VMS Software

VSI OpenVMS

Media Robot Utility Application Programming Interface Guide

Operating System and Version: VSI OpenVMS IA-64 Version 8.4-1H1 or higher
VSI OpenVMS Alpha Version 8.4-2L1 or higher
VSI OpenVMS x86-64 Version 9.2-1 or higher

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

Media Robot Utility Application Programming Interface Guide

Legal Notice

Table of Contents

Preface ..... **ix**
1. About VSI ix
2. OpenVMS Documentation ix
3. VSI Encourages Your Comments ix

Chapter 1. Media Robot Driver Library **1**
1.1. Description 1
1.1.1. Element Address Naming 1
1.1.2. Description 1
1.2. MRD Routine Summary 2
1.2.1. Common Routines 2
1.2.2. Operating System Routines 4
1.2.3. About Return Values 6
1.2.3.1. Common Values 6
1.2.3.2. Windows 2000/Windows XP Codes 12
1.2.3.3. Tru64 UNIX Codes 13
1.2.3.4. OpenVMS Codes 15
1.2.4. Related Information 16

Chapter 2. mrd_eject **19**
2.1. Media Robot Driver Library 19
2.2. Parameters 19
2.3. Description 19
2.3.1. Example 20
2.3.2. Return Values 21
2.3.2.1. Common Codes 21
2.3.2.2. Windows 2000/Windows XP Codes 22
2.3.2.3. Tru64 UNIX Codes 22
2.3.2.4. OpenVMS Codes 23
2.3.3. Related Information 24

Chapter 3. mrd_find_cartridge **25**
3.1. Media Robot Driver Library 25
3.2. Parameters 25
3.3. Description 26
3.3.1. Element Info 26
3.3.2. Example 27
3.3.3. Return Values 28
3.3.3.1. Common Values 28
3.3.3.2. Windows 2000/Windows XP Codes 29
3.3.3.3. Tru64 UNIX Codes 30
3.3.3.4. OpenVMS Codes 30
3.3.4. Restrictions 31
3.3.5. Related Information 31

Chapter 4. mrd_home **33**
4.1. Media Robot Driver Library 33
4.2. Parameters 33
4.3. Description 34
4.3.1. Example 34
4.3.2. Return Values 36
4.3.2.1. Common Codes 36

4.3.2.2. Windows 2000/Windows XP Codes 36
4.3.2.3. Tru64 UNIX Codes 37
4.3.2.4. OpenVMS Codes 38
4.3.3. Related Information 38
Chapter 5. mrd_initialize 41
5.1. Media Robot Driver Library 41
5.2. Parameters 41
5.3. Description 41
5.3.1. Example 41
5.3.2. Return Values 42
5.3.2.1. Common Codes 42
5.3.3. Related Information 44
Chapter 6. mrd_initialize_element 45
6.1. Media Robot Driver Library 45
6.2. Parameters 45
6.3. Description 45
6.3.1. Example 46
6.3.2. Return Values 47
6.3.2.1. Common Codes 47
6.3.2.2. Windows 2000/Windows XP Codes 48
6.3.2.3. Tru64 UNIX Codes 48
6.3.2.4. OpenVMS Codes 48
6.3.3. Related Information 48
Chapter 7. mrd_inject 49
7.1. Media Robot Driver Library 49
7.2. Parameters 49
7.3. Description 49
7.3.1. Example 50
7.3.2. Return Values 51
7.3.2.1. Common Codes 51
7.3.2.2. Windows 2000/Windows XP Codes 52
7.3.2.3. Tru64 UNIX Codes 53
7.3.2.4. OpenVMS Codes 53
7.3.3. Related Information 54
Chapter 8. mrd_load 55
8.1. Media Robot Driver Library 55
8.2. Parameters 55
8.3. Description 55
8.3.1. Example 56
8.3.2. Return Values 57
8.3.2.1. Common Codes 57
8.3.2.2. Windows 2000/Windows XP Codes 58
8.3.2.3. Tru64 UNIX Codes 58
8.3.2.4. OpenVMS Codes 59
8.3.3. Related Information 59
Chapter 9. mrd_lock 61
9.1. Media Robot Driver Library 61
9.2. Parameters 61
9.3. Description 61
9.3.1. Return Values 61

9.3.1.1. Common Codes 62
9.3.1.2. Windows 2000/Windows XP Codes 62
9.3.1.3. Tru64 UNIX Codes 63
9.3.1.4. OpenVMS Codes 63
9.3.2. Related Information 64
Chapter 10. mrd_map_element 65
10.1. Media Robot Driver Library 65
10.2. Parameters 65
10.3. Description 65
10.3.1. Example 65
10.3.2. Return Values 67
10.3.3. Related Information 67
Chapter 11. mrd_message 69
11.1. Media Robot Driver Library 69
11.2. Parameters 69
11.3. Description 69
11.3.1. Codes Translated 70
11.3.2. Example 71
11.3.3. Return Values 72
11.3.4. Related Information 72
Chapter 12. mrd_move 73
12.1. Media Robot Driver Library 73
12.2. Parameters 73
12.3. Description 74
12.3.1. Example 74
12.3.2. Return Values 75
12.3.2.1. Common Codes 75
12.3.2.2. Windows 2000/Windows XP Codes 76
12.3.2.3. Tru64 UNIX Codes 77
12.3.2.4. OpenVMS Codes 78
12.4. Related Information 78
Chapter 13. mrd_move_medium 81
13.1. Media Robot Driver Library 81
13.2. Parameters 81
13.3. Description 81
13.3.1. Absolute Element Addresses 82
13.3.2. Example 83
13.3.3. Return Values 84
13.3.3.1. Common Codes 85
13.3.3.2. Windows 2000/Windows XP Codes 85
13.3.3.3. Tru64 UNIX Codes 86
13.3.3.4. OpenVMS Codes 86
13.3.4. Related Information 86
Chapter 14. mrd_position 87
14.1. Media Robot Driver Library 87
14.2. Parameters 87
14.3. Description 87
14.3.1. Example 88
14.3.2. Return Values 89
14.3.2.1. Windows 2000/Windows XP Codes 91

14.3.2.2. Tru64 UNIX Codes 91
14.3.2.3. OpenVMS Codes 92
14.3.3. Related Information 93
Chapter 15. mrd_position_to_element 95
15.1. Media Robot Driver Library 95
15.2. Parameters 95
15.3. Description 95
15.3.1. Absolute Element Addresses 96
15.3.2. Example 97
15.3.3. Return Values 99
15.3.3.1. Common Codes 99
15.3.3.2. Windows 2000/Windows XP Codes 100
15.3.3.3. Tru64 UNIX Codes 100
15.3.3.4. OpenVMS Codes 101
15.3.4. Related Information 101
Chapter 16. mrd_prevent_allow 103
16.1. Media Robot Driver Library 103
16.2. Parameters 103
16.3. Description 103
16.3.1. Example 104
16.3.2. Return Values 105
16.3.2.1. Common Codes 105
16.3.2.2. Windows 2000/Windows XP Codes 106
16.3.2.3. Tru64 UNIX Codes 106
16.3.2.4. OpenVMS Codes 106
16.3.3. Related Information 106
Chapter 17. mrd_ready 107
17.1. Media Robot Driver Library 107
17.2. Parameters 107
17.3. Description 107
17.3.1. Example 108
17.3.2. Return Values 109
17.3.2.1. Common Codes 109
17.3.2.2. Windows 2000/Windows XP Codes 110
17.3.2.3. Tru64 UNIX Codes 110
17.3.2.4. OpenVMS Codes 111
17.3.3. Related Information 111
Chapter 18. mrd_ready_inport 113
18.1. Media Robot Driver Library 113
18.2. Parameters 113
18.3. Description 113
18.3.1. Example 113
18.3.2. Return Values 114
18.3.2.1. Common Codes 114
18.3.2.2. Windows 2000/Windows XP Codes 115
18.3.3. Related Information 116
Chapter 19. mrd_read_element_status 119
19.1. Media Robot Driver Library 119
19.2. Parameters 119
19.3. Description 119

19.3.1. Absolute Element Addresses 120
19.3.2. Example 121
19.3.3. Return Values 126
19.3.3.1. Common Codes 126
19.3.3.2. Windows 2000/Windows XP Codes 127
19.3.3.3. Tru64 UNIX Codes 127
19.3.3.4. OpenVMS Codes 127
19.3.4. Related Information 127
Chapter 20. mrd_request_sense 129
20.1. Media Robot Driver Library 129
20.2. Parameters 129
20.3. Description 129
20.3.1. Example 130
20.3.2. Return Values 131
20.3.2.1. Common Codes 132
20.3.2.2. Windows 2000/Windows XP Codes 135
20.3.2.3. Tru64 UNIX Codes 136
20.3.2.4. OpenVMS Codes 137
20.3.3. Related Information 138
Chapter 21. mrd_show 139
21.1. Media Robot Driver Library 139
21.2. Parameters 139
21.3. Description 140
21.3.1. Element Info 140
21.3.2. Example 140
21.3.3. Return Values 142
21.3.3.1. Common Codes 142
21.3.3.2. Windows 2000/Windows XP Codes 144
21.3.3.3. Tru64 UNIX Codes 144
21.3.3.4. OpenVMS Codes 145
21.4. Related Information 147
Chapter 22. mrd_startup 149
22.1. Media Robot Driver Library 149
22.2. Parameters 149
22.3. Description 149
22.3.1. Example 152
22.3.2. Return Values 153
22.3.2.1. Common Codes 153
22.3.2.2. Windows 2000/Windows XP Codes 154
22.3.2.3. Tru64 UNIX Codes 155
22.3.2.4. OpenVMS Codes 157
22.4. Related Information 158
Chapter 23. mrd_test_unit_ready 161
23.1. Media Robot Driver Library 161
23.2. Parameters 161
23.3. Description 161
23.3.1. Example 162
23.3.2. Return Values 164
23.3.2.1. Common Codes 164
23.3.2.2. Windows 2000/Windows XP Codes 164

23.3.2.3. Tru64 UNIX Codes 165
23.3.2.4. OpenVMS Codes 165
23.3.3. Related Information 166
23.3.4. Tru64 UNIX Restriction 166
Chapter 24. mrd_unload 167
24.1. Media Robot Driver Library 167
24.2. Parameters 167
24.3. Description 167
24.3.1. Tru64 UNIX 168
24.4. OpenVMS Example 171
24.4.1. Return Values 172
24.4.1.1. Common Codes 172
24.4.1.2. OpenVMS Codes 173
24.4.2. Related Information 173
Chapter 25. mrd_utility 175
25.1. Media Robot Driver Library 175
25.2. Parameters 175
25.3. Description 175
25.3.1. Example 176
25.3.2. Return Values 178
25.3.2.1. Common Codes 178
25.3.2.2. Windows 2000/Windows XP Codes 181
25.3.2.3. Tru64 UNIX Codes 182
25.3.2.4. OpenVMS Codes 183
25.3.3. Related Information 184

Preface

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

3. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: <docinfo@vmssoftware.com>. Users who have VSI OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product.

Chapter 1. Media Robot Driver Library

Media Robot Driver Library is a programming interface for controlling SCSI-2 medium-changers.

1.1. Description

The Media Robot Driver library is a callable interface for controlling SCSI-2 medium-changers. The interface consists of two include files and an object library which are installed in an operating system specific location. The operating system specific locations are shown in the table below.

Table 1.1. Library File Locations

/Windows 2000/Windows XP	\ Program Files\ MRU\ mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	SYS$LIBRARY:MRD$RTL.EXE

The distribution also includes examples showing how each callable routine can be used and a manual page for each routine.

The <mrd_common.h> include file defines data structures used to provide information about the medium-changer and its elements. The <mrd_common.h> file also defines a large number of symbolic constants for element type codes, offsets within SCSI structures, masks for SCSI bit fields, and other useful structures. The <mrd_common.h> file also includes prototype definitions of all the medium-changer functions provided in the interface.

The <mrd_message.h> include file defines constants for each error code returned by the MRD interface. Function prototypes are also included for routines that will return a string corresponding to the error code.

On Tru64 UNIX, these strings are retrieved from an I18N message catalog that is part of the installed software. Code and routines are also included for words and element exception messages that might be commonly used by a medium-changer application.

1.1.1. Element Address Naming

The first OpenVMS implementation of MRD supported the TF and TA family DLT media-changers. It used Mass Storage Control Protocol Display commands to indicate what cartridge should be moved. The MSCP uses cartridge address names instead of numbers as SCSI does. When SCSI support was added to the MRD, the convention of using strings for the address was kept and thus it has been since.

1.1.2. Description

In the common interface example programs, the character strings for the addresses are taken directly from the command line arguments and no special formatting is necessary. But, in practice, a program will probably keep SCSI addresses in numeric form and will have to convert those to strings. In the MRU command line interface and graphic user interface we use sprintf(3) for this:

```
int element_number ;
```

```
char element[MRD_NAME_SIZE+1] ;

element_number = 5 ;

sprintf(element, "%d", element_number) ;
```

1.2. MRD Routine Summary

The media robot driver library routines comprise two categories, the common routines and operating system specific routines.

1.2.1. Common Routines

The following list identifies the common routines.

- **mrd_eject(3mrd)**
- **mrd_find_cartridge(3mrd)**
- **mrd_home(3mrd)**
- **mrd_initialize(3mrd)**
- **mrd_inject(3mrd)**
- **mrd_load(3mrd)**
- **mrd_lock(3mrd)**
- **mrd_map_element(3mrd)**
- **mrd_move(3mrd)**
- **mrd_position(3mrd)**
- **mrd_ready_inport(3mrd)**
- **mrd_scsi_decode(3mrd)**
- **mrd_startup(3mrd)**
- **mrd_show(3mrd)**
- **mrd_shutdown(3mrd)**
- **mrd_strelement(3mrd)**
- **mrd_strexcept(3mrd)**
- **mrd_unload(3mrd)**

The common routines will open a robot to perform their operations. All these routines will close the robot when successfully completed, except for **mrd_ show(3mrd)**. The **mrd_show(3mrd)** routine closes the robot only when it encounters an error.

The routine **mrd_startup(3mrd)** is used to open a medium-changer. It will fill in a **robot_info_t** data structure that contains the number of elements of each type, their addresses and the medium-changer SCSI Inquiry data. Thus, it is unnecessary (and often not desirable) to keep the robot open while it is being used. The routine **mrd_shutdown(3mrd)** can be used to close the robot. Aside from closing the file and setting the *channel* field to BAD_CHANNEL, it has no effect on the other data in the **robot_info_t** data structure.

Use the **mrd_show(3mrd)** routine to obtain information about the contents and state of the slots, drive, ports and transports of the medium-changer. The **mrd_ show(3mrd)** routine will open a robot, but it will also work if the robot is already open when the routine is called. For each element requested, an **element_info_t** data structure will be set if the element exists. The **mrd_show(3mrd)** function will accept the address of a **robot_info_t** data structure. If the robot has already been opened by **mrd_startup(3mrd)**, this open robot will be used by the routine. If the robot isn't open (indicated by the *channel* field set to BAD_CHANNEL), the medium-changer indicated by the *robot_name* will be opened. If the routine completes successfully, the medium-changer will remain open. On an error, the medium-changer will be closed and the *channel* field reset to BAD_CHANNEL. By keeping the medium-changer open, multiple calls can be made to **mrd_ show(3mrd)** without incurring the time to call **mrd_startup(3mrd)** each time.

The routine **mrd_move(3mrd)** is a general interface to the SCSI Move Medium command. It allows the specification of source and destination elements for the move, whether the medium should be inverted and an optional volume tag. On medium-changers which have a vision system to read bar-codes, the volume tag can be used to verify that the medium in the source slot is the one desired.

The routines **mrd_load(3mrd)**, **mrd_unload(3mrd)**, **mrd_inject(3mrd)** and **mrd_eject(3mrd)** are specialized interfaces to the SCSI Move Medium command. Load will move a medium from a slot to a drive. Unload will move a medium from a drive to a slot. Inject moves a medium from a port to a slot and Eject from a slot to a port. On the TL82x family of libraries, **mrd_eject(3mrd)** can also be used to clear a medium from the Pass-Through Mechanism.

The routine **mrd_lock(3mrd)** enables sending a SCSI Prevent/Allow Media Removal command. Whether this command is supported, and its effect, depends on the robot.

The routine **mrd_initialize(3mrd)** sends a SCSI Initialize Element Status command. The effect of this command varies among robots, but it typically causes complete reinventory of the medium-changer.

The routine **mrd_position(3mrd)** sends a SCSI Position to Element command.

The routine **mrd_ready_inport(3mrd)** sends a vendor unique, Ready Inport command. On the TL82n family of libraries, this command enables the button which opens the Inport/Outport Device inport door. Other libraries and loaders may silently ignore this command or treat it as an illegal command.

On medium-changers which keep track of a medium's previous element location, the routine **mrd_home(3mrd)** returns a medium to that location.

On medium-changers with vision systems to read bar-code labels, the routine **mrd_find_cartridge(3mrd)** will search for a specified volume tag. The routine will search the entire library, or just a subset of elements according to the arguments used.

The routine **mrd_map_element(3mrd)** accepts an element's absolute address and returns the element type and zero relative address.

The routine **mrd_strstatus(3mrd)** accepts an MRD error status code and returns the corresponding message text. The routine **mrd_strelement(3mrd)** accepts an MRD_ELEMENT code for various

words which apply to SCSI-2 medium-changer elements and returns the corresponding string. The routine **mrd_strexcept(3mrd)** accepts the Additional Sense Code and Additional Sense Code Qualifier for an element with an exception and returns the corresponding message text.

1.2.2. Operating System Routines

The following list identifies the operating system specific routines.

- **mrd_initialize_element(3mrd)**
- **mrd_move_medium(3mrd)**
- **mrd_position_to_element(3mrd)**
- **mrd_prevent_allow(3mrd)**
- **mrd_read_element_status(3mrd)**
- **mrd_ready(3mrd)**
- **mrd_request_sense(3mrd)**
- **mrd_test_unit_ready(3mrd)**

The operating system interface routines can be called directly and share three common traits.

Trait 1

Instead of a medium changer name, they accept a **robot_info_t** data structure that has been opened by **mrd_startup(3mrd)**. This allows them to be called without the repeated start-up time of **mrd_startup(3mrd)** and allows keeping the medium changer open by a single application.

Trait 2

Instead of zero-relative element addresses, these routines all use absolute element addresses. These address can be calculated by adding the zero-relative address of a specific element to the element start address from the **robot_info_t** structure.

For example:

```
/*
 * Given an robot_info_t initialized with mrd_startup(3mrd)
 * or mrd_show(3mrd), an element type and a relative element
 * address, convert it to an absolute address.
 */
convert_relative(robot_info_t *robot_info, int type, int element)
{
 switch( type )
 case SLOT:
  return element + robot_info->slot_start ;
 case TRANSPORT:
  return element + robot_info->transport_start ;
 case DRIVE:
  return element + robot_info->device_start ;
 case PORT:
  return element + robot_info->port_start ;
 default:
  return -1 ;
```

```
  }
}
```

The routine **mrd_move_medium(3mrd)** is used by **mrd_move(3mrd)**, **mrd_ load(3mrd)**, **mrd_unload(3mrd)**, **mrd_eject(3mrd)** and **mrd_inject(3mrd)**. These routines accepts the absolute transport, source and destination element addresses for a Move Medium command, as well as a value to indicate whether the medium should be inverted when moved.

The routine **mrd_read_element_status(3mrd)** is used by **mrd_show(3mrd)** and a variety of internal utility functions. It offers direct access to the SCSI Read Element Status command. However, the data returned is also uninterpreted Read Element Status data, so the application using it must interpret the data for itself. Since **mrd_show(3mrd)** allows keeping the medium changer open as well, it is usually easier to use, except for simple requests.

The routine **mrd_position_to_element(3mrd)** is used by **mrd_position(3mrd)**. It offers direct access to the SCSI Position to Element command, accepting absolute element addresses for the transport and destination elements. It can also invert the transport where this is supported.

The routine **mrd_initialize_element(3mrd)** is used by **mrd_initialize(3mrd)**. It offers direct access to the SCSI Initialize Element Status command.

The routine **mrd_ready(3mrd)** is used by **mrd_ready_inport(3mrd)**. It offers direct access to the SCSI Position to Element command, accepting the absolute addresse of the port to be readied.

The routine **mrd_prevent_allow(3mrd)** is used by **mrd_lock(3mrd)**. It offers direct access to the SCSI Prevent Allow Media Removal command, accepting a value to indicate which is desired.

The **mrd_test_unit_ready(3mrd)** routine performs a SCSI Test Unit Ready command, or equivalent if some other I/O architecture is supported. It is used by the **mrd_startup(3mrd)** and the OpenVMS implementation of **mrd_ ready(3mrd)**.

The **mrd_request_sense(3mrd)** routine performs a SCSI Request Sense command, or equivalent if some other I/O architecture is supported. It is used by all MRD API routines to determine the cause of a command failure.

Trait 3

Finally, these routines accept the address of a **dev_status_t** structure for holding error status, instead of a the *log_info* string used by the other routines. This allows custom formatting of errors.

The **dev_status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

Compiling a Tru64 UNIX Application with MRD

The files <mrd_common.h> and <mrd_message.h> must be included by any source module wishing to use the MRD interface. The library was compiled with the - migrate and default optimization options available with the DEC OSF/1 V1.3 C compiler.

If the calling program is not compiled with the -migrate option, it will be necessary to link with the OTS library, </usr/lib/libots.a>. If not, the following symbols will be unresolved:

```
% make mrd_move
cc -O -c mrd_move.c
cc -O mrd_move.o -lmrd -o mrd_move
ld:U
nresolved:
_OtsDivide64Unsigned
_OtsMove
_OtsDivide32
*** Exit 1
Stop.
```

The subset containing the <libots.a> object library has changed over the versions. In DEC OSF/1 V1.3 it is part of the OTABASE. subset. By DEC OSF/1 V3.0 it had moved to OSFCMPLRS300, where it has remained through Tru64 UNIX V4.0.

1.2.3. About Return Values

Upon successful completion, the Media Robot Driver library routines that access a medium-changer return the value MRD_STATUS_SUCCESS. On a failure, one of the following errors may be returned. The Media Robot Driver library will attempt to map SCSI failures to one of a small group of error codes, but not all errors have been anticipated.

Many of the MRD routines accept a *log_info* argument that is a character array. When a SCSI error occurs, the the Sense Key, additional Sense Code and Additional Sense Code Qualifier are formatted into the space provided. If the error is an operating system specific error, then the text corresponding to the error will be copied into the space provided.

1.2.3.1. Common Values

Common return values.

1. **MRD_STATUS_PARAM**

   This error is returned when a pointer argument passed to an MRD routine is NULL, unless the routine is documented as one allowing a NULL pointer.

2. **MRD_STATUS_CART_INVALID**

For routines that accept a *volume_tag* argument to perform volume tag verification, this error indicates that the volume tag of the media doesn't match that passed to the function.

3. **MRD_STATUS_CART_NOT_AVAIL**

   This error can occur on the TL81n and TL82n family of DLT libraries when the source of a move is a drive and the cartridge in the drive is still on-line. These robots do not allow moving the cartridge until the drive is taken offline.

4. **MRD_STATUS_CART_SIDE_INVALID**

   For routines that use the *cartridge_side* argument, this error indicates that the value is neither one (1) nor two (2).

5. **MRD_STATUS_PORT_INVALID**

   This error is returned when the element address for a port is less than zero or greater than the number of ports.

6. **MRD_STATUS_SLOT_INVALID**

   This error is returned when the element address for a slot is less than zero or greater than the number of slots.

7. **MRD_STATUS_TRANSPORT_INVALID**

   This error is returned when the element address for a transport is less than zero or greater than the number of transports.

8. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives.

9. **MRD_STATUS_INVALID_TYPE**

   For routines that allow the specification of an element type argument, this error indicates that specified type was not one of SLOT, DRIVE, PORT or TRANSPORT.

10. **MRD_STATUS_DESTINATION_FULL**

    On routines that perform a SCSI Move Medium command, this error indicates that the destination element already has a cartridge in it.

11. **MRD_STATUS_SOURCE_EMPTY**

    On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

12. **MRD_STATUS_AUTOCLEAN**

    This error occurs when a SCSI command fails with the ASC set to 0x30 and the ASCQ set to 0x3. On TL8nn libraries supporting Auto-clean, it indicates that a command was attempted while an auto-clean was in progress.

13. **MRD_STATUS_CART_DAMAGED**

This error occurs when a SCSI command fails with the ASC set to 0x30, but the ASCQ is NOT a value of 0x3. The *log_info* will contain the ASCQ.

14. **MRD_STATUS_CART_NOT_FOUND**

    This error is returned by **mrd_ find_cartridge(3mrd)** when it can't find the cartridge with the desired volume tag.

15. **MRD_STATUS_ELEMENT_INVALID**

    This error occurs when a SCSI command fails with the ASC set to 0x21. The *log_info* will contain the ASCQ. This indicates that an invalid element address reached the medium-changer. For example, specifying the 13th slot when only 12 slots are present.

16. **MRD_STATUS_INSFMEM**

    The **mrd_show(3mrd)** and **mrd_find_ cartridge(3mrd)** functions allocate virtual memory using malloc(3) to store temporary element data. If the attempt to allocate the memory fails, these routines will return this error.

17. **MRD_STATUS_NO_ELEMENTS**

    This error occurs in **mrd_ show(3mrd)**, **mrd_find_cartridge(3mrd)** and **mrd_home(3mrd)** when the medium-changer has no elements within the range and type specified by the arguments.

18. **MRD_STATUS_NO_VISION**

    This error occurs in **mrd_find_ cartridge(3mrd)** when the medium-changer has no vision system with which to read bar-code labels.

19. **MRD_STATUS_RES_INVALID**

    This error occurs in **mrd_ home(3mrd)** when the element data returned from **mrd_show(3mrd)** is not valid.

20. **MRD_STATUS_ROBOT_ATTENTION**

    This error occurs when a SCSI command fails with the ASC set to one of 0x29, 0x2A or 0x2F. The *log_info* contains the ASCQ. The SCSI translations for these error codes are:

    - 0x29 - Power-on, Reset or Bus device reset occurred
    - 0x2A - Mode Parameters Changed
    - 0x2F - Command cleared by another initiator

    This error also occurs when the ASC and ASCQ are zero, but the SCSI sense key is 6h.

