

Kernel Logging in HP-UX 11i Version 1.5



Manufacturing Part Number:

April 30, 2001

© Copyright 2001 © Hewlett-Packard Company. All rights reserved..

1 Kernel Logging

What is Kernel Logging?

Kernel Logging is an infrastructure for tracking and logging specified behaviors during HP-UX system operation. It enhances the high availability of the operating system by providing enough kernel trace information to enable a system administrator or other troubleshooter to diagnose the root cause of a system problem without needing to reboot the system or reproduce the problem.

Kernel Logging has two main parts: an instrumented kernel, and a framework of tools to configure and access the information that is logged. Likewise, this document has two main sections:

- ❑ “Instrumenting Kernel Subsystems for Kernel Logging” on page 4, is intended for kernel developers.
- ❑ “Using Kernel Logging” on page 10, is intended for system administrators and other troubleshooters.

Instrumenting Kernel Subsystems for Kernel Logging

Although the KL (Kernel Logging) infrastructure is maintained by the KL team (email: KL_team_NJ@fpk.hp.com), it makes sense that the instrumentation of specific kernel subsystems be accomplished by those who know them best: the subsystem owners.

Instrumenting a kernel subsystem for kernel logging comprises the three main tasks listed here. Each task is explained in the sections that follow.

- ❑ Ascertain that an appropriate ID is defined for your subsystem
- ❑ Create a local header file for event IDs specific to your subsystem
- ❑ Add KL instrumentation points to your subsystem

Ascertain That an Appropriate ID is Defined for Your Subsystem

A subsystem ID is a unique string that identifies your subsystem to the KL infrastructure. It must be unique among the subsystem IDs, and evaluate to an integer in the range [513-1023]. When it is defined, as described in this procedure, your subsystem will be assigned the next available integer.

Several subsystem IDs have already been defined: they are listed, under the comment `/* KL supported subsystems */`, in the file `/ux/core/kern/common/sys/subsys_id.h`. For example:

```
/* KL supported subsystems */
#define KL_FORMATTER          512

/* The values for KL subsystems must be within 513 and 1023 */
/* since 512 value is reserved for the KL formatter */
#define KL_VM                 513
#define KL_PKM               514
#define KL_DLKM              515
#define KL_PM                516
#define KL_VFS               517
#define KL_VXFS              518
etc.
```

- Step 1.** Send mail to the KL team (KL_team_NJ@fpk.hp.com) describing the code to be instrumented and discuss with them what the ID for your subsystem should be.
- Step 2.** If you and the KL team decide to use a pre-existing subsystem ID for your instrumentation work, you are done with this task.

Step 3. If you and the KL team determine that a new subsystem ID must be defined--because there is no pre-existing subsystem ID suitable for your code, or because the pre-existing subsystem IDs are too general (for example, in a huge subsystem like VM, a subsystem ID that further subdivides it might be very useful)--then the

following source files must be modified to support the new subsystem ID:

- `/ux/core/kern/common/sys/subsys_id.h` -- this file must include a line for your new subsystem like the lines shown in the example in the introduction to this procedure.
- `/ux/core/lan/src/NETTRACELOG/conf/NETTL-MIN/nettlgen.conf` -- this file must include a line for your new subsystem like the following sample line taken from the section of the file captioned # Subsystems supported by the Kernel Logging facility:

```
SS:513:KL_VM:8:k:NULL:klfmt:NULL:NULL:Kernel Logging
```

Clear explanations of the fields in this entry are given in the section of notes captioned # SUBSYSTEM RECORD, earlier in the file.

Step 4. Copy the modified `nettlgen.conf` file to the `/etc` directory on your target machine.

Step 5. Before you submit these modified files to the final product, notify the KL team so a final review can be done.

Create a Local Header File for Kernel Events Specific to Your Subsystem

You will be adding instrumentation points to your kernel subsystem code wherever you want to identify an event to be logged. Each such point must be given an “event ID,” which is a string that identifies that event to the KL infrastructure. Each must be unique among the event IDs for your subsystem, and is assigned an integer value in the range [1-65535]. You, not the KL team, assign integer values to event IDs in your subsystem.

