used. **Note:** More than one checkbox can be checked; selecting both "initial" and "return" will generate a notification whenever the usage crosses above or below the threshold.
polling interval	Displays the interval, in minutes, between polling of resource usage.
notification	Displays the notification method (`console, opcmsg, syslog, textlog, email, snmp, tcp, udp`).
comment	Indicates the comment field.

Resource usage commands

The *kcalarm* command is used to add, delete, or list selected kernel tunable alarms, as well as turn kernel tunable monitoring on and off.

kcalarm is used to manage selected kernel tunable alarms and monitors; alarms and monitors are implemented in the *kcmond* daemon. Users can create, modify, delete, and list selected kernel tunable alarms. Alarms send a notification though various notification targets when a kernel tunable crosses a specified percentage threshold of its current setting.

Monitoring is the process of collecting historical tunable data. When this feature is turned on, historical data is collected on the usage of supported tunables. The data is used by the *kcusage* command to generate usage tables (including top consumers) for supported kernel tunables. The data also enables usage graphs in the *kcweb* tool. Monitoring is turned on by default when the *kcweb* tool is installed.

For more information, see the *kcalarm*(1M), *kcmond*(1M), and *kcusage*(1M) man pages.

Managing the running configuration with *kconfig*

The *kconfig* command has two options that are useful for dealing with changes to the currently running kernel configuration being held for next boot. Configuration changes are held for next boot when requested (using the `-h` (hold) option to *kcmodule* or *kctune*, or the `-n` (next boot) option to *kconfig*). Configuration changes are also held for next boot when they cannot be applied to the currently running system.

To get a list of changes being held for next boot, run `kconfig -D` (Differences). This is really just a short cut for running `kcmodule -D` and `kctune -D`. Similarly, to get a list of configuration settings that are set to non-default values, run `kconfig -S` (Set). This is a short cut for running `kcmodule -S` and `kctune -S`[2].

If you decide that you don't want those changes to be applied at next boot after all, run `kconfig -H` (unHold). All changes being held for next boot will be discarded.

For more information on changes being held for next boot, see the section "Persistence of changes" on page 7.

Managing saved configurations using *kconfig*

When you have an HP-UX kernel configuration that satisfies your needs, you may want to save a copy of it to protect yourself against inadvertent configuration changes. Or you may want to have multiple kernel configurations so you can switch between them easily. HP-UX allows you to save as many kernel configurations as you wish (subject to available disk space in `/stand`), and to modify them and use them at will.

Getting information about saved configurations

When you run *kconfig* with no options, it shows you the saved configurations on your system. There will always be a saved configuration called `backup`, which is automatically maintained by the system; any other saved configurations on the system will also be listed. (For more information on the `backup` configuration, see the section "Recovering from errors" on page 30.)

The output of *kconfig* can be varied with several options. The output can be restricted to specific configurations by listing them on the command line.

To control the output format, use the `-a` (all), `-v` (verbose), or `-P` (Parse) flags. Without these flags, the output looks like this:

```
Configuration   Title
backup          Automatic Backup
day             Configuration for daytime multiuser processing
night           Configuration for nighttime batch processing
```

[2] Device binding changes are not included in these outputs. See the section "Managing device bindings" on page 27.

Using the `-v` flag gives verbose, multi-line information about each saved configuration:

```
Configuration backup
Title               Automatic Backup
Save Time           Sun Jan 12 07:46:40 2003
Modify Time         Sun Jan 12 07:46:40 2003

Configuration day
Title               Configuration for daytime multiuser processing
Save Time           Sun Jan 12 07:49:00 2003
Modify Time         Sun Jan 12 07:49:00 2003

Configuration night
Title               Configuration for nighttime batch processing
Save Time           Sun Jan 12 07:52:12 2003
Modify Time         Sun Jan 12 07:52:12 2003
```

Using the `-a` flag gives the same output as using the `-v` flag, except that after each saved configuration, the entire outputs of `kcmodule -a -v` and `kctune -v` for that configuration are displayed. This gives a record of all settings in the configuration (except device bindings).

The `-P` flag, which is designed for use by scripts or programs, gives complete control over what information is printed:

```
# kconfig -P name,title
name                backup
title               Automatic Backup

name                day
title               Configuration for daytime multiuser processing

name                night
title               Configuration for nighttime batch processing
```

For more information on the `-P` flag and its use by scripts or programs, see the section "Parsing command output" on page 29, or the *kconfig*(5) man page.

Interpreting saved configuration information

Referring to the examples above, each saved configuration has a name. The names must start with a letter; contain only letters, digits, and underscores; and be at most 32 characters long. Except for the `backup` configuration, you choose the name for each saved configuration when you create it, and you can rename it at will.

Each saved configuration can also have a title. The title can be used to provide a longer description of the configuration's purpose or settings. It is optional.

Each saved configuration also has a pair of timestamps. "Save Time" indicates when the configuration was last saved (`kconfig -s`). "Modify Time" indicates when the configuration was last changed.

Associated with each saved configuration is a complete set of module state settings, tunable value settings, and device bindings. These can be seen using

`kcmodule -c` *configname*

and

`kctune -c` *configname*

or by using

`kconfig -a` *configname*

(Device bindings are visible only by looking at the system file for the saved configuration, located in `/stand/`*configname*`/system`.)

Using and modifying saved configurations

Creating saved configurations

Saved kernel configurations can be created in three ways: by saving the currently running configuration, by copying an existing saved configuration, or by reading a system file.

To save the currently running configuration, use `kconfig -s` (save). The resulting saved configuration will include any changes to the currently running configuration that are being held for next boot.

An existing saved configuration can be copied using `kconfig -c` (copy).

For information on working with system files, see the section "Managing configurations with system files" on page 25.

Using saved configurations

A saved configuration can be loaded using `kconfig -l` (load). This changes the configuration of the currently running kernel to match what was saved. If the configuration can be changed without a reboot, the changes will take effect immediately. Otherwise, all of the changes will be held for next boot.

