

---

# **dmrplib Documentation**

*Release 0.99.3*

**Wijnand Modderman-Lenstra**

September 03, 2016



<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Documentation</b>	<b>3</b>
2.1	bits: bit and byte manipulation . . . . .	3
2.2	crc: cyclic redundancy checks . . . . .	4
2.3	fec: forward error correction . . . . .	5
2.4	log: logging . . . . .	5
2.5	malloc: memory allocation . . . . .	6
2.6	packet: DMR packet handling . . . . .	7
2.7	packetq: DMR packet queue . . . . .	9
2.8	protocol: Site Connect protocols . . . . .	10
2.9	queue: Berkeley queues . . . . .	12
2.10	raw: raw byte buffers . . . . .	15
2.11	tree: data trees . . . . .	17
<b>3</b>	<b>Indices and tables</b>	<b>21</b>



**Overview**

---

dmrlib is a multi-platform support library to build applications for Ham Radio applications of the Digital Mobile Radio (DMR) protocol.



## 2.1 bits: bit and byte manipulation

### 2.1.1 Data types

**dmr\_bit**

Single bit values.

**dmr\_dibit**

Dibit (or double bit) values.

**dmr\_tribit**

Tribit (or triple bit) values.

### 2.1.2 API

**Bit sizes****DMR\_SLOT\_TYPE\_HALF****DMR\_SLOT\_TYPE\_BITS****Byte manipulation****DMR\_UINT16\_LE** (b0, b1)**DMR\_UINT16\_BE** (b0, b1)**DMR\_UINT16\_LE\_PACK** (b, n)**DMR\_UINT16\_BE\_PACK** (b, n)**DMR\_UINT32\_LE** (b0, b1, b2, b3)**DMR\_UINT32\_BE** (b0, b1, b2, b3)**DMR\_UINT32\_BE\_UNPACK** (b)**DMR\_UINT32\_BE\_UNPACK3** (b)**DMR\_UINT32\_LE\_PACK** (b, n)**DMR\_UINT32\_LE\_PACK3** (b, n)

**DMR\_UINT32\_BE\_PACK** (b, n)

**DMR\_UINT32\_BE\_PACK3** (b, n)

char \***dmr\_byte\_to\_binary** (uint8\_t *byte*)

uint8\_t **dmr\_bits\_to\_byte** (bool *bits[8]*)

void **dmr\_bits\_to\_bytes** (bool \**bits*, size\_t *bits\_length*, uint8\_t \**bytes*, size\_t *bytes\_length*)

void **dmr\_byte\_to\_bits** (uint8\_t *byte*, bool *bits[8]*)

void **dmr\_bytes\_to\_bits** (uint8\_t \**bytes*, size\_t *bytes\_length*, bool \**bits*, size\_t *bits\_length*)

## 2.2 crc: cyclic redundancy checks

### 2.2.1 API

**DMR\_CRC8\_MASK\_VOICE\_PI**

**DMR\_CRC8\_MASK\_VOICE\_LC**

**DMR\_CRC8\_MASK\_TERMINATOR\_WITH\_LC**

void **dmr\_crc9** (uint16\_t \*, uint8\_t, uint8\_t)

void **dmr\_crc9\_finish** (uint16\_t \*, uint8\_t)

void **dmr\_crc16** (uint16\_t \*, uint8\_t)

void **dmr\_crc16\_finish** (uint16\_t \*)

void **dmr\_crc32** (uint32\_t \*, uint8\_t)

void **dmr\_crc32\_finish** (uint32\_t \*)



## 2.3 fec: forward error correction

### 2.3.1 Block Product Turbo Code (196, 96)

### 2.3.2 Golay (20, 8)

### 2.3.3 Quadratic Residue (16, 7)

### 2.3.4 Reed-Solomon

### 2.3.5 Reed-Solomon (12, 9)

### 2.3.6 Trellis

### 2.3.7 Variable Block Product Turbo Code (16, 11)

## 2.4 log: logging

### 2.4.1 Data types

`dmr_log_priority_t`

`DMR_LOG_PRIORITY_TRACE`

`DMR_LOG_PRIORITY_DEBUG`

`DMR_LOG_PRIORITY_INFO`

`DMR_LOG_PRIORITY_WARN`

`DMR_LOG_PRIORITY_ERROR`

`DMR_LOG_PRIORITY_CRITICAL`

`DMR_LOG_PRIORITIES`

`void (*dmr_log_cb_t) (void *, dmr_log_priority_t, const char *)`

### 2.4.2 API

`DMR_LOG_MESSAGE_MAX`

`DMR_LOG_TIME_FORMAT`

`DMR_LOG_BOOL(x)`

`bool dmr_log_color (void)`

`void dmr_log_color_set (bool)`

`dmr_log_priority_t dmr_log_priority (void)`

`void dmr_log_priority_set (dmr_log_priority_t)`

`void dmr_log_priority_reset (void)`

`const char *dmr_log_prefix (void)`

void **dmr\_log\_prefix\_set** (const char \*)  
void **dmr\_log** (const char \*, ...)  
void **dmr\_log\_mutex** (const char \*, ...)  
void **dmr\_log\_trace** (const char \*, ...)  
void **dmr\_log\_debug** (const char \*, ...)  
void **dmr\_log\_info** (const char \*, ...)  
void **dmr\_log\_warn** (const char \*, ...)  
void **dmr\_log\_error** (const char \*, ...)  
void **dmr\_log\_errno** (const char \**msg*)  
void **dmr\_log\_critical** (const char \*, ...)  
void **dmr\_log\_message** (*dmr\_log\_priority\_t*, const char \*, ...)  
void **dmr\_log\_messagev** (*dmr\_log\_priority\_t*, const char \*, va\_list)  
void **dmr\_log\_cb\_get** (*dmr\_log\_cb\_t* \*, void \*\*)  
void **dmr\_log\_cb** (*dmr\_log\_cb\_t*, void \*)

## 2.5 malloc: memory allocation

### 2.5.1 API

**dmr\_palloc** (void \*, type)  
Allocate a 0-initialized structure with parent context.

**dmr\_palloc\_size** (void \*, size)  
Allocate a 0-initialized untyped buffer with parent context.

**dmr\_malloc** (type)  
Allocate a 0-initialized structure.

**dmr\_malloc\_size** (size)  
Allocate a 0-initialized untyped buffer.

**dmr\_realloc** (void \*, void\*, type, size)  
Resize an untyped buffer with parent context.

**dmr\_strdup** (void \*, char \*)  
Duplicate a string with parent context.

**dmr\_free** (void \*)  
Free a previously allocated structure.

**DMR\_NULL\_CHECK** (expr)  
Checks an expression for NULL returns, sets ENOMEM.

**DMR\_NULL\_CHECK\_FREE** (expr, var)  
Checks an expression for NULL returns, sets ENOMEM and frees var.

## 2.6 packet: DMR packet handling

### 2.6.1 Data types

`dmr_color_code`

`dmr_data_type`

DMR data type. Also includes pseudo data types for internal usage.

