12 Dynamically Loadable Kernel Modules
This chapter describes how to set up a single-user or multiuser system. The following topics are discussed:
Managing Dynamically Loadable Kernel Modules

This section presents the concepts and procedures which are necessary to understand, configure, and manage Dynamically Loadable Kernel Modules (DLKMs).

This section is divided into the following three topical sections:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLKM Concepts</td>
<td>This section provides an introduction to DLKM, important DLKM terms, and detailed technical DLKM concepts.</td>
</tr>
<tr>
<td>DLKM Tools</td>
<td>This section provides a summary of tools collectively known as the Kernel Configuration Tool Set which are used when installing, configuring, and managing DLKM modules.</td>
</tr>
<tr>
<td>DLKM Procedures</td>
<td>This sections presents the key DLKM procedures used in the three phases of managing DLKM modules: Preparation, Loading, and Maintenance.</td>
</tr>
</tbody>
</table>

This section focuses on configuring and managing loadable device drivers, as they constitute the majority of supported module types for HP-UX release 11.0 and later.

**NOTE**

The HP-UX kernel infrastructure provides the ability to dynamically load and unload DLKM drivers. While the base set of drivers shipped with HP-UX release 11i are not DLKM enabled, many Independent Software Vendors (ISVs) are coding DLKM enabled drivers for the hardware they provide.
Check the documentation that shipped with any 3rd-party drivers you have to determine if they are DLKM enabled.

**DLKM Concepts**

This section provides a conceptual overview of DLKM features and functionality by:

- defining DLKM at a high level
- explaining terms and concepts essential to understanding DLKM
- describing how DLKM modules are packaged in HP-UX
- identifying the types of kernel modules currently supported by DLKM
- describing the advantages of writing kernel modules in DLKM format
- examining DLKM module functions and configuration parameters

**What is DLKM?**

The *Dynamically Loadable Kernel Modules Infrastructure* is an HP-UX operating system feature that allows “DLKM Enabled” kernel modules to be dynamically loaded into, or unloaded from, the HP-UX kernel without having to re-link the entire kernel or reboot the system.

Previously, to install a new driver you had to edit the `system` file, run the `config` or `mk_kernel` commands to create a new kernel, shut down the system, and then bring the system back up before you could use the new driver.

The DLKM feature not only provides the infrastructure to load kernel modules into a running system, but it also allows a kernel module to be statically linked when rebuilding the kernel. Setting a flag in one of the DLKM module’s configuration files determines whether the module is to be configured as dynamically loadable or statically linked.
Important Terms and Concepts

The DLKM infrastructure allows kernel modules to be configured in a number of different ways. The following table considers the different ways a kernel module can be configured and loaded, and clearly defines each as a term. It also clarifies the relationship between each term as seen by the HP-UX kernel.

Table 12-2 Important Terms and Concepts

<table>
<thead>
<tr>
<th>Term / Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Module</td>
<td>A Kernel Module is a section of kernel code responsible for supporting a specific capability or feature. For example, file system types and device drivers are kernel modules. In the kernel configuration context, a kernel module may be viewed as an object that can be installed, removed, configured or built on a system, either statically or dynamically. There are two categories of kernel modules:</td>
</tr>
<tr>
<td></td>
<td>• Traditional Module</td>
</tr>
<tr>
<td></td>
<td>• Modularly-packaged Module</td>
</tr>
</tbody>
</table>

| Traditional Module | A Traditional Module is a Kernel Module whose configuration data has not been modularized and can only be statically linked to the kernel. In the kernel configuration context, configuration information about Traditional Modules is maintained in the shared master and system files, and can only be accessed upon booting a kernel in which they have been statically-configured. |
### Modularly-packaged Module

A Modularly-packaged Module is a Kernel Module whose configuration data has been modularized (not shared with other kernel modules), which is a pre-requisite for DLKM-enabling the Kernel Module.

In the kernel configuration context, this means that the module uses its own `master` and `system` files (as opposed to the shared `master` and `system` files in which Traditional Modules are configured).

In order to be classified as a Modularly-packaged Module, the module must contain its own `master` and `system` files, as well as an individual object file, `mod.o`, that implements the module.

A Modularly-packaged Module can be dynamically loaded into the HP-UX kernel only if that module includes the module wrapper code and additional data structures.

For this reason, we place Modularly-packaged Modules in two categories:

- **Static Modularly-packaged Modules**
- **Loadable Modules (or DLKM Modules)**

The terms Loadable Module and DLKM Module are interchangeable.

### Static Modularly-packaged Module

A Static Modularly-packaged Module is a Modularly-packaged Module that can only be linked statically to the kernel.

In the kernel configuration context, this means that the module uses its own `master` and `system` files but does not contain the module wrapper code and additional data structures that provide the dynamic loading and unloading ability.