Sometimes you may want to force the configuration change to be held for next boot rather than applying it immediately. In these cases, mark the saved configuration for use at next boot using `kconfig -n` (next boot). HP recommends that this flag be used only when the next boot is expected to be soon. If the next boot doesn't happen for months after making such a change, the system administrator could be unpleasantly surprised at the effect of a pending change that had been forgotten.

To find out which saved configuration is marked for use at next boot, use `kconfig -w` (which). This command also identifies the saved configuration that was most recently loaded or booted, or the system file that was most recently imported.

Modifying saved configurations

To modify the module state settings and tunable value settings in a saved configuration, use the `-c` (configuration) option to the *kcmodule* and *kctune* commands, respectively. Saved configurations can also be changed by changing their system file and then importing it; see the section "Making configuration changes with system files" on page 25.

Several options to *kconfig* allow other changes to saved configurations. The `-r` (rename) option will rename a saved configuration. (The `backup` configuration cannot be renamed.) The `-t` option will change the title on a saved configuration. The `-d` (delete) option will delete a saved configuration.

If a configuration has been marked for use at next boot and you decide you want to continue using the currently running configuration instead, use `kconfig -H` (unHold) to discard all changes being held for next boot.

Managing configurations with system files

Every kernel configuration has a corresponding system file. A system file is a flat text file that describes all of the configuration settings in a compact, machine-readable, portable format. The format of a system file is described in detail in the *system*(4) man page. It is an enhancement of the format used in previous releases of HP-UX; the previous formats are still accepted.

Making configuration changes with system files

System files provide an alternate mechanism for kernel configuration because configuration changes can be made by editing a system file and then telling the kernel configuration tools to apply the changes. This is the kernel configuration method most familiar to users of older versions of HP-UX.

To make configuration changes using a system file, start with the system file corresponding to the configuration you want to change.[3] The system automatically maintains system files for each configuration. The system file for the currently running configuration is located at `/stand/system`. The system file for any saved configuration is located at `/stand/`*`confignam`*`e/system`. If you want to create a new system file for a configuration, use the `kconfig -e` (export) command. This command takes two forms:

```
kconfig -e filename                    (export the running configuration)
```

[3] You will be asked to confirm your changes if the system file comes from a different configuration than the one you're changing, or if it's out of date with respect to the configuration you're changing.

```
kconfig -e configname filename          (export a saved configuration)
```

Note: `/stand/system`, and any system file created by exporting the running configuration, always reflects any changes that are being held for next boot.

Once you have a system file, you can edit it using any text editor, making the changes you desire. After editing it, you can apply the changes with the `kconfig -i` (import) command. This command takes three forms:

```
kconfig -i filename                     (import to running configuration, now)
kconfig -h -i filename                  (import and hold for next boot)
kconfig -i configname filename          (import to saved configuration)
```

In the first form, if the changes cannot be applied to the running system, they will be held for next boot.

For backward compatibility, the *mk_kernel* command is still available to apply changes made in a system file. Note that its name is no longer accurate since it will apply configuration changes without making a kernel, if it can. This command has the form:

```
mk_kernel [-o target] [-s filename]
```

filename is the name of the system file to read; if not specified, `/stand/system` is used. To import to a saved configuration, *target* should be the name of the configuration. To import to the currently running system, taking effect immediately if possible, *target* should be `/stand/vmunix`. (Changes will be held until next boot if they cannot be applied immediately.) If *target* is omitted, the changes will be made to a saved configuration called `hpux_test`. It is not possible to import to the currently running system using *mk_kernel*, forcing changes to be held for next boot. Use `kconfig -h -i` for this purpose.

It is important to note that the system files at `/stand/system` and `/stand/`*configname*`/system` are automatically recreated after every configuration change. In this process, comments in the system file are not preserved. Also, the ordering of lines in the file is not preserved. Therefore, HP recommends against putting comments in the system files. Instead, to add your comments directly to the kernel configuration log file, use the `-C` (Comment) option when importing the configuration. (See the section "The kernel configuration log file" on page 28.)

Most changes made in system files can be made using the kernel configuration commands, and vice versa. Following are the equivalents:

System file line	Kernel configuration command
modulename	`kcmodule` *modulename*`=best`
`module` *modulename* best	`kcmodule` *modulename*`=best`
`module` *modulename state* [*version*][4]	`kcmodule` *modulename*`=`*state*
(no entry for *modulename*)	`kcmodule` *modulename*`=unused`
tunablename tunablevalue	`kctune` *tunablename*`=`*tunablevalue*
`tunable` *tunablename tunablevalue*	`kctune` *tunablename*`=`*tunablevalue*
(no entry for *tunablename*)	`kctune` *tunablename*`=default`
`swap` *swapdevice*	(no equivalent)
`dump` *dumpdevice*	(no equivalent)
`driver` *devicename drivername*	(no equivalent)

[4] System files created by the kernel configuration tools always list the version number for each module. However, it is not required. Administrators adding `module` lines to a system file need not give version numbers.

Uses for system files

System files are primarily useful in four situations. First, they are useful for system administrators who are familiar with them from previous releases of HP-UX. If you are used to editing `/stand/system` and running *mk_kernel* to make configuration changes, it will still work.

Second, system files are the only mechanism through which device bindings can be seen or changed. See the section "Managing device bindings" on page 27 for more details.

Third, system files are useful if you want to apply multiple configuration changes simultaneously. You can edit `/stand/system` and change three tunable values and two module states, and have all of those changes take effect together when you import the system file with `kconfig -i` or `mk_kernel`. By contrast, each invocation of one of the kernel configuration commands applies changes separately (although multiple changes listed on the same configuration command line are applied together).