`DMR_DATA_TYPE_VOICE_PI`

`DMR_DATA_TYPE_VOICE_LC`

`DMR_DATA_TYPE_TERMINATOR_WITH_LC`

`DMR_DATA_TYPE_CSBK`

`DMR_DATA_TYPE_MBC_HEADER`

`DMR_DATA_TYPE_MBC_CONTINUATION`

`DMR_DATA_TYPE_DATA_HEADER`

`DMR_DATA_TYPE_RATE12_DATA`

`DMR_DATA_TYPE_RATE34_DATA`

`DMR_DATA_TYPE_IDLE`

`DMR_DATA_TYPE_INVALID`

`DMR_DATA_TYPE_SYNC_VOICE`

`DMR_DATA_TYPE_VOICE_SYNC`

`DMR_DATA_TYPE_VOICE`

`dmr_id`

DMR identifier.

`dmr_fid`

DMR function identifier.

`DMR_FID_ETSI`

`DMR_FID_DMRA`

`DMR_FID_HYTERA`

`DMR_FID_MOTOROLA`

`dmr_flco`

Full Link Control Opcode.

`DMR_FLCO_GROUP`

`DMR_FLCO_PRIVATE`

`DMR_FLCO_INVALID`

`dmr_ts`

DMR time slot.

`DMR_TS1`

`DMR_TS2`

## Packets

### **dmr\_packet**

Raw DMR packet.

### **dmr\_packet\_data**

Raw DMR packet data bits.

### **dmr\_packet\_data\_block\_bits**

Raw DMR packet data block bits.

### **dmr\_packet\_data\_block**

Parsed DMR packet data block.

int (**\*dmr\_packet\_cb**) (*dmr\_packet*, void \*)

Callback for raw DMR packets.

### **dmr\_parsed\_packet**

Parsed DMR packet with meta data.

*dmr\_packet* **dmr\_parsed\_packet.packet**

*dmr\_ts* **dmr\_parsed\_packet.ts**

*dmr\_flco* **dmr\_parsed\_packet.flco**

*dmr\_id* **dmr\_parsed\_packet.src\_id**

Source (originating) DMR ID.

*dmr\_id* **dmr\_parsed\_packet.dst\_id**

Target DMR ID.

*dmr\_id* **dmr\_parsed\_packet.repeater\_id**

DMR ID of the repeater or hotspot.

*dmr\_data\_type* **dmr\_parsed\_packet.data\_type**

*dmr\_color\_code* **dmr\_parsed\_packet.color\_code**

uint8\_t **dmr\_parsed\_packet.sequence**

Sequential number for frames that belong to the same *dmr\_parsed\_packet.stream\_id*.

uint32\_t **dmr\_parsed\_packet.stream\_id**

Unique identifier for the stream.

uint8\_t **dmr\_parsed\_packet.voice\_frame**

Voice frame index.

bool **dmr\_parsed\_packet.parsed**

int (**\*dmr\_parsed\_packet\_cb**) (*dmr\_parsed\_packet* \*, void \*)

Callback for parsed DMR packets.

## 2.6.2 API

### **DMR\_PACKET\_LEN**

Size of a single DMR packet (frame) in bytes.

### **DMR\_PACKET\_BITS**

Size of a single DMR packet (frame) in bits.

### **DMR\_DATA\_TYPE\_COUNT**

Number of valid data types, without pseudo data types.

---

**Note:** The `dmr_dump_packet()` and `dmr_dump_parsed_packet()` functions are not ABI stable.

---

```
void dmr_dump_packet (dmr_packet)
    Debug function that decodes a raw DMR packet.

void dmr_dump_parsed_packet (dmr_parsed_packet *)
dmr_parsed_packet *dmr_packet_decode (dmr_packet)
char *dmr_flco_name (dmr_flco)
char *dmr_ts_name (dmr_ts)
char *dmr_fid_name (dmr_fid)
char *dmr_data_type_name (dmr_data_type)
char *dmr_data_type_name_short (dmr_data_type)
int dmr_payload_bits (dmr_packet, void *)
int dmr_slot_type_decode (dmr_packet, dmr_color_code *, dmr_data_type *)
int dmr_slot_type_encode (dmr_packet, dmr_color_code, dmr_data_type)
```

## 2.7 packetq: DMR packet queue

### 2.7.1 Data types

```
dmr_packetq_entry
dmr_packetq_entry.parsed
dmr_packetq_entry.entries
dmr_packetq
dmr_packetq.head
```

### 2.7.2 API

```
dmr_packetq * dmr_packetq_new ()
    Allocates a new packet queue. Free with dmr_free().

int dmr_packetq_add (dmr_packetq *, dmr_parsed_packet *)
int dmr_packetq_add_packet (dmr_packetq *, dmr_packet)
int dmr_packetq_shift (dmr_packetq *, dmr_parsed_packet **)
int dmr_packetq_shift_packet (dmr_packetq *, dmr_packet)
int dmr_packetq_foreach (dmr_packetq *, dmr_parsed_packet_cb, void *)
int dmr_packetq_foreach_packet (dmr_packetq *, dmr_packet_cb, void *)
```

## 2.8 protocol: Site Connect protocols

### 2.8.1 Homebrew IP Site Connect

#### Data types

`dmr_homebrew_state`

`DMR_HOMEBREW_AUTH_NONE`

`DMR_HOMEBREW_AUTH_INIT`

`DMR_HOMEBREW_AUTH_CONFIG`

`DMR_HOMEBREW_AUTH_DONE`

`DMR_HOMEBREW_AUTH_FAILED`

`dmr_homebrew`

struct `dmr_homebrew.config`

char \*`dmr_homebrew.config.call`

*dmr\_id* `dmr_homebrew.config.repeater_id`

uint16\_t `dmr_homebrew.config.rx_freq`

uint16\_t `dmr_homebrew.config.tx_freq`

uint8\_t `dmr_homebrew.config.tx_power`

*dmr\_color\_code* `dmr_homebrew.config.color_code`

double `dmr_homebrew.config.latitude`

double `dmr_homebrew.config.longitude`

uint16\_t `dmr_homebrew.config.altitude`

char \*`dmr_homebrew.config.location`

char \*`dmr_homebrew.config.description`

char \*`dmr_homebrew.config.url`

char \*`dmr_homebrew.config.software_id`

char \*`dmr_homebrew.config.package_id`

char \*`dmr_homebrew.id`

Protocol identificaiton string.

void \*`dmr_homebrew.sock`

uint8\_t `dmr_homebrew.peer_ip[16]`

uint16\_t `dmr_homebrew.peer_port`

uint8\_t `dmr_homebrew.bind_ip[16]`

uint16\_t `dmr_homebrew.bind_port`

uint8\_t `dmr_homebrew.state`

char \*`dmr_homebrew.secret`

Authentication secret set by `dmr_homebrew_auth()`.

```

uint8_t      dmr_homebrew_nonce[8]
    Nonce sent by repeater.

dmr_packetq *dmr_homebrew_rxq
dmr_packetq *dmr_homebrew_txq
dmr_rawq *dmr_homebrew_rrq
dmr_rawq *dmr_homebrew_trq

struct timeval dmr_homebrew_last_ping
struct timeval dmr_homebrew_last_pong    /* last pong received */

```

## API

### DMR\_HOMEBREW\_PORT

Default Homebrew protocol UDP/IP port.

```

dmr_homebrew * dmr_homebrew_new(dmr_id repeater_id, uint8_t peer_ip[16], uint16_t peer_port,
                                uint8_t bind_ip[16], uint16_t bind_port)

```

Setup a new Homebrew instance.