---

**Table 12-2** Important Terms and Concepts (Continued)

<table>
<thead>
<tr>
<th>Term / Concept</th>
<th>Definition</th>
</tr>
</thead>
</table>
Dynamically Loadable Kernel Modules
Managing Dynamically Loadable Kernel Modules

Table 12-2 Important Terms and Concepts (Continued)

<table>
<thead>
<tr>
<th>Term / Concept</th>
<th>Definition</th>
</tr>
</thead>
</table>
| Loadable Module (DLKM Module) | A Loadable Module (or DLKM Module) is a Modularly-packaged Module with the capability to be dynamically loaded into a running kernel. In the kernel configuration context, this means that the DLKM module uses its own master and system files and contains the module wrapper code and additional data structures that provide the dynamic loading and unloading ability. However, when a DLKM module is written with self-contained module wrapper code and packaged with module-specific master and system files, it can still be statically-configured into the kernel. For this reason, we place Loadable Modules in two categories:  
  - *Statically-configured Loadable Module*  
  - *Dynamically-configured Loadable Module* |
| Statically-Configured Loadable Module | A Statically-configured Loadable Module is a DLKM module that has the capability to be dynamically loaded but instead is configured to be statically built into the kernel. In the kernel configuration context, this means that the module-specific system file was updated to indicate static configuration. Because it is now statically built into the kernel, it cannot be unloaded from or reloaded into loaded into the kernel dynamically. |
Managing Dynamically Loadable Kernel Modules

Table 12-2  Important Terms and Concepts (Continued)

<table>
<thead>
<tr>
<th>Term / Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamically-configured Loadable Module</td>
<td>A Dynamically-configured Loadable Module is a loadable module which has been fully configured to be dynamically loaded into or unloaded from the kernel without having to re-link the entire kernel or reboot the system.</td>
</tr>
<tr>
<td></td>
<td>To summarize the terminology presented in this table, a Dynamically-configured Kernel Module is all of the following:</td>
</tr>
<tr>
<td></td>
<td>• a Modularly-packaged Module (Which is a Kernel Module that uses module-specific master and system files.)</td>
</tr>
<tr>
<td></td>
<td>• a Loadable Module (or DLKM Module) (Which is a Modularly-packaged Module that contains the wrapper code and additional data structures and uses module-specific master and system files, but still could be configured as dynamic or statically-linked.)</td>
</tr>
<tr>
<td></td>
<td>• a Dynamically-configured Loadable Module (Which is a DLKM Module that has been configured to be fully capable of dynamic loading into, and unloading from the running kernel.)</td>
</tr>
<tr>
<td>Module Wrapper</td>
<td>The additional code and data structures added to a kernel module which enable the DLKM mechanism to logically connect and disconnect a loadable module to and from the running kernel.</td>
</tr>
</tbody>
</table>

DLKM Module Packaging

The DLKM infrastructure specifies that:

- a kernel module must be packaged modularly with at least:
  - its own master and system files
  - its own mod.o object file that implements only that module
the mod.o object file must contain the Module Wrapper code
(although full optimization is optional).

NOTE
See the master (4) manpage for descriptions of the two kinds of master
files, and the config (1M) manpage for a description of the traditional and
modular system files.

Kernel modules written as traditional modules are still fully supported
in HP-UX. Driver developers are encouraged to re-package their static
modules according to the module packaging architecture introduced with
DLKM modules.

DLKM Module Types
The DLKM feature currently supports the following types of kernel
modules:
- WSIO class drivers
- WSIO interface drivers
- STREAMS drivers
- STREAMS modules
- Miscellaneous modules—for example, modules containing support
  functions not required in the statically-configured kernel but shared
  among multiple loadable modules

DLKM Advantages
DLKM modules provide many advantages relative to static modules,
including:
- reducing time spent on device driver development by streamlining
  the driver installation process
- making it easier for administrators to install device drivers from
  other vendors
- improving system availability by allowing device drivers and other
  modules to be configured into the kernel while the system is running
- conserving system resources by unloading infrequently used modules
  when not in use
providing administrators with the ability to demand load and unload modules

providing the kernel with the ability to automatically load modules

Auto loading occurs when the kernel detects a particular loadable module is required to accomplish some task, but the module is not currently loaded. The kernel automatically loads the module.

**DLKM Driver Loading Concepts**

When a module is dynamically loaded, its object file is read from disk and loaded into newly allocated kernel memory. Once in memory, the module's symbols are relocated and any external references are resolved. Special code in the module is then executed to perform any required module-specific setup. Then the code specific to the module's type, if any, is executed, making the newly loaded module accessible to the rest of the kernel.

A module can be loaded in the following ways:

- **Demand Load**
  
  A demand load is a user level request for a specific module to be loaded. The load is accomplished through the `kmadmin` command.

- **Autoload Event**
  
  An autoload occurs when the kernel detects that a specific module is required to provide the functionality necessary to perform a task. The load is triggered by the initiation of the task. Once the required module is loaded, the task continues.

  A loadable module's `_load()` function performs any initialization tasks required by the module before the module is logically connected to the kernel. Typical initialization tasks include acquiring private memory for the module and initializing devices and data structures.

  If the module is unable to initialize itself, the `_load()` function must free any memory that it allocated and undo any other action that it took prior to the failure including canceling all outstanding calls to `timeout`.