21. **MRD_STATUS_ROBOT_DOOR_OPENED**

    This occurs when a SCSI command fails with the ASC set to 0x80 and the ASCQ set to 0x0. On TL8nn libraries this typically indicates that the cabinet door was opened during a command operation.

22. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

This error occurs for a variety of reasons.

It is used when a sanity check fails in the code that attempts to move a cartridge to the Pass-Through Mechanism, when the robot type isn't a TL82n.

It is used in the **mrd_lock(3mrd)** code when the value is not one of ALLOW_REMOVAL or PREVENT_REMOVAL.

It is used when the medium changer does not support the Prevent/Allow Medium Removal command or the lock value is not one or zero. The specific cause can be determined by examining the ASC/ASCQ values in the *status* data.

It is used when a call to **mrd_initialize_element(3mrd)** is issued against a medium changer that does not support the Initialize Element Status command.

It is used when the medium changer does not support the Position To Element command. The seven and five slot DLT loaders do not support the command, though the TL820 and TL810 family libraries do. Some models of TLZ6L and TLZ7L do not support the command and may take a long time to fail.

It is used when the medium changer does not support the Ready Inport command.

The TL820 family of DLT libraries support this command. The TL810 family of DLT libraries allows this command to succeed, but it doesn't perform any function.

It is also used for a SCSI command failure, when the ASC is set to one of:

- 0x1A - Parameter list length error
- 0x20 - Invalid command operation code
- 0x22 - Unsupported command
- 0x24 - Illegal field in CDB
- 0x25 - Logical unit not supported
- 0x26 - Threshold parameters not supported
- 0x28 - Import or Export element accessed
- 0x2C - Command sequence error
- 0x39 - Saving parameters not supported
- 0x3D - Invalid bits in Identify message
- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

23. **MRD_STATUS_ROBOT_MECH_ERROR**

    This error occurs as the result of a SCSI command failure, when the ASC is set to one of:

    - 0x15 - Positioning error.

- 0x8B - Vendor unique; Pass-through mechanism errors on the TL82n

24. **MRD_STATUS_SOURCE_INVALID**

    This error occurs in **mrd_home(3mrd)** when the return address in the element data isn't valid.

25. **MRD_STATUS_VENDOR_UNIQUE_ERROR**

    This error occurs when the internal routine used to decode SCSI-2 errors encounters an error that it has not been written to antipicate.

    This error is also returned when the ASC is zero and the ASCQ is not one of zero or six, and when ASC/ASCQ are both zero and the *key* is 9h.

26. **MRD_STATUS_NO_SENSE**

    This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc, ascq* and *key* values are all zero (0). It is also returned when the *key* value is less than zero or greater than 15.

27. **MRD_STATUS_RECOVERED_ERROR**

    This error occurs when a SCSI device returns only a sense key of 1h. This indicates that although a command successfully completed, the target device had performed some internal error recovery.

28. **MRD_STATUS_MEDIUM_ERROR**

    This error occurs when ASC and ASCQ are zero, but the sense key is 3h. This occurs when the target encounters a nonrecoverable error due to a flaw in the medium.

29. **MRD_STATUS_ROBOT_HW_ERROR**

    This error occurs when ASC and ASCQ are zero, but the sense key is 4h. This occurs when the target encounters a nonrecoverable hardware error.

30. **MRD_STATUS_DATA_PROTECT**

    This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is seven (7).

31. **MRD_STATUS_BLANK_CHECK**

    This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is eight (8).

32. **MRD_STATUS_COPY_ABORTED**

    This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is ten (10).

33. **MRD_STATUS_SENSE_EQUAL**

    This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Ch (12).

34. **MRD_STATUS_VOLUME_OVERFLOW**

This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Dh (13).

35. **MRD_STATUS_MISCOMPARE**

    This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Eh (14).

36. **MRD_STATUS_SENSE_RESERVED**

    This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Fh (15).

37. **MRD_STATUS_SCSI_CHECK**

    The SCSI Check Condition error should never occur. It indicates that it is safe to use a Request Sense command and that you are likely to get a different error.

38. **MRD_STATUS_SCSI_CONDMET**

    The SCSI Condition Met status indicates a SCSI command completed with the status "Condition Met".

39. **MRD_STATUS_SCSI_BUSY**

    The SCSI Device is Busy status code indicates a SCSI command completed with the status "Busy". Some TZ87x media changers are known to cause this condition.

40. **MRD_STATUS_SCSI_INTER**

    The SCSI Intermediate Command Completed status code indicates a SCSI command completed with the status "Intermediate".

41. **MRD_STATUS_SCSI_INTER_CONDMET**

    The SCSI Intermediate-Condition Met status code indicates a SCSI command completed.

42. **MRD_STATUS_SCSI_RESCON**

    The SCSI Reservation Conflict status code indicates a SCSI command completed with the status "Reservation Conflict".

43. **MRD_STATUS_SCSI_TERM**

    The SCSI Command Terminated status code indicates a SCSI command completed with the status "Terminated".

44. **MRD_STATUS_SCSI_QUEUE**

    The SCSI Queue Full status code indicates a SCSI command completed with the status "Queue Full".

45. **MRD_STATUS_SCSI_RESERVED**

    The SCSI Status Code Reserved return indicates a SCSI command completed with a status that wasn't listed in Chapter 7 of the SCSI-2 specification and is "Reserved".

1.2.3.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_NO_SUCH_DEVICE**

   This error is returned when a regular file or robot was specified without the ‘‘:BnTnLn" string.

3. **MRD_STATUS_ROBOT_CMD_ABORTED**

   This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* is zero and the *ascq* is six, or when the *asc* and *ascq* are zero and the *key* is eleven (11).

4. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

5. **MRD_STATUS_EINVAL**

   This error code is returned when the ":BnTnLn" string points to an invalid or nonexistent SCSI address.

6. **MRD_STATUS_ENOENT**

   This error is returned when the system cannot find the specified device.

7. **MRD_STATUS_ROBOT_NOT_READY**

   Under Microsoft Windows 2000/Windows XP, this error code is returned when the specified robot exists but is not responding.

1.2.3.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x80 - When the ASCQ is not zero (0).
   - 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
   - 0x04 - Logical unit not ready
   - 0x3E - Logical unit has not been self configured
   - 0x40 - Diagnostic failure; ASCQ indicates component
   - 0x42 - Power-on self test failure
   - 0x44 - Internal target failure
   - 0x46 - Unsuccessful soft reset
   - 0x4C - Logical unit failed self-configuration

   This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

3. **MRD_STATUS_ROBOT_CMD_ABORTED**

   This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* is zero and the *ascq* is six, or when the *asc* and *ascq* are zero and the *key* is eleven (11).

4. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

5. **MRD_STATUS_EINVAL**

   This error is returned by **mrd_map_os_ error(3mrd)** when the *os_status* is EINVAL. This typically occurs during **mrd_ startup(3mrd)** when the special file is not a SCSI device: for example, /dev/tty.

6. **MRD_STATUS_STARTUP_ERROR**

   This error is returned by **mrd_ map_os_error(3mrd)** when the *os_status* is ENODEV. This typically occurs during **mrd_startup(3mrd)** when the special file is not a SCSI device; /dev/null.

7. **MRD_STATUS_NO_SUCH_DEVICE**

   This error occurs when a UNIX system call returns ENXIO, to indicate that the device corresponding to the special device does not exist.

8. **MRD_STATUS_EBUSY**

   This error occurs when a UNIX system call returns EBUSY, to indicate that some other process is using that medium-changer device.

9. **MRD_STATUS_EINTR**

   This error occurs when a UNIX system call returns EINTR. This error corresponds to an interrupted system call, but also occurs when the SCSI CAM Layered Components Medium-Changer driver is not configured into the running system.

10. **MRD_STATUS_EIO**

    This error occurs when a UNIX system call returns EIO to indicate that there was an I/O error. In most cases an I/O error on a SCSI medium-changer indicates a SCSI error which be translated to another MRD error.

11. **MRD_STATUS_ENOENT**

    This error occurs when a UNIX system call returns ENOENT to indicate that a special device file doesn't exist.

12. **MRD_STATUS_EACCES**

    This error occurs when a UNIX system call returns EACCES to indicate that the caller does not have sufficient permission to open the special device file corresponding to the medium-changer. MRD expects to have read permission on the special device file.

13. **MRD_STATUS_OS_ERROR**

    This error occurs when a UNIX system call returns an error that is not among those previously mentioned. The routine strerror(3) will be used to translate the error code into a standard text message which will be copied to *log_info*.

14. **MRD_STATUS_INVALID**

    This error is a catch-all for MRD failures. All cases where this error is returned are those instances where MRD should have caught and reported the true cause of the failure.

1.2.3.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS$_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

3. **MRD_STATUS_ROBOT_NOT_READY** Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x80 - When the ASCQ is not zero (0).
   - 0x81 - Vendor unique; gripper errors on the TL82X and TL81X

- 0x04 - Logical unit not ready
- 0x3E - Logical unit has not been self configured
- 0x40 - Diagnostic failure; ASCQ indicates component
- 0x42 - Power-on self test failure
- 0x44 - Internal target failure
- 0x46 - Unsuccessful soft reset
- 0x4C - Logical unit failed self-configuration

This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

4. **MRD_STATUS_ROBOT_CMD_ABORTED** This error code is used when an OpenVMS system service fails with the status SS$_ABORT.
5. **MRD_STATUS_NOPRIV** This error code is used when an OpenVMS system service fails with the status SS$_NOPRIV. This will typically occur in **mrd_startup(3mrd)** when the caller doesn't have sufficient privilege to assign a channel to the device.
6. **MRD_STATUS_IVCHAN** This error code is used when an OpenVMS system service fails with the status SS$_IVCHAN. It is likely when an operating system specific routine is used on a device that hasn't been opened by **mrd_ startup(3mrd)**.
7. **MRD_STATUS_MOUNTED** This error code is used when an OpenVMS system service fails with the status SS$_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in mrd_startup(3mrd) when the caller tries to open a device which doesn't exist or uses an invalid device name.
8. **MRD_STATUS_PAGE_CODE** This error occurs in mrd_startup(3mrd) when a SCSI Mode Sense command fails to return the expected data. It uses the SCSI Element Address Assignment mode page to fill in the element count and base address fields of the robot_info_t structure. If the data returned by the medium changer does not have the expected page code, this error is returned. This error has been seen when medium changers are connected to HS family array controllers running V2.7 firmware.
9. **MRD_STATUS_EBUSY** This error code is used when an OpenVMS system service fails with the status SS$_DEVALLOC. This generally happens in **mrd_startup(3mrd)** when another process already has the device allocated.
10. **MRD_STATUS_DEVOFFLINE** This error code is used when an OpenVMS system service fails with the status SS$_DEVOFFLINE and SS$_ MEDOFL.
11. **MRD_STATUS_ACCVIO** This error indicates an internal application failure.
12. **MRD_STATUS_EXQUOTA** This error occurs when an operation requested of the application causes you to exceed a process quota. To correct this problem, increase your process quotas.
13. **MRD_STATUS_ILLEFC** For more information about this error, refer to the OpenVMS system documentation.

1.2.4. Related Information

Functions:

- **mrd_eject**(3mrd)
- **mrd_find_cartridge**(3mrd)
- **mrd_home**(3mrd)
- **mrd_initialize**(3mrd)
- **mrd_inject**(3mrd)
- **mrd_map_element**(3mrd)
- **mrd_move**(3mrd)
- **mrd_load**(3mrd)
- **mrd_lock**(3mrd)
- **mrd_position**(3mrd)
- **mrd_ready_inport**(3mrd)
- **mrd_show**(3mrd)
- **mrd_shutdown**(3mrd)
- **mrd_startup**(3mrd)
- **mrd_strelement**(3mrd)
- **mrd_strexcept**(3mrd)
- **mrd_unload**(3mrd)
- **mtio**(7) For Tru64 UNIX systems.

Chapter 2. mrd_eject

mrd_eject - Move a cartridge from a slot to a port

2.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_eject(
    const char *robot_name,
    const char *volume_tag,
    const char *slot,
    const char *port,
    char       *log_info) ;
```

2.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *volume_tag* — A NUL terminated character string that is the expected volume tag on the cartridge to be moved. On robots with vision support this string will be compared with the volume tag of the cartridge in the source slot and if it doesn't match the call will fail. This feature will not be used if the *volume_tag* is NULL or the empty string.
- *slot* — A NUL terminated character string that is the zero relative address of the slot which is to be used as the source of the move.
- *port* — A NUL terminated character string that is the zero relative address of the port which is to be used as the destination of the move.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

2.3. Description

The **mrd_eject(3mrd)** function is a specialized interface to the SCSI Move Medium command. For the robot specified by *robot_name*, the routine will attempt to move the cartridge in the specified *slot* to the specified *port*. Element addresses are zero based.

The robot will be opened and the arguments to the function will be verified to make sure they are safe and appropriate. The *slot* and *port* address will be verified they are within the valid range of those elements on the robot.

The *cartridge_name* argument can be used to perform cartridge volume tag verification before the move. If the cartridge volume tag at the slot doesn't match that specified by this argument, then **mrd_eject(3mrd)** will fail with the status MRD_STATUS_CART_INVALID. If *cartridge_name* argument is a NULL pointer, an empty string or used on a robot without vision support this argument is silently ignored and the volume tag check will not be made.

If the slot string is an empty string and the library is a TL820 family member, this routine will attempt to move a cartrige on the PTM to the port specified by the *port* argument. This is the equivalent of the Eject Port command of the CLI.

2.3.1. Example

```
/*
 * Example mrd_eject(3mrd).
 *
 * This example is slightly different from the others since it
 * also demonstrates the Eject Port feature of mrd_eject(3mrd).
 * This feature can be used on the TL820 family to move a tape
 * from the Pass-through mechanism (PTM) to the outport.
 *
 * The command usage is:
 *
 * mrd_eject robot [ slot port [ volume_tag ] ]
 */
#ifndef   lint
static   char   SccsId[] = "@(#)mrd_eject.c   1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int    status ; /* Status from mrd_eject(3mrd) */
   char    *robot ; /* Name of the robot to use */
   char    *volume_tag = NULL ; /* Volume tag to check */
   char    *slot ; /* Source slot */
   char    *port ; /* Destination port */
   char    log_info[MRD_MAX_LOG_STRING+1] ; /* Error text */

   /*
    *    Allow the command to only have the robot name specified.
    */
   if( argc < 2 ) {
      printf("usage: %s robot [ slot port [ volume_tag ] ]\n",
         argv[0]) ;

      exit(1) ;
   }
   else
      robot = argv[1] ;
/*
 * If the slot and port aren't specified assume that
 * the target robot is a TL820 and fill in default
 * values for an Eject Port. Otherwise take the
 * desired values directly from the command line.
 */
if( argc >= 4 ) {
   slot = argv[2] ;
   port = argv[3] ;
```

```
   /*
    * Collect the volume_tag name if the user wants it.
    */
   if( argc > 4 )
      volume_tag = argv[4] ;
}
/*
 * We also observe that this case catches the command:
 *
 * mrd_eject robot_name address
 *
 * It can't hurt to let the user specify the outport,
 * since an invalid one simply won't work. In this case
 * the 3rd argument is the port name instead of the slot
 * name.
 *
 * The user could get the same affect by using a quoted
 * empty string for the slot argument on the command line:
 *
 * robot /dev/mc54 "" 1
*/
else {
   if( argc == 3 )
      port = argv[2] ;
   else
      port = "1" ;

   slot = "" ;
}

/*
 * Do the operation.
 */
status = mrd_eject(robot, volume_tag, slot, port, log_info) ;

if( status == MRD_STATUS_SUCCESS )
   printf("Ejected the media in slot #%d to port #%d.\n",
      slot, port) ;
else
   printf("Eject failed: %s: %s.\n", mrd_strstatus(status),
      log_info[0] ? log_info : "none") ;

   return 0 ;
}
```

2.3.2. Return Values

Upon successful completion, the **mrd_eject(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_eject(3mrd)** fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

2.3.2.1. Common Codes

1. 1. **MRD_STATUS_PARAM**

      This error is returned if the *robot_name*, *slot*, *port*, or *log_info* are NULL pointers.

   2. **MRD_STATUS_PORT_INVALID**

      This error is returned when the element address for a port is less than zero or greater than the number of ports.

3. **MRD_STATUS_SLOT_INVALID**

   This error is returned when the element address for a slot is less than zero or greater than the number of slots.

4. **MRD_STATUS_CART_INVALID**

   For routines that accept a *volume_tag* argument to perform volume tag verification, this error indicates that the volume tag of the media doesn't match that passed to the function.

5. **MRD_STATUS_SOURCE_EMPTY**

   On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

6. **MRD_STATUS_DESTINATION_FULL**

   On routines that perform a SCSI Move Medium command, this error indicates that the destination element already has a cartridge in it.

2.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR** This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR** This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR** This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted

- 0x54 - SCSI to host system interface failure

2.3.3. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_load**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_inject**(3mrd)

Chapter 3. mrd_find_cartridge

mrd_find_cartridge - Search for a cartridge by volume tag.

3.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_find_cartridge(
    const char     *robot_name,
    const char     *volume_tag,
    const int       element_type,
    const char     *element_start,
    const int       element_count,
    element_info_t *result,
    char           *result_name,
    int            *result_type,
    char           *log_info);
```

3.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *volume_tag*— A NUL terminated character string that is the volume tag for which to search.
- *element_type* — The type of robot element on which the operation takes place. If an element type of zero (0) is used, all elements will be searched starting at element 0 of each type and searching all the elements of that type on the robot. The order of this search is Slot, Drive, Transport and finally Ports.
- *element_start*— A NUL terminated character string that is the zero relative address of the element where the search should be started. This argument is not used when the *element_type* is zero (0).
- *element_count* — A volume tag search in a large library can take a long time. Some applications (a graphic user interface for example) may want to break up a large search into smaller, quicker sub-searches. When a specific *element_type* is specified only a range specified by the *element_name* and *element_count* will be searched. This argument is ignored when the *element_ type* is zero (0).
- *result_param*— If an element matching the *volume_tag* string is found, the element_info_t will copied into the space pointed to by *result*.
- *result_name*— The zero relative element address of the matching element will be copied into the space pointed to by *result_name*. This space should be a character array of size MRD_NAME_SIZE.
- *result_type*— The element type of the matching element will be copied into the space pointed to by *result_type*.

- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

3.3. Description

This routine allows searching for the element location of a piece of media using the volume tag as a search key. If the *element_type* value is zero (0), all elements will searched in the order Slot, Drive, Transport and Port. The *element_name* and *element_count* arguments will be ignored in this case.

When a specific element type is specified, the search will be limited to that element type. The *element_name* will used as the starting location for a search and *element_count* as the number of elements from that address to search. Using these arguments a search of a large number of elements may be broken up into a number of smaller searches.

When a matching element is found, the element_info_t data for that element will copied into the space pointed to by *result*. The zero relative element address and element type will also be copied into the space provided.

3.3.1. Element Info

The **element_info_t** data structure is defined in the include file **<mrd_ common.h>**. The fields of this data structure are described below:

- *name* — The name field holds the volume tag of the media if applicable.

- *state* — The state field can have one of the following values: ELEMENT_FULL,

ELELMENT_EMPTY, or ELEMENT_EXCEPT.

- *port_type* — If the *element_type* parameter specifies PORT, the port_type field will have one of the following values:

IN_OUT_PORT, INPORT, OUTPORT.

- *status* — The status field can have one of the following values:

MRD_STATUS_SLOT_INVALID, MRD_STATUS_DEVICE_INVALID,
MRD_STATUS_TRANSPORT_INVALID, MRD_STATUS_PORT_INVALID, or

MRD_STATUS_SUCCESS.

- *flags* — Use the ELEMENT_VALID mask on the *flags* field to indicate whether or not the full Read Element Status data is valid. The ELEMENT_ PVOLTAG and ELEMENT_AVOLTAG indicate whether the primary or alternate volume tags of the Read Element Status data are valid.

- *element_addr* — This is the address of the element, unadjusted for the starting address. The routine **mrd_map_element(3mrd)** can be used to convert an absolute element address to a relative address and type. This field will be set to -1 when the information is not valid.

- *source_addr* — On most SCSI-2 medium-changers, this is the address where a cartridge resided before being moved to its current location. The routine **mrd_map_element(3mrd)** can be used to

convert an absolute element address to a relative address and type. This field will be set to -1 when the information is not valid. On some SCSI-2 medium-changers (the DLT family loaders) this will be the element address of the slot itself.

- *data* — This a copy of the SCSI-2 Read Element Status data when the ELEMENT_VALID bit is set in the *flags* field. A byte-order neutral declaration of this data structure is included in the **<mrd_common.h>** include file as the **mrd_reades_t** data structure.

3.3.2. Example

```
/*
 * Example of mrd_find_cartridge(3mrd). The command usage is:
 *
 * mrd_find robot_name volume_tag
 */
#ifndef lint
static char SccsId[] = "@(#)mrd_find.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   element_info_t element ;    /* Element data result */
   int   status ;     /* status from mrd_find_cartridge(3mrd) */
   char   *robot ;     /* Medium changer to search */
   char   *volume_tag ;     /* Volume tag for which to search */
   int   type ;     /* Element type result */
   char   *content ;     /* element content */
   char   *format ;     /* format to print element data */
   char   address[MRD_NAME_SIZE+1] ;   /* Element name result */
   char   log_info[MRD_MAX_LOG_STRING+1] ;     /* error text */
   char   exception[BUFSIZ+1] ;     /* exception buffer */

   /*
    * There are two required arguments; robot name and volume tag.
    */
   if( argc < 3 ) {
      printf("usage: %s robot volume-tag\n", argv[0]) ;
      exit(1) ;
   }

   robot = argv[1] ;
   volume_tag = argv[2] ;

   /*
    * Search all of the elements at the same time. With the
    * type set to zero, the values of element_address ("")
    * and element_count (0), don't matter.
    */
   status = mrd_find_cartridge(robot, volume_tag, 0, "", 0, &element,
         address, &type, log_info) ;

   if( status != MRD_STATUS_SUCCESS )
     printf("Can't find volume %s: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;

   /*
    * Need to print out the results of the find. This is
    * similar to that used by mrd_show, but is a bit more
```

```
     * extensive to show more features.
     */
    format = "%s\t%s\t%s\n" ; /* default format */

    if( element.name[0] )
       content = element.name ;
    else if( element.state & ELEMENT_FULL )
       content = "Full" ;
    else if( element.state & ELEMENT_EXCEPT ) {
       format = "%s\t%s\t%s\t%s\n" ;
       content = "Exception" ;

       (void)mrd_strexcept(element.data.asc, element.data.ascq,
          exception, BUFSIZ) ;
    }
    else
       content = "Empty" ;

    if( element.state & ELEMENT_EXCEPT )
       printf(format, mrd_strelement(type), address, content,
          exception) ;
    else
       printf(format, mrd_strelement(type), address, content) ;

    return 0 ;
}
```

3.3.3. Return Values

Upon successful completion, the **mrd_find_cartridge(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_find_cartridge(3mrd)** fails the returned status value may be set to one of the following values. This routine may also return any of the errors descibed in the **mrd_show(3mrd)** manual page.

Other values that correspond to specific SCSI errors may also be possible, but the ones below are most likely.

3.3.3.1. Common Values

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name*, *volume_tag*, *log_info*, *result*, *result_name*, *element_start* or *result_type* arguments are NULL pointers.

2. **MRD_STATUS_NO_VISION**

   This error occurs in **mrd_find_ cartridge(3mrd)** when the medium-changer has no vision system with which to read bar-code labels.

3. **MRD_STATUS_SLOT_INVALID**

   This error is returned when the element address for a slot is less than zero or greater than the number of slots.

4. **MRD_STATUS_PORT_INVALID**

   This error is returned when the element address for a port is less than zero or greater than the number of ports.

5. **MRD_STATUS_TRANSPORT_INVALID**

   This error is returned when the element address for a transport is less than zero or greater than the number of transports.

6. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives.

7. **MRD_STATUS_INVALID_TYPE**

   For routines that allow the specification of an element type argument, this error indicates that specified type was not one of SLOT, DRIVE, PORT or TRANSPORT.

8. **MRD_STATUS_INSFMEM**

   The **mrd_show(3mrd)** and **mrd_find_ cartridge(3mrd)** functions allocate virtual memory using malloc(3) to store temporary element data. If the attempt to allocate the memory fails, these routines will return this error.

9. **MRD_STATUS_CART_NOT_FOUND**

   This error is returned by **mrd_ find_cartridge(3mrd)** when it can't find the cartridge with the desired volume tag.

3.3.3.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

3.3.3.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

3.3.3.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR** This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error

- 0x48 - Initiator detected error message received

- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_DEVICE_INVALID** This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS$_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

3.3.4. Restrictions

The SCSI-2 specification includes two commands which allow a medium-changer to perform most of the work that this routine does by brute force. Unfortunately, a reliable implementation of these commands was unavailable at the time MRD V1.2 was written. A future version of the API may be able to make use of these routines to speed up a search.

Unlike **mrd_show(3mrd)** this routine will open and close the robot at each iteration.

3.3.5. Related Information

Functions:

- **mrd_show**(3mrd)
- **mrd_map_element**(3mrd)

Chapter 4. mrd_home

mrd_home - Return a cartridge whence it came.

