# Advanced Debugging Messages Filter and Logger

## Description

Th **DbgLog** library filters and logs debugging/error messages according to errno, calling module/function, warning keyword and debugging level. Particular errno/keyword combinations can be made fatal. Output log files depend on keyword.
Typical output looks like:

```
IPC.....  <04>  1016120237s|706271us  [000004] :  0003  @  main   >  No such process   (0000000543)

  dbg     dbg          time             msg     errno      module     errno message    opt args
keywd   level   seconds|milisec     number
```

But the output format can also be changed at will by setting a format string.

This tool offers a tunable insight into the hierarchy of methods called within the libraries using it. It provides a way to trace and debug invalid data as well as misbehaving functions. The ability to turn off or filter whole segments of the debugging traces ensures that debugging will not be hindered by useless data. It filters output much in the same way as system calls debugging tools (such as strace) allow.

## Programming Interface

The API definition is entirely contained in the <DbgLog.h> include file.

### Overview

At the development stage, the main entry point in the API is the DbgLog funtion, that generates a log entry. To do that, this call must be supplied with contextual information, like function name, type of entry, and severity level. While this may seem cumbersome at first, this call needs only be used in error contexts, meaning its use is not too frequent.

The synopsis of the variadic function **DbgLog** is:

```
#include <DbgLog.h>

void DbgLog(const char * mod, const char * key, unsigned char lev, const char * mes, ...) ;
```

**mod** is the calling module or function name. This is the main context data, as this information will structure the actual hierarchy of calls. **key** is a debugging key, an identifier tag that will direct logging and filtering of this message. **lev** is a numeric indication of severity (the lower, the more severe, with 0 having sense as "system-level error")

At the degugging stage, several calls provide the tools to fine-tune the logging output: **DbgLog_SetFormat**, **DbgLog_SetLogFd**, **DbgLog_UpDbgLvl** and **DbgLog_DnDbgLvl**, **DbgLog_AddFatal** and **DbgLog_DelFatal**.
It is also possible to set the active degugging levels with a simple .properties file, using the DbgCfg library.

**DbgLog_SetFormat** has a single string argument, determining the format of log entries. The remaining functions have a debug key as first argument, standing for a particular class of filtering and, more generally, handling of that class of messages. The second argument is, for **DbgLog_SetLogFd**, an output file descriptor in which the messages will be written. The **...DbgLvl** calls use a binary mask of levels (between 0 and 31) to set on or off. The **...Fatal** calls set or unset some system error values to trigger immediate termination.

## Output format control

### Function DbgLog_SetFormat

**void DbgLog_SetFormat(char * fmt) ;**
**Sets the format of output messages.**

**Args:**

|  |  |
|---|---|
| _ **fmt** | : format string |

**Note:**

The format string is printed as-is for each log entry, except for escape sequences of the form '%d*?', where d* desings an optionnal decimal number indicating the size of this field, and ? is a character indicating the field type. Here is a list of valid types, and their function:

|  |  |
|---|---|
| _ **%** | : a litteral '%' character |
| _ **k** | : log key |
| _ **l** | : log level |
| _ **s** | : time of error (seconds component) |
| _ **u** | : time of error (microseconds component) |
| _ **n** | : entry number |
| _ **c** | : calling function or module |
| _ **e** | : error number (LibC errno) |
| _ **r** | : error message related to the error number |
| _ **m** | : user message |

The default format is '%8k<%2l> %10ss|%6uus [%6n] : %4e @ %c > %r (%m)'.

## Filters control

### Function DbgLog_SetLogFd

**void DbgLog_SetLogFd(char * key, int fd) ;**
**Sets log file descriptor.**

**Args:**

|  |  |
|---|---|
| _ **key** | : filter key identifier |
| _ **fd** | : file descriptor representing an open stream to use to print traces |

**Note:**

If the key argument is the empty string, it designs the default (generic) filter, whose caracteristics are used as a basis for new filters, or when filter is unspecified. When the key is NULL, it designs ALL filters except the default.

### Function DbgLog_DbgLvl

**unsigned long DbgLog_DbgLvl(char * key) ;**
**Gets the state of debugging levels.**

**Args:**

|  |  |
|---|---|
| _ **key** | : filter key identifier |

**Returns:**

|  |  |
|---|---|
| _ **0** | : if key was NULL, or did not refer to an existing filter |

else the current mask of active debugging levels.

**Note:**

If the key argument is the empty string, it designs the default (generic) filter, whose caracteristics are used as a basis for new filters, or when filter is unspecified. When the key is NULL, it designs ALL filters except the default.

The levels mask set bits positions (as in '<<') design debugging levels set ON.

## Function DbgLog_UpDbgLvl

**void DbgLog_UpDbgLvl(char * key, unsigned long lev) ;**
**Sets debugging levels ON.**

**Args:**

| | |
|---|---|
| _ **key** | : filter key identifier |
| _ **lev** | : debugging levels mask |

**Note:**
If the key argument is the empty string, it designs the default (generic) filter,
whose caracteristics are used as a basis for new filters, or when filter is unspecified.
When the key is NULL, it designs ALL filters except the default.
The levels mask set bits positions (as in '<<') design debugging levels set ON.