**DLKM Driver Unloading Concepts**

When the functionality provided by a module is no longer needed the module can be unloaded, thus freeing its resources for later use.
When a module is unloaded, the code specific to the module's type, if any, is executed to disconnect the module from the kernel. Then, special code in the module is executed to perform any module-specific cleanup. Finally, the memory allocated to the module is freed.

A module may be unloaded only by a user level request specifying the module to be unloaded. The unload is accomplished through the `kmadmin` command. This request may fail for a number of reasons, the most common being that the module is busy at the time. An example of this would be attempting to unload a device while there are outstanding opens on the device.

A loadable module's `_unload()` function is called by the DLKM mechanism whenever the module is about to be removed from active memory. The function may be given any name (typically `module_name_unload`); a pointer to the `_unload()` function is obtained from the module's wrapper.

The module's `_unload()` function cleans up any resources that were allocated to the module, and it must remove all references to the module. Typical cleanup tasks include releasing private memory acquired by the module, removing device interrupts, disabling interrupts from the device, and canceling any outstanding timeout requests made by the module.

The module's `_unload()` function returns 0 on success and an `errno` value on failure. In the event of failure, the function leaves the module in a sane state, since the module will remain loaded after the return.

The system will never attempt to unload a module that it thinks is busy. However, the system cannot determine under all cases when the module is in use. Currently, a module is considered to be busy when another module that depends on it is also loaded. In addition, WSIO class drivers and STREAMS drivers track the `open()` and `close()` calls; these types of modules are busy whenever there is at least one open on the device using the driver. Under most other circumstances, the module determines for itself whether it is appropriate for it to be unloaded. When a module is still in use, its `_unload()` function returns a non-zero value to cancel the unload.

The argument passed to the `_unload()` function is the same type-specific value that was passed to the module's `_load()` function. The use of this argument is described in section “STREAMS Drivers”.

---

Dynamically Loadable Kernel Modules
Managing Dynamically Loadable Kernel Modules

- When a module is unloaded, the code specific to the module's type, if any, is executed to disconnect the module from the kernel. Then, special code in the module is executed to perform any module-specific cleanup. Finally, the memory allocated to the module is freed.

- A module may be unloaded only by a user level request specifying the module to be unloaded. The unload is accomplished through the `kmadmin` command. This request may fail for a number of reasons, the most common being that the module is busy at the time. An example of this would be attempting to unload a device while there are outstanding opens on the device.

A loadable module's `_unload()` function is called by the DLKM mechanism whenever the module is about to be removed from active memory. The function may be given any name (typically `module_name_unload`); a pointer to the `_unload()` function is obtained from the module's wrapper.

- The module's `_unload()` function cleans up any resources that were allocated to the module, and it must remove all references to the module. Typical cleanup tasks include releasing private memory acquired by the module, removing device interrupts, disabling interrupts from the device, and canceling any outstanding timeout requests made by the module.

- The module's `_unload()` function returns 0 on success and an `errno` value on failure. In the event of failure, the function leaves the module in a sane state, since the module will remain loaded after the return.

- The system will never attempt to unload a module that it thinks is busy. However, the system cannot determine under all cases when the module is in use. Currently, a module is considered to be busy when another module that depends on it is also loaded. In addition, WSIO class drivers and STREAMS drivers track the `open()` and `close()` calls; these types of modules are busy whenever there is at least one open on the device using the driver. Under most other circumstances, the module determines for itself whether it is appropriate for it to be unloaded. When a module is still in use, its `_unload()` function returns a non-zero value to cancel the unload.

- The argument passed to the `_unload()` function is the same type-specific value that was passed to the module's `_load()` function. The use of this argument is described in section “STREAMS Drivers”.

---
DLKM Driver Configuration Concepts

Since kernel modules written in the DLKM format can be configured as either dynamically loadable or statically-configured, DLKM-compatible device drivers must accommodate either configuration.

Through the use of configurable module attributes, System Administrators can control the various functions of a DLKM driver, including whether it is dynamically loaded or statically-configured.

This section provides attributes and keywords for:

- required components of a DLKM driver
- optional components of a DLKM driver

It also presents a brief description of STREAMS and Miscellaneous drivers. See the section DLKM Tools for detailed instructions on how to modify the configurable module attributes presented here.

NOTE

The system must be in a run-time state before dynamic module loading is available. Thus, kernel modules required during system boot must be configured as statically-configured.

master File Definition

Each DLKM module has its own master file. The format of the master file includes the following section keywords:

- $VERSION—indicates the version number for the file format. Version is defined as an integer and starts from one. A single line containing the only supported version (version 1) is entered.

- $LOADABLE—indicates that the module supports dynamic loading. If this section keyword does not exist, the module can only be statically-configured into the kernel.

- $INTERFACE—identifies the interface names and versions on which the module is built. For HP-UX, versions 11.0 and higher, a single line is entered containing the word base.