4.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_home(
    const char   *robot_name,
    const char   *volume_tag,
    const char   *source_name,
    const int     source_type,
    char         *destination_name,
    int          *destination_type,
    char         *log_info) ;
```

4.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *volume_tag* — A NUL terminated character string that is the expected volume tag on the cartridge to be moved. On robots with vision support this string will be compared with the volume tag of the cartridge in the source slot and if it doesn't match the call will fail. This feature will not be used if the *volume_tag* is NULL or the empty string.
- *source_name* — A NUL terminated character string that is the zero relative address of the element which is to be used as the source of the move.
- *source_type* — The *source_type* is an integer value to indicate the type of the *source_name* address. The **<mrd_common.h>** include file defines constants for different element types; SLOT, DRIVE, PORT and TRANSPORT.
- *destination_name* — The address of space where the name of the destination address will be written if the move is successful. An character array of MRD_NAME_SIZE bytes should be used. If the *destination_name* address is NULL, the address will not be returned.
- *destination_type* — The address of space where the type of the destination will be copied if the move is successful. If the *destination_type* address is NULL, the type will not be returned.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

4.3. Description

The SCSI-2 specification for medium-changer devices allows an element to remember the source element of the current piece of media. For example, if a **mrd_load(3mrd)** is performed from slot 17 to drive 2, the element information for drive 2 will remember that the media came from slot 17. Where this feature is implemented, it allows an application to query an element to learn the original source of the media in it and return it.

The **mrd_home(3mrd)** function does this. Given a robot name and element address it will see if the source address is valid and when it is return that media to its original location. If the source address is invalid or the element unavailable an error will be returned. The routine will also check to see if the media was inverted when placed in the current element and restore it to its original orientation. When the move is complete, the resulting address and element type will be copied into *destination_name* and *destination_type*.

If the *volume_tag* argument is used, the routine will verify that a cartridge with the volume tag is present in the element before performing the move.

4.3.1. Example

```
/*
 *  Example to do slot to slot moves. The command usage is:
 *
 *    mrd_home robot_name type address [ volume-tag ]
 *
 *  Type can be one of:
 *
 *    slot, port, drive or transport
 *
 *  The optional transport argument can be a transport address
 *  number, the word "default" or an empty string.
 */
#ifndef lint
static char SccsId[] = "@(#)mrd_home.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mrd_common.h>
#include <mrd_message.h>

/*
 * Given a string, resembling one of the element types,
 * return the SCSI type code for it.
 */

struct {
   int   code ;
   char   *string ;
} etypes[] = {
   TRANSPORT,   "transport",
   SLOT,   "slot",
   DRIVE,   "drive",
   PORT,   "port",
} ;

convert_type(char *etype)
{
   register i ;
```

```
   /*
    * For each entry in the array.
    */

   for(i = 0; i < sizeof(etypes)/sizeof(etypes[0]); i++)
      /*
       * Do a case insensitive comparison, allowing
       * abbreviations. Return as soon as a match is
       * found. Return -1 if one isn't found.
       */
#ifdef vms
      if( strncmp(etypes[i].string, etype, strlen(etype)) == 0)
#else
      if( strncasecmp(etypes[i].string,etype,strlen(etype)) == 0)
#endif
         return etypes[i].code ;
   return -1 ;
}

main(int argc, char *argv[])
{

   int   status ;   /* Status from mrd_home(3mrd) */
   char   *robot ;   /* Robot to use */
   char   *element ;   /* Element address */
   char   *volume_tag = NULL ;   /* Optional volume tag */
   int   type ;   /* Element type */
   char   home[MRD_NAME_SIZE+1] ;   /* space for return address */
   int   home_type ;   /* return element type */
   char   log_info[MRD_MAX_LOG_STRING+1] ;   /* error string */

   /*
    * Three required arguments; robot, element type and address.
    */
   if( argc < 4 ) {
      printf("usage: %s robot type address [ volume_tag ]\n",
         argv[0]) ;

      exit(1) ;
   }
   robot   = argv[1] ;
   type    = convert_type(argv[2]) ;
   element = argv[3] ;
   /*
    * Optional volume tag.
    */
   if( argc > 4 )
      volume_tag = argv[4] ;
   /*
    * Do the operation.
    */
   status = mrd_home(robot, volume_tag, element, type,
         home, &home_type, log_info) ;

   if( status != MRD_STATUS_SUCCESS )
      printf("Home failed: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;
   else
      printf("The cartridge in %s %s was returned to %s %s.\n",
         mrd_strelement(type), element,
         mrd_strelement(home_type), home) ;
   return 0 ;
}
```

4.3.2. Return Values

Upon successful completion, the **mrd_home(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_home(3mrd)** fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

4.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_info*, *source_name*, or *log_info* arguments are NULL pointers.

2. **MRD_STATUS_SLOT_INVALID**

   This error is returned when the element address for a slot is less than zero or greater than the number of slots.

3. **MRD_STATUS_PORT_INVALID**

   This error is returned when the element address for a port is less than zero or greater than the number of ports.

4. **MRD_STATUS_TRANSPORT_INVALID**

   This error is returned when the element address for a transport is less than zero or greater than the number of transports.

5. **MRD_STATUS_INVALID_TYPE**

   For routines that allow the specification of an element type argument, this error indicates that specified type was not one of SLOT, DRIVE, PORT or TRANSPORT.

6. **MRD_STATUS_RES_INVALID**

   This error occurs in **mrd_home(3mrd)** when the element data returned from **mrd_show(3mrd)** is not valid.

7. **MRD_STATUS_SOURCE_INVALID**

   This error occurs in **mrd_ home(3mrd)** when the return address in the element data isn't valid.

8. **MRD_STATUS_CART_INVALID**

   For routines that accept a *volume_tag* argument to perform volume tag verification, this error indicates that the volume tag of the media doesn't match that passed to the function.

9. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives.

4.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

4.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

4.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

- 0x08 - Logical unit communcation errors.
- 0x43 - Message error
- 0x45 - Select or Reselect failure
- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

1. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS$_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

Caution

Strict interpretation of the SCSI-2 specification by devices will require that the device only report the address of the last SLOT a medium was in.

4.3.3. Related Information

Functions:

- **mrd_show**(3mrd)

- **mrd_find_cartridge**(3mrd)
- **mrd_map_element**(3mrd)

Chapter 5. mrd_initialize

mrd_initialize - Send a SCSI Initialize Element Status command.

5.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_initialize(
    const char *robot_name,
    char *log_info) ;
```

5.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

5.3. Description

The function sends a SCSI Initialize Element Status command to the specified robot. This command is not qualified by the TA and TF loaders. On robots where this command is qualified, it forces a physical reinventory of the library or loader. On some library systems this may take a long time.

Most library subsystems will perform an inventory when they are powered on or have detected that the configuration may have changed (doors opened, panels removed, etc). For this reason, this routine is rarely needed.

5.3.1. Example

```
/*
 *   Example of mrd_initialize(3mrd). The command usage is:
 *
 *   mrd_init robot_name
 *
 *   It has been observed on an empty TL820 with all the
 *   bin-packs in place that this command takes just under
 *   23 minutes.
 */
#ifndef   lint
static   char SccsId[] = "@(#)mrd_init.c 1.2 3/5/97" ;
#endif
```

```
#include <stdio.h>
#include <stdlib.h>

#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int   status ;   /* Status from mrd_inject(3mrd) */
   char   *robot ;   /* The name of the robot */
   char   log_info[MRD_MAX_LOG_STRING+1] ;   /* error string */
   /*
    *  Accept one required argument; robot name
    */
   if( argc < 2 ) {
      printf("usage: %s robot\n", argv[0]) ;
      exit(1) ;
   }

   /*
    *   Just use this directly from the command line.
    */
   robot = argv[1] ;

   /*
    *   Because this routine can take a long time we'll
    *   provide some positive feed-back that is doing
    *   something.
    */
   printf("Reinventory library %s...", robot); fflush(stdout) ;
   /*
    *   Call the function. Because this routine can take a
    */
   status = mrd_initialize(robot, log_info) ;
   /*
    *   Done.
    */
   putchar('\n') ;
   /*
    *   Print an error message if there is a failure. The
    *   routine mrd_strstatus(3mrd) will accept an MRD
    *   error status and return the corresponding string.
    *   If the log_info data has something other than a
    *   NULL as the first character print it as well. It
    *   typically be the SCSI sense data or a operating
    *   system specific message for the error.
    */
   if( status != MRD_STATUS_SUCCESS )
      printf("Initialize failed: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;
   return 0 ;
}
```

5.3.2. Return Values

Upon successful completion, the **mrd_initialize(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_initialize(3mrd)** fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

5.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

This error is returned if the *robot_name* or *log_info* arguments are NULL pointers.

5.3.2.1.1. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

5.3.2.1.2. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error

- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

5.3.2.1.3. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

5.3.3. Related Information

- **MRD**(3mrd)
- **mrd_initialize_element**(3mrd)

Chapter 6. mrd_initialize_element

mrd_initialize_element - Force a robot inventory operation

6.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_initialize_element(
    robot_info_t *robot_info,
    dev_status_t *dev_status) ;
```

6.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.

- *dev_status* — The *dev_status* is the address of a dev_status_t structure, which is used to pass back detailed error information in the event of a command failure.

6.3. Description

This routine performs a SCSI Initialize Element Status command. It is used by **mrd_initialize(3mrd).** On qualified medium changers this typically causes the medium changer to perform a physical inventory of its contents. This routine can take a long time to complete. The longest time ever observed on a qualified medium changer was approximately 23 minutes on an empty TL820 with all bin-packs in place. The DLT and RDAT changers may take only a few seconds.

The *robot_info* argument is the address of a robot_info_t that has been opened by mrd_startup(3mrd).

This routine uses the **dev_status_t** structure for handing errors. The **dev_ status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

6.3.1. Example

```
/*
 *   Example of mrd_initialize_element(3mrd). The command usage is:
 *
 *   Usage:
 *
 *      mrd_initialize_element robot [ robot... ]
 *      robot_name
 *
 *   It has been observed on an empty TL820 with all the bin-packs
 *   in place that this command takes just under 23 minutes.
 */
#ifndef lint
static char SccsId[] = "@(#)mrd_initialize_element.c 1.3) 6/20/97";
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int     rc ;
   int     status ; /* return status */
   char     *robot ; /* Robot to open */
   robot_info_t   robot_info ; /* Robot data */
   dev_status_t   dev_status ; /* Device status */
   char     log_info[MRD_MAX_LOG_STRING+1] ;

   /*
    *   Check that there are enough arguments.
    */
   if( argc < 4 ) {
      printf("usage: %s robot [ robot... ]\n", argv[0]) ;
      exit(1) ;
   }

   /*
    *   Initialize the channel field of the robot_info, so
    *   mrd_startup(3mrd) will actually open the robot.
    */
   robot_info.channel = BAD_CHANNEL ;

   for(rc = 1; rc < argc; rc++) {
      /*
```

```
        *   Save the current robot name.
        */
        robot = argv[rc] ;

        status = mrd_startup(robot, &robot_info, log_info) ;

        if( status != MRD_STATUS_SUCCESS ) {
           printf("Startup failed: %s: %s.\n",
              mrd_strstatus(status),
              log_info[0] ? log_info : "none") ;

           continue ;
        }

        printf("Initialize Element Status on %s...", robot) ;
        fflush(stdout) ;

        status = mrd_initialize_element(&robot_info, &dev_status) ;

        if( status != MRD_STATUS_SUCCESS )
           printf("Failed: %s.\n", mrd_strstatus(status)) ;
        else
           printf("Success.\n") ;

        (void)mrd_shutdown(&robot_info) ;
    }

    return 0 ;
}
```

6.3.2. Return Values

Upon successful completion **mrd_initialize_element(3mrd)** will return MRD_ STATUS_SUCCESS. On a failure, an MRD_STATUS value corresponding to the error will be returned. Common errors are:

6.3.2.1. Common Codes

1. 1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_info* or *dev_ status* arguments are NULL pointers. The *dev_status* structure is unchanged, even if a valid address is provided.

2. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   This error occurs when the medium changer does not qualify the Initialize Element Status command.

   It is also used for a SCSI command failure, when the ASC is set to one of:

   - 0x1A - Parameter list length error
   - 0x20 - Invalid command operation code
   - 0x22 - Unsupported command
   - 0x24 - Illegal field in CDB
   - 0x25 - Logical unit not supported
   - 0x26 - Threshold parameters not supported

- 0x28 - Import or Export element accessed
- 0x2C - Command sequence error
- 0x39 - Saving parameters not supported
- 0x3D - Invalid bits in Identify message
- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

6.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

6.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

6.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is used when an OpenVMS system service fails with the status SS$_IVCHAN. It is likely when an operating system specific routine is used on a device that hasn't been opened by **mrd_ startup(3mrd)**.

6.3.3. Related Information

Functions:

mrd_initialize(3mrd)

Chapter 7. mrd_inject

mrd_inject - Move a cartridge from an inport to a slot

7.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
include <mrd_common.h>
include <mrd_message.h>

int mrd_inject(
    const char *robot_name,
    const char *volume_tag,
    const char *port,
    const char *slot,
    char       *log_info) ;
```

7.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *volume_tag* — A NUL terminated character string that is the expected volume tag on the cartridge to be moved. On robots with vision support this string will be compared with the volume tag of the cartridge in the source slot and if it doesn't match the call will fail. This feature will not be used if the *volume_tag* is NULL or the empty string.
- *port* — A NUL terminated character string that is the zero relative address of the port which is to be used as the destination of the move.
- *slot* — A NUL terminated character string that is the zero relative address of the slot which is to be used as the source of the move.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

7.3. Description

The **mrd_inject(3mrd)** function is a specialized interface to the SCSI Move Medium command. For the robot specified by *robot_name,* the routine will attempt to move the cartridge in the specified *port* to the specified *slot*. Element addresses are zero based.

The robot will be opened and the arguments to the function will be verified to make sure they are safe and appropriate. The *port* and *slot* address will be verified they are within the valid of those elements on the robot.

The *volume_tag* argument can be used to perform cartridge volume tag verification before the move. If the cartridge volume tag at the port doesn't match that specified by this argument, then **mrd_inject(3mrd)** will fail with the status MRD_STATUS_CART_INVALID. If *volume_tag* argument is a NULL pointer, an empty string or used on a robot without vision support this argument is silently ignored and the volume tag check will not be made.

The TL820 family requires special handling within the **mrd_inject(3mrd)** routine, because of the way the Input/Output Device (IOD) works. This routine will explicitly check the specified inport to see if it is full. If empty and the robot is a TL820, the Pass-through mechanism will then be checked. If the PTM is full the source address will be reset to the PTM. If both are empty, the routine will send a Ready Inport command to enable the IOD. A one minute polling loop will be performed waiting for the inport to become full, five seconds between polls. If this first loop fails, the Ready Inport will be sent again and the loop repeated. This allows the user two minutes to put a tape into the inport.

If volume tag verification is desired on the TL820, the cartridge will be moved to the PTM so the volume tag can be read. If the check fails, the cartridge will left on the PTM. If the **mrd_inject(3mrd)** is repeated with the correct volume tag or without one, the cartridge will be found on the PTM and the Move Medium command will proceed from there.

7.3.1. Example

```
/*
* Example of mrd_inject(3mrd). The command usage is:
*
* mrd_inject robot_name port slot [ volume_tag ]
*/
#ifndef lint
static char SccsId[] = "@(#)mrd_inject.c 1.2 3/5/97" ;
#endif
#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>
main(int argc, char *argv[])
{
   int   status ;   /* Status from mrd_inject(3mrd) */
   char   *robot ;   /* The name of the robot */
   char   *volume_tag = NULL ; /* Optional volume tag to check */
   char   *port ;   /* Source port */
   char   *slot ;   /* Destination slot */
   char   log_info[MRD_MAX_LOG_STRING+1] ;   /* error string */
   /*
    * Accept three required argument; robot, port and slot. The
    * volume tag is optional.
    */
   if( argc < 4 ) {
      printf("usage: %s robot port slot [ volume-tag ]\n", argv[0]);
      exit(1) ;
   }

   /*
    * Just use these directly from the command line.
    */
   robot = argv[1] ;
   port  = argv[2] ;
   slot  = argv[3] ;

   /*
    * If the volume tag is present use it.
    */
   if( argc > 4 )
```

```
      volume_tag = argv[4] ;

   /*
    * Call the function.
    */
   status = mrd_inject(robot, volume_tag, port, slot, log_info) ;

   /*
    *   Print an error message if there is a failure.
    */
   if( status != MRD_STATUS_SUCCESS )
      printf("Inject failed: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;
   else
      printf("Injected media from Port #%s to Slot #%s.\n",
         port, slot) ;

   return 0 ;
}
```

7.3.2. Return Values

Upon successful completion, the **mrd_inject(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_inject(3mrd)** fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

7.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name*, *log_ info*, *slot*, or *port* arguments are NULL pointers.

2. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   It is used when a sanity check fails in the code that attempts to move a cartridge to the Pass-Through Mechanism, when the robot type isn't a TL82n.

   It is also used for a SCSI command failure, when the ASC is set to one of:

   - 0x1A - Parameter list length error
   - 0x20 - Invalid command operation code
   - 0x22 - Unsupported command
   - 0x24 - Illegal field in CDB
   - 0x25 - Logical unit not supported
   - 0x26 - Threshold parameters not supported
   - 0x28 - Import or Export element accessed
   - 0x2C - Command sequence error
   - 0x39 - Saving parameters not supported
   - 0x3D - Invalid bits in Identify message

- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

3. **MRD_STATUS_PORT_INVALID**

   This error is returned when the element address for a port is less than zero or greater than the number of ports.

4. **MRD_STATUS_SLOT_INVALID**

   This error is returned when the element address for a slot is less than zero or greater than the number of slots.

5. **MRD_STATUS_SOURCE_EMPTY**

   On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

6. **MRD_STATUS_DESTINATION_FULL**

   On routines that perform a SCSI Move Medium command, this error indicates that the destination element already has a cartridge in it.

7. **MRD_STATUS_CART_INVALID**

   For routines that accept a *volume_tag* argument to perform volume tag verification, this error indicates that the volume tag of the media doesn't match that passed to the function.

8. **MRD_STATUS_ROBOT_MECH_ERROR**

   This error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x15 - Positioning error.
   - 0x8B - Vendor unique; Pass-through mechanism errors on the TL82n

9. **MRD_STATUS_VENDOR_UNIQUE_ERROR**

   This error occurs when the internal routine used to decode SCSI-2 errors encounters an error that it has not been written to antipicate.

   This error is also returned when the ASC is zero and the ASCQ is not one of zero or six, and when ASC/ASCQ are both zero and the *key* is 9h.

7.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

- 0x08 - Logical unit communcation errors.
- 0x43 - Message error
- 0x45 - Select or Reselect failure
- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

7.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

7.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

7.3.3. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_load**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_eject**(3mrd)
- **mrd_ready_inport**(3mrd)

Chapter 8. mrd_load

mrd_load - Move a piece of media from slot to drive.

8.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_load(
    const char   *robot_name,
    const char   *volume_tag,
    const char   *slot,
    const short   cartridge_side,
    const char   *drive,
    char         *log_info) ;
```

8.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *volume_tag* — A NUL terminated character string that is the expected volume tag on the cartridge to be moved. On robots with vision support this string will be compared with the volume tag of the cartridge in the source slot and if it doesn't match the call will fail. This feature will not be used if the *volume_tag* is NULL or the empty string.
- *slot* — A NUL terminated character string that is the zero relative address of the slot which is to be used as the source of the move.
- *cartridge_side* — The *cartridge_side* indicates whether the media should be inverted as it is being to moved to the destination element. If the value 1 is used, the media will not be inverted. If the value 2 is used the media will be inverted.
- *drive* — A NUL terminated character string that is the zero relative address of the drive which is to be used as the destination of the move.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

8.3. Description

The **mrd_load(3mrd)** function is a specialized interface to the SCSI Move Medium command (or DSA equivalent). For the robot specified by *robot_name*, the routine will attempt to move the cartridge in the specified *slot* to the specified *drive*. Element addresses are zero based. On subsystems that

support inverting a cartridge during a move, the *cartridge_side* argument can be used to indicate that the cartridge should be inverted.

The robot will be opened and the arguments to the function will be verified to make sure they are safe and appropriate. The *slot* and *drive* address will be verified they are within the valid of those elements on the robot.

The *volume_tag* argument can be used to perform cartridge volume tag verification before the move. If the cartridge volume tag at the port doesn't match that specified by this argument, then **mrd_load(3mrd)** will fail with the status MRD_STATUS_CART_INVALID. If *volume_tag* argument is a NULL pointer, an empty string or used on a robot without vision support this argument is silently ignored and the volume tag check will not be made.

8.3.1. Example

```
/*
 * Example of mrd_load(3mrd). The command usage is:
 *
 *   mrd_load robot_name slot drive [ volume_tag ]
 */
#ifndef   lint
static   char   SccsId[] = "@(#)mrd_load.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int   status ;   /* Status from mrd_load(3mrd) */
   short   side = 1 ;   /* Only support single sided media */
   char   *robot ;   /* The name of the robot */
   char   *volume_tag = NULL ; /* Optional volume tag to check */
   char   *slot ;   /* Source slot */
   char   *drive ;   /* Destination drive */
   char   log_info[MRD_MAX_LOG_STRING+1] ;   /* error string */

   /*
    *   Accept three required argument; robot, port and slot. The
    *   volume tag is optional.
    */
   if( argc < 4 ) {
      printf("usage: %s robot slot drive [ volume-tag ]\n", argv[0]);
      exit(1) ;
   }

   /*
    * Just use these directly from the command line.
    */
   robot = argv[1] ;
   slot  = argv[2] ;
   drive = argv[3] ;

   /*
    *   If the volume tag is present use it.
    */
   if( argc > 4 )
      volume_tag = argv[4] ;

   /*
    *   Call the function.
```

```
    */
   status = mrd_load(robot, volume_tag, slot, side, drive, log_info);

   /*
    *   Print an error message if there is a failure.
    */
   if( status != MRD_STATUS_SUCCESS )
      printf("Load failed: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;
   else
      printf("Loaded media in Slot #%s to Drive #%s\n",
         slot, drive) ;

   return 0 ;
}
```

8.3.2. Return Values

Upon successful completion, the **mrd_load(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_load(3mrd)** fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

8.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name*, *log_ info*, *slot*, or *drive* arguments are NULL pointers.

2. **MRD_STATUS_SLOT_INVALID**

   This error is returned when the element address for a slot is less than zero or greater than the number of slots.

3. **MRD_STATUS_CART_SIDE_INVALID**

   For routines that use the *cartridge_side* argument, this error indicates that the value is neither one (1) nor two (2).

4. **MRD_STATUS_CART_INVALID**

   For routines that accept a *volume_tag* argument to perform volume tag verification, this error indicates that the volume tag of the media doesn't match that passed to the function.

5. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives.

6. **MRD_STATUS_SOURCE_EMPTY**

   On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

7. **MRD_STATUS_DESTINATION_FULL**

   On routines that perform a SCSI Move Medium command, this error indicates that the destination element already has a cartridge in it.

8.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

8.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error

- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

8.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS$_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

8.3.3. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_inject**(3mrd)
- **mrd_eject**(3mrd)

Chapter 9. mrd_lock

mrd_lock - Send a SCSI Prevent-Allow Media Removal command

9.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_lock(
    const char *robot_name,
    const int   lock_value,
    char       *log_info) ;
```

9.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *lock_value* — This value indicates whether the the routine should prevent or allow media removal for the robot. The include file <mrd_common.h> defines two constants PREVENT_REMOVAL and ALLOW_REMOVAL that will set the correct value.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

9.3. Description

This routine sends a SCSI Prevent-Allow Media Removal command. Some robots have been observed to silently ignore the command, others will fail with an illegal request and others will do something particular to the robot. This command is not supported on the DSA medium-changers (TA and TF drives). SCSI defines this a single command where the value of a single bit determines the affect.

On some versions of TL820 firmware, the command PREVENT_REMOVAL will disable Move Medium commands to the outport. Other versions will also disable Move Medium commands from the inport. Other models will ignore the command entirely. The TL810 family of libraries use the command to disable the front panel button which allows opening the port door. Please refer to your robot's documentation to see what affect this command will have.

9.3.1. Return Values

Upon successful completion, the **mrd_lock(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_lock(3mrd)** fails the returned status value may be set to one of

the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

9.3.1.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name* or *log_info* arguments are NULL pointers.

2. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   This is used in the **mrd_lock(3mrd)** code when the value is not one of ALLOW_REMOVAL or PREVENT_REMOVAL.

   It is also used for a SCSI command failure, when the ASC is set to one of:

   - 0x1A - Parameter list length error
   - 0x20 - Invalid command operation code
   - 0x22 - Unsupported command
   - 0x24 - Illegal field in CDB
   - 0x25 - Logical unit not supported
   - 0x26 - Threshold parameters not supported
   - 0x28 - Import or Export element accessed
   - 0x2C - Command sequence error
   - 0x39 - Saving parameters not supported
   - 0x3D - Invalid bits in Identify message
   - 0x53 - Medium removal prevented

   This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

9.3.1.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure

- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

9.3.1.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

9.3.1.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

- 0x08 - Logical unit communcation errors.
- 0x43 - Message error
- 0x45 - Select or Reselect failure
- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

9.3.2. Related Information

Functions:

mrd_prevent_allow(3mrd)

Chapter 10. mrd_map_element

mrd_map_element - Map an absolute element address to a zero relative one.