This function sets up the internal structure as well as the communication sockets, the caller must provide data to the internal config struct before `dmr_homebrew_auth()` is called.

```

int dmr_homebrew_auth(dmr_homebrew *homebrew, char *secret)

```

Initiate authentication with the repeater.

```

int dmr_homebrew_close(dmr_homebrew *homebrew)

```

Close the link with the repeater.

```

int dmr_homebrew_read(dmr_homebrew *homebrew, dmr_parsed_packet **parsed_out)

```

Read a packet from the repeater.

This also processes communications with the Homebrew repeater, if the received frame does not contain a DMR packet, the function will set the destination packet pointer to `NULL`.

```

int dmr_homebrew_send(dmr_homebrew *homebrew, dmr_parsed_packet *parsed)

```

Send a DMR frame to the repeater.

```

int dmr_homebrew_send_buf(dmr_homebrew *homebrew, uint8_t *buf, size_t len)

```

Send a raw buffer to the repeater.

```

int dmr_homebrew_send_raw(dmr_homebrew *homebrew, dmr_raw *raw)

```

Send a raw packet to the repeater.

```

int dmr_homebrew_parse_dmr(dmr_homebrew *homebrew, dmr_raw *raw,
                           dmr_parsed_packet **parsed_out)

```

Parse a Homebrew protocol DMR data frame.

```

dmr_protocol dmr_homebrew_protocol

```

Protocol specification.

## 2.8.2 Multi Mode Digital Voice Modem

### Supported hardware

The following MMDVM variants are supported by dmrlib:

- G4KLX MMDVM

- PE1PLM AMBE3000
- PE1PLM DVMEGA

## 2.9 queue: Berkeley queues

This file defines five types of data structures: singly-linked lists, lists, simple queues, tail queues and XOR simple queues.

### 2.9.1 API

#### Singly-linked list

A singly-linked list is headed by a single forward pointer. The elements are singly linked for minimum space and pointer manipulation overhead at the expense of  $\mathcal{O}(n)$  removal for arbitrary elements. New elements can be added to the list after an existing element or at the head of the list. Elements being removed from the head of the list should use the explicit macro for this purpose for optimum efficiency. A singly-linked list may only be traversed in the forward direction. Singly-linked lists are ideal for applications with large datasets and few or no removals or for implementing a LIFO queue.

**DMR\_SLIST\_HEAD** (name, type)  
**DMR\_SLIST\_HEAD\_INITIALIZER** (head)  
**DMR\_SLIST\_ENTRY** (type)  
**DMR\_SLIST\_FIRST** (head)  
**DMR\_SLIST\_END** (head)  
**DMR\_SLIST\_EMPTY** (head)  
**DMR\_SLIST\_NEXT** (elm, field)  
**DMR\_SLIST\_FOREACH** (var, head, field)  
**DMR\_SLIST\_FOREACH\_SAFE** (var, head, field, tvar)  
**DMR\_SLIST\_INIT** (head)  
**DMR\_SLIST\_INSERT\_AFTER** (slistelm, elm, field)  
**DMR\_SLIST\_INSERT\_HEAD** (head, elm, field)  
**DMR\_SLIST\_REMOVE\_AFTER** (elm, field)  
**DMR\_SLIST\_REMOVE\_HEAD** (head, field)  
**DMR\_SLIST\_REMOVE** (head, elm, type, field)

#### List

A list is headed by a single forward pointer (or an array of forward pointers for a hash table header). The elements are doubly linked so that an arbitrary element can be removed without a need to traverse the list. New elements can be added to the list before or after an existing element or at the head of the list. A list may only be traversed in the forward direction.

**DMR\_LIST\_HEAD** (name, type)



**DMR\_LIST\_HEAD\_INITIALIZER** (head)  
**DMR\_LIST\_ENTRY** (type)  
**DMR\_LIST\_FIRST** (head)  
**DMR\_LIST\_END** (head)  
**DMR\_LIST\_EMPTY** (head)  
**DMR\_LIST\_NEXT** (elm, field)  
**DMR\_LIST\_FOREACH** (var, head, field)  
**DMR\_LIST\_FOREACH\_SAFE** (var, head, field, tvar)  
**DMR\_LIST\_INIT** (head)  
**DMR\_LIST\_INSERT\_AFTER** (listelm, elm, field)  
**DMR\_LIST\_INSERT\_BEFORE** (listelm, elm, field)  
**DMR\_LIST\_INSERT\_HEAD** (head, elm, field)  
**DMR\_LIST\_REMOVE** (elm, field)  
**DMR\_LIST\_REPLACE** (elm, elm2, field)

## Simple queue

A simple queue is headed by a pair of pointers, one to the head of the list and the other to the tail of the list. The elements are singly linked to save space, so elements can only be removed from the head of the list. New elements can be added to the list before or after an existing element, at the head of the list, or at the end of the list. A simple queue may only be traversed in the forward direction.

**DMR\_SIMPLEQ\_HEAD** (name, type)  
**DMR\_SIMPLEQ\_HEAD\_INITIALIZER** (head)  
**DMR\_SIMPLEQ\_ENTRY** (type)  
**DMR\_SIMPLEQ\_FIRST** (head)  
**DMR\_SIMPLEQ\_END** (head)  
**DMR\_SIMPLEQ\_EMPTY** (head)  
**DMR\_SIMPLEQ\_NEXT** (elm, field)  
**DMR\_SIMPLEQ\_FOREACH** (var, head, field)  
**DMR\_SIMPLEQ\_FOREACH\_SAFE** (var, head, field, tvar)  
**DMR\_SIMPLEQ\_INIT** (head)  
**DMR\_SIMPLEQ\_INSERT\_HEAD** (head, elm, field)  
**DMR\_SIMPLEQ\_INSERT\_TAIL** (head, elm, field)  
**DMR\_SIMPLEQ\_INSERT\_AFTER** (head, listelm, elm, field)  
**DMR\_SIMPLEQ\_REMOVE\_HEAD** (head, field)  
**DMR\_SIMPLEQ\_REMOVE\_AFTER** (head, elm, field)  
**DMR\_SIMPLEQ\_CONCAT** (head1, head2)

## XOR simple queue

An XOR simple queue is used in the same way as a regular simple queue. The difference is that the head structure also includes a “cookie” that is XOR’d with the queue pointer (first, last or next) to generate the real pointer value.