- $TYPE—indicates the module type and the type specific information. Valid types are wsio_class, wsio_intfc, streams_mod, streams_drv, and misc.
Other sections (if required)—$DRIVER_DEPENDENCY, $TUNABLE, and $DRIVER_INSTALL.

The $DRIVER_DEPENDENCY section, defines the names of all other modules that this module depends upon.

The $TUNABLE section defines the names and default values of the tunable parameters (variables) for the module. Default (and optionally minimum) values for tunable parameters are entered here.

The $DRIVER_INSTALL section defines the module’s name and associated block and/or character major device number(s).

**system File Definition**

Every DLKM module requires a system file. The system file includes the following three mandatory and one optional section keywords:

- **$VERSION**—indicates the version number for the file format. Version 1 is the only supported file-format.

  **NOTE**

  The version number for the master file and system file must be the same.

- **$CONFIGURE**—indicates if the module is to be configured into the system. If $CONFIGURE is Y or y, the module will be configured on the next build; if $CONFIGURE is N or n, the module will not be configured on the next build. kmsystem (1M) provides the interface to modify the flag.

- **$LOADABLE**—indicates how the module will be configured. If $LOADABLE is Y or y, the module will be configured as a Dynamically-configured Loadable Module; if $LOADABLE is N or n, the module will be statically configured into the kernel, requiring a reboot. kmsystem provides the interface to modify the flag.

  - If $CONFIGURE is N or n, $LOADABLE is ignored.

- **$TUNABLE (empty)**—place holder for any tunable parameter specified in the associated master file for which you want to specify a value other than the default value. Nothing is entered here.
kmtune (1M) is the interface to modify tunable parameters in the module's system description file and the HP-UX system file (/stand/system by default).

**Modstub.o File Definition**

An optional component, the `Modstub.o` file is statically-configured into the kernel as a “place holder” for functions implemented in a loadable module that will be loaded at a later time. Its purpose is to enable the kernel to resolve references to the absent module’s functions. Configuring a module that uses stubs requires a full kernel build so that the stubs can be statically linked to the kernel.

`Modstub.o` contains stubs for entry points defined in the associated loadable module that can be referenced by other statically-configured kernel modules currently configured in the system. Access to a stub causes the kernel to auto load the associated loadable module.

**space.h File Definition**

An optional component, the `space.h` file contains storage allocations and initialization of data structures associated with a DLKM module when the size or initial value of the data structures depend on configurable values such as tunable parameters. In order to communicate these values to the rest of the DLKM module, the values are stored in global variables and accessed by the module via external declarations in the module's `mod.o` file.

---

**NOTE**

All tunable parameters specified in the `master` file are defined as global variables in the `space.h` file.

---

**STREAMS Drivers**

Initialization of STREAMS drivers is very similar for both the loadable and statically-configured module cases. The only difference is that loadable drivers must use the `drv_info_t` structure that is passed as an argument to the `_load()` function.

STREAMS drivers, like WSIO class drivers, automatically track `open()` and `close()` system calls for the STREAMS device. The system will prevent a STREAMS driver from unloading whenever the device has one or more open file handles. Of course, the driver can still disallow an unload if this check is insufficient for its needs.
Miscellaneous Modules

Miscellaneous modules can implement any feature within the kernel. As such, a miscellaneous module's _load() function must address all of the module's specific needs. Similarly, the module's _unload() function must determine for itself if it is safe to unload. The system will not allow a module to be unloaded if other loaded modules are dependent upon the module. Other than this check, the system performs no other checks when the administrator attempts to remove a miscellaneous module from the kernel.

The argument to the _load() function is not meaningful and should be ignored.

DLKM Tools

There are a number of HP-UX commands known collectively as the kernel configuration tool set for installing, configuring, and managing DLKM modules. These commands are presented with descriptions and applicable command line options in this section.

Why you should use the kernel configuration tools instead of manually editing the system files

Although the HP-UX static kernel environment has not changed, it is affected by the configuration of kernel modules within the DLKM infrastructure. Specifically, DLKM requires that a kernel module have its own master and system files, and contain a Module Wrapper.

To the overall HP-UX kernel configuration environment this means:

1. The configurable module information is distributed among several files:
   - traditional modules use the /stand/system file
   - modularly-packaged modules use their own module-specific system file

2. The kernel structure is extended:
   - static kernel executable file /stand/vmunix
associated DLKM kernel components under /stand/dlkm:
  — kernel symbol table
  — dynamic loadable modules

Because of the effects that the DLKM infrastructure has on the overall kernel configuration environment, it is best to configure any type of kernel module using the tools described in this section.

Avoid editing the system file, or replacing the kernel file manually, as doing so increases the chance of introducing configuration errors.

For more detailed information regarding the master and system files, refer to the master (4) manpage and the config (1M) manpages.

Kernel Configuration Tools Description

The system administrator uses the kernel configuration tools to install, configure, load, unload, update, or remove kernel modules from the system; and to build new kernels. You can use the commands described in this tool set to configure kernel modules of any type (static or loadable).