10.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_map_element(
    const robot_info_t *robot_info,
    const int           address,
    char               *result) ;
```

10.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.
- *address* — This is the absolute element address that is to be mapped.
- *result* — This is the address where the zero relative element address is to be written. Like other element addresses used by the Media Robot Driver Library, it is a character string. A character array of MRD_NAME_SIZE bytes should be used.

10.3. Description

Given a *robot_info_t* structure and absolute element address, this routine figures out the corresponding element type and zero relative address. The relative address is formatted into the space provided by *result* and the element type is returned.

A valid *robot_info_t* structure can be obtained by using **mrd_startup(3mrd)** or **mrd_show(3mrd)** to open the robot and fill in the robot_info_t structure.

The SCSI-2 specification allows an absolute address of zero (0) to refer to a default transport, when a medium-changer has more than one. When handed zero as the absolute address, this routine will reflect this convention even if the particular medium-changer doesn't.

10.3.1. Example

```
/*
 *   For the specified robot, walk through the remainder of
 *   argument list and have mrd_map_element(3mrd) convert
 *   each address to a relative element address and type.
 *
 *     mrd_map_element robot address [ address... ]
 */
```

```
#ifndef   lint
static   char   SccsId[] = "@(#)mrd_map_element.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   char   *robot ;   /* Robot for command */
   int   status ;   /* status from mrd_startup(3mrd) */
   int   address ;   /* Input argument */
   int   type ;   /* element type */
   int   i ;   /* index counter */
   robot_info_t robot_info ;   /* Set by mrd_startup(3mrd) */
   char   result[MRD_NAME_SIZE+1] ;   /* relative address */
   char   log_info[MRD_MAX_LOG_STRING+1] ;   /* error text */

   /*
    *   Two required arguments, many optional ones.
    */
   if( argc < 3 ) {
      printf("usage: %s robot address [ address... ]\n", argv[0]) ;
      exit(1) ;
   }
   else
      robot = argv[1] ;

   /*
    *   Open the robot. Must set channel to BAD_CHANNEL so
    *   it will really open the robot.
    */
   robot_info.channel = BAD_CHANNEL ;

   status = mrd_startup(robot, &robot_info, log_info) ;

   if( status != MRD_STATUS_SUCCESS ) {
      printf("Can't open robot %s: %s: %s.\n", robot,
         mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;

      exit(1) ;
   }

   /*
    *   We don't need to keep the robot for the remainder of
    *   the example.
    */
   (void)mrd_shutdown(&robot_info) ;
   /*
    *   For each address in the list, call mrd_map_element(3mrd).
    */
   for(i = 2; i < argc; i++) {
      address = atoi(argv[i]) ;

      type = mrd_map_element(&robot_info, address, result) ;

      if( type == 0 )
         printf("Can't map %d on robot %s.\n", address, robot) ;
      else
         printf("Element %d -> %s %s\n", address,
            mrd_strelement(type), result) ;
   }
```

```
   return 0 ;
}
```

10.3.2. Return Values

Upon successful completion, the **mrd_map_element(3mrd)** function returns the element type, which is one of SLOT, PORT, DRIVE or TRANSPORT. On an error it returns zero (0). The two possible errors are the *robot_info* address being NULL and the *address* not one used by this medium-changer.

10.3.3. Related Information

Functions:

- **mrd_show**(3mrd)
- **mrd_home**(3mrd)
- **mrd_find_cartridge**(3mrd)

Chapter 11. mrd_message

11.1. Media Robot Driver Library

- **mrd_strstatus** - Maps MRD error codes to message strings.
- **mrd_strelement** - Maps MRD_ELEMENT codes to descriptive words.
- **mrd_strexcept** - I18N MRD messages and strings.

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

char *mrd_strstatus(int code) ;

char *mrd_strexcept(
      unsigned char  asc,
      unsigned char  ascq,
      char          *buffer,
      size_t         length) ;

char *mrd_strelement(int code) ;
```

11.2. Parameters

- *code* — For **mrd_strstatus(3mrd)** this is the status code returned from one of the MRD functions. For **mrd_strelement(3mrd)** this is a code number corresponding to a number of common strings that may occur in an appliation using MRD.
- *asc* — This is the SCSI Additional Sense Code (ASC) in the element status data (*data.asc*) obtained from the **mrd_show(3mrd)** or **mrd_find_ cartridge(3mrd)** functions.
- *ascq* — This is the SCSI Additional Sense Code Qualifer (ASCQ) in the element status data (*data.ascq*) obtained from the **mrd_show(3mrd)** or **mrd_find_cartridge(3mrd)** functions.
- *buffer* — This is the address of a user supplied buffer, into which the message corresponding to the provided ASC/ASCQ code will be copied. This message is formatted to include the ASC/ASCQ codes in hexadecimal.
- *length* — This is size of the buffer provided. No more than *length* characters will be copied from the internal temporary buffer to the user's buffer. The internal buffer is limited to BUFSIZ characters.

11.3. Description

These routines offer an interface to convert MRD status codes, element exception codes and constants for other common words into internationalized text. The message catalogs used by these routines are the

same ones used by the Media Robot Utility command line interface. However, if no message catalog is available, a standard default message will be used for each code.

The routine **mrd_strstatus(3mrd)** accepts MRD error codes and returns the corresponding message string from the catalog.

The routine **mrd_strelement(3mrd)** accepts one of the MRD_ELEMENT codes defined in <mrd_message.h> and returns the corresponding word. An effort has been made to ensure that the first four code values correspond to the SCSI element types of SLOT, DRIVE, PORT and TRANSPORT, but the routine will remap these values to corresponding MRD_ELEMENT codes and return that string.

The routine **mrd_strexcept(3mrd)** accepts the ASC/ASCQ code set in the element_info_t structure from an **mrd_show(3mrd)** or **mrd_find_**

cartridge(3mrd) when the ELEMENT_EXCEPT bit is set in the *data.state* field. Using the user provided buffer and length it will format the corresponding message to include the ASC/ASCQ values and return a pointer to the space.

Since many of these code are vendor specific, we're unable to provide translations for all them, but we have made an effort to include the translations for many of the exception codes on supported medium-changers.

11.3.1. Codes Translated

The following MRD_ELEMENT codes are those currently supported by **mrd_ strelement(3mrd),** with their corresponding default strings.

Most of the codes listed below are self-explanatory:

- MRD_ELEMENT_TRANSPORT
- MRD_ELEMENT_SLOT
- MRD_ELEMENT_PORT
- MRD_ELEMENT_DRIVE
- MRD_ELEMENT_EMPTY
- MRD_ELEMENT_FULL
- MRD_ELEMENT_EXCEPT
- MRD_ELEMENT_ACCESS
- MRD_ELEMENT_INPORT

  This code is suitable for use when the ELEMENT_IMPENB bit is set in the *state* and *data.state* field of the **element_info_t** data.

- MRD_ELEMENT_OUTPORT - OUTPORT.

  This code is suitable for use when the ELEMENT_EXPENB bit is set in the *state* and *data.state* field of the **element_info_t** data.

- MRD_ELEMENT_IOPORT

  This code suitable for use when the ELEMENT_IMPENB and ELEMENT_EXPENB bits are set in the *state* and *data.state* field of the **element_info_t** data.

- MRD_ELEMENT_PASS

  This code suitable in those cases where the Pass-through mechanism of the TL820 can be identified. Currently, the only way to do this is to know the absolute address of the PTM depending on whether the library is in Multi-unit Single-LUN (Transport 1) mode or Single-unit Single-LUN (Port 2) mode.

- MRD_ELEMENT_MYSTERY

  This code is suitable when neither the ELEMENT_IMPENB nor ELEMENT_EXPENB bits are set in the *state* and *data.state* field of the **element_info_t** data. Most medium-changers identify their ports, when they have one or the other, or both.

11.3.2. Example

```
/*
 *   Illustrate the different mrd_str*(3mrd) functions. For each
 *   case a code from one the mrd_message.h is selected and the
 *   resulting string printed. The command doesn't require
 *   any arguments.
 */
#ifndef   lint
static   char   SccsId[] = "@(#)mrd_string.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   /*
    *   This happens to be an obscure VMS system service error code.
    */
   int status = MRD_STATUS_UNASEFC ;
   /*
    *   The code for the TL820 Pass-through mechanism.
    */
   int word = MRD_ELEMENT_PASS ;
   /*
    *   The codes for when a TL820 doesn't a have a bin-pack
    *   installed for a certain range of slots.
    */
   char   asc  = 0x80 ;
   char   ascq = 0x2 ;
   /*
    * Buffer for the message.
    */
   char buffer[BUFSIZ] ;

   printf("Status:    %s\n", mrd_strstatus(status)) ;
   printf("Word:      %s\n", mrd_strelement(word)) ;
   printf("Exception: %s\n", mrd_strexcept(asc,ascq,buffer,BUFSIZ));

   return 0 ;
}
```

11.3.3. Return Values

These routines try very hard to return the corresponding error text, even to the point of formatting the integer value into the provided string, into a static buffer or returning a default string. These routines should never return NULL, but they rely on **catgets(3)** to do the underlying work of looking the error code in the message catalogs.

11.3.4. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_load**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_inject**(3mrd)
- **mrd_eject**(3mrd)
- **mrd_show**(3mrd)
- **mrd_ready**(3mrd)
- **mrd_position**(3mrd)
- **mrd_initialize**(3mrd)
- **mrd_home**(3mrd)
- **mrd_find_cartridge**(3mrd)
- **mrd_lock**(3mrd)
- **mrd_unlock**(3mrd)

File:

These additional files support programming in the Tru64 UNIX environment.

</usr/lib/nls/msg/en_LOCALE/mrd.cat>

</usr/include/mrd_message.h>

Chapter 12. mrd_move

mrd_move - Move media from one location to another

12.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_move(
    const char *robot_name,
    const char *volume_tag,
    const char *source,
    const int   source_type,
    const char *destination,
    const int   destination_type,
    const int   cartridge_side,
    char       *log_info) ;
```

12.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *volume_tag* — A NUL terminated character string that is the expected volume tag on the cartridge to be moved. On robots with vision support this string will be compared with the volume tag of the cartridge in the source slot and if it doesn't match the call will fail. This feature will not be used if the *volume_tag* is NULL or the empty string.
- *source* — The *source* is a character string which is the zero based element address that is to be used as the source of the move.
- *source_type* — The *source_type* is an integer value to indicate the type of the *source_name* address. The **<mrd_common.h>** include file defines constants for different element types; SLOT, DRIVE, PORT and TRANSPORT.
- *destination* — The *destination* is a character string which is the zero based element address that is to be used as the destination of the move.
- *destination_type* — The *destination_type* is an integer value to indicate the element type of the *destination* address. The **<mrd_common.h>** include file defines constants for different element types; SLOT, DRIVE, PORT and TRANSPORT.
- *cartridge_side* — The *cartridge_side* indicates whether the media should be inverted as it is being to moved to the destination element. If the value 1 is used, the media will not be inverted. If the value 2 is used the media will be inverted.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense

data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

12.3. Description

The **mrd_move(3mrd)** function is an interface to the SCSI Move Medium command. For the robot specified by *robot_name,* the routine will attempt to move the cartridge in the element specified by the source address and type to that specified by the destination address and type.

Element addresses are zero based. On subsystems that support inverting a cartridge during a move, the *cartridge_side* argument can be used to indicate that the cartridge should be inverted.

The robot will be opened and the arguments to the function verified that they are appropriate. Element addresses and types will be checked that they are within the valid range of elements on the robot. The *cartridge_side* argument will be verified that it is either the value one (1) or two (2). All pointer arguments, except *cartridge_name*, are checked to verify they are not NULL pointers.

The *cartridge_name* argument can be used to perform cartridge volume tag verification before the move. If the cartridge volume tag at the source doesn't match that specified by this argument, then **mrd_move(3mrd)** will fail with the status MRD_STATUS_CART_INVALID. If the *cartridge_name* argument is a NULL pointer, an empty string or used on a robot without vision support this argument is silently ignored and the volume tag check will not be made.

On the TL820 family of libraries, the tape will be moved to the pass-through read station if the source is a Port. If this move fails the status will be appropriate to that of a failed Move Medium. Likely error codes are documented in the Return Values section.

After the volume tag check, the destination address is checked in the same way the source was. The same error codes are returned if the destination address

is out of range. With the range checks completed the Move Medium command is issued. If successful MRD_STATUS_SUCCESS is returned. If the command

failed, the SCSI error will be mapped to the appropriate MRD_STATUS message.

The DLT libraries (TL820 and TL810 families) require the host issue a SCSI Unload command before a cartridge may be removed from the drive. The function **mrd_move(3mrd),** does not offer this feature. Thus, the calling program must do this itself. On Tru64 UNIX this may done with the MTOFFL opcode of the MTIOCTOP I/O control.

12.3.1. Example

```
/*
 *   Example to do slot to slot moves, using mrd_move(3mrd). The
 *   reason for only doing slot to slot, is that it simplifies
 *   having to figure out element types. The mrd_position(3mrd)
 *   example shows how part of this may be done.
 *
 *   The command usage is:
 *
 *      mrd_move source-slot destination-slot [ volume-tag ]
 */
#ifndef   lint
static   char   SccsId[] = "@(#)mrd_move.c 1.2 3/5/97" ;
#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int   status ;   /* Status from mrd_move(3mrd) */
   int   side = 1 ;   /* Only support side one */
   char   *robot ;   /* Name of the robot to use */
   char   *volume_tag = NULL ;   /* Volume tag to check */
   char   *source ;   /* Source slot */
   char   *destination ;   /* Destination slot */
   char   log_info[MRD_MAX_LOG_STRING+1] ;   /* error string */

   /*
    * Three required arguments; robot name, source slot and
    * destination slot.
    */
   if( argc < 4 ) {
   printf("usage: %s robot src dest [ volume-tag ]\n", argv[0]) ;
   exit(1) ;
   }

   robot       = argv[1] ;
   source      = argv[2] ;
   destination = argv[3] ;

   /*
    * Volume tag is optional.
    */
   if( argc > 4 )
      volume_tag = argv[4] ;

   /*
    * Do the operation.
    */
   status = mrd_move(robot, volume_tag, source, SLOT, destination,
         SLOT, side, log_info) ;

   if( status != MRD_STATUS_SUCCESS )
      printf("Move failed: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;
   else
      printf("Moved media from Slot #%s to Slot #%s\n",
         source, destination) ;

   return 0 ;
}
```

12.3.2. Return Values

Upon successful completion, the **mrd_move(3mrd)** function returns the value MRD_STATUS_SUCCESS. If **mrd_move(3mrd)** fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

12.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name*, *source*, *destination*, or *log_info* arguments are NULL pointers.

2. **MRD_STATUS_CART_SIDE_INVALID**

   For routines that use the *cartridge_side* argument, this error indicates that the value is neither one (1) nor two (2).

3. **MRD_STATUS_PORT_INVALID**

   This error is returned when the element address for a port is less than zero or greater than the number of ports.

4. **MRD_STATUS_TRANSPORT_INVALID**

   This error is returned when the element address for a transport is less than zero or greater than the number of transports.

5. **MRD_STATUS_SLOT_INVALID**

   This error is returned when the element address for a slot is less than zero or greater than the number of slots.

6. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives.

7. **MRD_STATUS_INVALID_TYPE**

   For routines that allow the specification of an element type argument, this error indicates that specified type was not one of SLOT, DRIVE, PORT or TRANSPORT.

8. **MRD_STATUS_CART_INVALID**

   For routines that accept a *volume_ tag* argument to perform volume tag verification, this error indicates that the volume tag of the media doesn't match that passed to the function.

9. **MRD_STATUS_SOURCE_EMPTY**

   On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

10. **MRD_STATUS_DESTINATION_FULL**

    On routines that perform a SCSI Move Medium command, this error indicates that the destination element already has a cartridge in it.

11. **MRD_STATUS_CART_NOT_AVAIL**

    This error can occur on the TL81n and TL82n family of DLT libraries when the source of a move is a drive and the cartridge in the drive is still on-line. These robots do not allow moving the cartridge until the drive is taken offline.

12.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

12.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

12.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS$_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

3. **MRD_STATUS_CART_NOT_AVAIL**

   This error can occur on the TL81n and TL82n family of DLT libraries when the source of a move is a drive and the cartridge in the drive is still on-line. These robots do not allow moving the cartridge until the drive is taken offline.

   On OpenVMS this can be done with $DISMOU system service.

12.4. Related Information

Functions:

- **mrd_load**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_inject**(3mrd)
- **mrd_eject**(3mrd)
- **mrd_ready**(3mrd)

Chapter 13. mrd_move_medium

mrd_move_medium - Move media from one location to another

13.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_move_medium(
    robot_info_t *robot_info,
    int           transport,
    int           source,
    int           destination,
    int           invert,
    dev_status_t *dev_status) ;
```

13.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.
- *transport* — The *transport* is the numeric value of the transport which will be moved.
- *source* — The *source* is an absolute element address.
- *destination*— The *destination* is an absolute element address.
- *invert* — The *invert* is a numeric value used to indicate if the medium should be inverted when it is moved. A value of one (1) is used to indicate that the medium should be inverted.
- *dev_status* — The *dev_status* is the address of a dev_status_t structure, which is used to pass back detailed error information in the event of a command failure.

13.3. Description

This routine performs a SCSI Move Medium command, or equivalent if some other I/O architecture is supported. It is used by **mrd_move(3mrd)**, **mrd_ load(3mrd)**, **mrd_unload(3mrd)**, **mrd_inject(3mrd)** and **mrd_eject(3mrd)**. Since it accepts a robot_info_t structure associated with an open medium changer it can be used to perform multiple move commands, without having to re-open the medium changer as the other functions that use it do.

The *robot_info* argument is the address of a robot_info_t that has been opened by **mrd_startup(3mrd)**. If the medium changer isn't opened, the Move Medium command will fail with the operating system error for trying to use an unopened device. On SCSI medium changers, it maps directly to the SCSI Move Medium command.

The *transport* address is the absolute address of the transport element to be used for the command. Many medium changers allow the use of address zero (0) as the default transport, but some may require a transport address that is valid for the medium changer. For single transport medium changers, the transport base address in the robot_info_t structure, *transport_start* is a suitable address.

The *source* and *destination* addresses are absolute addresses to be used as the source and destination for the move. The absolute address can be calculated from a zero relative address by adding it to the base address for the element type. The routine makes no checks for the validity of the address, relying on the medium changer to do this.

A *invert* value of one (1) can be used on medium changers that support inverting the media, when this is desired; an optical drive with two sided media. Otherwise a value of zero should be used.

This routine uses the **dev_status_t** structure for handing errors. The **dev_ status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

13.3.1. Absolute Element Addresses

The operating system interface routines use absolute SCSI element addresses, instead of zero relative address as used by the higher level functions. A zero based element address can be converted to an absolute address by adding the element base address from the robot_info_t structure.

```
int slot ;
robot_info_t robot_info ;

/*
 * An relative starting address.
 */
slot = 3 ;

/*
 * Becoming an absolute address.
 */
```

```
slot += robot_info.slot_start ;
```

13.3.2. Example

```
/*
 *   This is an example of using mrd_move_medium(3mrd) directly to
 *   move a cartridge from one slot to another. To simplify the
 *   example, it only supports slot to slot moves, but it shows
 *   how the absolute element addresses are calcuated. For each
 *   additional destination address given, the previous (successful)
 *   destination address is used as the source.
 *
 *   Usage:
 *
 *   mrd_move_medium robot source dest [ dest... ]
 */
#ifndef   lint
static   char SccsId[] = "@(#)mrd_move_medium.c 1.4 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int   i ; /* counter */
   int   source ; /* Source address */
   int   dest ; /* Destination address */
   int   invert = 0 ; /* Assume it can't invert */
   int   transport ; /* Transport address */
   int   status ; /* return status */
   char   *robot ; /* Robot to open */
   robot_info_t   robot_info ; /* Robot data */
   dev_status_t   dev_status ; /* Device status */
   char   log_info[MRD_MAX_LOG_STRING+1] ;
/*
 *   Check that there are enough arguments.
 */
if( argc < 4 ) {
   printf("usage: %s robot source dest [ dest... ]\n", argv[0]) ;
   exit(1) ;
}
else {
   robot = argv[1] ;
   source = atoi(argv[2]) ;
}

/*
 *   Initialize the channel field of the robot_info, so
 *   mrd_startup(3mrd) will actually open the robot.
 */
robot_info.channel = BAD_CHANNEL ;

status = mrd_startup(robot, &robot_info, log_info) ;

if( status != MRD_STATUS_SUCCESS ) {
   printf("Startup failed: %s: %s.\n", mrd_strstatus(status),
      log_info[0] ? log_info : "none") ;

   exit(1) ;
}

/*
```

```
 *   Set the transport address. If there is only one
 *   transport use the correct address. If there is
 *   more than one assume that the medium changer
 *   supports zero as the default.
 */
if( robot_info.transport_count == 1 )
   transport = robot_info.transport_start ;
else
   transport = 0 ;

/*
 *   Turn the relative slot address into an absolute slot
 *   address by adding the slot start address.
 */
source += robot_info.slot_start ;
/*
 *   For each destination address on the command line,
 *   move the the cartridge in the source to the
 *   destination. After each (successful) move, replace
 *   the previous source with this destination.
 */
for(i = 3; i < argc; i++) {
   /*
    * Calculate the absolute address.
    */
   dest = atoi(argv[i]) + robot_info.slot_start ;

   /*
    *   Print an audit as we go. Since we know these
    *   are slots, convert back to relative addresses
    *   for the audit.
    */
   printf("Move the medium in slot %d to slot %d\n",
      source - robot_info.slot_start,
      dest - robot_info.slot_start) ;

   status = mrd_move_medium(&robot_info, transport, source,
         dest, invert, &dev_status) ;
   if( status != MRD_STATUS_SUCCESS ) {
      printf("Move failed on %s: %s\n", robot,
         mrd_strstatus(status)) ;

        /*
         * Since the cartridge didn't move, don't
         * reset the source, by skipping the remainder
         * of the loop.
         */
        continue ;
     }
    /*
     * Make the next source, this destination.
     */
    source = dest ;
  }
  (void)mrd_shutdown(&robot_info) ;

  return 0 ;
}
```

13.3.3. Return Values

Upon successful completion, **mrd_move_medium(3mrd)** will return MRD_ STATUS_SUCCESS. On a failure, an MRD_STATUS value corresponding to the error will be returned. Common errors are:

13.3.3.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_info* or *dev_status* arguments are NULL pointers.

2. **MRD_STATUS_CART_NOT_AVAIL**

   This error can occur on the TL81n and TL82n family of DLT libraries when the source of a move is a drive and the cartridge in the drive is still on-line. These robots do not allow moving the cartridge until the drive is taken offline.

3. **MRD_STATUS_DESTINATION_FULL**

   On routines that perform a SCSI Move Medium command, this error indicates that the destination element already has a cartridge in it.

4. **MRD_STATUS_SOURCE_EMPTY**

   On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

5. **MRD_STATUS_ELEMENT_INVALID**

   This error occurs when a SCSI command fails with the ASC set to 0x21. The *log_info* will contain the ASCQ. This indicates that an invalid element address reached the medium-changer. For example, specifying the 13th slot when only 12 slots are present.

13.3.3.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

2. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error

- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

13.3.3.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

13.3.3.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_CART_NOT_AVAIL**

   This error can occur on the TL81n and TL82n family of DLT libraries when the source of a move is a drive and the cartridge in the drive is still on-line. These robots do not allow moving the cartridge until the drive is taken offline.

   On OpenVMS this can be done with $DISMOU system service.

2. **MRD_STATUS_IVCHAN** This error code is used when an OpenVMS system service fails with the status SS$_IVCHAN. It is likely when an operating system specific routine is used on a device that hasn't been opened by **mrd_ startup(3mrd)**.

13.3.4. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_load**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_inject**(3mrd)
- **mrd_eject**(3mrd)

Chapter 14. mrd_position

mrd_position - Position a transport to an element.

14.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_position(
    const char   *robot_name,
    const char   *transport,
    const char   *element,
    const int     element_type,
    const int     cartridge_side,
    char         *log_info) ;
```

14.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *transport* — The *transport* is the numeric value of the transport which will be moved.
- *element* — A NUL terminated character string that is the zero relative address of the element to which the transport is to be moved.
- *element_type* — An integer value indicating the type of element to which **mrd_position(3mrd)** places the changer. Element types include: PORT, DRIVE, SLOT, TRANSPORT.
- *cartridge_side* — The *cartridge_side* indicates whether the media should be inverted as it is being to moved to the destination element. If the value 1 is used, the media will not be inverted. If the value 2 is used the media will be inverted.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

14.3. Description

The **mrd_position(3mrd)** routine provides access to the SCSI Position to Element command. For the robot specified by the *robot_name,* the routine will attempt to position the specified transport to the specified element. The transport and element addresses are zero based. On subsystems that support inverting a cartridge during a move, the *cartridge_side* argument can be used to indicate that the transport should be inverted to invert a cartridge.

The robot will be opened and the arguments to the function verified that they are appropriate. The element address and type will be checked that they are within the valid range of elements on the robot.

The *cartridge_side* argument will be verified that it is either the value one (1) or two (2). All pointer arguments, except *transport,* are checked to verify they are not NULL pointers.

Many robot subsystems support an absolute transport address of zero for the Position to Element command so that the robot can select the appropriate transport if multiple are available. This routines allows the default transport address to be specified by using a NULL pointer for the transport string.

14.3.1. Example

```
/*
 *   Example to do slot to slot moves. The command usage is:
 *
 *     mrd_position robot_name type address [ transport ]
 *
 *   Type can be one of:
 *
 *     slot, port, drive or transport
 *
 *   The optional transport argument can be a transport address
 *   number, the word "default" or an empty string.
 */
#ifndef lint
static   char   SccsId[] = "@(#)mrd_position.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <mrd_common.h>
#include <mrd_message.h>

/*
 * Given a string, resembling one of the element types,
 * return the SCSI type code for it.
 */
struct {
   int code ;
   char *string ;
} etypes[] = {
   TRANSPORT, "transport",
   SLOT, "slot",
   DRIVE, "drive",
   PORT, "port",
} ;

convert_type(char *etype)
{
   register i ;
   /*
    * For each entry in the array.
    */
   for(i = 0; i < sizeof(etypes)/sizeof(etypes[0]); i++)
      /*
       * Do a case insensitive comparison, allowing
       * abbreviations. Return as soon as a match is
       * found. Return -1 if one isn't found.
       */
#ifdef vms
      if( strncmp(etypes[i].string, etype, strlen(etype)) == 0 )
#else
      if( strncasecmp(etypes[i].string, etype, strlen(etype)) == 0 )
#endif
         return etypes[i].code ;
```

```
return -1 ;
}

main(int argc, char *argv[])
{
   int   status ;
   int   side = 1 ;
   char   *robot ;
   char   *cart = NULL ;
   char   *element ;
   char   *transport ;
   int   type ;
   char   log_info[MRD_MAX_LOG_STRING+1] ;

   if( argc < 4 ) {
      printf("usage: %s robot type address [ transport ]\n",
         argv[0]) ;

      exit(1) ;
   }

   robot   = argv[1] ;
   type    = convert_type(argv[2]) ;
   element = argv[3] ;

   if( argc > 4 ) {
      transport = argv[4] ;

      /*
       * If "default" or a suitable abbreviation is used
       * use NULL for the transport name, to indicate that
       * the SCSI default transport should be used.
       */
#ifdef   vms
      if( strncmp("default", transport, strlen(transport)) == 0 )
#else
      if( strncasecmp("default", transport, strlen(transport)) == 0 )
#endif
         transport = NULL ;
   }
   else
      transport = "0" ;

   status = mrd_position(robot, transport, element, type,
                         side, log_info) ;
   if( status != MRD_STATUS_SUCCESS )
      printf("Position failed: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;
   else if ( transport == NULL )
      printf("Positioned default Transport to %s #%s\n",
         mrd_strelement(type), element) ;
   else
      printf("Positioned Transport #%s to %s #%s\n",
         mrd_strelement(type), element) ;

   return 0 ;
}
```

14.3.2. Return Values

Upon successful completion, the **mrd_position(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_position(3mrd)** fails the returned status value may be set to

one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name*, *log_info*, or *element* arguments are NULL pointers.

2. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   It is used when the medium changer does not support the Position To Element command. The seven and five slot DLT loaders do not support the command, though the TL820 and TL810 family libraries do. Some models of TLZ6L and TLZ7L do not support the command and may take a long time to fail.

   It is also used for a SCSI command failure, when the ASC is set to one of:

   - 0x1A - Parameter list length error
   - 0x20 - Invalid command operation code
   - 0x22 - Unsupported command
   - 0x24 - Illegal field in CDB
   - 0x25 - Logical unit not supported
   - 0x26 - Threshold parameters not supported
   - 0x28 - Import or Export element accessed
   - 0x2C - Command sequence error
   - 0x39 - Saving parameters not supported
   - 0x3D - Invalid bits in Identify message
   - 0x53 - Medium removal prevented

   This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

3. **MRD_STATUS_CART_SIDE_INVALID**

   For routines that use the *cartridge_side* argument, this error indicates that the value is neither one (1) nor two (2).

4. **MRD_STATUS_INVALID_TYPE**

   For routines that allow the specification of an element type argument, this error indicates that specified type was not one of SLOT, DRIVE, PORT or TRANSPORT.

5. **MRD_STATUS_PORT_INVALID**

   This error is returned when the element address for a port is less than zero or greater than the number of ports.

6. **MRD_STATUS_SLOT_INVALID**

This error is returned when the element address for a slot is less than zero or greater than the number of slots.

7. **MRD_STATUS_TRANSPORT_INVALID**

   This error is returned when the element address for a transport is less than zero or greater than the number of transports.

8. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives.

14.3.2.1. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

14.3.2.2. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

- 0x08 - Logical unit communcation errors.
- 0x43 - Message error
- 0x45 - Select or Reselect failure
- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

14.3.2.3. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_DEVICE_INVALID**

This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS $_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

14.3.3. Related Information

Functions:

mrd_position_to_element(3mrd)

Chapter 15. mrd_position_to_element

mrd_position_to_element - Move transport to specified location

15.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_position_to_element(
    robot_info_t *robot_info,
    int           transport,
    int           destination,
    int           invert,
    dev_status_t *dev_status) ;
```

15.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.
- *transport* — The *transport* is the numeric value of the transport which will be moved.
- *destination*— The *destination* is an absolute element address.
- *invert* — The *invert* is a numeric value used to indicate if the medium should be inverted when it is moved. A value of one (1) is used to indicate that the medium should be inverted.
- *dev_status* — The *dev_status* is the address of a dev_status_t structure, which is used to pass back detailed error information in the event of a command failure.