Applying multiple changes together is particularly valuable when modules are moved into or out of `static` state, because each command that does this will run for quite a while. This is because such changes require that the kernel executable be relinked. If you have multiple such changes to make, it is best that you list them all on the same *kcmodule* command line, or make the changes in a system file and import it. Either of these techniques will ensure that the kernel executable is only relinked once.

The other primary use for system files is copying configurations from one system to another. It is not safe to copy a kernel configuration directory from one machine to another, and HP does not support doing that. However, it is perfectly safe to export a system file from a configuration on one system, move that system file to a different system, and import it there. This is an appropriate and effective way to ensure that two machines are running compatible configurations. (Compatible means they have the same set of kernel modules, but they may have different versions of those modules due to patch installations.)

In some cases, running compatible configurations is not enough—you need to be sure that two machines are running *exactly* the same configuration. In that case, use the `-V` (Version match) flag while importing the system file on the target system. This flag turns on strict version checking, and the import will fail if the two machines have different versions of kernel modules installed.

Managing device bindings

Device bindings are infrequently used configuration settings that can only be configured using system files (see the section "Managing configurations with system files" on page 25). Device bindings are notations about how particular hardware devices should be used or controlled. There are three basic types of device bindings supported by HP-UX: primary swap device specifications, dump device specifications, and device driver specifications. Most kernel configurations have no device bindings.

Primary swap device

Each kernel configuration is allowed to have a primary swap device specification. In essence, this specifies which disk volume should be used by the system for paging. At present, only the primary swap device is specified using the kernel configuration mechanisms; other swap devices, if desired, are configured after boot using the *swapon* command or system call, or through entries in `/etc/fstab`. (See *swapon*(1M), *swapon*(2), and *fstab*(4) for details.)

The primary swap device is specified in a system file as a line with one of the following forms:

`swap` *deviceID* [5]

```
swap lvol
swap none
swap default
```

Only one such line is allowed. If no such line is specified, `swap default` is assumed.

The first form explicitly identifies the disk device to use for paging. The disk device may not contain a file system, and must not be an LVM or VxVM physical volume. Disks are presently identified using hardware paths (see *ioscan*(1M) for details), but this may change in future HP-UX releases.

The second form (`swap lvol`) specifies that the primary swap device is one of the logical volumes in the root LVM volume group, and that the *lvlnboot*(1M) command has been used to identify the logical volume.

[5] Earlier versions of HP-UX allowed the specification of a starting offset and size of the paging area on the specified device. These specifications are still accepted for backward compatibility; see *system*(4) for details. New installations should not use these obsolescent features.

The third form (`swap none`) specifies that there should be no primary swap device. The system will be unable to perform paging activities.

The fourth form (`swap default`) specifies the default behavior. It is equivalent to `lvol` if the system boots from an LVM volume group. Otherwise, paging is directed to the disk containing the root file system, in the area between the end of the file system and the end of the disk.

Dump devices

Each kernel configuration is allowed to have any number of dump devices. These are devices to which a system crash dump should be written, if a system crash occurs. The dump devices specified in the kernel configuration are typically only used during the boot process; once the boot process completes, the system uses the dump devices specified in `/etc/fstab` instead. (See *crashconf*(1M) for more details.)

Dump devices are specified in a system file as lines with the following forms:

`dump` *deviceID*

`dump lvol`

`dump none`

`dump default`

Any number of such lines can be specified. If no such lines are specified, `dump default` is assumed.

The first form (`dump` *deviceID*) explicitly identifies the disk device to use for crash dumps. The disk device may not contain a file system, and must not be an LVM or VxVM physical volume. Disks are presently identified using hardware paths (see *ioscan*(1M) for details), but this may change in future HP-UX releases.

The second form (`dump lvol`) specifies that the crash dump device(s) are logical volumes in the root LVM volume group, and that the *lvlnboot*(1M) command has been used to identify the logical volume(s).

The third form (`dump none`) specifies that there should be no crash dump device. The system will be unable to save crash dump information in the event of a system crash.

The fourth form (`dump default`) specifies the default behavior. Crash dumps will be written to the primary swap device. (Using the same device for primary swap and for crash dumps is common and accepted.)

Device driver specifications

Most of the time, the system can correctly choose the device driver module that should control each hardware device in your system. In some circumstances, you may need to force a particular hardware device to be controlled by a particular device driver module. If so, you can specify an explicit attachment of the device to the driver in question. Most installations have no need to specify explicit device driver specifications.

Explicit device driver bindings are specified in a system file as lines with the following form:

`driver` *deviceID drivername*

The *deviceID* is the identification of the hardware device in question. Devices are presently identified using hardware paths (see *ioscan*(1M) for details), but this may change in future HP-UX releases. The *drivername* is the name of the kernel module that is the desired driver for the device.