Step 1. Add the following line to both your local header file (if you have one) and your `.c` file:

```
#include <sys/net_diag.h>
```

Step 2. Construct an event ID for each instrumentation point in your subsystem. Event IDs are constructed according to the following convention:

```
subsystemID_eventName
```

where *subsystemID* is the subsystem identifier you and the KL team previously agreed upon, and *eventName* is an identifier for the particular event you are instrumenting. For example, the Dynamic Tuning subsystem has the ID `KL_DYNTUNE`. In that subsystem, one of the kernel events is setting a tunable

parameter. The kernel developer has assigned the name `SETTUNE_EVENT` to this event. Thus, the event ID for this kernel event becomes

```
KL_DYNTUNE_SETTUNE_EVENT
```

Step 3. Add your event IDs to your local header file, or to your `.c` file, and assign each an integer value. It is good practice to keep all `#define` statements of a particular subsystem in one file, typically in a local header file. However, if you do not have a local header file, you can define event IDs in your `.c` file.

Use the following format:

```
eventID=n
```

where *eventID* is constructed as described in Step 2, and *n* is the unique value assigned to this event.

In addition to being unique to this event ID in this subsystem, the assigned integer value *n* also serves to specify the “event class” of the event. Four event classes for Kernel Logging are defined in the `subsys_id.h` file. The following table lists and defines them, and identifies the range of integer values that can be assigned in each class:

Table 1-1

Event Class	Definition	Recommended Integer Values
DISASTER	The DISASTER class signals an event or condition which affects the operation of an entire subsystem, or the kernel, causing several programs to fail or the machine to panic	1000 - 1999
ERROR	The ERROR class signals an event or condition which does not affect the overall operation of an entire subsystem, but may cause an application to fail.	2000 - 2999
WARNING	The WARNING class indicates an abnormal event, possibly caused by problems in an individual subsystem.	3000 - 4999
INFORMATIVE	The INFORMATIVE class describes important routine operations and current system values.	5000 and up

Continuing the example of the Dynamic Tuning subsystem, and assuming that the developer had decided the event is an INFORMATIVE event, and that it is the first event of that class in the subsystem, the entry in the local header file for this event would be as follows:

```
KL_DYNTUNE_SETTUNE_EVENT=5000
```

Add KL Instrumentation Points to Your Subsystem

The three macros listed in the table below are the entry points to the Kernel Logging infrastructure. (These macros represent Kernel Logging Instrumentation Points.) Use them to record the dynamic parts of the user-friendly messages that will be logged from your subsystem. User-friendly messages are discussed in the next section of this document.

Table 1-2

Macro Name	Description
KL_LOG_INFO ()	This macro passes up to five long integers as user-defined data.
KL_LOG_STR ()	This macro passes a string as user-defined data. It can also be used if no user-defined data is supplied; in such a case the pointer to a string should be NULL.
KL_LOG_DATA ()	This macro passes a data structure as user-defined data. This information is dumped in hexadecimal by the Kernel Logging infrastructure. For information on how to format it for readability, see <code>netfmt(1M)</code> or the <i>Device Drivers Guide</i> .

Step 1. Select the point in your subsystem code to be instrumented for Kernel Logging.

IMPORTANT Do not instrument `MALLOC` and `FREE` macros: They are used in the KL infrastructure itself, and recursion will occur if you instrument them.

Step 2. Select the KL macro you want to call, based upon the specific type of information you want to pass to the kernel log. (See the manual page at the end of this document for complete information on the use of the macros.)

Step 3. Construct a user-friendly message for this instrumentation point, and add it to the message catalog file. (See the following procedure for details on doing this.)

Construct a User-Friendly Message to be Associated with an Instrumentation Point

The data passed to the KL infrastructure can be combined with a user-friendly message describing the event and its implications. These user-friendly messages are composed, stored, and built by you, in a message catalog file. The steps to complete this task are given below. But first, a couple of examples.

Suppose a kernel developer instruments some part of the VFS subsystem (subsystem ID `KL_VFS`) and wants to log a message when there is something wrong with a file whose name is specified, for example, through an `open()` system call. The developer wants the full message about the event to be “Could not open `foo.bar` file due to I/O related error.” Such a message has a static part: “Could not open ... file due to I/O related error”; and a dynamic part: *filename*, in this case, `foo.bar`. To avoid consuming memory, the static part of the message is not logged in the kernel. The dynamic part, *filename*, is logged, however, via the `KL_LOG_STR` macro with a string pointer pointing to the `foo.bar` string. At the same time, the following string is added to the local header file as the static part of the message associated with this instrumentation point: `Could not open %s file due to I/O related error`.

Or, suppose a kernel developer wants to log the following message: “I/O related error while accessing file, inode is 505, errno is 22.” In this case, the instrumentation point uses the `KL_LOG_INFO` macro and the user-friendly message is: `I/O related error while accessing file, inode is %lld, errno is %lld`.

