



Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'libdmmp.h.3'

\$ man libdmmp.h.3

libdmmp.h(3) Library Functions Manual libdmmp.h(3)

NAME

libdmmp.h - Device Mapper Multipath API.

SYNOPSIS

```
#include <libdmmp/libdmmp.h>
```

DESCRIPTION

All the libdmmp public functions ships its own man pages. Use 'man 3 <function_name>' to check the detail usage.

USAGE

To use libdmmp in your project, we suggest to use the 'pkg-config' way:

* Add this line into your configure.ac:

```
PKG_CHECK_MODULES([LIBDMMP], [libdmmp])
```

* Add these lines into your Makefile.am:

```
foo_LDFLAGS += $(LIBDMMP_LIBS)
```

```
foo_CFLAGS += $(LIBDMMP_CFLAGS)
```

LOG HANDLING

The log handler function could be set via 'dmmp_context_log_func_set()'. The log priority could be set via 'dmmp_context_log_priority_set()'.
By default, the log priorities is 'DMMP_LOG_PRIORITY_WARNING'. By default, the log handler is print log to STDERR, and its code is listed below in case you want to create your own log handler.

```
static int _DMMP_LOG_STRERR_ALIGN_WIDTH = 80;
```

```
static void _log_stderr(struct dmmp_context *ctx,
```

```

        enum dmmp_log_priority priority,
        const char *file, int line,
        const char *func_name,
        const char *format, va_list args)
{
    int printed_bytes = 0;
    printed_bytes += fprintf(stderr, "libdmmp %s: ",
        dmmp_log_priority_str(priority));
    printed_bytes += vfprintf(stderr, format, args);
    userdata = dmmp_context_userdata_get(ctx);
    if (userdata != NULL)
        fprintf(stderr, "(with user data at memory address %p)",
            userdata);
    if (printed_bytes < _DMMP_LOG_STRERR_ALIGN_WIDTH) {
        fprintf(stderr, "%*s # %s:%s():%d0,
            _DMMP_LOG_STRERR_ALIGN_WIDTH - printed_bytes, "", file,
            func_name, line);
    } else {
        fprintf(stderr, " # %s:%s():%d0, file, func_name, line);
    }
}

```

SAMPLE CODE

```

#include <libdmmp/libdmmp.h>

int main(int argc, char *argv[]) {
    struct dmmp_context *ctx = NULL;
    struct dmmp_mpath **dmmp_mps = NULL;
    struct dmmp_path_group **dmmp_pgs = NULL;
    struct dmmp_path **dmmp_ps = NULL;
    uint32_t dmmp_mp_count = 0;
    uint32_t dmmp_pg_count = 0;
    uint32_t dmmp_p_count = 0;
    const char *name = NULL;
    const char *wwid = NULL;

```

```

uint32_t i = 0;

int rc = DMMP_OK;

ctx = dmmp_context_new();

dmmp_context_log_priority_set(ctx, DMMP_LOG_PRIORITY_DEBUG);

// By default, log will be printed to STDERR, you could
// change that via dmmp_context_log_func_set()

rc = dmmp_mpath_array_get(ctx, &dmmp_mps, &dmmp_mp_count);

if (rc != DMMP_OK) {

    printf("dmmp_mpath_array_get() failed with %d: %s", rc,
           dmmp_strerror(rc));

    goto out;

}

for (i = 0; i < dmmp_mp_count; ++i) {

    name = dmmp_mpath_name_get(dmmp_mps[i]);

    wwid = dmmp_mpath_wwid_get(dmmp_mps[i]);

    printf("dmmp_mpath_array_get(): Got mpath: %s %s0, name,
           wwid);

    // You could use dmmp_path_group_array_get() to retrieve
    // path group information and then invoke dmmp_path_array_get()
    // for path information.

}

out:

dmmp_context_free(ctx);

dmmp_mpath_array_free(dmmp_mps, dmmp_mp_count);

if (rc != DMMP_OK)

    exit(1);

exit(0);

}

```

LICENSE

GPLv2+

BUG

Please report bug to <dm-devel@redhat.com>