15.3. Description

This routine performs a SCSI Position to Element command. This command positions the trasport to the specified destination element. It is used by **mrd_ position(3mrd)** function.

The *robot_info* argument is the address of a robot_info_t that has been opened by **mrd_startup(3mrd).** This allows multiple position commands (and other commands) to be executed without having to repeat the startup for each command.

The *transport* address is the absolute address of the transport element to be used for the command. Many medium changers allow the use of address zero (0) as the default transport, but some may require a transport address that is valid for the medium changer. For single transport medium changers, the transport base address of the robot_info_t structure, *transport_start* is a suitable address.

The *destination* address is the absolute addresses to be used as the destination for the command . The absolute address can be calculated from a zero relative address by adding it to the base address for the element type. The routine makes no checks for the validity of the address, relying on the medium changer to do this.

A *invert* value of one (1) can be used on medium changers that support inverting the transport, when this is desired; an optical drive with two sided media.

Otherwise a value of zero should be used.

This routine uses the **dev_status_t** structure for handing errors. The **dev_ status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

15.3.1. Absolute Element Addresses

The operating system interface routines use absolute SCSI element addresses, instead of zero relative address as used by the higher level functions. A zero based element address can be converted to an absolute address by adding the element base address from the robot_info_t structure.

```
int slot ;
robot_info_t robot_info ;

/*
 * An relative starting address.
 */
slot = 3 ;

/*
 * Becoming an absolute address.
 */
```

```
slot += robot_info.slot_start ;
```

15.3.2. Example

```
/*
 *   This is an example of using mrd_position_to_element(3mrd)
 *   to perform multiple Position To Element commands. For
 *   each pair of arguments after the robot and transport
 *   address, it will position the transport to that location.
 *
 *     mrd_position_to_element robot transport type address
 *
 *   Type can be one of:
 *
 *     slot, port, drive or transport
 *
 *   The optional transport argument can be a transport address
 *   number, the word "default" or an empty string. To keep the
 *   example as simple as possible, it doesn't try to invert the
 *   transport.
 */
#ifndef   lint
static char SccsId[] = "@(#)mrd_position_to_element.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <mrd_common.h>
#include <mrd_message.h>

/*
 *   Given a string, resembling one of the element types,
 *   return the SCSI type code for it.
 */
struct {
   int   code ;
   char   *string ;
} etypes[] = {
   TRANSPORT, "transport",
   SLOT, "slot",
   DRIVE, "drive",
   PORT, "port",
} ;

convert_type(char *etype)
{
   register i ;
   /*
    *   For each entry in the array.
    */
   for(i = 0; i < sizeof(etypes)/sizeof(etypes[0]); i++)
      /*
       *   Do a case insensitive comparison, allowing
       *   abbreviations. Return as soon as a match is
       *   found. Return -1 if one isn't found.
       */
#ifdef vms
      if( strncmp(etypes[i].string, etype, strlen(etype)) == 0 )
#else
      if( strncasecmp(etypes[i].string, etype, strlen(etype)) == 0 )
#endif
        return etypes[i].code ;
```

```
   return -1 ;
}

main(int argc, char *argv[])
{
   int   el ; /* Counter */
   int   status ; /* Status from MRD calls */
   int   invert = 0 ; /* Don't invert */
   char   *robot ; /* Robot name */
   int   type ; /* Element type */
   int   element ; /* Relative element addr */
   int   address ; /* Absolute element addr */
   int   transport ; /* Transport address */
   char   *transport_name ; /* Tranport name */
   robot_info_t   robot_info ;
   dev_status_t   dev_status ;
   char     log_info[MRD_MAX_LOG_STRING+1] ;

   if( argc < 5 ) {
     printf("usage: %s robot transport type address ...\n", argv[0]);
     exit(1) ;
   }

   /*
    *   Get the medium changer name.
    */
   robot = argv[1] ;

   /*
    *   Get the transport number. We'll keep it as a name
    *   so we can detect the default transport. Once we
    *   know the element addresses, we can add the base
    *   base address if appropriate.
    */
   if( strcmp(argv[2], "default") == 0 )
      transport_name = NULL ;
   else
      transport_name = argv[2] ;
   /*
    *   Make sure there are pairs of arguments left. There
    *   should be an odd number.
    */
   if((argc % 2) == 0 ) {
      printf("Pairs of arguments are required.\n") ;
      exit(1) ;
   }

   /*
    *   Open the robot.
    */
   robot_info.channel = BAD_CHANNEL ;

   status = mrd_startup(robot, &robot_info, log_info) ;

   if( status != MRD_STATUS_SUCCESS ) {
      fprintf(stderr, "Can't start %s: %s\n", robot,
         mrd_strstatus(status)) ;

      exit(1) ;
   }

   if( transport_name == NULL )
      transport = 0 ;
   else
      transport = atoi(transport_name) + robot_info.transport_start;
```

```
   /*
    * Look at the element addresses in pairs.
    */
   for(el = 3; el < argc; el += 2) {
      type = convert_type(argv[el]) ;
      element = atoi(argv[el + 1]) ;

      switch( type ) {
      case SLOT:
         address = element + robot_info.slot_start ;
         break ;
      case DRIVE:
         address = element + robot_info.device_start ;
         break ;
      case TRANSPORT:
         address = element + robot_info.transport_start ;
         break ;
      case PORT:
         address = element + robot_info.port_start ;
         break ;
      default:
         printf("Unknown element type: %s %s\n", argv[el],
            argv[el + 1]) ;
         continue ;
      }

      /*
       * Audit the command.
       */
      printf("Position transport to %s #%d.\n", mrd_strelement(type),
         element) ;

      /*
       * Do the command.
       */
      status = mrd_position_to_element(&robot_info, transport,
                address, invert, &dev_status) ;
      if( status != MRD_STATUS_SUCCESS )
         printf("Position to Element failed: %s: %s.\n", robot,
            mrd_strstatus(status)) ;
   }

   (void)mrd_shutdown(&robot_info) ;

   return 0 ;
}
```

15.3.3. Return Values

Upon successful completion, **mrd_position_to_element(3mrd)** will return MRD_STATUS_SUCCESS. On a failure, an MRD_STATUS value corresponding to the error will be returned.

15.3.3.1. Common Codes

Common errors are:

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_info* or *dev_status* arguments are NULL pointers.

2. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

This error occurs when the medium changer does not support the Position To Element command. The seven and five slot DLT loaders do not support the command, though the TL820 and TL810 family libraries do. Some models of TLZ6L and TLZ7L do not support the command and may take a long time to fail.

It is also used for a SCSI command failure, when the ASC is set to one of:

- 0x1A - Parameter list length error
- 0x20 - Invalid command operation code
- 0x22 - Unsupported command
- 0x24 - Illegal field in CDB
- 0x25 - Logical unit not supported
- 0x26 - Threshold parameters not supported
- 0x28 - Import or Export element accessed
- 0x2C - Command sequence error
- 0x39 - Saving parameters not supported
- 0x3D - Invalid bits in Identify message
- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

3. **MRD_STATUS_ELEMENT_INVALID**

   This error occurs when a SCSI command fails with the ASC set to 0x21. The *log_info* will contain the ASCQ. This indicates that an invalid element address reached the medium-changer. For example, specifying the 13th slot when only 12 slots are present.

15.3.3.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

15.3.3.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_EBADF**

This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

15.3.3.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is used when an OpenVMS system service fails with the status SS$_IVCHAN. It is likely when an operating system specific routine is used on a device that hasn't been opened by **mrd_startup(3mrd)**.

15.3.4. Related Information

Functions:

mrd_position(3mrd)

Chapter 16. mrd_prevent_allow

mrd_prevent_allow - Send a Prevent/Allow Media Removal command

16.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_prevent_allow(
    robot_info_t *robot_info,
    int           lock,
    dev_status_t *dev_status) ;
```

16.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.
- *lock* — The *lock* argument indicates whether media removal should be prevented or allowed.
- *dev_status* — The *dev_status* is the address of a dev_status_t structure, which is used to pass back detailed error information in the event of a command failure.

16.3. Description

This routine performs a SCSI Prevent/Allow Media Removal. It is used by **mrd_lock(3mrd)**. The *robot_info* argument is the address of a robot_info_t that has been opened by **mrd_startup(3mrd)**.

When a *lock* value of one (1) is specified media removal is prevented. When the value zero (0) is used, media removal is allowed. If other values for *lock* are used, the routine will create a status which corresponds to an illegal request.

This routine uses the **dev_status_t** structure for handing errors. The **dev_ status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

16.3.1. Example

```
/*
 *   This is an example of using mrd_prevent_allow(3mrd) to
 *   prevent or allow media removal, where allowed. For
 *   robot name on the command line, the desired version
 *   of the command will be used according lock value.
 *   name.
 *
 *   Usage:
 *
 *   mrd_prevent_allow lock-value robot [ robot... ]
 */
#ifndef   lint
static   char   SccsId[] = "@(#)mrd_prevent_allow.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int     rc ;   /* Counter */
   int     lock ;   /* Lock value */
   int     status ;   /* return status */
   char     *robot ;   /* Robot to open */
   robot_info_t   robot_info ;   /* Robot data */
   dev_status_t   dev_status ;   /* Device status */
   char     log_info[MRD_MAX_LOG_STRING+1] ;
   /*
    *   Check that there are enough arguments.
    */
   if( argc < 3 ) {
      printf("usage: %s lock-value robot [ robot... ]\n", argv[0]) ;
      exit(1) ;
   }
   else
      lock = atoi(argv[1]) ;

   /*
    *   Initialize the channel field of the robot_info, so
```

```
     *   mrd_startup(3mrd) will actually open the robot.
     */
    robot_info.channel = BAD_CHANNEL ;

    for(rc = 2; rc < argc; rc++) {
    /*
     *   The robot for this command.
     */
      robot = argv[rc] ;

      status = mrd_startup(robot, &robot_info, log_info) ;

      if( status != MRD_STATUS_SUCCESS ) {
         printf("Startup failed on %s: %s.\n", robot,
            mrd_strstatus(status)) ;

         continue ;
      }

      printf("Lock value %d on %s.\n", lock, robot) ;

      status = mrd_prevent_allow(&robot_info, lock, &dev_status) ;

      if( status != MRD_STATUS_SUCCESS )
         printf("Prevent/Allow failed on %s: %s.\n", robot,
            mrd_strstatus(status)) ;

      (void)mrd_shutdown(&robot_info) ;
    }

    return 0 ;
}
```

16.3.2. Return Values

Upon successful completion, **mrd_prevent_allow(3mrd)** will return MRD_STATUS_SUCCESS. On a failure, one of the following status values will be returned.

16.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_info* or *dev_status* arguments are NULL pointers.

2. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   This error occurs when the medium changer does not support the Prevent/Allow Medium Removal command or the lock value is not one or zero. The specific cause can be determined by examining the ASC/ASCQ values in the *status* data.

   It is also used for a SCSI command failure, when the ASC is set to one of:

   - 0x1A - Parameter list length error
   - 0x20 - Invalid command operation code
   - 0x22 - Unsupported command
   - 0x24 - Illegal field in CDB
   - 0x25 - Logical unit not supported

- 0x26 - Threshold parameters not supported
- 0x28 - Import or Export element accessed
- 0x2C - Command sequence error
- 0x39 - Saving parameters not supported
- 0x3D - Invalid bits in Identify message
- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

16.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

16.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

16.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is used when an OpenVMS system service fails with the status SS$_IVCHAN. It is likely when an operating system specific routine is used on a device that hasn't been opened by **mrd_startup(3mrd)**.

16.3.3. Related Information

Functions:

mrd_lock(3mrd)

Chapter 17. mrd_ready

mrd_ready - Send a Ready Inport command

17.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_ready(
    robot_info_t *robot_info,
    int           port,
    dev_status_t *dev_status) ;
```

17.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.
- *port* — The absolute integer element address of the port which is to be used as the destination of the move.
- *dev_status* — The *dev_status* is the address of a dev_status_t structure, which is used to pass back detailed error information in the event of a command failure.

17.3. Description

It is used by **mrd_ready_inport(3mrd)**. This command is used by the TL820 family of DLT libraries to enable the button on the I/O device (IOD) which opens the Inport door.

The *robot_info* argument is the address of a robot_info_t that has been opened by **mrd_startup(3mrd)**.

The *port* argument is the absolute address of the port to be readied. On supported TL820 configurations which use a left mounted IOD this will always be 64.

This routine uses the **dev_status_t** structure for handing errors. The **dev_ status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

17.3.1. Example

```
/*
 *   This is an example of using mrd_move_medium directly to move
 *   a cartridge from one slot to another. To simplify the
 *   example, it only supports slot to slot moves, but it shows
 *   how the absolute element addresses are calcuated. For each
 *   additional destination address given, the previous (successful)
 *   destination address is used as the source.
 *
 *   Usage:
 *
 *     mrd_ready robot port [ port... ]
 */
#ifndef lint
static char SccsId[] = "@(#)mrd_ready.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int   pc ;   /* counter */
   int   port ;   /* Port number */
   int   address ;   /* Port address */
   int   status ;   /* return status */
   char   *robot ;   /* Robot to open */
   robot_info_t   robot_info ;   /* Robot data */
   dev_status_t   dev_status ;   /* Device status */
   char   log_info[MRD_MAX_LOG_STRING+1] ;

   /*
    *   Check that there are enough arguments.
    */
   if( argc < 3 ) {
      printf("usage: %s robot port [ port... ]\n", argv[0]) ;
      exit(1) ;
   }
   else
```

```
      robot = argv[1] ;

   /*
    *   Initialize the channel field of the robot_info, so
    *   mrd_startup(3mrd) will actually open the robot.
    */
   robot_info.channel = BAD_CHANNEL ;

   status = mrd_startup(robot, &robot_info, log_info) ;

   if( status != MRD_STATUS_SUCCESS ) {
      printf("Startup failed on %s: %s.\n", robot,
         mrd_strstatus(status)) ;

      exit(1) ;
   }

   /*
    *   For each destination address on the command line,
    *   move the the cartridge in the source to the
    *   destination. After each (successful) move, replace
    *   the previous source with this destination.
    */
   for(pc = 2; pc < argc; pc++) {
      /*
       *   Get the port number.
       */
      port = atoi(argv[pc]) ;

   /*
    *   Now the absolute address.
    */
   address = port + robot_info.port_start ;
   /*
    *   Print an audit as we go. Since we know these
    *   are slots, convert back to relative addresses
    *   for the audit.
    */
   printf("Ready Inport #%d of %s\n", port, robot) ;

   status = mrd_ready(&robot_info, address, &dev_status) ;

   if( status != MRD_STATUS_SUCCESS ) {
      printf("Ready Inport failed on %s: %s.\n", robot,
         mrd_strstatus(status)) ;
      /*
       *   Since the cartridge didn't move, don't
       *   reset the source, by skipping the remainder
       *   of the loop.
       */
      continue ;
     }
   }
   (void)mrd_shutdown(&robot_info) ;

   return 0 ;
}
```

17.3.2. Return Values

17.3.2.1. Common Codes

Upon successful completion, **mrd_ready(3mrd)** will return MRD_STATUS_SUCCESS. On a failure, one of the following status values will be returned.

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_info* or *dev_status* arguments are NULL pointers.

2. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   This error occurs when the medium changer does not support the Ready Inport command. The TL820 family of DLT libraries support this command. The TL810 family of DLT libraries allows this command to succeed, but it doesn't perform any function.

   It is also used for a SCSI command failure, when the ASC is set to one of:

   - 0x1A - Parameter list length error
   - 0x20 - Invalid command operation code
   - 0x22 - Unsupported command
   - 0x24 - Illegal field in CDB
   - 0x25 - Logical unit not supported
   - 0x26 - Threshold parameters not supported
   - 0x28 - Import or Export element accessed
   - 0x2C - Command sequence error
   - 0x39 - Saving parameters not supported
   - 0x3D - Invalid bits in Identify message
   - 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

17.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

17.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

17.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is used when an OpenVMS system service fails with the status SS$_IVCHAN. It is likely when an operating system specific routine is used on a device that hasn't been opened by **mrd_startup(3mrd)**.

17.3.3. Related Information

Functions:

mrd_ready_inport(3mrd)

Chapter 18. mrd_ready_inport

mrd_ready_inport - Enable access to the Inport of a TL820

18.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_ready_inport(
    const char *robot_name,
    const char *inport,
    char *log_info) ;
```

18.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *inport* — This is the address of the character string containing a zero based address of the port to be readied.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

18.3. Description

The TL820 family of libraries uses an Input/Output Device (IOD) to allow putting tapes into the library and taking them out. The inport part of the IOD holds a single cartridge and has a door on top which must be opened before a cartridge can be placed in it. A button on the front of the IOD opens the door, but only after it has been enabled by a Ready IOD command. This is a vendor unique command specific to the TL820 family.

The **mrd_ready_inport(3mrd)** routine allows sending a Ready IOD command to the robot and port specified by the *robot_name* and *inport* arguments. For all currently shipping TL820 family libraries there is only one Inport, so the *inport* argument can always be "0". On a TL820, when the source for a move in the

Inport, this routine should be called first and time allowed for the operator to place a tape in the inport. The routine **mrd_move(3mrd)** expects the element to be full when it starts.

18.3.1. Example

```
/*
```

```
 *   Send a Ready Inport command to the robot. This is specific
 *   the TL82X family and causes the Inport door to be enabled
 *   for one minute (the period the light is on). A future
 *   version of firmware may allow enableing the button to be
 *   on all the time, making this command obsolete. The command
 *   usage is:
 *
 *     mrd_ready_inport robot
 */

#ifndef   lint
static   char   SccsId[] = "@(#)mrd_ready_inport.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   int   status ;   /* status from mrd_ready_inport(3mrd) */
   char   *robot ;   /* Robot for command */
   char   log_info[MRD_MAX_LOG_STRING+1] ;   /* error text */

   /*
    * Only one argument; the robot name.
    */
   if( argc == 1 ) {
      printf("usage: %s robot\n", argv[0]) ;
      exit(1) ;
   }
   else
      robot = argv[1] ;

   /*
    *   While the interface of Ready Inport allows the specification
    *   of any port address, the Inport of the TL820 is always "0",
    *   and this command is very robot specific.
    */
   status = mrd_ready_inport(robot, "0", log_info) ;

   if( status != MRD_STATUS_SUCCESS )
      printf("Ready Inport failed: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;

   return 0 ;
}
```

18.3.2. Return Values

Upon successful completion, the **mrd_ready_inport(3mrd)** function returns the value MRD_STATUS_SUCCESS. If **mrd_ready_inport(3mrd)** fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

18.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the any of the arguments are NULL pointers.

2. **MRD_STATUS_PORT_INVALID**

This error is returned when the element address for a port is less than zero or greater than the number of ports.

18.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

18.3.2.2.1. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received

- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

18.3.2.2.2. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

18.3.3. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_load**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_inject**(3mrd)

- **mrd_eject**(3mrd)

Chapter 19. mrd_read_element_status

mrd_read_element_status - Obtain information about elements.

19.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_read_element_status(
    robot_info_t  *robot_info,
    int            type,
    int            start,
    int            count,
    unsigned char *data,
    int            length,
    dev_status_t  *dev_status) ;
```

19.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.
- *type* — This is the element type code about which information is desired. The **mrd_common.h** include file defines the constants SLOT, PORT, DRIVE and TRANSPORT which may be used.
- *start* — The absolute element address of the first element for which information is desired.
- *count* — The number of elements for which the information is desired.
- *data* — This is the address of an array of unsigned characters where the element status data will be written. Interpretation of the data is left to the caller.
- *length* — This is the amount of *element_status* data, in bytes, available for the Read Element Status request. If the requested number of elements requires more space only data for as many elements as will fit will be copied.
- *dev_status* — The *dev_status* is the address of a dev_status_t structure, which is used to pass back detailed error information in the event of a command failure.

19.3. Description

This routine performs a SCSI Read Element Status command, or equivalent if some other I/O architecture is supported. It is used by **mrd_show(3mrd)** and the routines doing volume tag checks. However, since it provides uninterpreted Read Element Status data, **mrd_show(3mrd)** will nearly always be easier to use.

It requires that the medium changer be opened by **mrd_startup(3mrd)** and uses absolute element addresses. On SCSI medium changers, it maps directly to the SCSI Read Element Status command. Since it uses a robot_info_t structure for an open robot, it is suitable in applications where it is desirable to hold the robot open and not incur the robot startup time on each command.

The *type* argument specifies the type of element about which information is to be obtained. It should be one of SLOT, TRANSPORT, PORT or DRIVE as defined in **mrd_common.h.** The *start* argument is the absolute address of the first element and *count* the number of elements for which data is to be obtained.

The *data* argument is an array of unsigned characters where the resulting data will be copied. The *length* is the amount of space available. If more data is required than there is space available, the device will only data for as many element as will fit into *length* bytes.

Medium Changers which are SCSI-2 compliant support Read Element Status commands which request only eight bytes of data. In this case the returned data will indicate how many bytes of data are needed for the entire request. This feature allows an application to find how much space is needed for a specific request, allocate that much and then request all of it.