NOTE Because the Kernel Logging infrastructure accepts 64-bit integers and the decoder of the KL files is a 32-bit command, use `%lld` (or `%llx`, `%llu`, etc.) templates to display the integers.

Step 1. Compose the user-friendly message that will be logged when this instrumentation point is reached by a running system.

Step 2. Edit the message catalog file,

`/us/core/lan/src/NETTRACELOG/format/klfmt.msg`, on the view server and add your message(s). The format of this file is as follows:

```
$setN subsystem_ID
n message
.
.
.
```

where *N* is the number and *subsystem_ID* is the ID assigned to your kernel subsystem by the KL team; *n* is the number you have assigned to an event ID, and *message* is the user-friendly message you have composed for that event.

The `$setn ...` line signals the beginning a new set of messages, and only needs to be added if `subsystem_ID` is a new one.

Step 3. Add a new `n message` line for each new instrumentation point you have introduced. The following example messages are from the Machine Check (KL_MC) subsystem;

```
$set525 KL_MC
1000 Machine Check Abort (MCA occurred)
2000 Corrected Machine Check (CMC) occurred
3000 System Abstraction Layer (SAL) returned unexpected Error Record (%s)
3001 Memory SBE address FIFO is full. Address 0x%01611x cannot be queued
5000 Error Record obtained from System Abstraction Layer (SAL)
6000 Bus check occurred. check info: 0x%01611x req: 0x1611x res: 0x%1611x tgt: 0x%01611x \
ip: 0x%01611x
```

Step 4. Be in the `/ux/core/lan/src/NETTRACELOG/format` directory, and issue the following command to build the message catalog:

```
clearmake all
```

When the compilation is complete, the KL message catalog file, `/ux/core/lan/obj/locales/NETTRACE/format/klfmt.cat`, will include your messages.

Step 5. Copy the `klfmt.cat` file to the following directory on the target machine:

```
/usr/lib/nls/C
```

Using Kernel Logging

Kernel Logging contributes to the high-availability of your system by giving system administrators and other troubleshooters the ability to collect the information necessary to diagnose problems in the HP-UX kernel while the system is still running. The cause of a system problem can often be tracked down without having to try to reproduce the problem or reboot the system. This is possible because, working in the background, while the system is running, KL logs messages detailing certain events that have taken place. The contents of the log are available to you at any time.

These three commands enable you to use and administer Kernel Logging:

<code>kl</code>	The <code>kl</code> command configures, and monitors the status of, the Kernel Logging infrastructure itself.
<code>netfmt</code>	The <code>netfmt</code> command formats the KL log files (which are binary) and allows you to get a human-readable report.
<code>nettlconf</code>	The <code>nettlconf</code> command configures the Kernel Logging subsystem database (allows you to control what gets logged).

Using `kl` to Enable, Disable, and Monitor Kernel Logging

The `kl` command controls the Kernel Logging infrastructure. It allows you to start and stop logging and determine its current status. `kl` can be used to specify the levels of events to be logged and the kernel subsystems that will write messages to memory or disk. `kl` also can also be used to manage the contents of the logfile in memory and on disk.

Enabling KL on a Running System

Step 1. Enable the KL infrastructure by executing the following command:

```
#kl -e
```

Note that, if the KL infrastructure is already enabled, reissuing this command does nothing and an error message (which may be disregarded) is issued.

Disabling KL on a Running System

Step 1. Disable the KL infrastructure by executing the following command:

```
#kl -d
```

Obtaining Configuration and Status Information About KL

You can obtain information, in the form of a report, about the status of the KL infrastructure. A typical report looks similar to this:

```
# kl -i
Kernel Logging Information:
Kernel Logging:                ON
Kernel Logging Disk Writer:    ON
Kernel Logging Picture (Snapshot): OFF
Current Queue Size:           100010485
Number of Messages Queued:    0
Log File name:                 /var/adm/kl.KLOG0
Log File size:                 1048576 (minimum: 8192)
Subsystem Name:                Log Class:
KL_VM      ( 513)              DISASTER
KL_PKM     ( 514)              DISASTER
KL_DLKM    ( 515)              DISASTER
KL_PM      ( 516)              DISASTER
KL_VFS     ( 517)              DISASTER
KL_VXFS    ( 518)              DISASTER
etc.
#
```

Step 1. Obtain a status report on the KL infrastructure itself by executing the following command:

```
#kl -i
```

Using kl to Temporarily Reconfigure Kernel Logging

The file `/etc/nettlgen.conf` contains the default values for

- the level of messages that will be logged
- the subsystems from which that level of messages will be accepted
- the maximum size of the log file
- the maximum number of individual messages that will be logged

The only way these defaults can be permanently changed for your system is with the `nettlconf` command (see “Using `nettlconf` to Reconfigure the Default Level of Message Logging for a Kernel Subsystem” on page 13).