**DMR\_XSIMPLEQ\_HEAD** (name, type)  
**DMR\_XSIMPLEQ\_ENTRY** (type)  
**DMR\_XSIMPLEQ\_XOR** (head, ptr)  
**DMR\_XSIMPLEQ\_FIRST** (head)  
**DMR\_XSIMPLEQ\_END** (head)  
**DMR\_XSIMPLEQ\_EMPTY** (head)  
**DMR\_XSIMPLEQ\_NEXT** (head, elm, field)  
**DMR\_XSIMPLEQ\_FOREACH** (var, head, field)  
**DMR\_XSIMPLEQ\_FOREACH\_SAFE** (var, head, field, tvar)  
**DMR\_XSIMPLEQ\_INIT** (head)  
**DMR\_XSIMPLEQ\_INSERT\_HEAD** (head, elm, field)  
**DMR\_XSIMPLEQ\_INSERT\_TAIL** (head, elm, field)  
**DMR\_XSIMPLEQ\_INSERT\_AFTER** (head, listelm, elm, field)  
**DMR\_XSIMPLEQ\_REMOVE\_HEAD** (head, field)  
**DMR\_XSIMPLEQ\_REMOVE\_AFTER** (head, elm, field)

## Tail queue

A tail queue is headed by a pair of pointers, one to the head of the list and the other to the tail of the list. The elements are doubly linked so that an arbitrary element can be removed without a need to traverse the list. New elements can be added to the list before or after an existing element, at the head of the list, or at the end of the list. A tail queue may be traversed in either direction.

**DMR\_TAILQ\_HEAD** (name, type)  
**DMR\_TAILQ\_HEAD\_INITIALIZER** (head)  
**DMR\_TAILQ\_ENTRY** (type)  
**DMR\_TAILQ\_FIRST** (head)  
**DMR\_TAILQ\_END** (head)  
**DMR\_TAILQ\_NEXT** (elm, field)  
**DMR\_TAILQ\_LAST** (head, headname)  
**DMR\_TAILQ\_PREV** (elm, headname, field)  
**DMR\_TAILQ\_EMPTY** (head)  
**DMR\_TAILQ\_FOREACH** (var, head, field)  
**DMR\_TAILQ\_FOREACH\_SAFE** (var, head, field, tvar)  
**DMR\_TAILQ\_FOREACH\_REVERSE** (var, head, headname, field)  
**DMR\_TAILQ\_FOREACH\_REVERSE\_SAFE** (var, head, headname, field, tvar)

**DMR\_TAILQ\_INIT** (head)  
**DMR\_TAILQ\_INSERT\_HEAD** (head, elm, field)  
**DMR\_TAILQ\_INSERT\_TAIL** (head, elm, field)  
**DMR\_TAILQ\_INSERT\_AFTER** (head, listelm, elm, field)  
**DMR\_TAILQ\_INSERT\_BEFORE** (listelm, elm, field)  
**DMR\_TAILQ\_REMOVE** (head, elm, field)  
**DMR\_TAILQ\_REPLACE** (head, elm, elm2, field)  
**DMR\_TAILQ\_CONCAT** (head1, head2, field)

## 2.10 raw: raw byte buffers

### 2.10.1 Data types

**dmr\_raw**  
 Raw byte buffer with preallocated size.

`uint8_t *dmr_raw.buf`  
 The actual buffer.

`uint64_t dmr_raw.len`  
 Number of bytes stored in the buffer.

`int64_t dmr_raw.allocated`  
 Number of bytes allocated in the buffer.

**DMR\_TAILQ\_ENTRY(dmr\_raw) dmr\_raw.entries**  
 If this raw buffer is part of a `c:type:dmr_rawq`, this stores pointers to the sibling entries.

**dmr\_rawq**  
 Zero or more `dmr_raw` buffers in a tail queue.

**DMR\_TAILQ\_HEAD dmr\_rawq.head**  
 Head of the tail queue.

`size_t dmr_rawq.limit`  
 Maximum number of buffers in the tail queue.

`void (*dmr_raw_cb) (dmr_raw *, void *)`

### 2.10.2 API

`dmr_raw *dmr_raw_new (uint64_t len)`  
 Setup a new raw buffer.

`void dmr_raw_free (dmr_raw *raw)`  
 Destroy a raw buffer.

`int dmr_raw_add (dmr_raw *raw, const void *buf, size_t len)`  
 Add to the raw buffer.

`int dmr_raw_add_hex (dmr_raw *raw, const void *buf, size_t len)`  
 Add to the raw buffer and hex encode.

`int dmr_raw_add_uint8 (dmr_raw *raw, uint8_t in)`  
Add an `uint8_t` to the buffer.

`int dmr_raw_add_uint16 (dmr_raw *raw, uint16_t in)`  
Add an `uint16_t` to the buffer.

`int dmr_raw_add_uint24 (dmr_raw *raw, uint24_t in)`  
Add an `uint24_t` to the buffer.

`int dmr_raw_add_uint32 (dmr_raw *raw, uint32_t in)`  
Add an `uint32_t` to the buffer.

`int dmr_raw_add_uint32_le (dmr_raw *raw, uint32_t in)`  
Add an `uint32_t` to the buffer (little endian).

`int dmr_raw_add_uint64 (dmr_raw *raw, uint64_t in)`  
Add an `uint64_t` to the buffer.

`int dmr_raw_add_xuint8 (dmr_raw *raw, uint8_t in)`  
Add a hex encoded `uint8_t` to the buffer.

`int dmr_raw_add_xuint16 (dmr_raw *raw, uint16_t in)`  
Add a hex encoded `uint16_t` to the buffer.

`int dmr_raw_add_xuint24 (dmr_raw *raw, uint24_t in)`  
Add a hex encoded `uint24_t` to the buffer.

`int dmr_raw_add_xuint32 (dmr_raw *raw, uint32_t in)`  
Add a hex encoded `uint32_t` to the buffer.

`int dmr_raw_add_xuint32_le (dmr_raw *raw, uint32_t in)`  
Add a hex encoded `uint32_t` to the buffer (little endian).

`int dmr_raw_add_xuint64 (dmr_raw *raw, uint64_t in)`  
Add a hex encoded `uint64_t` to the buffer.

`int dmr_raw_addf (dmr_raw *raw, size_t len, const char *fmt, ...)`  
Add a formatted string to the buffer.

`int dmr_raw_grow (dmr_raw *raw, uint64_t len)`  
Resize a raw buffer.

`int dmr_raw_grow_add (dmr_raw *raw, uint64_t add)`  
Resize a raw buffer if `add` number of bytes can't be added.

`int dmr_raw_reset (dmr_raw *raw)`  
Reset a raw buffer.

`int dmr_raw_zero (dmr_raw *raw)`  
Zero a raw buffer, maintaining the size.

`dmr_rawq *dmr_rawq_new (size_t limit)`  
Setup a new raw queue.  
A limit of 0 means unlimited queue size.

`void dmr_rawq_free (dmr_rawq *rawq)`  
Destroy a raw queue.

`int dmr_rawq_add (dmr_rawq *rawq, dmr_raw *raw)`  
Add a buffered element to the raw queue.