The kernel configuration log file

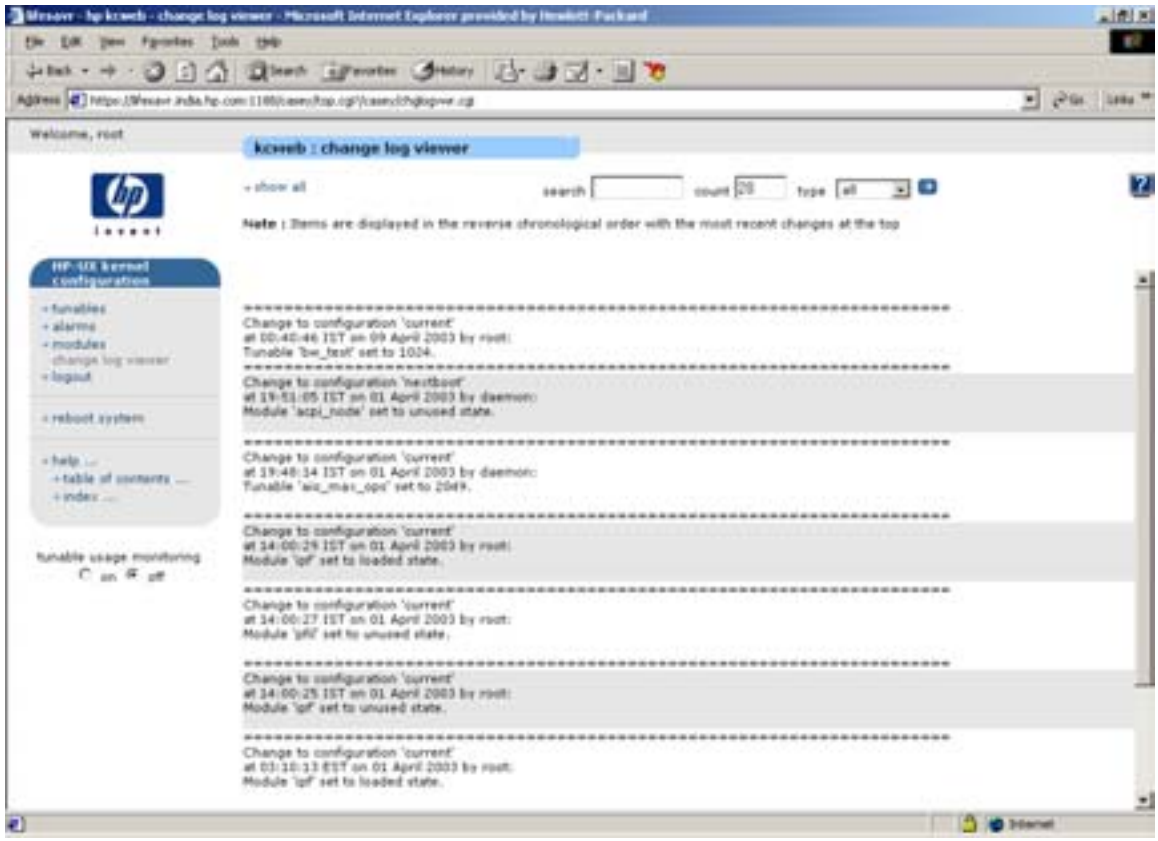
It is often useful to know what configuration changes have been made on a system. For this purpose, the kernel configuration tools automatically maintain a log file at `/var/adm/kc.log`. This file lists every change made using the kernel configuration commands. (Some configuration changes can be made by calling kernel system calls directly. These changes are not logged. Changes made through the *kcweb*, the Web-based GUI for kernel configuration, are logged since *kcweb* uses the kernel configuration commands to make the changes.)

The log file is a plain text file that you can view directly. The *kclog* command is provided for when you want to do an intelligent search of the log file, but its use is optional. (More information on the *kclog* command can be found in the *kclog*(1M) man page.)

All of the kernel configuration commands accept a `-C` (Comment) option when they are being used to make configuration changes. The `-C` option allows you to specify a comment that will be included in the log entry for your change. This can help readers of the log understand the reasons for your changes.

To add a comment to the log without making a configuration change, use `kclog -C`.

In the *kcweb* tool, you can select the **change log viewer** menu item from the navigation column to see the kernel configuration log file (in reverse order).



Parsing command output

Improvements to HP-UX often require changes in the output formats of commands like those described in this paper. This can be troublesome when applications or scripts have been written that parse the outputs of those commands. For this reason, each of the primary kernel configuration commands (*kcmodule, kctune*, and *kconfig*) have a special output format, selected using the `-P` (Parse) flag, designed for parsing by applications. In addition to providing release-to-release compatibility, it is also easier to parse than human-readable output.

Note: HP reserves the right to change the other output formats of these commands at any time. HP will not support applications and scripts that parse the output of these commands unless they use the `-P` option.

The `-P` option to each of these commands takes a list of field names, identifying the fields that the application wants to have appear in the output. The available field names are different for each command and are documented in the man pages for the commands. The list is comma-separated and cannot contain spaces. Examples are shown in the sections above.

The output format consists of one line per field, containing the field name, a single tab character (ASCII 9), the field value, and a newline (ASCII 12). The fields are printed in the order requested for each item, with empty lines between the items.

Some fields have multiple values. In these cases, there will be one line for each value of the field, each starting with the field name in the manner described.

Some fields do not have values under some circumstances. For example, the “value at last boot” tunable field has no meaning for tunables in a saved configuration. In these cases, no line will be printed for that field.

The special field name `ALL` can be used to retrieve all available data. When this field name is used, the output may include fields that are not listed in the man page. The order of fields in the output is undefined.

Recovering from errors

Occasionally, kernel configuration changes are made that are undesirable. Also, hardware failures and changes can ruin a previously acceptable kernel configuration. HP-UX has several mechanisms available to system administrators who need to recover from such issues. They include the kernel configuration log file (described above); saved configurations, including the automatically maintained `backup` configuration; and failsafe boot mode.

The automatic `backup` configuration

The system automatically maintains a saved configuration called `backup`. Generally, any time you use the kernel configuration tools to make a change to the currently running configuration, the previous (pre-change) configuration is saved to `backup`. Therefore, the `backup` configuration is somewhat like the "Undo" command in a word processor. In these cases, if you load the `backup` configuration using `kconfig -l backup`, it will reverse the last change you made to the currently running configuration using the kernel configuration commands.

Some changes can be made to the currently running configuration by calling kernel system calls directly. The `backup` configuration is not updated when those changes are made.

There are cases in which you may not want this automatic `backup` behavior. For example, if you have made an undesirable change and are trying to fix it, you do not want the kernel configuration commands to replace a good `backup` configuration with the one containing your undesirable change. The `-K` option (keep the existing backup) can be given to any kernel configuration command to disable the automatic update of the `backup` configuration. When making changes using *kcweb*, you can turn off the "back up the current configuration before applying change" checkbox to disable the automatic backup behavior.

When your system first boots, the `backup` configuration mirrors the configuration that was in use before the reboot. You may not want this replaced by the first kernel configuration change you make, especially since the first kernel configuration change could be made by a startup script before you even get a login prompt.