However, you can temporarily set new values for these aspects of KL, as described in this procedure.

Step 1. Temporarily set the appropriate level of message logging for a specific subsystem(s) by executing the following command:

```
#kl -l level subsystem_ID ...
```

For example, the following command sets *level* to INFORMATION for the KL_DLKM subsystem: `#kl -l i KL_DLKM`.

That means that messages of classes DISASTER, ERROR, WARNING, and INFORMATIVE, coming from the KL_DLKM subsystem, will now be logged. Classes of messages coming from other subsystems remain unchanged.

Complete definitions of *level* and *subsystem_ID* are given on the `kl(1M)` manpage.

- Step 2.** Temporarily set the size of the file used to store logged messages when write-to-disk is enabled, by executing the following command:

```
#kl -s fsize
```

The value of *fsize* is an integer in the range [8KB-1GB], optionally followed by the letter *k*, *m*, or *g*, to explicitly specify kilobytes, megabytes, or gigabytes (the default is bytes). *fsize* should be as large as your system can handle.

- Step 3.** Temporarily set the number of individual messages that will be stored in the log file by executing the following command:

```
#kl -q qsize
```

qsize must be an integer in the range [100-10000] representing the number of messages that can be logged. *qsize* should be as large as your system can handle.

- Step 4.** To restore the default values (as listed in the file `/etc/nettlgen.conf`) for any parameters you have temporarily changed:

- a. Disable Kernel Logging (`kl -d`).
- b. Enable Kernel Logging (`kl -e`).

Using `kl` and `netfmt` to Obtain and Format KL Reports

Taking a Picture of the KL Logfile

Taking an isolated picture (a snapshot) of what's in memory and writing it out to disk is useful when you don't want to use system resources by having KL write to disk full-time.

- Step 1.** Take a picture of the KL logfile and save it in *filename*, by executing the following command:

```
#kl -p filename
```

This causes all messages in memory to be dumped to *filename* and removed from memory.

Obtaining Human-readable KL Reports

The KL logfile is a binary file. Before you can display or print logfile data in human-readable form, it must be formatted by the `netfmt` command. The formatted data is written to standard output by default, or to *filename*, if the `-f` option is specified.

Step 1. Obtain human-readable output of the collected KL messages by executing the following command:

```
# netfmt -k -v -f filename
```

The following is an example of the output of this command:

```
# netfmt -k -v -f kl.KLOG0
      Time           CPU      SS   PID   TID  CL  ID
-----@##%
06/21/2000 19:18:53.969947    0    KL_MC   -1   -1   E  /ux/core/kern/em/svc/cmc.c: 441
  0: 00 00 00 01 00 00 00 00 e0 00 00 00 00 17 a4 b8 .....
 16: 00 00 00 00 00 00 00 ff f6 83 55 ac e3 c8 97 db 10 .....U.....
 32: 00 00 00 00 00 00 00 ff f6 00 00 00 00 00 00 00 1f .....
-----@##%
06/21/2000 19:19:02.553831    0    KL_MC   -1   -1   E  /ux/core/kern/em/svc/cmc.c: 441
  0: 00 00 00 01 00 00 00 00 e0 00 00 00 00 17 a4 b8 .....
 16: 00 00 00 00 00 00 00 ff f6 83 55 ac e3 c8 97 db 10 .....U.....
 32: 00 00 00 00 00 00 00 ff f6 00 00 00 00 00 00 00 1f .....

===== Kernel Log file Summary =====
Node: hydra
HP-UX Version: B.11.20 A   Machine Type: ia64

Total number of messages: 2
Messages dropped: 0       Data dropped(bytes): 0

First Message                Last Message
Time: 19:18:53.969947        Time: 19:19:02.553831
Date: 06/21/00                Date: 06/21/00

Message distribution:
Disaster: 0                  Error: 2
Warning: 0                   Informative: 0