The action carried out by a kernel configuration tool depends upon the options you specify during the tool’s invocation. This information is presented in the section Commands and Options in the Kernel Configuration Tool Set.

The following list describes the basic function of each of the commands that make up the kernel configuration tool set.

Tools For Building Static or Dynamic Kernels

- `kmsystem` (1M)

  Provides interface to set a module’s configurable attributes, to indicate whether a module should be configured, and whether it should be built as loadable or static.

- `kmtune` (1M)

  Provides interface to set the tunable parameters
Dynamically Loadable Kernel Modules

Managing Dynamically Loadable Kernel Modules

- *kmupdate* (1M)
  Updates the system with the newly built kernel and/or associated DLKM files.

**Tools That Provide an Interface to DLKM**

- *kminstall* (1M)
  Install, remove, or update a module's component files on a system

- *kmadmin* (1M)
  Provides general administrative interface for DLKM. Allows administrators to load, unload, and query loadable modules.

**Commands and Options in the Kernel Configuration Tool Set**

This section the command line options with descriptions for each of the kernel configuration tools.

**NOTE**

If you need further information regarding the functionality, usage, or command line options for any of the kernel configuration tools, refer to their respective manpages.

**Table 12-3 Kernel Configuration Tool Set**

<table>
<thead>
<tr>
<th>Tool/Command</th>
<th>Action</th>
</tr>
</thead>
</table>
| **config**   | • First form—generates both the static kernel and associated Dynamically-configured Loadable Modules; a system reboot is necessary.  
• Second form, `-m` option—generates the specified loadable module for use with the currently running kernel. The newly configured service is available immediately, without requiring a system reboot. |

### Table 12-3  Kernel Configuration Tool Set (Continued)

<table>
<thead>
<tr>
<th>Tool/Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmadmin</td>
<td>- <code>k</code> option—prints a list of all statically-configured modules in the running kernel.</td>
</tr>
<tr>
<td></td>
<td>- <code>L</code> option—loads the specified loadable module into the running kernel.</td>
</tr>
<tr>
<td></td>
<td>- <code>Q</code>, <code>-q</code> option—prints the status of the specified loadable module.</td>
</tr>
<tr>
<td></td>
<td>- <code>S</code>, <code>-s</code> option—prints the status of all currently loaded or registered loadable modules.</td>
</tr>
<tr>
<td></td>
<td>- <code>U</code>, <code>-u</code> option—unloads the specified loadable module from the running kernel.</td>
</tr>
<tr>
<td>kmininstall</td>
<td>- <code>a</code> option—adds a module’s component files to certain subdirectories of <code>/usr/conf</code> and <code>/stand</code>.</td>
</tr>
<tr>
<td></td>
<td>- <code>d</code> option—deletes a module’s component files from the subdirectories of <code>/usr/conf</code> and <code>/stand</code>.</td>
</tr>
<tr>
<td></td>
<td>- <code>u</code> option—copies a module’s <em>updated</em> component files into the subdirectories of <code>/usr/conf</code> and <code>/stand</code>.</td>
</tr>
</tbody>
</table>
### Table 12-3  Kernel Configuration Tool Set (Continued)

<table>
<thead>
<tr>
<th>Tool/Command</th>
<th>Action</th>
</tr>
</thead>
</table>
| **kmsystem**  | • `-c` option—assigns a value (Y or N) to the configuration (`$CONFIGURE`) flag of the specified module in preparation for the next system configuration.  
• `-l` option—assigns a value (Y or N) to the loadable (`$LOADABLE`) flag of the specified module in preparation for the next system configuration.  
• `-q` option—prints the values of the configuration and loadable flags of the specified module. Prints a “-” (signifies “does not apply”) for the loadable flag of a static module.  
• no options or `-S` option only—prints the values of the configuration and loadable flags of all modules. Prints a “-” for the loadable flags of static modules. |
| **kmtune**    | • `-l` option—prints the values of all system parameters.  
• `-q` option—queries the value of the specified system parameter.  
• `-r` option—resets the value of the specified parameter to its default value in preparation for the next system configuration.  
• `-s` option—assigns a value to the specified system parameter in preparation for the next system configuration. |
DLKM Procedures for Dynamically Configured Loadable Modules

This section provides detailed procedures for configuring, loading, and unloading DLKM Enabled kernel modules. Procedural information is shown in three different ways. The first two are summary formats and the third provides detailed procedure steps.

1. DLKM Procedural Flowchart

Use this chart as a reference to view all of the procedures and to determine the correct sequence in which to perform them.

2. Tables of Loadable Module Configuration and Management Procedures

These tables group the procedures into 3 phases: Preparing, Loading, and Maintaining procedures. There is one table for each Loadable Module type: Dynamically-configured and Statically-configured.

3. DLKM Procedures

This section presents step-by-step instructions for preparing, configuring, loading and unloading (or activating) loadable modules.