For this reason, the first configuration changes after a boot are handled specially. Instead of automatically replacing the `backup` configuration, the kernel configuration commands will ask you whether or not to do so. [6] They will continue to ask each time you make a change, until the first time you say "yes." From that point on, until next boot, they will automatically replace the `backup` configuration with each change as described above.

If you want to disable the automatic replacement of the `backup` configuration for a particular change, specify `-K`. If you want to force an automatic replacement of the `backup` configuration, specify `-B` (Backup). These flags work with any kernel configuration command that makes configuration changes.

Booting a saved configuration

In extreme circumstances, a mistaken configuration change can result in a kernel configuration that won't boot. In these cases, you have two options: boot a different configuration, such as the automatic `backup` configuration, and/or boot in failsafe mode (described below).

To boot a saved configuration on an Itanium® Processor Family system, interrupt the automatic boot process when it reaches the point that it has started the HP-UX boot loader. (On most systems, this is during the *second* 10-second countdown.) At the `HPUX>` prompt, type

```
HPUX> boot configname
```

To boot a saved configuration on a PA-RISC system, interrupt the automatic boot process when you arrive at the boot console handler. Tell it to boot from the desired device (typically with a `boot pri` command). When it asks if you want to interact with the ISL or IPL, say yes. (The exact mechanism to get to this point varies; consult your system's hardware manual or the *hpux*(1M) man page for details.) At the `ISL>` prompt, type

```
ISL> hpux configname /vmunix
```

In either case, this will boot the saved configuration named *configname*. When the boot is complete, it will be the currently running configuration; the previous configuration is lost (unless it had been automatically saved as `backup`).

[6] If the command is being run non-interactively, such as from a startup script, the answer is assumed to be "no" for *kcmodule*, *kctune*, and *kcdevice*, and "yes" for *kconfig*.

Booting in failsafe mode

The other alternative for recovering from an unbootable configuration is booting in failsafe mode. When you boot the system in failsafe mode, your configuration settings are ignored. All kernel tunables are given failsafe values, default device bindings are used, and no kernel modules are dynamically loaded during boot. This method is particularly useful when a hardware change or failure has caused all of your saved configurations to be unbootable.

To boot an Itanium Processor Family system in failsafe mode, get to the `HPUX>` prompt as described above and type

```
HPUX> boot -tm
```

To boot a PA-RISC system in failsafe mode, get to the `ISL>` prompt as described above and type

```
ISL> hpux -f0x40000
```

(The two methods can be combined if you want to boot a saved configuration in failsafe mode. This uses the kernel executable built for the saved configuration, including all of its static modules but none of its dynamically loaded modules.)

When you boot the system in failsafe mode, the previous kernel configuration will be automatically saved for you with a configuration name something like `saved_3DE78FA0`. The exact name will be printed for you in the boot messages on the console.

When you boot the system in failsafe mode, the boot will stop when you reach single user mode. At this time you should take any necessary steps to repair your system or your configuration and then reboot onto a valid configuration. HP does not recommend continuing to boot to multiuser mode after a failsafe boot.

Guidelines for recovering from errors

If you have an undesirable or unbootable kernel configuration, HP recommends the following approach to resolving the problem.

If your system is up:

- If you know which configuration change caused the problem:
  - If your `backup` configuration hasn't been updated since the bad change:
    - Load the `backup` configuration with `kconfig -l backup`.
  - If your `backup` configuration also has the problem in it:
    - Try to reverse the change using *kcmodule* or *kctune*.
    - Always specify the `-K` flag to preserve the `backup` configuration.
- If you don't know what change caused the problem, or the above didn't work:
  - Load a known good configuration using `kconfig -l`.
  - Try the `backup` configuration first.

If your system is down:

- If you have had a hardware failure and now the system won't boot or if you need to preserve the bad configuration:
  - Try booting in failsafe mode (see above).
  - Repair the configuration or the hardware, then reboot.
- If no hardware failure, no need to preserve bad configuration:
  - Try booting a known good configuration, such as `backup`.

Of course, depending on the level of your support contract with HP, you can call on HP field-service personnel to perform these steps, if needed.

If you get to a point where you cannot boot any of your saved configurations, even in failsafe mode, your last resort is to boot from the HP-UX installation media. If that succeeds, you do not necessarily have to reinstall HP-UX; you can open a shell and try to repair your system.

Kernel configuration example

```
demo [HP Release B.11.23]
Console Login: root
Password:
Please wait...checking for disk quotas
...
WARNING: YOU ARE SUPERUSER!!
```

In this example, Susie Admin is setting up a new HP-UX system to run a database server called "Prophet." It has just finished booting after the initial install.

```
# kconfig -C "Save initial installation config" -s installed
          * The current configuration has been saved to
            'installed'.
# kconfig -t installed "Initial installation"
          * The title of the configuration 'installed' has
            been set to "Initial installation".
```

The first thing Susie does is save a copy of the initial kernel configuration in case she needs it later. She puts comments on all of her changes (with –C). She also puts a title on the saved configuration to remind her what it contains.

```
# kctune enable_idds maxdsiz
Tunable               Value   Expression     Changes
enable_idds               0   Default
maxdsiz          0x40000000   Default        Immed
# kctune -d semmni shmmni
Tunable      Value    Expression       Changes
          Description
semmni       2048     Default
          Maximum number of semaphore sets on the system
shmmni       400      Default          Immed
          Maximum number of shared memory segments on
          the system
```

The manual for "Prophet" tells Susie to set `maxdsiz` to at least ½ TB, to set `semmni` to 3000, and to add 50 to whatever value she's using for `shmmni`. Being a security-minded system administrator, she knows she also wants to turn on Intrusion Detection by setting the `enable_idds` flag. Susie starts by looking at the current values of these tunables and the descriptions of ones she's unfamiliar with.

```
# kctune -C "Tunable settings for Prophet" "enable_idds=1" \
> "maxdsiz>=512000000" "semmni=3000" "shmmni+=50"
WARNING:    The requested changes cannot be made
            to the running system.
            They will be held until next boot.
            * The automatic 'backup' configuration has
              been updated.
NOTE:       No change to 'maxdsiz' was needed.
            * The requested changes have been saved,
              and will take effect at next boot.
```