`int dmr_rawq_addb (dmr_rawq *rawq, uint8_t *buf, size_t len)`  
Add a buffer to the raw queue.

int **dmr\_rawq\_each** (*dmr\_rawq* \*rawq, *dmr\_raw\_cb* cb, void \*userdata)

Iterate over all the items in a raw queue.

bool **dmr\_rawq\_empty** (*dmr\_rawq* \*rawq)

Check if the rawq is empty.

size\_t **dmr\_rawq\_size** (*dmr\_rawq* \*rawq)

Size of the rawq.

*dmr\_raw* \***dmr\_rawq\_shift** (*dmr\_rawq* \*rawq)

Shift the first element from the raw queue.

int **dmr\_rawq\_unshift** (*dmr\_rawq* \*rawq, *dmr\_raw* \*raw)

Prepend a raw buffer to the raw queue.

## 2.11 tree: data trees

This module defines data structures for different types of trees: splay trees and red-black trees.

### 2.11.1 API

#### Splay tree

A splay tree is a self-organizing data structure. Every operation on the tree causes a splay to happen. The splay moves the requested node to the root of the tree and partly rebalances it.

This has the benefit that request locality causes faster lookups as the requested nodes move to the top of the tree. On the other hand, every lookup causes memory writes.

The Balance Theorem bounds the total access time for  $m$  operations and  $n$  inserts on an initially empty tree as  $\mathcal{O}((m+n) \log n)$ . The amortized cost for a sequence of  $m$  accesses to a splay tree is  $\mathcal{O}(\log n)$ .

**DMR\_SPLAY\_HEAD** (name, type)

**DMR\_SPLAY\_INITIALIZER** (root)

**DMR\_SPLAY\_INIT** (root)

**DMR\_SPLAY\_ENTRY** (type)

**DMR\_SPLAY\_LEFT** (elm, field)

**DMR\_SPLAY\_RIGHT** (elm, field)

**DMR\_SPLAY\_ROOT** (head)

**DMR\_SPLAY\_EMPTY** (head)

**DMR\_SPLAY\_ROTATE\_RIGHT** (head, tmp, field)

**DMR\_SPLAY\_LINKLEFT** (head, tmp, field)

**DMR\_SPLAY\_LINKRIGHT** (head, tmp, field)

**DMR\_SPLAY\_ASSEMBLE** (head, node, left, right, field)

**DMR\_SPLAY\_GENERATE** (name, type, field, cmp)

**DMR\_SPLAY\_NEGINF**

**DMR\_SPLAY\_INF**

**DMR\_SPLAY\_INSERT** (name, x, y)

**DMR\_SPLAY\_REMOVE** (name, x, y)

**DMR\_SPLAY\_FIND** (name, x, y)

**DMR\_SPLAY\_NEXT** (name, x, y)

**DMR\_SPLAY\_MIN** (name, x)

**DMR\_SPLAY\_MAX** (name, x)

**DMR\_SPLAY\_FOREACH** (x, name, head)

## Red-black tree

A red-black tree is a binary search tree with the node color as an extra attribute. It fulfills a set of conditions:

- every search path from the root to a leaf consists of the same number of black nodes,
- each red node (except for the root) has a black parent,
- each leaf node is black.

Every operation on a red-black tree is bounded as  $\mathcal{O}(\log n)$ . The maximum height of a red-black tree is  $2 \log (n + 1)$ .

**DMR\_RB\_BLACK**

**DMR\_RB\_RED**

**DMR\_RB\_NEGINF**

**DMR\_RB\_INF**

**DMR\_RB\_HEAD** (name, type)

**DMR\_RB\_INITIALIZER** (root)

**DMR\_RB\_INIT** (root)

**DMR\_RB\_ENTRY** (type)

**DMR\_RB\_LEFT** (elm, field)

**DMR\_RB\_RIGHT** (elm, field)

**DMR\_RB\_PARENT** (elm, field)

**DMR\_RB\_COLOR** (elm, field)

**DMR\_RB\_ROOT** (head)

**DMR\_RB\_EMPTY** (head)

**DMR\_RB\_SET** (elm, parent, field)

**DMR\_RB\_SET\_BLACKRED** (black, red, field)

**DMR\_RB\_AUGMENT** (x)

**DMR\_RB\_ROTATE\_LEFT** (head, elm, tmp, field)

**DMR\_RB\_ROTATE\_RIGHT** (head, elm, tmp, field)

**DMR\_RB\_PROTOTYPE** (name, type, field, cmp)

**DMR\_RB\_PROTOTYPE\_STATIC** (name, type, field, cmp)

**DMR\_RB\_GENERATE** (name, type, field, cmp)  
**DMR\_RB\_GENERATE\_STATIC** (name, type, field, cmp)  
**DMR\_RB\_INSERT** (name, x, y)  
**DMR\_RB\_REMOVE** (name, x, y)  
**DMR\_RB\_FIND** (name, x, y)  
**DMR\_RB\_NFIND** (name, x, y)  
**DMR\_RB\_NEXT** (name, x, y)  
**DMR\_RB\_PREV** (name, x, y)  
**DMR\_RB\_MIN** (name, x)  
**DMR\_RB\_MAX** (name, x)  
**DMR\_RB\_FOREACH** (x, name, head)  
**DMR\_RB\_FOREACH\_FROM** (x, name, y)  
**DMR\_RB\_FOREACH\_SAFE** (x, name, head, y)  
**DMR\_RB\_FOREACH\_REVERSE** (x, name, head)  
**DMR\_RB\_FOREACH\_REVERSE\_FROM** (x, name, y)  
**DMR\_RB\_FOREACH\_REVERSE\_SAFE** (x, name, head, y)