---

Table 12-3 Kernel Configuration Tool Set (Continued)

<table>
<thead>
<tr>
<th>Tool/Command</th>
<th>Action</th>
</tr>
</thead>
</table>
| kmupdate | • First form—prepares the system to move the specified static kernel and its associated files to the `/stand/vmunix` file and `/stand/dlkm` directory, respectively, during the next system shutdown and startup.  
• Second form, `-M` option—moves the configured image of the specified loadable module to the location where the DLKM loader can find it, and registers the module with the kernel either (1) immediately or (2) later at system shutdown. |
The detailed procedure steps are presented in two sections:

a. Dynamically-configured Loadable Module Procedures

b. Statically-configured Loadable Module Procedures
Dynamically Loadable Kernel Modules
Managing Dynamically Loadable Kernel Modules

Figure 12-1  DLKM Procedural Flowchart

Start

Dynamically-configured
Loadable Module

Dynamically-configured
Loadable Module

Statically-configured
Loadable Module

Prepare module as Dynamically
Configured Loadable Module using
the command: kmsystem -c Y -l Y

Prepare module as Statically
Configured Loadable Module using
the command: kmsystem -c Y -l N

OPTIONAL: Tune system parameter(s) supplied by
module or static kernel using the command: kmtnue -s

Configure loadable module into
system using command: config -M

Move loadable module’s image into
place and register module using
command: kmupdate -M

If necessary, create device special
file(s) for loadable module using
command: mknod [NTA] To manage

Load loadable module using
command: kmadmin -L

OPTIONAL: Query loadable module
using command kmadmin -q

OPTIONAL: Unload loadable module
using command: kmadmin -U

OPTIONAL: Remove module’s
components from system using
command: kminstall -d

Configure statically linked module into
system by building new kernel using
command: config /stand/system

Prepare system to move new kernel into
place during next system shutdown and
startup using command:
kmupdate /stand/build/vmunix_test

Activate statically linked module by
booting new kernel using command:
shutdown -r

OPTIONAL: Query statically linked
module using command:
kmadmin -k

OPTIONAL: Create device special
file(s) for statically linked module using
command: mknod

Move loadable module’s image into
place and register module using
command: kmupdate -M

Load loadable module using
command:

OPTIONAL: Tune system parameter(s) supplied by
module or static kernel using the command:
kmtune -s

Done

522  Chapter 12
### Table 12-4  Dynamically-configured Loadable Module Procedures

<table>
<thead>
<tr>
<th>Phase</th>
<th>Configuration Option</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparing</td>
<td>Prepare Loadable Module as a Dynamically-configured Module</td>
<td>Prepare a loadable module for dynamic loading into the HP-UX kernel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional: Query and/or Tune the system parameters supplied by a loadable module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configure a loadable module for dynamic loading</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Register a Dynamically-configured Loadable Module with the kernel</td>
</tr>
<tr>
<td>Loading</td>
<td>Demand-Load</td>
<td>Load a Dynamically-configured Loadable Module into the kernel</td>
</tr>
<tr>
<td>Maintaining</td>
<td>Unload</td>
<td>Unload a Dynamically-configured Loadable Module</td>
</tr>
<tr>
<td></td>
<td>Tune</td>
<td>Tune a Dynamically-configured Loadable Module</td>
</tr>
<tr>
<td></td>
<td>Update a module</td>
<td>Update a Dynamically-configured Loadable Module’s image</td>
</tr>
<tr>
<td></td>
<td>Query a module</td>
<td>Determine which Dynamically-configured Loadable Modules are currently loaded</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Obtain information about a loaded Dynamically-configured Loadable Modules</td>
</tr>
</tbody>
</table>
Table 12-5  Statically-configured Loadable Modules Procedures

<table>
<thead>
<tr>
<th>Phase</th>
<th>Configuration Option</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparing</td>
<td>Prepare Loadable Module as a Statically-configured Module</td>
<td>Prepare a loadable module for static linking to the HP-UX kernel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional: Query and/or Tune the system parameters for a Statically-configured Module present in the Static Kernel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configure Kernel to include Statically-configured Module</td>
</tr>
<tr>
<td>Loading</td>
<td>Activate a Statically-configured Module</td>
<td>Activate a Statically-configured Module by rebooting</td>
</tr>
<tr>
<td>Maintaining</td>
<td>Tune a module</td>
<td>Tune a loadable module</td>
</tr>
<tr>
<td></td>
<td>Query a module</td>
<td>Determine which Statically-configured Module are currently loaded</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Obtain information about a currently loaded Statically-configured Module</td>
</tr>
</tbody>
</table>

All DLKM modules that are required to boot the kernel must be configured as *statically* configured modules.

If the module you are configuring is required to boot the kernel, refer to the configuration procedure in the section *Statically-configured Loadable Modules*.
How to prepare a loadable module for dynamic loading into the HP-UX kernel

Use the `kmsystem` command to assign values (Y or N) to the configuration ($CONFIGURATION) and loadable ($LOADABLE) flags in the module’s system description file. If the loadable flag is not present in the system description file and you attempt to assign it a value, `kmsystem` exits with an error.

You can use the `kmsystem` command to prepare a DLKM module for configuration as either (1) *dynamically-configured* or (2) *statically-configured*.