Having done that, she sets the values as directed. She sets them all on the same command line so that they will all take effect at the same time. Since two of the changes cannot be made immediately, all of the changes are held for next boot.

```
Tunable                          Value    Expression    Changes
enable_idds    (now)                 0    Default
               (next boot)           1    1
maxdsiz                    0x40000000    Default       Immed
semmni         (now)              2048    Default
               (next boot)        3000    3000
shmmni         (now)               400    400           Immed
               (next boot)         450    450
```

```
# kcmodule -d idds
Module    State     Cause
        Description
idds      unused
        Intrusion Detection Data Source
```

To use Intrusion Detection, Susie knows she needs to have the `idds` module in her kernel configuration. She checks and sees that it is currently unused, so she adds it to her configuration.

```
# kcmodule -C "Add Intrusion Detection to the kernel."
idds=best
WARNING:  The requested changes cannot be made to the
          running system.
          They will be held until next boot.
          * The automatic 'backup' configuration has
            been updated.
          * Building a new kernel for configuration
            'nextboot'...
          * Adding version information to new kernel...
          * The requested changes have been saved, and
            will take effect at next boot.
Module    State     Cause
idds      static    best
```

`idds` needs to be built into the kernel executable itself, so a new kernel is built and marked for use at next boot.

```
# kconfig -D
Module             State       Cause
idds          (now)           unused
              (next boot)     static    best
Tunable                        Value    Expression    Changes
enable_idds   (now)                 0    Default
              (next boot)           1    1
```

Susie checks a summary of all of her changes that will take effect when she reboots.

```
semmni       (now)            2048   Default
             (next boot)      3000   3000
shmmni       (now)             400   Default      Immed
             (next boot)       450   450
```

Satisfied, she reboots. The system confirms that her changes will be applied.

```
# shutdown -r
...
      * The kernel registry database has been saved to disk.
      * The configuration changes that were being held for
        next boot have been applied.
...
The system is ready.

demo [HP Release B.11.23]
Console Login: root
Password:
Please wait...checking for disk quotas
...
WARNING: YOU ARE SUPERUSER!!
```

After the reboot, Susie saves the new kernel configuration under the name "good" so she can go back to it if needed. She gives it a title to help recognize it later.

```
# kconfig -C "Good configuration for Prophet" -s good
       * The current configuration has been saved to 'good'
# kconfig -t good "Good configuration for Prophet"
       * The title of the configuration 'good' has been set
         to "Good configuration for Prophet".
```

After some time, one of her users asks here to increase the size of the buffer cache, hoping to speed up the application. She complies—after all, it doesn't need a reboot, so she can do it without disturbing anyone. Since it's the first change after boot, the system asks whether to make automatic backups.

```
# kctune -C "Bigger buffer cache for better performance"
dbc_max_pct=20
WARNING:  The automatic 'backup' configuration currently
          contains the configuration that was in use before
          the last reboot of this system.
     ==>  Do you wish to update it to contain the current
          configuration before making the requested change?
          yes
        * The automatic 'backup' configuration has been
          updated.
        * The requested changes have been applied to the
          currently running system.
```

It's a good thing she said "yes." The larger buffer cache actually slowed things down—but all she has to do is restore the automatic backup.

```
Tunable                  Value   Expression    Changes
dbc_max_pct    (before)     10   Default       Immed
               (now)        20   20
# kconfig -C "Putting buffer cache back; performance was
  worse." -l backup
                * The configuration 'backup' has been loaded.
# kctune dbc_max_pct
Tunable        Value   Expression   Changes
dbc_max_pct    10      Default      Immed
```

While Susie's on vacation, her colleague Fred decides to use the machine for billing software during the night. This software needs to execute code on the stack (a security risk), so he enables that behavior (which is prohibited by default). No reboot is needed to do so.

```
# kctune -d executable_stack
  Tunable      Value   Expression   Changes
    Description
executable_stack   0   Default      Immed
    Enables execution of code on a stack (0 = no, 1 = yes,
    2 = yes but warn)
# kctune -C "Nightly billing s/w needs execute-on-stack"
executable_stack=1
       * The automatic 'backup' configuration has been
         updated.
       * The requested changes have been applied to the
         currently running system.
Tunable                      Value  Expression    Changes
executable_stack  (before)       0  Default       Immed
                  (now)          1  1
```

The billing software also uses the kernel Random Number Generator module. Fred checks and sees that it's not in use, but since it's loadable he doesn't need to reboot to use it.

```
# kcmodule -d rng
Module    State    Cause     Notes
   Description
rng      unused              loadable, unloadable
    Strong Random Number Generator
```

He goes ahead and loads the module.

```
# kcmodule -C "Random Number Generator needed for nightly
  billing jobs" rng=best
       * The automatic 'backup' configuration has been
         updated.
       * The requested changes have been applied to the
         currently running system.
Module               State    Cause    Notes
rng       (before)   unused            loadable, unloadable
          (now)      loaded   best
```