~~~~~Message distribution by Subsystem~~~~~

Subsystem Name: KL_MC        Group Name: Kernel Logging
Disaster: 0                  Error: 2
Warning: 0                   Informative: 0
```

Using `nettlconf` to Reconfigure the Default Level of Message Logging for a Kernel Subsystem

The `nettlconf` command is used primarily by developers to configure Kernel Logging for

their product or kernel subsystem during installation. However, a system administrator or other users with appropriate privileges can use it to reconfigure the default level of messages that are logged for a given subsystem. Refer to the `nettlconf(1M)` manpage for complete information on using this command.

NOTE It is not recommended that `nettlconf` be used to change any other aspects of the Kernel Logging infrastructure specified in the `/etc/nettlgen.conf` file. If this file becomes corrupted, the `kl` and `netfmt` commands will not function.

Using SAM to Administer Kernel Logging

Many of the system administrator's tasks discussed in the previous sections can be accomplished with SAM (System Administration Manager), a tool that provides an easy-to-use graphical interface for system setup and other administrative tasks. If you have appropriate privileges, or are logged in as `root`, you can access SAM by entering the command

```
/usr/sbin/sam
```

Once SAM is up and running, the KL administrative interface can be reached by the following path: select Kernel Configuration -> Configurable Parameters -> Actions -> Kernel Logging.

SAM has online help to assist you. You can also refer to the chapter "Using System Administration Manager" in "Managing Systems and Workgroups: A Guide for HP-UX System Administrators" for complete information on its use.

Kernel Logging Manual Pages

NAME

KL_LOG_INFO(), KL_LOG_STR(), KL_LOG_DATA() – Kernel Logging Macros

Synopsis

```
KL_LOG_INFO (
    unsigned short evt,
    short cl,
    unsigned short ss,
    unsigned long p0, p1, p2, p3, p4
);
```

```
KL_LOG_STR (
    unsigned short evt,
    short cl,
    unsigned short ss,
    char *cp
);
```

```
KL_LOG_DATA (
    unsigned short evt,
    short cl,
    unsigned short ss,
    caddr_t dp,
    int len
);
```

Parameters

evt **evt** is the ID of the event which triggered the instrumentation point to be executed. It will be cast to type unsigned short.

cl **cl** is the class (severity) of the event. It will be cast to type short. **cl** can be one of the following:

DISASTER The **DISASTER** class signals an event or condition which affects the operation of an entire subsystem, or the kernel, causing several programs to fail or the machine to panic

ERROR The **ERROR** class signals an event or condition which does not affect the overall operation of an entire subsystem, but may cause an application to fail.

	WARNING	The WARNING class signals an abnormal event, possibly caused by problems in an individual subsystem
	INFORMATIVE	The INFORMATIVE class signals important routine operations and current system values.
ss		ss is the ID of the subsystem the instrumentation point belongs to. It will be cast to type unsigned short.
p[0-4]		p[0-4] are five long integers logged by KL as user-defined data. They will be cast to type long.
cp		cp is a pointer to a string to be logged. It will be cast to type char*.
dp		dp is a pointer to a data structure. It will be cast to type caddr_t.
len		len is the length of the data structure. It will be cast to type int.

Description

The macros `KL_LOG_INFO()`, `KL_LOG_STR()`, and `KL_LOG_DATA()` are entry points to the Kernel Logging infrastructure. These macros are called from the variety of functions in the different subsystems within the kernel. Each of the three macros is used to record a message sent from the kernel with specific user-defined data:

`KL_LOG_INFO()` is used to pass five long integers as user-defined data.

`KL_LOG_STR()` is used to pass a string as user-defined data. This macro can also be used if no user-defined data is supplied; in such cases, the pointer to a string should be NULL.

`KL_LOG_DATA()` is used to pass an abstract data structure as user-defined data.

NOTE User-defined data should not exceed the value of `MAXPATHLEN`, which is currently 1024. This amount of data should be enough to store a filename (along with the full path) using the `KL_LOG_STR()` macro. If user-defined data exceeds `MAXPATHLEN` bytes, then only the first `MAXPATHLEN` bytes will be logged. Using the `KL_LOG_STR()` and `KL_LOG_DATA()` macros with long (more than 100 bytes) of user-defined data should be done sparingly. The `cp` and `dp` parameters must point to the strings and data structures which reside in kernel memory space, not user space.

These macros check whether a message issued by a particular instrumentation point should be logged by the Kernel Logging infrastructure. The check will be based on two parameters:

- whether the subsystem ID supplied by the instrumentation point falls into the KL range of

subsystem IDs, which is set to be from 513 to 1023

- whether the subsystem specified by the instrumentation point has been authorized (by you) to issue messages of the specified level, i.e., if subsystem X has been specified to log only messages of classes `DISASTER` and `ERROR`, then any messages issued by subsystem X with the class `WARNING` will not be logged.