This routine uses the **dev_status_t** structure for handing errors. The **dev_ status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

19.3.1. Absolute Element Addresses

The operating system interface routines use absolute SCSI element addresses, instead of zero relative address as used by the higher level functions. A zero based element address can be converted to an absolute address by adding the element base address from the robot_info_t structure.

```
int slot ;
robot_info_t robot_info ;

/*
 * An relative starting address.
 */
```

```
slot = 3 ;

/*
 * Becoming an absolute address.
 */
slot += robot_info.slot_start ;
```

19.3.2. Example

```
/*
 * This is an example of using mrd_read_element_status(3mrd).
 *
 * Due to the complexity of the SCSI Read Element Status data
 * all this example will do is format the headers found in the
 * data. It won't try to format the element data. It also
 * calls mrd_read_element_status(3mrd) twice, once to determine
 * the needed data size for the remaining data and again to get
 * the actual data.
 *
 *   Usage:
 *
 * mrd_read_element_status robot type start count
 *
 * If an unrecognized element type is used, the routine will use
 * a type of zero, which is allowed by the SCSI-2 specification.
 */

#ifndef lint
static char SccsId[] = "@(#)mrd_read_element_status.c 1.3
                        (mrd-example) 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

#include <mrd_common.h>
#include <mrd_message.h>

/*
 * The SCSI specification says that a request size of 8
 * bytes will have the underlying device only return a
 * header indicating the number of bytes needed for the
 * command.
 */
#define SCSI_RES_MIN (8)

/*
 * Given a string, resembling one of the element types,
 * return the SCSI type code for it.
 */
struct {
 int code ;
 char *string ;
} etypes[] = {
TRANSPORT,   "transport",
 SLOT,   "slot",
 DRIVE,   "drive",
 PORT,   "port",
} ;

convert_type(char *etype)
{
 register i ;
```

```
 /*
  * For each entry in the array.
  */
 for(i = 0; i < sizeof(etypes)/sizeof(etypes[0]); i++)
  /*
   * Do a case insensitive comparison, allowing
   * abbreviations. Return as soon as a match is
   * found. Return -1 if one isn't found.
   */
#ifdef vms
  if( strncmp(etypes[i].string, etype, strlen(etype)) == 0 )
#else
  if( strncasecmp(etypes[i].string, etype, strlen(etype)) == 0 )
#endif
   return etypes[i].code ;
 return 0 ;
}

/*
 * When an 8 byte Read Element Status command is handed
 * to a compliant medium changer, it is supposed to fill
 * enough of the header section to say how much data is
 * needed for the full command. We that here, if we get
 * a reasonable value, allocate sufficient space for the
 * real command and return the pointer to it. If there
 * is an error or command returns a zero byte report
 * return NULL.
 */
unsigned char *
res_size(robot_info_t *robot_info, int type, int start, int count, size_t *bytes)
{
 unsigned char data[SCSI_RES_MIN] ; /* minimum data */
 int status ; /* command status */
 dev_status_t dev_status ; /* In case of error */
 unsigned char *report ; /* Data space */

 /*
  * Read Element Status commands rarely fail, they just
  * succeed, but return no data. Clear all the fields
  * so we'll have an easier time seeing if any data was
  * returned.
  */
 memset((void *)data, 0, SCSI_RES_MIN) ;

 status = mrd_read_element_status(robot_info, type, start, count,
   data, SCSI_RES_MIN, &dev_status) ;
 /*
  * But sometimes they do fail.
  */
 if( status != MRD_STATUS_SUCCESS ) {
  printf("Size Read Element Status failed on %s: %s.\n",
   robot_info->robot_name, mrd_strstatus(status)) ;

 return NULL ;
 }

 /*
  * Calculate the report size.
  */
 *bytes = (data[RES_REPORT_MSB] << 16) | (data[RES_REPORT_ISB] << 8) |
   data[RES_REPORT_LSB] ;
 /*
  * The report size doesn't include the 8 bytes needed for
  * the first headers.
  */
```

```
 *bytes += RES_DATA_HEADER ;

 if( *bytes == 0 ) {
  printf("The report size is zero on %s.\n",
   robot_info->robot_name) ;

 return NULL ;
 }

 if((report = (unsigned char *)malloc(*bytes)) == NULL )
  printf("Can't allocate %ld bytes for %s: %s.\n", *bytes,
   robot_info->robot_name, strerror(errno)) ;

 return report ;
}
/*
 * Print out the Read Element Status data headers. There
 * is an 8 byte data header, that describes the remaining
 * data, which is zero or more Status Pages, one for each
 * element type.
 *
 * Each Element Status Page consists of an 8 byte header
 * for the element type described by that page and then
 * element descriptors.
 */
print_res_data(robot_info_t *robot_info, unsigned char *dp)
{
 int first ; /* First element reported */
 int elements ; /* Number of elements reported */
 int report ; /* Bytes in the total report */
 int descriptor ; /* Each element descriptor size */
 int bytes ; /* The per-element report size */

 /*
  * The first two bytes of the overall header is the
  * first element reported.
  */
 first = (dp[RES_FIRST_MSB] << 8) | dp[RES_FIRST_LSB] ;

 /*
  * The next two bytes are the number of elements in
  * the report.
  */
elements = (dp[RES_COUNT_MSB] << 8) | dp[RES_COUNT_LSB] ;
/*
 * Three of the remaining bytes are the total report size.
 */
report = (dp[RES_REPORT_MSB] << 16) | (dp[RES_REPORT_ISB] << 8) |
   dp[RES_REPORT_LSB] ;

printf("RES Data Header:\n") ;
printf("   First Element Address:  %d\n",  first) ;
printf("   Number of Elements:     %d\n",  elements) ;
printf("   Byte Count of Report:   %d\n",  report) ;

/*
 * As long as bytes of report remaining, print each
 * element type header.
 *
* Here we play a curious pointer game. The RES_
* constants defined in mrd_common.h for the element
* header assume a single element type with offsets
* from the beginning of the data. But real Read
* Element Status Data can have multiple element
* types in it. For successive element type we'll
* add the per-element report size to the address
```

```
* of the base data. This should put the per-element
* header at the right relative place.
*/
report -= RES_DATA_HEADER ;

while( report > 0 ) {
 /*
  * Calculate the descriptor size.
  */
 descriptor = (dp[RES_DESC_MSB] << 8) | dp[RES_DESC_LSB] ;

 /*
  * And the per element report.
  */
 bytes = (dp[RES_BYTES_MSB] << 16) | (dp[RES_BYTES_ISB] << 8) |
   dp[RES_BYTES_LSB] ;

 printf("   Descriptor Header:\n") ;
 printf("     Element Type: %s\n",
  mrd_strelement(dp[RES_TYPE])) ;

 printf("     Primary Volume Tag: %x\n",
  dp[RES_TAGS] & ELEMENT_PVOLTAG) ;

 printf("     Alternate Volume Tag: %x\n",
  dp[RES_TAGS] & ELEMENT_AVOLTAG) ;

 printf("     Descriptor Length: %d\n", descriptor) ;
 printf("     Descriptor Report: %d", bytes) ;

 /*
  * Include the number of elements of this type.
  */
 if( descriptor )
  printf(" (%d)\n", bytes / descriptor) ;
 else
  putchar('\n') ;

 /*
  * Protection against the odd insane loader.
  */
 if( bytes == 0 )
  break ;
  /*
   * Go to the next header. Formatting the element
   * data is left as an exercise to the reader.
   */
  dp += (bytes + RES_DATA_PAGE) ;

  /*
   * Lose some bytes...
   */
  report -= (bytes + RES_DATA_PAGE) ;
 }

}

main(int argc, char *argv[])
{
 int  status ;  /* return status */
 int  type ;  /* Element type */
 int  start ;  /* First element */
 int  count ;  /* Number of elements */
 size_t  bytes ;  /* Bytes of data */
 char  *robot ;  /* Robot to open */
 unsigned char *data ;  /* Element Status data */
```

```
robot_info_t robot_info ;  /* Robot data */
dev_status_t dev_status ; /* Device status */
char log_info[MRD_MAX_LOG_STRING+1] ;

/*
 * Check that there are enough arguments.
 */
if( argc < 5 ) {
 printf("usage: %s robot type start count\n", argv[0]) ;
 exit(1) ;
}
else {
 robot = argv[1] ;
 type = convert_type(argv[2]) ;
 start = atoi(argv[3]) ;
 count = atoi(argv[4]) ;
}

/*
 * Initialize the channel field of the robot_info, so
 * mrd_startup(3mrd) will actually open the robot.
 */
robot_info.channel = BAD_CHANNEL ;

status = mrd_startup(robot, &robot_info, log_info) ;

if( status != MRD_STATUS_SUCCESS ) {
 printf("Startup failed: %s: %s.\n", mrd_strstatus(status),
  log_info[0] ? log_info : "none") ;

 exit(1) ;
}

if( type == 0 )
 printf("Data size needed for all elements %d - %d...",
  start, start + count) ;
else
 printf("Data size needed for %s %d - %d...",
  mrd_strelement(type),
  start, start + count) ;

fflush(stdout) ;
switch( type ) {
case SLOT:
 start += robot_info.slot_start ;
 break ;
case PORT:
 start += robot_info.port_start ;
 break ;
case TRANSPORT:
 start += robot_info.transport_start ;
 break ;
case DRIVE:
 start += robot_info.device_start ;
 break ;
}

/*
 * Allocate sufficient space for the command. This
 * function prints its own error messages, so we can
 * just exit.
 */
data = res_size(&robot_info, type, start, count, &bytes) ;

if( data == NULL )
 exit(1) ;
```

```
 printf("%d bytes.\n", bytes) ;

 /*
  * Now do the full Read Element Status command.
  */
 status = mrd_read_element_status(&robot_info, type, start, count,
   data, bytes, &dev_status) ;

 if( status != MRD_STATUS_SUCCESS ) {
  printf("Read Element Status failed on %s: %s.\n", robot,
   mrd_strstatus(status)) ;

  free(data) ;

  exit(1) ;
 }

 /*
  * We appear to have valid Read Element Status data. Print
  * out the results.
  */
 print_res_data(&robot_info, data) ;

 (void)mrd_shutdown(&robot_info) ;

 return 0 ;
}
```

19.3.3. Return Values

Upon successful completion, **mrd_read_element_status(3mrd)** will return MRD_STATUS_SUCCESS. On a failure, one of the following status values will be returned.

Experience has shown that Read Element Status rarely fails on the supported SCSI-2 medium changers. When the command is unable to obtain the requested data it simply arranges for the element and byte counts of the report to contain no data.

19.3.3.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_info*, *data* or *dev_status* arguments are are NULL pointers. The *status* structure is unchanged, even if a valid address is provided.

2. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   This error occurs when *robot_info* structure indicates that the medium changer supports volume tags, when it doesn't. When **mrd_startup(3mrd)** opens the robot, it determines whether volume tags are support and sets up the structure appropriately. So, this error is only likely to occur when the structure has been changed.

   It is also used for a SCSI command failure, when the ASC is set to one of:

   - 0x1A - Parameter list length error
   - 0x20 - Invalid command operation code
   - 0x22 - Unsupported command

- 0x24 - Illegal field in CDB
- 0x25 - Logical unit not supported
- 0x26 - Threshold parameters not supported
- 0x28 - Import or Export element accessed
- 0x2C - Command sequence error
- 0x39 - Saving parameters not supported
- 0x3D - Invalid bits in Identify message
- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

19.3.3.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

19.3.3.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

19.3.3.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is used when an OpenVMS system service fails with the status SS$_IVCHAN. It is likely when an operating system specific routine is used on a device that hasn't been opened by **mrd_startup(3mrd)**.

19.3.4. Related Information

Functions:

mrd_show(3mrd)

Chapter 20. mrd_request_sense

mrd_request_sense - Get the status of a medium changer.

20.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_request_sense(
    robot_info_t *robot_info,
    dev_status_t *dev_status,
    int os_status);
```

20.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.

- *dev_status* — The *dev_status* is the address of a dev_status_t structure, which is used to pass back detailed error information in the event of a command failure.

- *os_status* — When **mrd_request_sense(3mrd)** is used directly by an application, this argument should be MRD_CHECK_SENSE, or the operating system specific error code that indicates a device failure. On Tru64 UNIX this is EIO.

20.3. Description

This routine performs a SCSI Request Sense command, or equivalent if some other I/O architecture is supported. It is used by all MRD API routines to determine the cause of a command failure.

The *robot_info* is the address of a *robot_info_t* structure that has been opened by **mrd_startup(3mrd)**. If the medium changer isn't opened, the Request Sense command will fail with the operating system error for trying to use an unopened device.

The **dev_status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

In typical usage by MRD, the *os_status* argument will be an operating system specific code. However, The SCSI-2 specification allows Request Sense to be used at any time to obtain status information about a device. To support this feature, the MRD implementation of **mrd_reqeust_sense(3mrd)** can be called with the code MRD_CHECK_SENSE to force a Request Sense command.

20.3.1. Example

```
/*
 *   This is an example of using mrd_request_sense(3mrd)
 *   to see what state a medium changer is in. The MRD
 *   implementation of Request Sense only collects the
 *   Sense Key, Additional Sense Code and Additional Sense
 *   Code Qualifier.
 *
 *   Usage:
 *
 *      mrd_request_sense robot [ more-robots... ]
 */

#ifndef   lint
static   char SccsId[] = "@(#)mrd_request_sense.c 1.1 4/16/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <mrd_common.h>
#include <mrd_message.h>

char  *device_sense = "Sense data for %s: %s (%d,0x%x,0x%x).\n" ;
char  *sense_failed = "Request Sense failed on %s: %s.\n" ;

main(int argc, char *argv[])
{
   int     rc ;    /* counter */
   int     status ;    /* return status */
   char     *robot ;    /* Robot to open */
   robot_info_t robot_info ;   /* Robot data */
   dev_status_t dev_status ;   /* Device status */
   char log_info[MRD_MAX_LOG_STRING+1] ;
```

```
   /*
    * Check that there are enough arguments.
    */
   if( argc < 2 ) {
      printf("usage: %s robot [ robot... ]\n", argv[0]) ;
      exit(1) ;
   }

   /*
    *   Initialize the channel field of the robot_info, so
    *   mrd_startup(3mrd) will actually open the robot.
    */
   robot_info.channel = BAD_CHANNEL ;

   for(rc = 1; rc < argc; rc++) {
      /*
       * The robot for this command.
       */
      robot = argv[rc] ;

      status = mrd_startup(robot, &robot_info, log_info) ;

      if( status != MRD_STATUS_SUCCESS ) {
         printf("Startup failed on %s: %s.\n", robot,
            mrd_strstatus(status)) ;
         continue ;
      }

      memset((void *)&dev_status, 0, sizeof(dev_status)) ;

      /*
       * mrd_request_sense(3mrd) will never return
       * MRD_STATUS_SUCCESS. If no Request Sense data
       * is available, it will return MRD_STATUS_NO_SENSE.
       */
      status = mrd_request_sense(&robot_info, &dev_status,
            MRD_CHECK_SENSE) ;

      /*
       * Print the Key/ASC/ASCQ data for device errors.
       */
      if( dev_status.valid )
         printf(device_sense, robot, mrd_strstatus(status),
            dev_status.key, dev_status.asc,
            dev_status.ascq) ;
      /*
       * Just print the MRD error.
       */
      else
         printf(sense_failed, robot, mrd_strstatus(status)) ;

      (void)mrd_shutdown(&robot_info) ;
   }

   return 0 ;
}
```

20.3.2. Return Values

The routine **mrd_request_sense(3mrd)** never returns MRD_STATUS_SUCCESS. If the *os_status* isn't the operating system specific code that forces a Reqeust Sense command or MRD_CHECK_SENSE, **mrd_map_os_error(3mrd)**, is used to map the *os_status* to an MRD status code. Otherwise, a Request Sense (or equivalent) is performed and the result mapped to an MRD status code with **mrd_scsi_decode(3mrd)**.

20.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned when a pointer argument passed to an MRD routine is NULL, unless the routine is documented as one allowing a NULL pointer.

2. **MRD_STATUS_NO_SENSE**

   This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc, ascq* and *key* values are all zero (0). It is also returned when the *key* value is less than zero or greater than 15.

3. **MRD_STATUS_RECOVERED_ERROR**

   This error occurs when a SCSI device returns only a sense key of 1h. This indicates that although a command successfully completed, the target device had performed some internal error recovery.

4. **MRD_STATUS_MEDIUM_ERROR**

   This error occurs when ASC and ASCQ are zero, but the sense key is 3h. This occurs when the target encounters a nonrecoverable error due to a flaw in the medium.

5. **MRD_STATUS_ROBOT_HW_ERROR**

   This error occurs when ASC and ASCQ are zero, but the sense key is 4h. This occurs when the target encounters a nonrecoverable hardware error.

6. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   This error occurs for a variety of reasons.

   It is used when a sanity check fails in the code that attempts to move a cartridge to the Pass-Through Mechanism, when the robot type isn't a TL82n.

   It is used in the **mrd_lock(3mrd)** code when the value is not one of ALLOW_REMOVAL or PREVENT_REMOVAL.

   It is used when the medium changer does not support the Prevent/Allow Medium Removal command or the lock value is not one or zero. The specific cause can be determined by examining the ASC/ASCQ values in the *status* data.

   It is used when a call to **mrd_initialize_element(3mrd)** is issued against a medium changer that does not support the Initialize Element Status command.

   It is used when the medium changer does not support the Position To Element command. The seven and five slot DLT loaders do not support the command, though the TL820 and TL810 family libraries do. Some models of TLZ6L and TLZ7L do not support the command and may take a long time to fail.

   It is used when the medium changer does not support the Ready Inport command.

   The TL820 family of DLT libraries support this command. The TL810 family of DLT libraries allows this command to succeed, but it doesn't perform any function.

   It is also used for a SCSI command failure, when the ASC is set to one of:

- 0x1A - Parameter list length error
- 0x20 - Invalid command operation code
- 0x22 - Unsupported command
- 0x24 - Illegal field in CDB
- 0x25 - Logical unit not supported
- 0x26 - Threshold parameters not supported
- 0x28 - Import or Export element accessed
- 0x2C - Command sequence error
- 0x39 - Saving parameters not supported
- 0x3D - Invalid bits in Identify message
- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

7. **MRD_STATUS_ROBOT_ATTENTION**

   This error occurs when a SCSI command fails with the ASC set to one of 0x29, 0x2A or 0x2F. The *log_info* contains the ASCQ. The SCSI translations for these error codes are:

   - 0x29 - Power-on, Reset or Bus device reset occurred
   - 0x2A - Mode Parameters Changed
   - 0x2F - Command cleared by another initiator

   This error also occurs when the ASC and ASCQ are zero, but the SCSI sense key is 6h.

8. **MRD_STATUS_DATA_PROTECT**

   This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is seven (7).

9. **MRD_STATUS_BLANK_CHECK**

   This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is eight (8).

10. **MRD_STATUS_VENDOR_UNIQUE_ERROR**

    This error occurs when the internal routine used to decode SCSI-2 errors encounters an error that it has not been written to antipicate.

    This error is also returned when the ASC is zero and the ASCQ is not one of zero or six, and when ASC/ASCQ are both zero and the *key* is 9h.

11. **MRD_STATUS_COPY_ABORTED**

This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is ten (10).

12. **MRD_STATUS_SENSE_EQUAL**

    This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Ch (12).

13. **MRD_STATUS_VOLUME_OVERFLOW**

    This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Dh (13).

14. **MRD_STATUS_MISCOMPARE**

    This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Eh (14).

15. **MRD_STATUS_SENSE_RESERVED**

    This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Fh (15).

16. **MRD_STATUS_ROBOT_COMM_ERROR**

    This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

    - 0x08 - Logical unit communcation errors.
    - 0x43 - Message error
    - 0x45 - Select or Reselect failure
    - 0x47 - SCSI parity error
    - 0x48 - Initiator detected error message received
    - 0x49 - Invalid message error
    - 0x4A - Command phase error
    - 0x4B - Data phase error
    - 0x4E - Overlapped commands attempted
    - 0x54 - SCSI to host system interface failure

17. **MRD_STATUS_ROBOT_MECH_ERROR**

    This error occurs as the result of a SCSI command failure, when the ASC is set to one of:

    - 0x15 - Positioning error.
    - 0x8B - Vendor unique; Pass-through mechanism errors on the TL82n

18. **MRD_STATUS_AUTOCLEAN**

This error occurs when a SCSI command fails with the ASC set to 0x30 and the ASCQ set to 0x3. On TL8nn libraries supporting Auto-clean, it indicates that a command was attempted while an auto-clean was in progress.

19. **MRD_STATUS_CART_DAMAGED**

    This error occurs when a SCSI command fails with the ASC set to 0x30, but the ASCQ is NOT a value of 0x3. The *log_info* will contain the ASCQ.

20. **MRD_STATUS_ELEMENT_INVALID**

    This error occurs when a SCSI command fails with the ASC set to 0x21. The *log_info* will contain the ASCQ. This indicates that an invalid element address reached the medium-changer. For example, specifying the 13th slot when only 12 slots are present.

21. **MRD_STATUS_CART_NOT_AVAIL**

    This error can occur on the TL81n and TL82n family of DLT libraries when the source of a move is a drive and the cartridge in the drive is still on-line. These robots do not allow moving the cartridge until the drive is taken offline.

22. **MRD_STATUS_DESTINATION_FULL**

    On routines that perform a SCSI Move Medium command, this error indicates that the destination element already has a cartridge in it.

23. **MRD_STATUS_SOURCE_EMPTY**

    On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

24. **MRD_STATUS_ROBOT_DOOR_OPENED**

    This occurs when a SCSI command fails with the ASC set to 0x80 and the ASCQ set to 0x0. On TL8nn libraries this typically indicates that the cabinet door was opened during a command operation.

20.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under Microsoft Windows 2000/Windows XP, this error code is returned when the specified robot exists but is not responding.

3. **MRD_STATUS_ROBOT_CMD_ABORTED**

   This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* is zero and the *ascq* is six, or when the *asc* and *ascq* are zero and the *key* is eleven (11).

20.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

2. **MRD_STATUS_EINVAL**

   This error is returned by **mrd_map_os_ error(3mrd)** when the *os_status* is EINVAL. This typically occurs during **mrd_ startup(3mrd)** when the special file is not a SCSI device: for example, /dev/tty.

3. **MRD_STATUS_STARTUP_ERROR**

   This error is returned by **mrd_ map_os_error(3mrd)** when the *os_status* is ENODEV. This typically occurs during **mrd_startup(3mrd)** when the special file is not a SCSI device; /dev/null.

4. **MRD_STATUS_NO_SUCH_DEVICE**

   This error occurs when a UNIX system call returns ENXIO, to indicate that the device corresponding to the special device does not exist.

5. **MRD_STATUS_EBUSY**

   This error occurs when a UNIX system call returns EBUSY, to indicate that some other process is using that medium-changer device.

6. **MRD_STATUS_EINTR**

   This error occurs when a UNIX system call returns EINTR. This error corresponds to an interrupted system call, but also occurs when the SCSI CAM Layered Components Medium-Changer driver is not configured into the running system.

7. **MRD_STATUS_EIO**

   This error occurs when a UNIX system call returns EIO to indicate that there was an I/O error. In most cases an I/O error on a SCSI medium-changer indicates a SCSI error which be translated to another MRD error.

8. **MRD_STATUS_ENOENT**

   This error occurs when a UNIX system call returns ENOENT to indicate that a special device file doesn't exist.

9. **MRD_STATUS_EACCES**

   This error occurs when a UNIX system call returns EACCES to indicate that the caller does not have sufficient permission to open the special device file corresponding to the medium-changer. MRD expects to have read permission on the special device file.

10. **MRD_STATUS_OS_ERROR**

This error occurs when a UNIX system call returns an error that is not among those previously mentioned. The routine strerror(3) will be used to translate the error code into a standard text message which will be copied to *log_info*.

11. **MRD_STATUS_ROBOT_NOT_READY**

    Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

    - 0x80 - When the ASCQ is not zero (0).
    - 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
    - 0x04 - Logical unit not ready
    - 0x3E - Logical unit has not been self configured
    - 0x40 - Diagnostic failure; ASCQ indicates component
    - 0x42 - Power-on self test failure
    - 0x44 - Internal target failure
    - 0x46 - Unsuccessful soft reset
    - 0x4C - Logical unit failed self-configuration

    This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

12. **MRD_STATUS_ROBOT_CMD_ABORTED**

    This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* is zero and the *ascq* is six, or when the *asc* and *ascq* are zero and the *key* is eleven (11).

20.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS $_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x80 - When the ASCQ is not zero (0).

- 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
- 0x04 - Logical unit not ready
- 0x3E - Logical unit has not been self configured
- 0x40 - Diagnostic failure; ASCQ indicates component
- 0x42 - Power-on self test failure
- 0x44 - Internal target failure
- 0x46 - Unsuccessful soft reset
- 0x4C - Logical unit failed self-configuration

This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

3. **MRD_STATUS_ROBOT_CMD_ABORTED**

   This error code is used when an OpenVMS system service fails with the status SS$_ABORT.

20.3.3. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_load**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_inject**(3mrd)
- **mrd_eject**(3mrd)
- **mrd_show**(3mrd)
- **mrd_ready_inport**(3mrd)
- **mrd_position**(3mrd)
- **mrd_initialize**(3mrd)
- **mrd_home**(3mrd)
- **mrd_find_cartridge**(3mrd)
- **mrd_startup**(3mrd)
- **mrd_shutdown**(3mrd)
- **mrd_lock**(3mrd)

Chapter 21. mrd_show

mrd_show - Obtain information from a media robot

21.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_show(
    const char *robot_name,
    robot_info_t *robot_info,
    int element_type,
    const char *element_name,
    int element_count,
    element_info_t *element_info,
    char *log_info) ;
```

21.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.
- *element_type* — The type of robot element for which **mrd_show(3mrd)** returns information. Element types include: PORT, DRIVE, SLOT, TRANSPORT, or ROBOT.
- *element_name* — A string used to specify the name of the first element about which to obtain information. While SCSI devices use integer numbers for element addresses, DSA robots use character strings. This allows the same interface to be used for both types where supported.
- *element_count* — The number of elements about which to obtain information.
- *element_info* — The array of **element_info_t** structures that is filled in with information on the type and number of elements requested with *element_type* and *element_count*. The information includes volume tag (if available), state, port type (if PORT information is requested), status, and when available a copy of the Read Element Status data for the element.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

21.3. Description

The **mrd_show(3mrd)** function can be used to obtain information about specific element types of a supported Medium-Changer. Medium-Changer element types that **mrd_show(3mrd)** can retrieve information about include PORT, DRIVE, SLOT and TRANSPORT. If a *element_type* of ROBOT is specified, the **mrd_ show(3mrd)** is equivalent to calling **mrd_startup(3mrd)** - that is, robot_info is filled in.

The **robot_info_t** data structure is described in **mrd_startup(3mrd)**.

The *element_name* parameter specifies the first element of the type *element_type* about which to obtain information. The *element_count* parameter specifies the number of elements of type *element_type* about which information is to be obtained.

21.3.1. Element Info

The **element_info_t** data structure is defined in the include file **<mrd_ common.h>**. The fields of this data structure are described below:

- *name* — The name field holds the volume tag of the media if applicable.
- *state* — The state field can have one of the following values: ELEMENT_FULL, ELELMENT_EMPTY, or ELEMENT_EXCEPT.
- *port_type* — If the *element_type* parameter specifies PORT, the port_type field will have one of the following values: IN_OUT_PORT, INPORT, OUTPORT.
- *status* — The status field can have one of the following values: MRD_STATUS_SLOT_INVALID, MRD_STATUS_DEVICE_INVALID, MRD_STATUS_TRANSPORT_INVALID, MRD_STATUS_PORT_INVALID, or MRD_STATUS_SUCCESS.
- *flags* — Use the ELEMENT_VALID mask on the *flags* field to indicate whether or not the full Read Element Status data is valid. The ELEMENT_ PVOLTAG and ELEMENT_AVOLTAG indicate whether the primary or alternate volume tags of the Read Element Status data are valid.
- *element_addr* — This is the address of the element, unadjusted for the starting address. The routine **mrd_map_element(3mrd)** can be used to convert an absolute element address to a relative address and type. This field will be set to -1 when the information is not valid.
- *source_addr* — On most SCSI-2 medium-changers, this is the address where a cartridge resided before being moved to its current location. The routine **mrd_map_element(3mrd)** can be used to convert an absolute element address to a relative address and type. This field will be set to -1 when the information is not valid. On some SCSI-2 medium-changers (the DLT family loaders) this will be the element address of the slot itself.
- *data* — This a copy of the SCSI-2 Read Element Status data when the ELEMENT_VALID bit is set in the *flags* field. A byte-order neutral declaration of this data structure is included in the **<mrd_common.h>** include file as the **mrd_reades_t** data structure.