Fred saves these new configuration

```
# kconfig -C "Settings for nightly billing jobs" -s night
```

```
        * The current configuration has been saved to
          'night'.

# kconfig -t night "Nightly billing jobs"
        * The title of the configuration 'night' has been
          set to "Nightly billing jobs".

# kconfig -r good day
        * The configuration 'good' has been renamed to
          'day'.
# kconfig
Configuration    Title
backup           Automatic Backup
day              Good configuration for Prophet
installed        Initial installation
night            Nightly billing jobs

# kconfig -l day
        * The automatic 'backup' configuration has been
          updated.
        * The requested changes have been applied to the
          currently running system.

# kconfig -l night
        * The automatic 'backup' configuration has been
          updated.
        * The requested changes have been applied to the
          currently running system.

# kclog 5
============================================================
Change to configuration 'current'
at 21:49:08 PST on 02 February 2003 by root:
Module 'rng' set to loaded state.

Random Number Generator needed for nightly billing jobs
============================================================
Change to configuration 'night'
at 21:53:03 PST on 02 February 2003 by root:
Configuration saved from currently running configuration.

Settings for nightly billing jobs
============================================================
Change to configuration 'day'
at 21:53:26 PST on 02 February 2003 by root:
Configuration created by renaming 'good'.
============================================================
Change to configuration 'current'
at 21:55:49 PST on 02 February 2003 by root:
Configuration loaded from 'day'.
============================================================
Change to configuration 'current'
at 21:56:09 PST on 02 February 2003 by root:
Configuration loaded from 'night'.

# kconfig -e night /tmp/system.night
        * The configuration 'day' has been exported
          to /tmp/system.night.

# kconfig -C "Move nightly billing jobs" -iV night
  /tmp/system.night
        * /tmp/system.night has been imported to 'night'.

# kconfig -l night
        * The automatic 'backup' configuration has been
          updated.
NOTE:     The configuration being loaded contains changes
          that cannot be applied immediately. The changes
          will be held for next boot.
# shutdown -r
...
        * The kernel registry database has been saved to
          disk.
        * The configuration 'night' will be used at next
          boot, as requested.
```

settings under the name “night” (with a descriptive title).

Since “good” isn’t a very helpful name for Susie’s configuration anymore, Fred renames it to “day”. He checks the list of configurations to make sure everything looks OK.

Finally, he tries loading first the “day” configuration, and then the “night” configuration, to make sure he can move back and forth at will.

When Susie returns from her vacation, the first thing she does is check the automatically maintained log file to see what Fred has done.

She can see that Fred has put a new application on her server, and worse, an insecure one. At least he tested and documented his changes.

Susie doesn’t want to leave her system the way Fred changed it, so she moves the nightly billing job to another system. First, she exports his “night” configuration to a text file.

Moving the file over to another machine, she imports the configuration there, using the `-V` flag to ensure that exactly the same kernel software is in use. Then she loads the configuration. Something about the configuration can’t be changed immediately—probably a tunable setting—so she has to reboot the machine. As intended, the machine uses Fred’s “night” configuration when it comes back up.

Kernel configuration quick reference card

Working with kernel configurations	
Choose the configuration to boot:	
before the reboot[7]	`kconfig [-f] –n` *configname*
at the boot loader prompt (Itanium Processor Family)	`boot` *configname*
at the boot loader prompt (PA-RISC)	`hpux` *configname*`/vmunix`
List all kernel configurations	`kconfig [-v]`
Save the currently running configuration	`kconfig [-f] –s` *new-name*
Copy a saved configuration	`kconfig -c` *src dest*
Rename a saved configuration	`kconfig -r` *old new*
Delete a saved configuration	`kconfig [-f] –d` *configname*
Load a saved configuration	`kconfig [-f] –l` *configname*
Set the title of a configuration	`kconfig -t` *configname* "*title*"

Working with system files	
Create a system file:	
for a saved configuration	`kconfig -e` *configname filename*
for the currently running configuration[8]	`kconfig -e` *filename*
Create/update a configuration from a system file:[9]	
create/update a saved configuration	`kconfig -i` *configname filename*
update the currently running configuration	`kconfig [-fhV] –i` *filename*

Working with changes held for next boot	
`kconfig -i`, `kcmodule`, and `kctune` hold their changes until next boot if they can't be applied immediately, or if `-h` is specified.	
List all changes being held for next boot	`kconfig -D`
Discard all changes being held for next boot	`kconfig -H`

Working with tunables	
List tunables and their values:	`kctune` [*tunable...*]
verbose output	`-v`
only tunables with changes held for next boot	`-D`
include derived tunables set to default values	`-a`
group by module name	`-g`
in a saved configuration	`-c` *configname*
Set a tunable value	`kctune` *tunable*="*expression*"
Set a tunable to default	`kctune` *tunable*=default
Increment a tunable value	`kctune` *tunable+=value*
Make sure tunable value is at least n	`kctune` "*tunable>=n*"
Hold change until next boot	`-h`
Apply change to saved configuration	`-c` *configname*
Create user-defined tunable	`-u`

[7] If this option is used, there is no need to interrupt the boot process to select the new kernel configuration.
[8] Includes any changes being held for next boot.
[9] `mk_kernel` can also be used for this purpose.

Working with kernel modules	
List modules and their states:	`kcmodule` [*module*...]
verbose output	`-v`
only modules with changes held for next boot	`-D`
include required modules	`-a`
in a saved configuration	`-c` *confignam*e
Add a module to the configuration:	
in default state	`kcmodule` *module*=best
statically bound into the kernel executable	`kcmodule` *module*=static
dynamically loaded, now and at each boot	`kcmodule` *module*=loaded
autoloaded at first use	`kcmodule` *module*=auto
Remove a module from the configuration	`kcmodule` *module*=unused
Hold change until next boot	`-h`
Apply change to saved configuration	`-c` *confignam*e

Working with the kernel configuration log file	
The log file is located at `/var/adm/kc.log`. The `kc*` commands add a log entry for every change.	
Add a comment to the log file:	
while making a change with a `kc*` command	add `-C` "*comment*" to the change command
without making a configuration change	`kclog -C` "*comment*"
View the last n entries in the log (default 1)	`kclog` *n*
counting only changes to a configuration	`-c` *confignam*e
counting only changes of a particular type	`-t module|tunable|device`
counting only changes to a particular item	`-n` *modulename*|*tunablename*|*hwpath*
counting only log entries containing a string	`-f` "*string*"

Kernel configuration file locations	
Saved configurations are stored at	`/stand/`*confignam*e
Kernel executable is at	`/stand/`*confignam*e`/vmunix`
System file is at	`/stand/`*confignam*e`/system`
Currently running configuration is at	`/stand/current`
Kernel executable is at	`/stand/current/vmunix`
System file is at	`/stand/current/system`
Never directly manipulate any of the files in a kernel configuration directory, except the system file. Always use the `kc*` commands.	