---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## D

- dmr\_bit (C type), 3
- dmr\_bits\_to\_byte (C function), 4
- dmr\_bits\_to\_bytes (C function), 4
- dmr\_byte\_to\_binary (C function), 4
- dmr\_byte\_to\_bits (C function), 4
- dmr\_bytes\_to\_bits (C function), 4
- dmr\_color\_code (C type), 7
- dmr\_crc16 (C function), 4
- dmr\_crc16\_finish (C function), 4
- dmr\_crc32 (C function), 4
- dmr\_crc32\_finish (C function), 4
- DMR\_CRC8\_MASK\_TERMINATOR\_WITH\_LC (C macro), 4
- DMR\_CRC8\_MASK\_VOICE\_LC (C macro), 4
- DMR\_CRC8\_MASK\_VOICE\_PI (C macro), 4
- dmr\_crc9 (C function), 4
- dmr\_crc9\_finish (C function), 4
- dmr\_data\_type (C type), 7
- DMR\_DATA\_TYPE\_COUNT (C macro), 8
- DMR\_DATA\_TYPE\_CSBK (C variable), 7
- DMR\_DATA\_TYPE\_DATA\_HEADER (C variable), 7
- DMR\_DATA\_TYPE\_IDLE (C variable), 7
- DMR\_DATA\_TYPE\_INVALID (C variable), 7
- DMR\_DATA\_TYPE\_MBC\_CONTINUATION (C variable), 7
- DMR\_DATA\_TYPE\_MBC\_HEADER (C variable), 7
- dmr\_data\_type\_name (C function), 9
- dmr\_data\_type\_name\_short (C function), 9
- DMR\_DATA\_TYPE\_RATE12\_DATA (C variable), 7
- DMR\_DATA\_TYPE\_RATE34\_DATA (C variable), 7
- DMR\_DATA\_TYPE\_SYNC\_VOICE (C variable), 7
- DMR\_DATA\_TYPE\_TERMINATOR\_WITH\_LC (C variable), 7
- DMR\_DATA\_TYPE\_VOICE (C variable), 7
- DMR\_DATA\_TYPE\_VOICE\_LC (C variable), 7
- DMR\_DATA\_TYPE\_VOICE\_PI (C variable), 7
- DMR\_DATA\_TYPE\_VOICE\_SYNC (C variable), 7
- dmr\_dibit (C type), 3
- dmr\_dump\_packet (C function), 9
- dmr\_dump\_parsed\_packet (C function), 9
- dmr\_fid (C type), 7
- DMR\_FID\_DMRA (C variable), 7
- DMR\_FID\_ETSI (C variable), 7
- DMR\_FID\_HYTERA (C variable), 7
- DMR\_FID\_MOTOROLA (C variable), 7
- dmr\_fid\_name (C function), 9
- dmr\_flco (C type), 7
- DMR\_FLCO\_GROUP (C variable), 7
- DMR\_FLCO\_INVALID (C variable), 7
- dmr\_flco\_name (C function), 9
- DMR\_FLCO\_PRIVATE (C variable), 7
- dmr\_free (C function), 6
- dmr\_homebrew (C type), 10
  - dmr\_homebrew.bind\_port (C member), 10
  - dmr\_homebrew.config (C member), 10
    - dmr\_homebrew.config.altitude (C member), 10
    - dmr\_homebrew.config.call (C member), 10
    - dmr\_homebrew.config.color\_code (C member), 10
    - dmr\_homebrew.config.description (C member), 10
    - dmr\_homebrew.config.latitude (C member), 10
    - dmr\_homebrew.config.location (C member), 10
    - dmr\_homebrew.config.longitude (C member), 10
    - dmr\_homebrew.config.package\_id (C member), 10
    - dmr\_homebrew.config.repeater\_id (C member), 10
    - dmr\_homebrew.config.rx\_freq (C member), 10
    - dmr\_homebrew.config.software\_id (C member), 10
    - dmr\_homebrew.config.tx\_freq (C member), 10
    - dmr\_homebrew.config.tx\_power (C member), 10
    - dmr\_homebrew.config.url (C member), 10
  - dmr\_homebrew.id (C member), 10
  - dmr\_homebrew.last\_ping (C member), 11
  - dmr\_homebrew.peer\_port (C member), 10
  - dmr\_homebrew.rrq (C member), 11
  - dmr\_homebrew.rxq (C member), 11
  - dmr\_homebrew.secret (C member), 10
  - dmr\_homebrew.sock (C member), 10
  - dmr\_homebrew.state (C member), 10
  - dmr\_homebrew.trq (C member), 11
  - dmr\_homebrew.txq (C member), 11
  - dmr\_homebrew\_auth (C function), 11

DMR\_HOMEBREW\_AUTH\_CONFIG (C variable), 10  
 DMR\_HOMEBREW\_AUTH\_DONE (C variable), 10  
 DMR\_HOMEBREW\_AUTH\_FAILED (C variable), 10  
 DMR\_HOMEBREW\_AUTH\_INIT (C variable), 10  
 DMR\_HOMEBREW\_AUTH\_NONE (C variable), 10  
 dmr\_homebrew\_close (C function), 11  
 dmr\_homebrew\_new (C function), 11  
 dmr\_homebrew\_parse\_dmr (C function), 11  
 DMR\_HOMEBREW\_PORT (C macro), 11  
 dmr\_homebrew\_protocol (C variable), 11  
 dmr\_homebrew\_read (C function), 11  
 dmr\_homebrew\_send (C function), 11  
 dmr\_homebrew\_send\_buf (C function), 11  
 dmr\_homebrew\_send\_raw (C function), 11  
 dmr\_homebrew\_state (C type), 10  
 dmr\_id (C type), 7  
 DMR\_LIST\_EMPTY (C macro), 13  
 DMR\_LIST\_END (C macro), 13  
 DMR\_LIST\_ENTRY (C macro), 13  
 DMR\_LIST\_FIRST (C macro), 13  
 DMR\_LIST\_FOREACH (C macro), 13  
 DMR\_LIST\_FOREACH\_SAFE (C macro), 13  
 DMR\_LIST\_HEAD (C macro), 12  
 DMR\_LIST\_HEAD\_INITIALIZER (C macro), 12  
 DMR\_LIST\_INIT (C macro), 13  
 DMR\_LIST\_INSERT\_AFTER (C macro), 13  
 DMR\_LIST\_INSERT\_BEFORE (C macro), 13  
 DMR\_LIST\_INSERT\_HEAD (C macro), 13  
 DMR\_LIST\_NEXT (C macro), 13  
 DMR\_LIST\_REMOVE (C macro), 13  
 DMR\_LIST\_REPLACE (C macro), 13  
 dmr\_log (C function), 6  
 DMR\_LOG\_BOOL (C macro), 5  
 dmr\_log\_cb (C function), 6  
 dmr\_log\_cb\_get (C function), 6  
 dmr\_log\_cb\_t (C type), 5  
 dmr\_log\_color (C function), 5  
 dmr\_log\_color\_set (C function), 5  
 dmr\_log\_critical (C function), 6  
 dmr\_log\_debug (C function), 6  
 dmr\_log\_errno (C function), 6  
 dmr\_log\_error (C function), 6  
 dmr\_log\_info (C function), 6  
 dmr\_log\_message (C function), 6  
 DMR\_LOG\_MESSAGE\_MAX (C macro), 5  
 dmr\_log\_messagev (C function), 6  
 dmr\_log\_mutex (C function), 6  
 dmr\_log\_prefix (C function), 5  
 dmr\_log\_prefix\_set (C function), 5  
 DMR\_LOG\_PRIORITIES (C variable), 5  
 dmr\_log\_priority (C function), 5  
 DMR\_LOG\_PRIORITY\_CRITICAL (C variable), 5  
 DMR\_LOG\_PRIORITY\_DEBUG (C variable), 5  
 DMR\_LOG\_PRIORITY\_ERROR (C variable), 5  
 DMR\_LOG\_PRIORITY\_INFO (C variable), 5  
 dmr\_log\_priority\_reset (C function), 5  
 dmr\_log\_priority\_set (C function), 5  
 dmr\_log\_priority\_t (C type), 5  
 DMR\_LOG\_PRIORITY\_TRACE (C variable), 5  
 DMR\_LOG\_PRIORITY\_WARN (C variable), 5  
 DMR\_LOG\_TIME\_FORMAT (C macro), 5  
 dmr\_log\_trace (C function), 6  
 dmr\_log\_warn (C function), 6  
 dmr\_malloc (C function), 6  
 dmr\_malloc\_size (C function), 6  
 DMR\_NULL\_CHECK (C macro), 6  
 DMR\_NULL\_CHECK\_FREE (C macro), 6  
 dmr\_packet (C type), 8  
 DMR\_PACKET\_BITS (C macro), 8  
 dmr\_packet\_cb (C type), 8  
 dmr\_packet\_data (C type), 8  
 dmr\_packet\_data\_block (C type), 8  
 dmr\_packet\_data\_block\_bits (C type), 8  
 dmr\_packet\_decode (C function), 9  
 DMR\_PACKET\_LEN (C macro), 8  
 dmr\_packetq (C type), 9  
 dmr\_packetq.head (C member), 9  
 dmr\_packetq\_add (C function), 9  
 dmr\_packetq\_add\_packet (C function), 9  
 dmr\_packetq\_entry (C type), 9  
 dmr\_packetq\_entry.entries (C member), 9  
 dmr\_packetq\_entry.parsed (C member), 9  
 dmr\_packetq\_foreach (C function), 9  
 dmr\_packetq\_foreach\_packet (C function), 9  
 dmr\_packetq\_new (C function), 9  
 dmr\_packetq\_shift (C function), 9  
 dmr\_packetq\_shift\_packet (C function), 9  
 dmr\_palloc (C function), 6  
 dmr\_palloc\_size (C function), 6  
 dmr\_parsed\_packet (C type), 8  
 dmr\_parsed\_packet.color\_code (C member), 8  
 dmr\_parsed\_packet.data\_type (C member), 8  
 dmr\_parsed\_packet.dst\_id (C member), 8  
 dmr\_parsed\_packet.flco (C member), 8  
 dmr\_parsed\_packet.packet (C member), 8  
 dmr\_parsed\_packet.parsed (C member), 8  
 dmr\_parsed\_packet.repeater\_id (C member), 8  
 dmr\_parsed\_packet.sequence (C member), 8  
 dmr\_parsed\_packet.src\_id (C member), 8  
 dmr\_parsed\_packet.stream\_id (C member), 8  
 dmr\_parsed\_packet.ts (C member), 8  
 dmr\_parsed\_packet.voice\_frame (C member), 8  
 dmr\_parsed\_packet\_cb (C type), 8  
 dmr\_payload\_bits (C function), 9  
 dmr\_raw (C type), 15  
 dmr\_raw.allocated (C member), 15  
 dmr\_raw.buf (C member), 15  
 dmr\_raw.len (C member), 15