21.3.2. Example

```
/*
 *   Example to do mrd_show(3mrd) on the first element of each type.
 *   The usage of this command is:
 *
 *   mrd_show robot
 *
```

```
 *   This examples show keeping the robot open across multiple
 *   calls to mrd_show(3mrd). In one happens to close it, the
 *   channel will be reset the BAD_CHANNEL and the next one will
 *   open it again. On some robot subsystems, opening the robot
 *   is fairly time consuming and if multiple "shows" are needed
 *   the time savings can be signficiant.
 *
 *   The subsystems where this is most noticable are HSJ and HSD
 *   connected robots, which aren't supported on Tru64 UNIX.
 */
#ifndef   lint
static   char   SccsId[] = "@(#)mrd_show.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{

   robot_info_t   robot_info ;  /* keep the robot open */
   element_info_t   element ;  /* place to put element data */
   int     el ;  /* type index */
   int     status ;  /* status from mrd_show(3mrd) */
   char     *robot ;  /* Robot to use */
   char     *content ;  /* pointer to a content string */
   char     log_info[MRD_MAX_LOG_STRING+1] ;  /* error text */

   /*
    * Only one argument is used; the robot name.
    */
   if( argc == 1 ) {
      printf("usage: %s robot\n", argv[0]) ;
      exit(1) ;
   }
   else
      robot = argv[1] ;

/*
 *   The channel number must be set to BAD_CHANNEL before
 *   mrd_startup or mrd_show is called, otherwise it will
 *   assume the robot is already open and not try to open
 *   it again.
 */
robot_info.channel = BAD_CHANNEL ;

/*
 *   In this case we want to open the robot once, and then
 *   call mrd_show(3mrd) in turn for each type of element.
 *   If there is an error and it happens to close the robot,
 *   the channel will be reset and the robot opened again on
 *   the next call.
 */
status = mrd_startup(robot, &robot_info, log_info) ;

if( status != MRD_STATUS_SUCCESS ) {
   printf("Startup failed: %s (%s).\n", mrd_strstatus(status),
      log_info[0] ? log_info : "none") ;

   exit(1) ;
}

/*
 * We rely on the fact that the element numbers are
```

```
 * are 1 through 4.
 */
for(el = 1; el <= 4; el++) {
   log_info[0] = '\0' ;

   status = mrd_show(robot, &robot_info, el, "0", 1,
         &element, log_info) ;

   if( status != MRD_STATUS_SUCCESS ) {
      printf("Can't show %s 0: %s (%s)\n",
         mrd_strelement(el), mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;

      continue ;
   }

   if( element.status != MRD_STATUS_SUCCESS ) {
      printf("Can't show %s 0: %s\n", mrd_strelement(el),
         mrd_strstatus(element.status)) ;

      continue ;
   }

   if( element.name[0] )
      content = element.name ;
   else if( element.state & ELEMENT_FULL )
      content = "Full" ;
   else if( element.state & ELEMENT_EXCEPT )
      content = "Exception" ;
   else
      content = "Empty" ;

   printf("%-9s 0: %s\n", mrd_strelement(el), content) ;
   }

   /*
    * Close it when done.
    */
   (void)mrd_shutdown(&robot_info) ;

   return 0 ;
}
```

21.3.3. Return Values

Upon successful completion, the **mrd_show(3mrd)** function returns the value MRD_STATUS_SUCCESS. If **mrd_show(3mrd)** fails, the returned status value will be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

21.3.3.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name*, *log_ info*, *element_name*, *element_info*, or *robot_info* arguments are NULL pointers.

2. **MRD_STATUS_RES_FAILED**

   The SCSI command Read Element Status failed.

3. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

This is error is returned when the *element_type* is not one of SLOT, PORT, DRIVE or TRANSPORT. This last case should return MRD_STATUS_INVALID_TYPE instead. It is also used for a SCSI command failure, when the ASC is set to one of:

- 0x1A - Parameter list length error
- 0x20 - Invalid command operation code
- 0x22 - Unsupported command
- 0x24 - Illegal field in CDB
- 0x25 - Logical unit not supported
- 0x26 - Threshold parameters not supported
- 0x28 - Import or Export element accessed
- 0x2C - Command sequence error
- 0x39 - Saving parameters not supported
- 0x3D - Invalid bits in Identify message
- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

4. **MRD_STATUS_SLOT_INVALID**

   This error is returned when the element address for a slot is less than zero or greater than the number of slots.

5. **MRD_STATUS_TRANSPORT_INVALID**

   This error is returned when the element address for a transport is less than zero or greater than the number of transports.

6. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives.

7. **MRD_STATUS_PORT_INVALID**

   This error is returned when the element address for a port is less than zero or greater than the number of ports.

8. **MRD_STATUS_NO_ELEMENTS**

   This error occurs in **mrd_ show(3mrd)**, **mrd_find_cartridge(3mrd)** and **mrd_home(3mrd)** when the medium-changer has no elements within the range and type specified by the arguments.

21.3.3.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under Microsoft Windows 2000/Windows XP, this error code is returned when the specified robot exists but is not responding.

3. **MRD_STATUS_INSFMEM**

   The **mrd_show(3mrd)** and **mrd_find_cartridge(3mrd)** functions allocate virtual memory using malloc(3) to store temporary element data. If the attempt to allocate the memory fails, these routines will return this error.

21.3.3.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.

- 0x43 - Message error
- 0x45 - Select or Reselect failure
- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x80 - When the ASCQ is not zero (0).
   - 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
   - 0x04 - Logical unit not ready
   - 0x3E - Logical unit has not been self configured
   - 0x40 - Diagnostic failure; ASCQ indicates component
   - 0x42 - Power-on self test failure
   - 0x44 - Internal target failure
   - 0x46 - Unsuccessful soft reset
   - 0x4C - Logical unit failed self-configuration

   This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

3. **MRD_STATUS_INSFMEM**

   The **mrd_show(3mrd)** and **mrd_find_cartridge(3mrd)** functions allocate virtual memory using malloc(3) to store temporary element data. If the attempt to allocate the memory fails, these routines will return this error.

21.3.3.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

- 0x08 - Logical unit communcation errors.
- 0x43 - Message error
- 0x45 - Select or Reselect failure
- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x80 - When the ASCQ is not zero (0).
   - 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
   - 0x04 - Logical unit not ready
   - 0x3E - Logical unit has not been self configured
   - 0x40 - Diagnostic failure; ASCQ indicates component
   - 0x42 - Power-on self test failure
   - 0x44 - Internal target failure
   - 0x46 - Unsuccessful soft reset
   - 0x4C - Logical unit failed self-configuration

   This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

3. **MRD_STATUS_DEVICE_INVALID**

This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS $_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

4. **MRD_STATUS_INSFMEM**

   The **mrd_show(3mrd)** and **mrd_find_cartridge(3mrd)** functions allocate virtual memory using malloc(3) to store temporary element data. If the attempt to allocate the memory fails, these routines will return this error.

21.4. Related Information

Functions:

- **mrd_shutdown**(3mrd)
- **mrd_startup**(3mrd)
- **mrd_map_element**(3mrd)

Chapter 22. mrd_startup

mrd_startup - Open a medium-changer robot

mrd_shutdown - Close a medium-changer robot

22.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_startup(
    const char *robot_name,
    robot_info_t *robot_info,
    char *log_info) ;

void mrd_shutdown(
    robot_info_t *robot_info) ;
```

22.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *robot_info* — This is the address of a *robot_info_t* structure initialized by **mrd_startup(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type. This is the address of the *robot_info_t* structure when **mrd_startup(3mrd)** is called.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

22.3. Description

The **mrd_startup(3mrd)** function can be used to obtain information about a supported Medium-Changer. Because the startup time on some robots (HSJ connected SCSI robots) can be relatively long, this routine can also be used to hold open the robot while **mrd_show(3mrd)** routines are used to collect information about the different robot elements.

The **mrd_shutdown(3mrd)** routine should be used to close a robot before other MRD routines are called. With the exception of **mrd_show(3mrd)** the MRD common routines call **mrd_startup(3mrd)** themselves and can't make use of **robot_info_t** filled in by **mrd_startup(3mrd)**.

Robot Information

The **robot_info_t** data structure is defined in the include file <mrd_common.h> The fields of this data structure are described below:

- *channel* — This is the file descriptor, channel number or other operating system specific handle assigned to the process for the robot, when **mrd_ startup(3mrd)** is successful. It should not be used directly and should only be closed through **mrd_shutdown(3mrd)**. When **mrd_show(3mrd)** is provided a **robot_info_t** where the channel is not BAD_CHANNEL, it will assume the robot is open and try to use that handle.

- *robot_name* — This is set to the address of the *robot_name* argument provided to **mrd_startup(3mrd)**.

- *robot_type* — MRD attempts to identify a robot using the SCSI inquiry data obtained during the startup. This is a value to indicate the family or type of medium-changer. Recognized types are:

DLT_ROBOT	TZ857, TZ867, TZ875, TZ877, TZ885, TZ887
RDAT_ROBOT	TLZ6L, TLZ7L, TLZ9L
TL820_ROBOT	TL820, TL822, TL826, TL893, TL896
TL810_ROBOT	TL810, TL812, TL894, TL895
TL800_ROBOT	TL891, TL892
OVERLAND_ROBOT	TKZ6x
RW5XX_ROBOT	RW500
UNKNOWN_ROBOT	A type not recognized

- *arch_type* — This indicates the I/O architecture used to communicate with the medium-changer. OpenVMS supports SCSI (ARCH_SCSI) and DSA (ARCH_DSA) connected medium-changers. HSJ and HSD connected robots are considered SCSI robots. Tru64 UNIX and Windows 2000 / Windows XP only support SCSI connected robots.

- *vision_present* — This flag indicates that the robot supports a vision system that can be used to read volume tags. It is set in the start-up and should not be changed. Most SCSI robots will reject commands asking for volume tags when the medium-changer doesn't support them.

- *robot_device_type* — This field is only used by the OpenVMS implementation of MRD to indicate the method the host uses to communicate with the medium-changer. This value is used internally by the OpenVMS MRD code to select the appropriate communcation path.

- *ptm_addr* and *ptm_type* — The TL820 family supports a pass-through mechanism (PTM) for moving cartridges from the inport to the bar-code reader station or from there to the outport. In multi-tower configurations the pass-through moves cartridges from one tower to another. The MRD uses the PTM to perform bar-code verification. Early versions of the TL820 family firmware present the PTM as a Import/Export element, while later versions may present it as a Transport. These fields are used to indicate the address and type.

- *maxecnt* — The OpenVMS drivers used to communicate with medium- changers support a limited I/O size that restricts the amount of data that can be transferred by a Read Element Status command. This field is used by the OpenVMS implementation of MRD to know where large transfers must be broken up by the software.

- *element_desc* — In addition to knowing the maximum number of elements that may be read in a single Read Element Status command, the element descriptor size is also needed to correctly break-up command. This field stores the element descriptor size on all implementations of the MRD.

The following fields are filled in from Element Address Assignment Page obtained via the SCSI Mode Sense command. When the robot is not a SCSI connect device, a suitable lie is filled by the operating system specific code supporting that type of robot.

- *slot_count* — This is the number of storage elements (slots) in the medium- changer. Some robots (TLZ7L) will change the number of slots presented depending on the type of magazine used. To detect changes in the size of the carrier, **mrd_startup(3mrd)** must be called and field checked for a change of value. When no magazine is in the drive, it may report 0 slots.

- *slot_start* — This is the element address of the first storage element. It is used by MRD to convert zero-relative element addresses to the actual element address used by the medium-changer.

- *device_count* — This is the number of data transfer elements (drives) in the medium-changer. Like storage elements it may be subject to change after a robot has been started.

- *device_start* — This is the element address of the first data transfer element. It is used by MRD to convert zero-relative element addresses to the actual element address used by the medium-changer.

- *port_count* — This is the number of import/export elements (ports) in the medium-changer. Like storage elements it may be subject to change after a robot has been started.

- *port_start* — This is the element address of the first import/export element. It is used by MRD to convert zero-relative element addresses to the actual element address used by the medium-changer.

- *transport_count* — This is the number of medium transport elements (transports) in the medium-changer. Like storage elements it may be subject to change after a robot has been started.

- *transport_start* — This is the element address of the first medium transport element. it is used by MRD to convert zero-relative element addresses to the actual element address used by the medium-changer.

- *inport_count* and *outport_count* — MRD V1.2 and earlier attempts to identify ports according to whether they are used for import-only, export-only or both. This is an artifact from the time that the TL820 was the only supported medium- changer with ports. Even though the TL810 has four ports these fields will report it having four inports and four outports.

- *inport_start* and *outport_start* — MRD V1.2 and earlier assumes that the arrangement of ports in the address space of the medium-changer has all the inports together and all the outports together. However, the two groups may be separated. No guarantee is made whether the addressees of the inports come before or after the outports. When the starting address of both types of ports the same value, it can be safely assumed that all the ports within the particular port count are both inport and outport elements.

The following fields are obtained as the result of a SCSI Inquiry Command. When the robot isn't a SCSI connected device, a suitable lie is filled in by the operating system specific code supporting that type of robot.

- *scsi_info* — These are the first eight (8) bytes of the SCSI Inquiry data for the robot. MRD doesn't make use of this information, but it is available if the calling application wants to use it. These bytes will be zero on non-SCSI devices.

- *inquiry* — This is the VendorID, ProductID and ProductRevisionLevel fields of the SCSI Inquiry data. They are collected as a single NUL terminated string. The data is edited to replace any non-printable character with a space.

These fields are not currently used:

- *bus*
- *target*
- *lun*
- *devcap*
- *transport_geometry*

22.3.1. Example

```
/*
 *   Example of using mrd_startup(3mrd) and mrd_shutdown(3mrd). This
 *   just opens the robot and prints the element counts and Inquiry
 *   string. The command usage is:
 *
 *      mrd_startup robot
 */

#ifndef   lint
static   char SccsId[] = "@(#)mrd_startup.c 1.2 3/5/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
   robot_info_t  robot_info ;  /* Place to put robot data */
   int  status ;  /* status from mrd_startup(3mrd) */
   char  *robot ;  /* robot name */
   char  log_info[MRD_MAX_LOG_STRING+1] ;  /* error text */
   /*
    *   Only one required argument; the robot name
    */
   if( argc == 1 ) {
      printf("usage: %s robot\n", argv[0]) ;
      exit(1) ;
   }
   else
      robot = argv[1] ;

   /*
   *   The channel number must be set to BAD_CHANNEL before
   *   mrd_startup is called, otherwise it will assume the
   *   robot is already open and not try to open it again.
   */
   robot_info.channel = BAD_CHANNEL ;

   status = mrd_startup(robot, &robot_info, log_info) ;

   if( status != MRD_STATUS_SUCCESS )
      printf("Startup failed: %s: %s.\n", mrd_strstatus(status),
         log_info[0] ? log_info : "none") ;
   else {
```

```
      printf("Inquiry: %s\n", robot_info.inquiry) ;
      printf(" Transports: %d\n", robot_info.transport_count) ;
      printf(" Slots: %d\n", robot_info.slot_count) ;
      printf(" Ports: %d\n", robot_info.port_count) ;
      printf(" Drives: %d\n", robot_info.device_count) ;
   }

   (void)mrd_shutdown(&robot_info) ;

   return 0 ;
}
```

22.3.2. Return Values

Upon successful completion, the **mrd_startup(3mrd)** and **mrd_shutdown(3mrd)** functions return the value MRD_STATUS_SUCCESS. If the **mrd_startup(3mrd)** fails the returned status value will be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

22.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name*, *log_info*, or *robot_info* arguments are NULL pointers.

2. **MRD_STATUS_SCSI_CHECK**

   The SCSI Check Condition error should never occur. It indicates that it is safe to use a Request Sense command and that you are likely to get a different error.

3. **MRD_STATUS_SCSI_CONDMET**

   The SCSI Condition Met status indicates a SCSI command completed with the status "Condition Met".

4. **MRD_STATUS_SCSI_BUSY**

   The SCSI Device is Busy status code indicates a SCSI command completed with the status "Busy". Some TZ87x media changers are known to cause this condition.

5. **MRD_STATUS_SCSI_INTER**

   The SCSI Intermediate Command Completed status code indicates a SCSI command completed with the status "Intermediate".

6. **MRD_STATUS_SCSI_INTER_CONDMET**

   The SCSI Intermediate-Condition Met status code indicates a SCSI command completed.

7. **MRD_STATUS_SCSI_RESCON**

   The SCSI Reservation Conflict status code indicates a SCSI command completed with the status "Reservation Conflict".

8. **MRD_STATUS_SCSI_TERM**

   The SCSI Command Terminated status code indicates a SCSI command completed with the status "Terminated".

9. **MRD_STATUS_SCSI_QUEUE**

   The SCSI Queue Full status code indicates a SCSI command completed with the status "Queue Full".

10. **MRD_STATUS_SCSI_RESERVED**

    The SCSI Status Code Reserved return indicates a SCSI command completed with a status that wasn't listed in Chapter 7 of the SCSI-2 specification and is "Reserved".

11. **MRD_STATUS_INIT_REQUIRED**

    LUN not ready, Initializing command required.

    This is for the ASC/ASCQ code of 4/2. It occurs when commands are sent to a TL810 family library that has auto-inventory on power-up turned off.

12. **MRD_STATUS_DIAG_FAILED** Diagnostic failure, component in ASCQ.

    This is the entire class of error codes with the ASC value set to 0x40.

13. **22.3.2.1.13 MRD_STATUS_IDE**

    Initiator detected error message received. This error code occurs when the ASC/ASCQ code is 0x48/0.

14. **MRD_STATUS_OPERATOR**

    Operator request. This error code occurs when the ASC code is 0x5A and the ASCQ code is 0 or 1.

15. **MRD_STATUS_LOG_ERROR**

    Device log error. This error code occurs when the ASC code is 0x5B and the ASCQ code is 0, 1, 2 or 3.

16. **MRD_STATUS_ELOG_OVERFLOW**

    Error log overflow. This error code occurs when the ASC code is 0xA and the ASCQ code is 0.

17. **MRD_STATUS_SYNC_XFER_ERROR**

    Synchronous data transfer error. This error code occurs when the ASC code is 0x1B and the ASCQ code is 0.

22.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.

- 0x43 - Message error
- 0x45 - Select or Reselect failure
- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under Microsoft Windows 2000/Windows XP, this error code is returned when the specified robot exists but is not responding.

3. **MRD_STATUS_NO_SUCH_DEVICE**

   This error is returned when a regular file or robot was specified without the ":BnTnLn" string.

22.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted

- 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_NO_SUCH_DEVICE**

   This error occurs when a UNIX system call returns ENXIO, to indicate that the device corresponding to the special device does not exist.

3. **MRD_STATUS_ROBOT_NOT_READY**

   Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x80 - When the ASCQ is not zero (0).
   - 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
   - 0x04 - Logical unit not ready
   - 0x3E - Logical unit has not been self configured
   - 0x40 - Diagnostic failure; ASCQ indicates component
   - 0x42 - Power-on self test failure
   - 0x44 - Internal target failure
   - 0x46 - Unsuccessful soft reset
   - 0x4C - Logical unit failed self-configuration

   This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

4. **MRD_STATUS_EBUSY**

   This error occurs when a UNIX system call returns EBUSY, to indicate that some other process is using that medium-changer device.

5. **MRD_STATUS_EINTR**

   This error occurs when a UNIX system call returns EINTR. This error corresponds to an interrupted system call, but also occurs when the SCSI CAM Layered Components Medium-Changer driver is not configured into the running system.

6. **MRD_STATUS_EIO**

   This error occurs when a UNIX system call returns EIO to indicate that there was an I/O error. In most cases an I/O error on a SCSI medium-changer indicates a SCSI error which be translated to another MRD error.

7. **MRD_STATUS_ENOENT**

   This error occurs when a UNIX system call returns ENOENT to indicate that a special device file doesn't exist.

8. **MRD_STATUS_EACCES**

This error occurs when a UNIX system call returns EACCES to indicate that the caller does not have sufficient permission to open the special device file corresponding to the medium-changer. MRD expects to have read permission on the special device file.

9. **MRD_STATUS_OS_ERROR**

   This error occurs when a UNIX system call returns an error that is not among those previously mentioned. The routine strerror(3) will be used to translate the error code into a standard text message which will be copied to *log_info*.

22.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

   This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x80 - When the ASCQ is not zero (0).
   - 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
   - 0x04 - Logical unit not ready

- 0x3E - Logical unit has not been self configured
- 0x40 - Diagnostic failure; ASCQ indicates component
- 0x42 - Power-on self test failure
- 0x44 - Internal target failure
- 0x46 - Unsuccessful soft reset
- 0x4C - Logical unit failed self-configuration

This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

3. **MRD_STATUS_NO_SUCH_DEVICE**

   This error is returned when a robot device name was specified for a robot that does not exist.

4. **MRD_STATUS_PAGE_CODE**

   This error occurs in mrd_startup(3mrd) when a SCSI Mode Sense command fails to return the expected data. It uses the SCSI Element Address Assignment mode page to fill in the element count and base address fields of the robot_info_t structure. If the data returned by the medium changer does not have the expected page code, this error is returned.

   This error has been seen when medium changers are connected to HS family array controllers running V2.7 firmware.

22.4. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_load**(3mrd)
- **mrd_unload**(3mrd)
- **mrd_inject**(3mrd)
- **mrd_eject**(3mrd)
- **mrd_show**(3mrd)
- **mrd_ready**(3mrd)
- **mrd_position**(3mrd)
- **mrd_initialize**(3mrd)
- **mrd_home**(3mrd)
- **mrd_find_cartridge**(3mrd)
- **mrd_error_decode**(3mrd)

- **mrd_strstatus**(3mrd)
- **mrd_map_element**(3mrd)
- **mrd_lock**(3mrd)
- **mrd_unlock**(3mrd)

Chapter 23. mrd_test_unit_ready

mrd_test_unit_ready - Verify a medium changer is ready to accept commands

23.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_test_unit_ready(
    robot_info_t *robot_info,
    dev_status_t *dev_status);
```

23.2. Parameters

- *robot_info* — This is the address of a *robot_info_t* structure initialized using **mrd_startup(3mrd)** or **mrd_show(3mrd)**. This data structure contains the element starting address and counts for each type of element, which are needed to map an absolute element to the correct zero relative address and type.

- *dev_status* — The *dev_status* is the address of a dev_status_t structure, which is used to pass back detailed error information in the event of a command failure.

23.3. Description

This routine performs a SCSI Test Unit Ready command, or equivalent if some other I/O architecture is supported. It is used by the **mrd_startup(3mrd)** and the OpenVMS implementation of **mrd_ready(3mrd)**. Since it accepts a *robot_ info_t* structure associated with an open medium changer it can used to perform Test Unit Ready command without having to re-open the medium changer each time.

The *robot_info_t* is the address of a *robot_info_t* that has been opened by **mrd_ startup(3mrd)**. If the medium changer isn't opened, the Test Unit Ready Command will fail with the operating system error for trying to use an unopened device.

The **dev_status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

23.3.1. Example

```
/*
 *   This is an example of using mrd_test_unit_ready(3mrd)
 *   to see if a media changer will accept commands. On
 *   Tru64 UNIX this particular example will always
 *   succeed whether the robot is ready or not. See the
 *   Restrictions section of the manual page for more
 *   information.
 *
 *   Usage:
 *
 *      mrd_test_unit_ready robot [ more-robots... ]
 */
#ifndef   lint
static char SccsId[] = "@(#)mrd_test_unit_ready.c 1.1 4/16/97" ;
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <mrd_common.h>
#include <mrd_message.h>

/*
 * Message string.
 */
char *tur_failed_dev =
        "Test unit ready failed on %s: %s (%d,0x%x,0x%x).\n ";
char *tur_failed_os  = "Test unit ready failed on %s: %s (%d).\n" ;
char *tur_failed     = "Test unit ready failed on %s: %s.\n" ;

/*
 *   The MRD can report three types of errors:
 *
 *   o Device errors - When the "valid" field is set, at least one
 *     of the key, asc and ascq field should have values set from
 *     a SCSI Request Sense data or equivalent.
 *
 *   o Operating System errors - When the valid field is zero, but
 *     the os_status field is set. Where possible an MRD error
 *     will be set if one corresponds to the error. If not, the
 *     MRD status will be MRD_STATUS_OS_ERROR. The os_status
 *     is the error specific to the operating system. On Tru64
 *     UNIX it is an errno value. On OpenVMS it is a system
 *     service return value.
 *
```

```
 *   o MRD Errors - The MRD error code is set explicitly.
 */
print_error(char *robot, int mrd_status, dev_status_t *dp)
{
   /*
    * Print the Key/ASC/ASCQ data for device errors.
    */
   if( dp->valid )
      printf(tur_failed_dev, robot, mrd_strstatus(mrd_status),
         dp->key, dp->asc, dp->ascq) ;
   /*
    * Try to decode the os_status according to the operating
    * system type.
    */
   else if( dp->os_status == MRD_STATUS_OS_ERROR )
      printf(tur_failed_os, robot, mrd_strstatus(mrd_status),
#ifdef   unix
         strerror(dp->os_status)) ;
#endif
#ifdef   vms
         strerror(EVMSERR, dp->os_status)) ;
#endif
   /*
    * Just print the message on others.
    */
   else
      printf(tur_failed, robot, mrd_strstatus(mrd_status)) ;
}

main(int argc, char *argv[])
{
   int   rc ;   /* counter */
   int   status ;   /* return status */
   char   *robot ;   /* Robot to open */
   robot_info_t   robot_info ;   /* Robot data */
   dev_status_t   dev_status ;   /* Device status */
   char   log_info[MRD_MAX_LOG_STRING+1] ;

   /*
    * Check that there are enough arguments.
    */
   if( argc < 2 ) {
      printf("usage: %s robot [ robot... ]\n", argv[0]) ;
      exit(1) ;
   }

   /*
    * Initialize the channel field of the robot_info, so
    * mrd_startup(3mrd) will actually open the robot.
    */
   robot_info.channel = BAD_CHANNEL ;

   for(rc = 1; rc < argc; rc++) {
      /*
       * The robot for this command.
       */
      robot = argv[rc] ;

      status = mrd_startup(robot, &robot_info, log_info) ;

      if( status != MRD_STATUS_SUCCESS ) {
         printf("Startup failed on %s: %s.\n", robot,
            mrd_strstatus(status)) ;

         continue ;
   }
```

```
      memset((void *)&dev_status, 0, sizeof(dev_status)) ;

      status = mrd_test_unit_ready(&robot_info, &dev_status) ;

      /*
       * Do some fancy error printing.
       */
      if( status != MRD_STATUS_SUCCESS )
         print_error(robot, status, &dev_status) ;
      else
         printf("%s is ready.\n", robot) ;
     (void)mrd_shutdown(&robot_info) ;
   }

   return 0 ;
}
```

23.3.2. Return Values

Upon successful completion, the **mrd_test_unit_ready(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_test_unit_ready(3mrd)**fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

23.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_info*, or *dev_status* are NULL pointers.

23.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error

- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

23.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted
   - 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

23.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_ROBOT_COMM_ERROR**

   This error code is used when an OpenVMS system service, such as $ASSIGN or $QIO, fails with a status of SS$_DRVERR. Generally SS$_DRVERR indicates a failure in the underlying device and the MRD can get the detailed device failure and return the correct MRD status code instead.