To prepare a loadable module to be dynamically loaded into the kernel, do the following:

**Step 1.** Execute this `kmsystem` command:

```
/usr/sbin/kmsystem -c Y -l Y module_name
```

How to query and tune the system parameters supplied by a loadable module

Use the `kmtune` command to query, set, or reset system (tunable) parameters used by the DLKM module or the static kernel. `kmtune` reads the master configuration files, the system description files, and the HP-UX system file.

For a Modularly packaged Module, `kmtune` writes any user-modified system parameter to the module’s system description file. For a Traditionally-packaged module using pre-11.0 module packaging, `kmtune` writes any user-modified system parameter to the HP-UX system file.

**Step 1.** To query the value of a specific system parameter, execute this `kmtune` command:

```
/usr/sbin/kmtune -q system_parameter_name
```

**Step 2.** To set the value of a specific system parameter, execute this `kmtune` command:

```
/usr/sbin/kmtune -s system_parameter_name=value
```

**Step 3.** To reset the value of a system parameter to its default value, execute this `kmtune` command:

```
/usr/sbin/kmtune -r system_parameter_name
```
At this point, you have set the values of the module’s system parameters for the next module configuration. The values of the system parameters supplied by the module will become effective with the running kernel after the loadable module is configured and registered (see procedures on following page).

**How to configure a loadable module for dynamic loading**

Upon completing the configuration procedure shown here, the dynamically-configured loadable module will be ready to load *immediately*, meaning that you do not have to wait for a reboot to be able to load it.

**Step 1.** To configure a loadable module for dynamic loading, execute this `config` command:

```
/usr/sbin/config -M module_name -u
```

This results in the generation of a loadable image. The `-u` option forces `config` to call the `kmupdate` command, which causes the system to move the newly generated image into place and register it with the running kernel.

**How to register a dynamically-configured loadable module with the HP-UX kernel.**

For a DLKM module configured as dynamically loadable, you use the `kmupdate` command to update its image and register it with the kernel. Updating a dynamically-configured loadable module’s image means moving its image into place and registering it with the kernel either (1) immediately or (2) later at system shutdown.

Call `kmupdate` after first calling `config`. If you include the `-u` option in the `config` invocation, there is no need to invoke `kmupdate`. The `config -M -u` command automatically invokes `kmupdate`.

**Step 1.** To update the image of a dynamically-configured loadable module *immediately*, execute this `kmupdate` command:

```
/usr/sbin/kmupdate -M module_name -i
```

After updating the specified module and assuming the module was loaded originally, `kmupdate` will reload the module before exiting.

**Step 2.** To update the image of a dynamically-configured loadable module *at system shutdown*, execute the following `kmupdate` command:

```
/usr/sbin/kmupdate -M module_name -a
```
If you do not specify the \texttt{-i} or \texttt{-a} option, \texttt{kmupdate} will attempt to update the specified loadable module immediately. If the module cannot be updated immediately (for example, the current module is in use and cannot be unloaded), the module will be updated at system shutdown.

To load a dynamically-configured loadable module, you use the \texttt{-L} option of the \texttt{kmadmin} command. The load operation initiated by the \texttt{kmadmin} \texttt{-L} command performs all tasks associated with link editing the module to the running kernel and making the module accessible to the system. Specifically, the load operation performs the following tasks:

- checks what other modules the loadable module depends upon and automatically loads any such module that is not currently loaded
- allocates space in active memory for the specified loadable module
- loads the specified loadable module from the disk and link-edits it into the running kernel
- relocates the loadable module's symbols and resolves any references the module makes to external symbols
- calls the module's \texttt{_load()} entry point to do any module-specific initialization and setup
- logically connects the module to the rest of the kernel, which is often accomplished with the help of module type-specific installation functions accessed through the module's wrapper code

\textbf{Step 1.} To load a dynamically-configured loadable module into the running kernel, execute the following \texttt{kmadmin} command:

\begin{verbatim}
/usr/sbin/kmadmin -L module_name
\end{verbatim}

When the loading completes, an identifier (ID) number prints on the standard output to identify the module that was loaded.

If you want the system to automatically load certain dynamically-configured loadable modules immediately after every system reboot, add the names of the modules to the \texttt{/etc/loadmods} file. At boot time, the \texttt{/sbin/init.d/kminit} script will execute the \texttt{kmadmin} command and load the modules listed in \texttt{/etc/loadmods}. 

Dynamically Loadable Kernel Modules
Managing Dynamically Loadable Kernel Modules

How to unload a dynamically-configured loadable module

Use the \(-U\) or \(-u\) option of the \texttt{kmadmin} command to unload a DLKM module configured as dynamically loadable. You have the choice of unloading the module by its name or its ID number.