- dmr\_raw\_add (C function), 15
- dmr\_raw\_add\_hex (C function), 15
- dmr\_raw\_add\_uint16 (C function), 16
- dmr\_raw\_add\_uint24 (C function), 16
- dmr\_raw\_add\_uint32 (C function), 16
- dmr\_raw\_add\_uint32\_le (C function), 16
- dmr\_raw\_add\_uint64 (C function), 16
- dmr\_raw\_add\_uint8 (C function), 15
- dmr\_raw\_add\_xuint16 (C function), 16
- dmr\_raw\_add\_xuint24 (C function), 16
- dmr\_raw\_add\_xuint32 (C function), 16
- dmr\_raw\_add\_xuint32\_le (C function), 16
- dmr\_raw\_add\_xuint64 (C function), 16
- dmr\_raw\_add\_xuint8 (C function), 16
- dmr\_raw\_addf (C function), 16
- dmr\_raw\_cb (C type), 15
- dmr\_raw\_free (C function), 15
- dmr\_raw\_grow (C function), 16
- dmr\_raw\_grow\_add (C function), 16
- dmr\_raw\_new (C function), 15
- dmr\_raw\_reset (C function), 16
- dmr\_raw\_zero (C function), 16
- dmr\_rawq (C type), 15
- dmr\_rawq.head (C member), 15
- dmr\_rawq.limit (C member), 15
- dmr\_rawq\_add (C function), 16
- dmr\_rawq\_addb (C function), 16
- dmr\_rawq\_each (C function), 16
- dmr\_rawq\_empty (C function), 17
- dmr\_rawq\_free (C function), 16
- dmr\_rawq\_new (C function), 16
- dmr\_rawq\_shift (C function), 17
- dmr\_rawq\_size (C function), 17
- dmr\_rawq\_unshift (C function), 17
- DMR\_RB\_AUGMENT (C macro), 18
- DMR\_RB\_BLACK (C macro), 18
- DMR\_RB\_COLOR (C macro), 18
- DMR\_RB\_EMPTY (C macro), 18
- DMR\_RB\_ENTRY (C macro), 18
- DMR\_RB\_FIND (C macro), 19
- DMR\_RB\_FOREACH (C macro), 19
- DMR\_RB\_FOREACH\_FROM (C macro), 19
- DMR\_RB\_FOREACH\_REVERSE (C macro), 19
- DMR\_RB\_FOREACH\_REVERSE\_FROM (C macro), 19
- DMR\_RB\_FOREACH\_REVERSE\_SAFE (C macro), 19
- DMR\_RB\_FOREACH\_SAFE (C macro), 19
- DMR\_RB\_GENERATE (C macro), 18
- DMR\_RB\_GENERATE\_STATIC (C macro), 19
- DMR\_RB\_HEAD (C macro), 18
- DMR\_RB\_INF (C macro), 18
- DMR\_RB\_INIT (C macro), 18
- DMR\_RB\_INITIALIZER (C macro), 18
- DMR\_RB\_INSERT (C macro), 19
- DMR\_RB\_LEFT (C macro), 18
- DMR\_RB\_MAX (C macro), 19
- DMR\_RB\_MIN (C macro), 19
- DMR\_RB\_NEGINF (C macro), 18
- DMR\_RB\_NEXT (C macro), 19
- DMR\_RB\_NFIND (C macro), 19
- DMR\_RB\_PARENT (C macro), 18
- DMR\_RB\_PREV (C macro), 19
- DMR\_RB\_PROTOTYPE (C macro), 18
- DMR\_RB\_PROTOTYPE\_STATIC (C macro), 18
- DMR\_RB\_RED (C macro), 18
- DMR\_RB\_REMOVE (C macro), 19
- DMR\_RB\_RIGHT (C macro), 18
- DMR\_RB\_ROOT (C macro), 18
- DMR\_RB\_ROTATE\_LEFT (C macro), 18
- DMR\_RB\_ROTATE\_RIGHT (C macro), 18
- DMR\_RB\_SET (C macro), 18
- DMR\_RB\_SET\_BLACKRED (C macro), 18
- dmr\_realloc (C function), 6
- DMR\_SIMPLEQ\_CONCAT (C macro), 13
- DMR\_SIMPLEQ\_EMPTY (C macro), 13
- DMR\_SIMPLEQ\_END (C macro), 13
- DMR\_SIMPLEQ\_ENTRY (C macro), 13
- DMR\_SIMPLEQ\_FIRST (C macro), 13
- DMR\_SIMPLEQ\_FOREACH (C macro), 13
- DMR\_SIMPLEQ\_FOREACH\_SAFE (C macro), 13
- DMR\_SIMPLEQ\_HEAD (C macro), 13
- DMR\_SIMPLEQ\_HEAD\_INITIALIZER (C macro), 13
- DMR\_SIMPLEQ\_INIT (C macro), 13
- DMR\_SIMPLEQ\_INSERT\_AFTER (C macro), 13
- DMR\_SIMPLEQ\_INSERT\_HEAD (C macro), 13
- DMR\_SIMPLEQ\_INSERT\_TAIL (C macro), 13
- DMR\_SIMPLEQ\_NEXT (C macro), 13
- DMR\_SIMPLEQ\_REMOVE\_AFTER (C macro), 13
- DMR\_SIMPLEQ\_REMOVE\_HEAD (C macro), 13
- DMR\_SLIST\_EMPTY (C macro), 12
- DMR\_SLIST\_END (C macro), 12
- DMR\_SLIST\_ENTRY (C macro), 12
- DMR\_SLIST\_FIRST (C macro), 12
- DMR\_SLIST\_FOREACH (C macro), 12
- DMR\_SLIST\_FOREACH\_SAFE (C macro), 12
- DMR\_SLIST\_HEAD (C macro), 12
- DMR\_SLIST\_HEAD\_INITIALIZER (C macro), 12
- DMR\_SLIST\_INIT (C macro), 12
- DMR\_SLIST\_INSERT\_AFTER (C macro), 12
- DMR\_SLIST\_INSERT\_HEAD (C macro), 12
- DMR\_SLIST\_NEXT (C macro), 12
- DMR\_SLIST\_REMOVE (C macro), 12
- DMR\_SLIST\_REMOVE\_AFTER (C macro), 12
- DMR\_SLIST\_REMOVE\_HEAD (C macro), 12
- DMR\_SLOT\_TYPE\_BITS (C macro), 3
- dmr\_slot\_type\_decode (C function), 9
- dmr\_slot\_type\_encode (C function), 9
- DMR\_SLOT\_TYPE\_HALF (C macro), 3