## Function DbgLog_DnDbgLvl

**void DbgLog_DnDbgLvl(char * key, unsigned long lev) ;**
**Sets debugging levels OFF.**

**Args:**

| | |
|---|---|
| _ **key** | : filter key identifier |
| _ **lev** | : debugging levels mask |

**Note:**
If the key argument is the empty string, it designs the default (generic) filter,
whose caracteristics are used as a basis for new filters, or when filter is unspecified.
When the key is NULL, it designs ALL filters except the default.
The levels mask set bits positions (as in '<<') design debugging levels set OFF.

## Function DbgLog_AddFatal

**void DbgLog_AddFatal(char * key, int err) ;**
**Adds a fatal errno.**

**Args:**

| | |
|---|---|
| _ **key** | : filter key identifier |
| _ **err** | : concerned errno |

**Note:**
If the key argument is the empty string, it designs the default (generic) filter,
whose caracteristics are used as a basis for new filters, or when filter is unspecified.
When the key is NULL, it designs ALL filters except the default.
When an error with this filter key AND this errno is encountered, this triggers
immediate termination (after the printing of a log entry), through _exit (2).

## Function DbgLog_DelFatal

**void DbgLog_DelFatal(char * key, int err) ;**
**Deletes a fatal errno.**

**Args:**

| | |
|---|---|
| _ **key** | : filter key identifier |
| _ **err** | : concerned errno |

**Note:**
If the key argument is the empty string, it designs the default (generic) filter,

whose caracteristics are used as a basis for new filters, or when filter is unspecified.
When the key is NULL, it designs ALL filters except the default.

## Log entries

### Function DbgLog

**void DbgLog(const char * mod, const char * key, unsigned char lev, const char * mes, ...) ;**
**Produces an error from module/function 'mod', filter key 'key' and message 'mes', at debugging level 'lev'.**

**Args:**

| | |
|---|---|
| _ **mod** | : name of the function/module from which this call was issued |
| _ **key** | : key of filter to be used |
| _ **lev** | : debugging level at which this error is raised |
| _ **mes** | : optional message to add to transcript |
| _ **...** | : arguments to be formatted in the mes string |

**Note:**

Use a NULL value for mes when no message needs to be printed.
Variable arguments are formatted following escapes in the message string, taking the form '%d*?' where d* is any number of digits and ? is either:

| | |
|---|---|
| _ **%** | : a litteral '%' character is printed |
| _ **c** | : the argument is a character |
| _ **s** | : the argument is a string, and will be printed as-is |
| _ **d** | : the argument is a signed number, printed in decimal form |
| _ **u** | : the argument is an unsigned number, printed in decimal form |
| _ **x** | : the argument is an unsigned number, printed in hexadecimal form |

For s,u,x, the digits before the conversion character give the maximum number of characters to be printed.

### Function DbgLog_

**void DbgLog_(const char * mod, const char * key, unsigned char lev, const char * mes, va_list ap) ;**
**Produces an error from module/function 'mod', filter key 'key' and message 'mes', at debugging level 'lev'.**

**Args:**

| | |
|---|---|
| _ **mod** | : name of the function/module from which this call was issued |
| _ **key** | : key of filter to be used |
| _ **lev** | : debugging level at which this error is raised |
| _ **mes** | : optional message to add to transcript |
| _ **ap** | : optional variable arguments list |

**Note:**

Use a NULL value for mes when no message needs to be printed.
Variable arguments are formatted following escapes in the message string, taking the form '%d*?' where d* is any number of digits and ? is either:

| | |
|---|---|
| _ **%** | : a litteral '%' character is printed |
| _ **c** | : the argument is a character |
| _ **s** | : the argument is a string, and will be printed as-is |
| _ **d** | : the argument is a signed number, printed in decimal form |
| _ **u** | : the argument is an unsigned number, printed in decimal form |
| _ **x** | : the argument is an unsigned number, printed in hexadecimal form |

For s,u,x, the digits before the conversion character give the maximum number of characters to be printed.
This function does not call va_start (of course) and va_end, so the value of ap is undefined after the call, and va_end must be called by the application.

### Block-wise logging

#### Function DbgLog_OpenBlock

**void DbgLog_OpenBlock(void) ;**
**Starts a new debugging block.**

#### Function DbgLog_CloseBlock

**void DbgLog_CloseBlock(void) ;**
**Closes the current debugging block, reactivating the last one.**

#### Function DbgLog_Flush

**void DbgLog_Flush(void) ;**
**Prints the last debugging block in log.**

## Download

The latest version of DbgLog can be obtained from
**http://www.varkhan.net/software/DbgLog**

## Author

| | |
|---|---|
| **Name:** | **Yann LANDRIN-SCHWEITZER aka Varkhan** |
| **Mail:** | **varkhan at varkhan dot net** |
| **Home:** | **http://www.varkhan.net/** |

## License