Transition from previous HP-UX releases

Experienced administrators of previous releases of HP-UX will find some aspects of the 11i v2 kernel configuration mechanisms unfamiliar. However, many of the underlying concepts are unchanged. The tables in this section give information to help administrators translate from the old kernel configuration mechanisms to 11i v2.

Older HP-UX technique	HP-UX 11i version 2	See page
Use SAM to configure the kernel	Use *kcweb* to configure the kernel	
Look at `/stand/system` to see the current configuration	Same	25
Run an unsupported command to make sure `/stand/system` is up to date	Not needed; `/stand/system` is automatically kept up to date	25
Make configuration changes by editing `/stand/system` and running *mk_kernel*	Same; changes will be applied to the running system (no reboot) if possible	25
Make configuration changes by running *kmtune* or *kmsystem*, then running *mk_kernel*	Make the changes with *kctune* or *kcmodule* (no *mk_kernel*), or edit `/stand/system` manually and then run *mk_kernel*	8, 14, or 25
Make configuration changes by editing `/stand/system` and running *config*	Use *mk_kernel* instead	25
Manage DLKMs with the *kminstall*, *kmsystem*, *kmmodreg*, *kmadmin*, *kmupdate*, and *config* commands	Manage DLKMs using *kcmodule*	8
View or change tunables using *kmtune*	Use *kctune* instead (see note below)	14

Older HP-UX command/option	HP-UX 11i version 2	See page
`config` (without `-M`)	`mk_kernel`	26
`config -M`	no longer needed	
`kmadmin -b`	no longer needed	
`kmadmin -k`	kcmodule	8
`kmadmin -L` *modulename*	`kcmodule` *modulename*`=loaded`	8
`kmadmin -U` *modulename*	`kcmodule` *modulename*`=unused`	8
`kmadmin -u` *module_id*	`kcmodule` *modulename*`=unused`	8
`kmadmin -q` *module_id*	`kcmodule -v` *modulename*	8
`kmadmin -Q` *modulename*	`kcmodule -v` *modulename*	8
`kmadmin -s`	`kcmodule`	8
`kmadmin -S`	`kcmodule -v`	8
`kminstall`	no longer needed	
`kmmodreg`	no longer needed	
`kmpath` (no options)	`kcpath -x` (see note below)	
`kmpath -k`	`kcpath -b`	
`kmpath -c`	`kcpath -d`	
`kmpath -i`	no longer needed	
`kmsystem` (no options)	`kcmodule`	8
`kmsystem -b`	no longer needed	
`kmsystem -c y -l y` *modulename*	`kcmodule` *modulename*`=loaded`	8
`kmsystem -c y -l n` *modulename*	`kcmodule` *modulename*`=static`	8
`kmsystem -c n` *modulename*	`kcmodule` *modulename*`=unused`	8
`kmsystem -q` *modulename*	`kcmodule -v` *modulename*	8
`kmtune` (no options)	`kctune` (see note below)	14
`kmtune -l`	`kctune -v`	14
`kmtune -q` *tunable*	`kctune` *tunable*	14
`kmtune -r` *tunable*	`kctune` *tunable*`=Default`	14
`kmtune -u -s` *tunable*=*value*	`kctune` *tunable*=*value*	14
`kmtune -u -s` *tunable*+*value*	`kctune` *tunable*+=*value*	14
`kmtune -s` *tunable*=*value*	`kctune -h` *tunable*=*value*	14
`kmupdate` (no options)	`kconfig -n` hpux_test	25
`kmupdate` *kernel*	`kconfig -n` *configuration*	25
`kmupdate -M` *module*	no longer needed	
`kmupdate -d` *kernel*	`kconfig -d` *configuration*	25
`mk_kernel` (without –M)	`mk_kernel`	26
`mk_kernel -M`	no longer needed	

Older HP-UX file/directory	HP-UX 11i version 2	See page
Currently running kernel `/stand/vmunix`	`/stand/vmunix`	
Backup kernel `/stand/vmunix.prev`	Backup configuration `backup`	30
Test kernel `/stand/build/vmunix_test` (default output of *mk_kernel*)	Test configuration `hpux_test`	25
Primary system file `/stand/system`	`/stand/system`	25
Module system files `/stand/system.d/*`	No longer used; the data is now in the primary system file `/stand/system`.	25
Master files `/usr/conf/master.d/*`	No longer used; the data is embedded into the kernel code, and is available through the *kcmodule* and *kctune* commands	8, 14

Note: HP-UX 11i v2 contains compatibility stubs for *kmpath* and *kmtune*, but they will be removed in a future release of HP-UX.

HP-UX 11i release names and release identifiers

With HP-UX 11i, HP delivers a highly available, secure, and manageable operating system that meets the demands of end-to-end Internet-critical computing. HP-UX 11i supports enterprise, mission-critical, and technical computing environments. HP-UX 11i is available on both PA-RISC systems and Itanium-based systems.

Each HP-UX 11i release has an associated release name and release identifier. The uname (1) command with the -r option returns the release identifier. The following table shows the releases available for HP-UX 11i.

Table 1. HP-UX 11i releases

Release name	Release identifier	Supported processor architecture
HP-UX 11i v1	B.11.11	PA-RISC
HP-UX 11i v1.5	B.11.20	Intel® Itanium
HP-UX 11i v1.6	B.11.22	Intel Itanium
HP-UX 11i v2	B.11.23	Intel Itanium

05/03

5981-7111EN