The unloading operation logically disconnects the module from the running kernel and calls the module's \_\texttt{unload}() entry point to perform any module-specific cleanup including:

1. canceling all outstanding calls to \texttt{timeout()}
2. disabling device interrupts
3. freeing all active memory allocated to the specified loadable module

\textbf{Step 1.} To unload a dynamically-configured loadable module by name, execute this \texttt{kmadmin} command:

\texttt{/usr/sbin/kmadmin -U module_name}

\textbf{Step 2.} To unload a dynamically-configured loadable module by ID number, execute this \texttt{kmadmin} command:

\texttt{/usr/sbin/kmadmin -u module_id}

How to determine which modules are currently loaded

Use the \(-S\) or \(-s\) option of the \texttt{kmadmin} command to view detailed information about all current registered DLKM module.

\textbf{Step 1.} To print the full status for all dynamically-configured loadable modules currently registered, execute this \texttt{kmadmin} command:

\texttt{/usr/sbin/kmadmin -S}

\textbf{Step 2.} To print the brief status for all dynamically-configured loadable modules currently loaded, execute this \texttt{kmadmin} command:

\texttt{/usr/sbin/kmadmin -s}

\textbf{Step 3.} To print a list of all statically-configured modules, execute the following \texttt{kmadmin} command:

\texttt{/usr/sbin/kmadmin -k}

How to obtain information about a loaded module

Use the \(-Q\) or \(-q\) option of the \texttt{kmadmin} command to view detailed information about the DLKM module. For a DLKM module configured as dynamically loadable, you have the choice of displaying information for the module by its name or ID number.
Step 1. To display a dynamically-configured loadable module’s status by name, execute this kmadmin command:

```
/usr/sbin/kmadmin -Q module_name
```

Step 2. To display a dynamically-configured loadable module’s status by ID, execute the following kmadmin command:

```
/usr/sbin/kmadmin -q module_id
```

Depending on the type of module, information on the module’s block major number, character major number, and flags may also be printed.

Information returned by the -Q and -q options includes:

- the module’s name
- the module’s ID
- the module’s pathname to its object file on disk
- the module’s status (LOADED or UNLOADED)
- the module’s size
- the module’s virtual load address
- the memory size of Block Started by Symbol (BSS) (the memory size of the un-initialized space of the data segment of the module’s object file)
- the base address of BSS
- the module’s reference or hold count (the number of processes that are currently using the module)
- the module’s dependent count (the number of modules that currently depend upon this module being loaded; depended upon modules are specified in the $DRIVER_DEPENDENCY section of the module’s file)
- the module’s unload delay value (currently not used—always 0 seconds)
- the module’s descriptive name
- the type of module (WSIO, STREAMS, or Misc)
DLKM Procedures for Statically Configured Loadable Modules

How to prepare a loadable module for static linking
You can use the `kmsystem` command to prepare a DLKM module for configuration as either (1) *dynamically loadable* or (2) *statically-configured*.

Use the `kmsystem` command to assign values (Y or N) to the configuration ($CONFIGURE) and loadable ($LOADABLE) flags in the module's system description file. If the loadable flag is not present in the system description file and you attempt to assign it a value, `kmsystem` exits with an error.

**Step 1.** To prepare a DLKM module for static linking to the HP-UX kernel, execute this `kmsystem` command:

```
/usr/sbin/kmsystem -c Y -l N module_name
```

How to query and tune the system parameters for a statically-configured loadable module present in the static kernel
Use the `kmtune` command to query, set, or reset system (tunable) parameters used by the DLKM module or the static kernel. `kmtune` reads the master configuration files, the system description files, and the HP-UX system file.

For a Modularly-packaged module or a Traditionally-packaged module using 11.0 module packaging, `kmtune` writes any user-modified system parameter to the module's system description file. For a Traditionally-packaged module using pre-11.0 module packaging, `kmtune` writes any user-modified system parameter to the HP-UX system file.

To query the value of a specific system parameter, do the following:

**Step 1.** Execute this `kmtune` command:

```
/usr/sbin/kmtune -q system_parameter_name
```

**Step 2.** To set the value of a specific system parameter, execute this `kmtune` command:

```
/usr/sbin/kmtune -s system_parameter_name=value
```

**Step 3.** To reset the value of a system parameter to its default value, execute this `kmtune` command:

```
/usr/sbin/kmtune -r system_parameter_name
```
At this point you have set the values of system parameters that will take effect after the next whole HP-UX kernel configuration, update and system reboot (see procedures below).

You can use the `config` command to configure a DLKM module into the system as either dynamically loadable or statically-configured. *Use this procedure to statically link the DLKM module to a new kernel.*

To configure the HP-UX kernel to include a statically-configured loadable module, do the following:

**Step 1.** Execute this `config` command:

```
/usr/sbin/config -u /stand/system
```

`config` builds a new kernel. The `-u` option forces `config` to call the `kmupdate` command, which causes the system to perform the following actions when you shutdown and restart the system:

- a. save the existing kernel file and its kernel function set directory as `/stand/vmunix.prev` and `/stand/dlkm.vmunix.prev`, respectively
- b. move the newly generated kernel file and its kernel function set directory to their default locations, `/stand/vmunix` and `/stand/dlkm`, respectively

After the system reboots, your DLKM module will be available as statically-configured in the new running kernel.