DMR\_SPLAY\_ASSEMBLE (C macro), 17  
DMR\_SPLAY\_EMPTY (C macro), 17  
DMR\_SPLAY\_ENTRY (C macro), 17  
DMR\_SPLAY\_FIND (C macro), 18  
DMR\_SPLAY\_FOREACH (C macro), 18  
DMR\_SPLAY\_GENERATE (C macro), 17  
DMR\_SPLAY\_HEAD (C macro), 17  
DMR\_SPLAY\_INF (C macro), 17  
DMR\_SPLAY\_INIT (C macro), 17  
DMR\_SPLAY\_INITIALIZER (C macro), 17  
DMR\_SPLAY\_INSERT (C macro), 17  
DMR\_SPLAY\_LEFT (C macro), 17  
DMR\_SPLAY\_LINKLEFT (C macro), 17  
DMR\_SPLAY\_LINKRIGHT (C macro), 17  
DMR\_SPLAY\_MAX (C macro), 18  
DMR\_SPLAY\_MIN (C macro), 18  
DMR\_SPLAY\_NEGINF (C macro), 17  
DMR\_SPLAY\_NEXT (C macro), 18  
DMR\_SPLAY\_REMOVE (C macro), 18  
DMR\_SPLAY\_RIGHT (C macro), 17  
DMR\_SPLAY\_ROOT (C macro), 17  
DMR\_SPLAY\_ROTATE\_RIGHT (C macro), 17  
dmr\_strdup (C function), 6  
DMR\_TAILQ\_CONCAT (C macro), 15  
DMR\_TAILQ\_EMPTY (C macro), 14  
DMR\_TAILQ\_END (C macro), 14  
DMR\_TAILQ\_ENTRY (C macro), 14  
DMR\_TAILQ\_FIRST (C macro), 14  
DMR\_TAILQ\_FOREACH (C macro), 14  
DMR\_TAILQ\_FOREACH\_REVERSE (C macro), 14  
DMR\_TAILQ\_FOREACH\_REVERSE\_SAFE (C macro), 14  
DMR\_TAILQ\_FOREACH\_SAFE (C macro), 14  
DMR\_TAILQ\_HEAD (C macro), 14  
DMR\_TAILQ\_HEAD\_INITIALIZER (C macro), 14  
DMR\_TAILQ\_INIT (C macro), 14  
DMR\_TAILQ\_INSERT\_AFTER (C macro), 15  
DMR\_TAILQ\_INSERT\_BEFORE (C macro), 15  
DMR\_TAILQ\_INSERT\_HEAD (C macro), 15  
DMR\_TAILQ\_INSERT\_TAIL (C macro), 15  
DMR\_TAILQ\_LAST (C macro), 14  
DMR\_TAILQ\_NEXT (C macro), 14  
DMR\_TAILQ\_PREV (C macro), 14  
DMR\_TAILQ\_REMOVE (C macro), 15  
DMR\_TAILQ\_REPLACE (C macro), 15  
dmr\_tribit (C type), 3  
dmr\_ts (C type), 7  
DMR\_TS1 (C variable), 7  
DMR\_TS2 (C variable), 7  
dmr\_ts\_name (C function), 9  
DMR\_UINT16\_BE (C macro), 3  
DMR\_UINT16\_BE\_PACK (C macro), 3  
DMR\_UINT16\_LE (C macro), 3  
DMR\_UINT16\_LE\_PACK (C macro), 3  
DMR\_UINT32\_BE (C macro), 3  
DMR\_UINT32\_BE\_PACK (C macro), 3  
DMR\_UINT32\_BE\_PACK3 (C macro), 4  
DMR\_UINT32\_BE\_UNPACK (C macro), 3  
DMR\_UINT32\_BE\_UNPACK3 (C macro), 3  
DMR\_UINT32\_LE (C macro), 3  
DMR\_UINT32\_LE\_PACK (C macro), 3  
DMR\_UINT32\_LE\_PACK3 (C macro), 3  
DMR\_XSIMPLEQ\_EMPTY (C macro), 14  
DMR\_XSIMPLEQ\_END (C macro), 14  
DMR\_XSIMPLEQ\_ENTRY (C macro), 14  
DMR\_XSIMPLEQ\_FIRST (C macro), 14  
DMR\_XSIMPLEQ\_FOREACH (C macro), 14  
DMR\_XSIMPLEQ\_FOREACH\_SAFE (C macro), 14  
DMR\_XSIMPLEQ\_HEAD (C macro), 14  
DMR\_XSIMPLEQ\_INIT (C macro), 14  
DMR\_XSIMPLEQ\_INSERT\_AFTER (C macro), 14  
DMR\_XSIMPLEQ\_INSERT\_HEAD (C macro), 14  
DMR\_XSIMPLEQ\_INSERT\_TAIL (C macro), 14  
DMR\_XSIMPLEQ\_NEXT (C macro), 14  
DMR\_XSIMPLEQ\_REMOVE\_AFTER (C macro), 14  
DMR\_XSIMPLEQ\_REMOVE\_HEAD (C macro), 14  
DMR\_XSIMPLEQ\_XOR (C macro), 14