This error is also returned when a SCSI Test Unit Ready command fails. The cause of the error can be determined by called **mrd_request_sense(3mrd)**. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

- 0x08 - Logical unit communcation errors.
- 0x43 - Message error
- 0x45 - Select or Reselect failure
- 0x47 - SCSI parity error
- 0x48 - Initiator detected error message received
- 0x49 - Invalid message error
- 0x4A - Command phase error
- 0x4B - Data phase error
- 0x4E - Overlapped commands attempted
- 0x54 - SCSI to host system interface failure

2. **MRD_STATUS_IVCHAN**

   This error code is used when an OpenVMS system service fails with the status SS$_IVCHAN. It is likely when an operating system specific routine is used on a device that hasn't been opened by **mrd_startup(3mrd)**.

3. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS $_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

23.3.3. Related Information

Functions:

mrd_startup(3mrd)

23.3.4. Tru64 UNIX Restriction

On Tru64 UNIX the SCSI CAM Layered Components Medium Changer driver doesn't offer easy access to the Test Unit Ready command. As a result this routine always returns true when a valid robot_info_t structure is passed to it. However, the open(2) implementation of the Medium Changer driver always waits for a Test Unit Ready to succeed before the open(2) succeeds.

Chapter 24. mrd_unload

mrd_unload - Move a cartridge from drive to slot

24.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_unload(
    const char  *robot_name,
    const char  *volume_tag,
    const char  *drive,
    const char  *slot,
    char        *log_info)) ;
```

24.2. Parameters

- *robot_name* — The name of the robot device to be opened. On Tru64 UNIX, if the leading character of the name is not a slash (/), /dev/ will be prepended to the name.
- *volume_tag* — A NUL terminated character string that is the expected volume tag on the cartridge to be moved. On robots with vision support this string will be compared with the volume tag of the cartridge in the source slot and if it doesn't match the call will fail. This feature will not be used if the *volume_tag* is NULL or the empty string.
- *drive* — A NUL terminated character string that is the zero relative address of the drive which is to be used as the destination of the move.
- *slot* — A NUL terminated character string that is the zero relative address of the slot which is to be used as the source of the move.
- *log_info* — This is a character array that should be at least MRD_MAX_LOG_ STRING in length. If this function fails as the result of a SCSI error, this will be filled with the formatted request sense data. If this function fails as the result of an operating system error, the operating system message particular to the error will be copied into the array.

24.3. Description

The **mrd_unload(3mrd)** function is a specialized interface to the SCSI Move Medium command (or DSA equivalent). For the robot specified by *robot_name,* the routine will attempt to move the cartridge in the specified *drive* to the specified *slot*. Element addresses are zero based.

The robot will be opened and the arguments to the function will be verified to make sure they are safe and appropriate. The *drive* and *slot* address will be verified they are within the valid range of those elements on the robot.

The *volume_tag* argument can be used to perform cartridge volume tag verification before the move. If the cartridge volume tag at the port doesn't match that specified by this argument, then **mrd_unload(3mrd)** will fail with the status MRD_STATUS_CART_INVALID. If *volume_tag* argument is a NULL pointer, an empty string or used on a robot without vision support this argument is silently ignored and the volume tag check will not be made.

The DLT libraries (TL82X and TL81X families) require the host issue a SCSI Unload command before a cartridge may be removed from the drive. The function **mrd_unload(3mrd),** does not offer this feature. Thus, the calling program must do this itself. The example below shows how this can be done on Tru64 UNIX.

24.3.1. Tru64 UNIX

This example applies to the Tru64 UNIX.

```
/*
 *   Example of mrd_unload(3mrd). The command usage is:
 *
 *      mrd_unload robot_name drive slot [ volume_tag tape ]
 *
 *   On libraries (TL81X and TL82X) that require the tape
 *   drive be explicitly taken offline before the tape can
 *   be unloaded, this command allows a tape drive name to
 *   be specified as an optional argument.
 *
 *   Note: While the examples are largely system independent,
 *   this one makes use of operating system features that are
 *   very system dependent.
 */
#ifndef   lint
static   char SccsId[] = "@(#)mrd_unload.c 1.2 3/5/97" ;
#endif

#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mtio.h>
#include <sys/file.h>

#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>

#include <mrd_common.h>
#include <mrd_message.h>

/*
 * This happens not be declared anywhere on the system where
 * these examples were developed.
 */
int ioctl(int, unsigned long, void *) ;

/*
 * Take the specified tape offline. This goes roughly:
 *
 * o Open the drive.
 * o Build the mtop structure for an offline.
 * o Use I/O control to send it.
 * o Close the tape.
 */
offline(char *tape)
{
```

```
   struct mtop op ;
   int   fd ;

   printf("Take %s offline..."); fflush(stdout) ;

   /*
    *   We have two choices when opening the tape.
    *
    *   1. Wait until the tape is ready, or the tape driver
    *      gives up and decides it never complete the open.
    *      Some drivers may never give up and this will
    *      block until interruped.
    *
    *   2. Open with the O_NDELAY flag, which will work
    *      tape is ready or not. But the follow-on MTOFFL
    *      will fail if the tape isn't ready.
    */
   if((fd = open(tape, O_RDONLY)) == -1 ) {
      printf("\nCan't open %s: %s.\n", tape, strerror(errno)) ;
      return ;
   }

   /*
    * Build the necessary mtop structure.
    */
   op.mt_op = MTOFFL ;
   op.mt_count = 1 ;

   /*
    * Let the driver handle the details.
    */
   if( ioctl(fd, MTIOCTOP, &op) == -1 )
      printf("\nCan't take %s offline: %s.\n", tape,
         strerror(errno)) ;

   putchar('\n') ;

   /*
    * Close the file when done.
    */
   if( close(fd) == -1 )
   printf("Can't close %s: %s.\n", tape, strerror(errno)) ;
}

main(int argc, char *argv[])
{
   int   status ;   /* Status from mrd_load(3mrd) */
   char   *robot ;   /* The name of the robot */
   char   *tape = NULL ;   /* Tape drive name */
   char   *cart = NULL ;   /* Optional volume tag to check */
   char   *slot ;   /* Source slot */
   char   *drive ;   /* Destination drive */
   char   log_info[MRD_MAX_LOG_STRING+1] ;   /* error string */
   /*
    * Accept three required argument; robot, port and slot. The
    * volume tag and tape drive name are optional.
    */
   if( argc < 4 ) {
      printf("usage: %s robot slot drive [ tape volume-tag ]\n",
         argv[0]) ;
      exit(1) ;
   }

   /*
    *   Just use these directly from the command line.
    */
```

```
robot = argv[1] ;
slot = argv[2] ;
drive = argv[3] ;

/*
 * Of the optional arguments are present, try to figure
 * them out. Two optional arguments are supported and
 * we can't use position as a clue. We could use getopt(3),
 * that is a little too much work for a simple example.
 *
 * The optional arguments are a tape drive if the drive
 * has to explicitly be taken offline before it can be
 * unloaded and a volume tag to verify this is the expected
 * cartridge. The tape drive name can be expected to have
 * the prefix of "/dev/", so we'll use that as the clue.
 */
if( argc > 4 ) {
   /*
    *   Check the first one. If the first five characters
    *   are /dev/, assume this is the tape name.
    */
   if( strncmp(argv[4], "/dev/", 5) == 0 )
      tape = argv[4] ;
   else
      cart = argv[4] ;

   /*
    * If there is another argument beyond that, repeat.
    */
   if( argc > 5 ) {
      if( strncmp(argv[5], "/dev/", 5) == 0 )
         tape = argv[5] ;
      else
         cart = argv[5] ;
   }
}

/*
 * Having initialize the tape name to NULL when declared,
 * if it has changed we can assume that we need to take
 * this drive offline first. If it fails, just run through
 * and try the unload anyway.
 */
if( tape )
   offline(tape) ;

/*
 *   Call the function.
 */
status = mrd_unload(robot, cart, slot, drive, log_info) ;
/*
 *   Print an error message if there is a failure. The
 *   routine mrd_strstatus(3mrd) will accept an MRD
 *   error status and return the corresponding string.
 *   If the log_info data has something other than a
 *   NULL as the first character print it as well. It
 *   typically be the SCSI sense data or a operating
 *   system specific message for the error.
 */
if( status != MRD_STATUS_SUCCESS )
   printf("Load failed: %s: %s.\n", mrd_strstatus(status),
      log_info[0] ? log_info : "none") ;
else
   printf("Unloaded media from Drive #%s to Slot #%s.\n",
      drive, slot) ;
```

```
   return 0 ;
}
```

24.4. OpenVMS Example

This example applies to the OpenVMS operating system.

```
/*
 * Example of mrd_unload(3mrd). The command usage is:
 *
 * mrd_unload robot_name drive slot [ volume_tag ]
 *
 * This version is VMS specific since it excludes the example
 * for taking a tape drive offline. That is moderately complicated
 * but something that should be in the VMS documentation.
 */
#ifndef lint
static char SccsId[] = "@(#)mrd_unload.c 1.2A (mrd-example) 3/5/97" ;
#endif

#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

#include <mrd_common.h>
#include <mrd_message.h>

main(int argc, char *argv[])
{
 int status ; /* Status from mrd_load(3mrd) */
 char *robot ; /* The name of the robot */
 char *cart = NULL ; /* Optional volume tag to check */
 char *slot ; /* Source slot */
 char *drive ; /* Destination drive */
 char log_info[MRD_MAX_LOG_STRING+1] ; /* error string */

 /*
  * Accept three required argument; robot, port and slot. The
  * volume tag and tape drive name are optional.
  */
 if( argc < 4 ) {
  printf("usage: %s robot slot drive [ volume-tag ]\n",
   argv[0]) ;
  exit(1) ;
 }
 /*
  * Just use these directly from the command line.
  */
 robot = argv[1] ;
 slot  = argv[2] ;
 drive = argv[3] ;

 /*
  * If there is an extra argument present, assume it is
  * a cartridge name. The habit of the DECC runtime
  * start-up mapping all characters to lower case,
  * might require special handling of the cartridge
  * name.
  */
 if( argc > 4 )
  cart = argv[4] ;

 /*
```

```
  * Call the function.
  */
 status = mrd_unload(robot, cart, slot, drive, log_info) ;

 /*
  * Print an error message if there is a failure. The
  * routine mrd_strstatus(3mrd) will accept an MRD
  * error status and return the corresponding string.
  * If the log_info data has something other than a
  * NUL as the first character print it as well. It
  * typically be the SCSI sense data or a operating
  * system specific message for the error.
  */
 if( status != MRD_STATUS_SUCCESS )
  printf("Load failed: %s: %s.\n", mrd_strstatus(status),
   log_info[0] ? log_info : "none") ;
 else
  printf("Unloaded media from Drive #%s to Slot #%s.\n",
   drive, slot) ;

 return 0 ;
}
```

24.4.1. Return Values

Upon successful completion, the **mrd_unload(3mrd)** function returns the value MRD_STATUS_SUCCESS. If the **mrd_unload(3mrd)** fails the returned status value may be set to one of the following values. Other values that correspond to specific SCSI errors may also be possible, but these are the most likely.

24.4.1.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *robot_name*, *drive*, *slot*, or *log_info* arguments are NULL pointers.

2. MRD_STATUS_ROBOT_COMM_ERROR

   This error occurs as the result of a failure to open the specified medium-changer. This may occur directly by calling **mrd_startup(3mrd)** or by a routine that calls **mrd_startup(3mrd)** internally. This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x08 - Logical unit communcation errors.
   - 0x43 - Message error
   - 0x45 - Select or Reselect failure
   - 0x47 - SCSI parity error
   - 0x48 - Initiator detected error message received
   - 0x49 - Invalid message error
   - 0x4A - Command phase error
   - 0x4B - Data phase error
   - 0x4E - Overlapped commands attempted

- 0x54 - SCSI to host system interface failure

3. **MRD_STATUS_SLOT_INVALID**

   This error is returned when theelement address for a slot is less than zero or greater than the number of slots.

4. **MRD_STATUS_CART_INVALID**

   For routines that accept a *volume_tag* argument to perform volume tag verification, this error indicates that the volume tag of the media doesn't match that passed to the function.

5. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number ofdrives.

6. **MRD_STATUS_SOURCE_EMPTY**

   On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

7. **MRD_STATUS_DESTINATION_FULL**

   On routines that perform aSCSI Move Medium command, this error indicates that the destination elementalready has a cartridge in it.

8. **MRD_STATUS_CART_NOT_AVAIL**

   This error can occur on theTL81n and TL82n family of DLT libraries when the source of a move is a driveand the cartridge in the drive is still on-line. These robots do not allow movingthe cartridge until the drive is taken offline.

24.4.1.2. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS $_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

24.4.2. Related Information

Functions:

- **mrd_move**(3mrd)
- **mrd_load**(3mrd)

- **mrd_inject**(3mrd)
- **mrd_eject**(3mrd)

Chapter 25. mrd_utility

mrd_scsi_decode, **mrd_map_os_error** - Various MRD utility functions

25.1. Media Robot Driver Library

The following table shows the names of the MRD library modules for each operating system.

/Windows 2000/Windows XP	mrd.dll
UNIX	/usr/lib/libmrd.a
OpenVMS	MRD$RTL.EXE

```
#include <mrd_common.h>
#include <mrd_message.h>

int mrd_scsi_decode(dev_status_t *dev_status);

int mrd_map_os_error(int os_status, char *log_info);
```

25.2. Parameters

- *dev_status* — The *dev_status* is the address of a dev_status_t structure. The fields in this structure are examined to map a SCSI error to an MRD_ STATUS code.
- *os_status* — The *os_status* is an operating system specific failure code that is used to find the matching MRD_STATUS code.
- *log_info* — The *log_info* is a character array that should be at least MRD_ MAX_LOG_STRING in length. On returning, it contains the operating system status.

25.3. Description

The routine **mrd_scsi_decode(3mrd)** is used by the low level MRD routines to map SCSI device errors to MRD_STATUS codes. It uses the Additional Sense Code (*asc*) and Additional Sense Code Qualifier (*ascq*) of the *status* structure to find an appropriate MRD_STATUS code. If both the *asc* and *ascq* are zero (0), the Sense Key (*key*) will be used to determine the code. The resulting MRD_STATUS code will be copied to the *code* field and returned.

The routine **mrd_map_os_error(3mrd)** is used to map operating system specific failures to MRD_STATUS codes. If the *os_status* isn't recognized, the routine will return MRD_STATUS_OS_ERROR. If the *log_info* argument is a valid pointer, a copy of the operating system text for the message will also copied to the space provided.

This routine uses the **dev_status_t** structure for handing errors. The **dev_ status_t** structure includes the *code*, *os_status*, and SCSI error fields. The following describes how to decode errors with the **dev_status_t** structure.

SCSI Errors

SCSI errors are indicated when the value of the *valid* field of the SCSI error is not equal to 0. The *key*, *asc*, and *ascq* fields provide additional information to help determine the cause of the error.

The *code* usually maps the Additional Sense Code and Additional Sense Code Qualifier (ASC/ASCQ) values to an MRD error. The *asc* and *ascq* values are copied from the request sense data returned by the target.

The Additional Sense Code (*asc*) indicates further information related to the error or exception condition reported in the sense key field. The Additional Sense Code Qualifier (*ascq*) indicates detailed information related to the additional sense code. For more information, consult the SCSI-2 Specification.

Operating System Errors

Operating system errors are indicated when the value of the *valid* field of the SCSI error is equal to 0 and the value of the *os_status* field is not equal to 0. This result is most likely caused by an operating system error, and probably has a mapped error in MRD.

MRD Errors

MRD errors are indicated when the value of the *os_status* field is 0, and the value of the *valid* field of the SCSI error is 0. This result is most likely caused when MRD encounters its own failure.

25.3.1. Example

```
/*
 *   This shows how the utility routines are used. For
 *   mrd_scsi_decode(3mrd), a selected SCSI-2 error will
 *   be filled into the key, asc and ascq fields of a
 *   dev_status_t structure and the resulting MRD status
 *   message printed. For mrd_map_os_error(3mrd) the
 *   will be done for a selected operating system error.
 *
 *   Usage:
 *
 *      mrd_utility
 */
#ifndef   lint
static   char   SccsId[] = "@(#)mrd_utility.c 1.2 3/5/97" ;
#endif

#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#include <mrd_common.h>
#include <mrd_message.h>

#ifdef   vms
#include   <ssdef.h>
#endif

main(int argc, char *argv[])
{
   dev_status_t   dev_status ;   /* Device status */
   int     status ;
   char     log_info[MRD_MAX_LOG_STRING+1] ;

   /*
    * Clear this for later.
    */
   log_info[0] = '\0' ;

   /*
    *  First, try mrd_scsi_decode(3mrd). SCSI-2 happens to
```

```
    *  have ASC/ASCQ codes for a cleaning cartridge being
    *  installed somewhere, presumably a drive. We'll
    *  see what MRD does with it.
    */
   dev_status.valid = SCSI_REQ_SENSE_VALID ;
   dev_status.key   = 1 ; /* Recovered Error */
   dev_status.asc   = 0x30 ;
   dev_status.ascq  = 3 ;

   status = mrd_scsi_decode(&dev_status) ;

   /*
    *  Now print the result. As it happens we map this
    *  code to MRD_STATUS_AUTOCLEAN, which is nearly
    *  right.
    */
   printf("Cleaning Cartridge Installed: (%d,%x,%x): %s\n",
      dev_status.key, dev_status.asc,
      dev_status.ascq, mrd_strstatus(status)) ;
   /*
    *   Now do one of completely random values. Seed the
    *   random number generator just so most get a different
    *   answer. Most of these are likely to end up as
    *   Vendor Unique errors.
    */
   srand(time(NULL)) ;

   dev_status.key  = rand() % 16 ; /* 0 - 15 */
   dev_status.asc  = rand() % 256 ; /* 0 - 255 */
   dev_status.ascq = rand() % 256 ; /* 0 - 255 */

   status = mrd_scsi_decode(&dev_status) ;

   /*
    *   Now print the result.
    */
   printf("Random SCSI Decode: (%d,%x,%x): %s\n",
      dev_status.key, dev_status.asc,
      dev_status.ascq, mrd_strstatus(status)) ;
   /*
    * Now an OS error. If #ifdef is handle the two example
    * operating systems.
    */
   dev_status.valid = 0 ;

#if defined(VMS)
   dev_status.os_status = SS$_UNASEFC ;
#elif defined(unix)
   dev_status.os_status = EINTR ;
#else
   dev_status.os_status = rand() % 100 ;
#endif

   status = mrd_map_os_error(dev_status.os_status, log_info) ;
   if( log_info[0] )
      printf("Map OS Error: %d: %s: %s\n", dev_status.os_status,
         mrd_strstatus(status), log_info) ;
   else
      printf("Map OS Error: %d: %s\n", dev_status.os_status,
         mrd_strstatus(status)) ;

   return 0 ;
}
```

25.3.2. Return Values

The status returned is always a valid MRD_STATUS code corresponding to the error given by the *status* or *os_status*. The errors and their mappings are:

25.3.2.1. Common Codes

1. **MRD_STATUS_PARAM**

   This error is returned if the *dev_status* argument is a NULL pointer.

2. **MRD_STATUS_NO_SENSE**

   This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc, ascq* and *key* values are all zero (0). It is also returned when the *key* value is less than zero or greater than 15.

3. **MRD_STATUS_RECOVERED_ERROR**

   This error occurs when a SCSI device returns only a sense key of 1h. This indicates that although a command successfully completed, the target device had performed some internal error recovery.

4. **MRD_STATUS_MEDIUM_ERROR**

   This error occurs when ASC and ASCQ are zero, but the sense key is 3h. This occurs when the target encounters a nonrecoverable error due to a flaw in the medium.

5. **MRD_STATUS_ROBOT_HW_ERROR**

   This error occurs when ASC and ASCQ are zero, but the sense key is 4h. This occurs when the target encounters a nonrecoverable hardware error.

6. **MRD_STATUS_ROBOT_ILLEGAL_REQUEST**

   This error occurs for a variety of reasons. It is used when a sanity check fails in the code that attempts to move a cartridge to the Pass-Through Mechanism, when the robot type isn't a TL82n.

   It is used in the **mrd_lock(3mrd)** code when the value is not one of ALLOW_REMOVAL or PREVENT_REMOVAL.

   It is used when the medium changer does not support the Prevent/Allow Medium Removal command or the lock value is not one or zero. The specific cause can be determined by examining the ASC/ASCQ values in the *status* data.

   It is used when a call to **mrd_initialize_element(3mrd)** is issued against a medium changer that does not support the Initialize Element Status command.

   It is used when the medium changer does not support the Position To Element command. The seven and five slot DLT loaders do not support the command, though the TL820 and TL810 family libraries do. Some models of TLZ6L and TLZ7L do not support the command and may take a long time to fail.

   It is used when the medium changer does not support the Ready Inport command.

   The TL820 family of DLT libraries support this command. The TL810 family of DLT libraries allows this command to succeed, but it doesn't perform any function.

It is also used for a SCSI command failure, when the ASC is set to one of:

- 0x1A - Parameter list length error
- 0x20 - Invalid command operation code
- 0x22 - Unsupported command
- 0x24 - Illegal field in CDB
- 0x25 - Logical unit not supported
- 0x26 - Threshold parameters not supported
- 0x28 - Import or Export element accessed
- 0x2C - Command sequence error
- 0x39 - Saving parameters not supported
- 0x3D - Invalid bits in Identify message
- 0x53 - Medium removal prevented

This status is also returned when the ASC and ASCQ are zero, but the key is five (5).

7. **MRD_STATUS_ROBOT_ATTENTION**

   This error occurs when a SCSI command fails with the ASC set to one of 0x29, 0x2A or 0x2F. The *log_info* contains the ASCQ. The SCSI translations for these error codes are:

   - 0x29 - Power-on, Reset or Bus device reset occurred
   - 0x2A - Mode Parameters Changed
   - 0x2F - Command cleared by another initiator

   This error also occurs when the ASC and ASCQ are zero, but the SCSI sense key is 6h.

8. **MRD_STATUS_DATA_PROTECT**

   This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is seven (7).

9. **MRD_STATUS_BLANK_CHECK**

   This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is eight (8).

10. **MRD_STATUS_VENDOR_UNIQUE_ERROR**

    This error occurs when the internal routine used to decode SCSI-2 errors encounters an error that it has not been written to antipicate.

    This error is also returned when the ASC is zero and the ASCQ is not one of zero or six, and when ASC/ASCQ are both zero and the *key* is 9h.

11. **MRD_STATUS_COPY_ABORTED**

    This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is ten (10).

12. **MRD_STATUS_SENSE_EQUAL**

    This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Ch (12).

13. **MRD_STATUS_VOLUME_OVERFLOW**

    This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Dh (13).

14. **MRD_STATUS_MISCOMPARE**

    This error is returned by **mrd_scsi_ decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Eh (14).

15. **MRD_STATUS_SENSE_RESERVED**

    This error is returned by **mrd_ scsi_decode(3mrd)** when the *asc* and *ascq* are zero, but the *key* value is Fh (15).

16. **MRD_STATUS_ROBOT_COMM_ERROR**

    This error also occurs as the result of a SCSI command failure, when the ASC is set to one of:

    - 0x08 - Logical unit communcation errors.
    - 0x43 - Message error
    - 0x45 - Select or Reselect failure
    - 0x47 - SCSI parity error
    - 0x48 - Initiator detected error message received
    - 0x49 - Invalid message error
    - 0x4A - Command phase error
    - 0x4B - Data phase error
    - 0x4E - Overlapped commands attempted
    - 0x54 - SCSI to host system interface failure

17. **MRD_STATUS_ROBOT_MECH_ERROR**

    This error occurs as the result of a SCSI command failure, when the ASC is set to one of:

    - 0x15 - Positioning error.
    - 0x8B - Vendor unique; Pass-through mechanism errors on the TL82n

18. **MRD_STATUS_AUTOCLEAN**

This error occurs when a SCSI command fails with the ASC set to 0x30 and the ASCQ set to 0x3. On TL8nn libraries supporting Auto-clean, it indicates that a command was attempted while an auto-clean was in progress.

19. **MRD_STATUS_CART_DAMAGED**

    This error occurs when a SCSI command fails with the ASC set to 0x30, but the ASCQ is NOT a value of 0x3. The *log_info* will contain the ASCQ.

20. **MRD_STATUS_ELEMENT_INVALID**

    This error occurs when a SCSI command fails with the ASC set to 0x21. The *log_info* will contain the ASCQ. This indicates that an invalid element address reached the medium-changer. For example, specifying the 13th slot when only 12 slots are present.

21. **MRD_STATUS_CART_NOT_AVAIL**

    This error can occur on the TL81n and TL82n family of DLT libraries when the source of a move is a drive and the cartridge in the drive is still on-line. These robots do not allow moving the cartridge until the drive is taken offline.

22. **MRD_STATUS_DESTINATION_FULL**

    On routines that perform a SCSI Move Medium command, this error indicates that the destination element already has a cartridge in it.

23. **MRD_STATUS_SOURCE_EMPTY**

    On routines that perform a SCSI Move Medium command, this error indicates that the source element is empty.

24. **MRD_STATUS_ROBOT_DOOR_OPENED**

    This occurs when a SCSI command fails with the ASC set to 0x80 and the ASCQ set to 0x0. On TL8nn libraries this typically indicates that the cabinet door was opened during a command operation.

25.3.2.2. Windows 2000/Windows XP Codes

The codes in this section apply only to the Windows 2000/Windows XP operating system variant of the Media Robot Driver.

1. **MRD_STATUS_IVCHAN**

   This error code is returned when the handle in NT parlance has been closed or is otherwise an invalid handle.

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under Microsoft Windows 2000/Windows XP, this error code is returned when the specified robot exists but is not responding.

3. **MRD_STATUS_ROBOT_CMD_ABORTED**

   This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* is zero and the *ascq* is six, or when the *asc* and *ascq* are zero and the *key* is eleven (11).

25.3.2.3. Tru64 UNIX Codes

The following error codes can occur when a **open(2)** or **ioctl(2)** system call fails. Open(2) is used by **mrd_startup(3mrd)** to open the medium-changer. The **ioctl(2)** system call is used to perform all other SCSI medium-changer commands.

1. **MRD_STATUS_EBADF**

   This error occurs when the medium changer has not been opened by **mrd_startup(3mrd)** or has been closed by **mrd_ shutdown(3mrd)**.

2. **MRD_STATUS_EINVAL**

   This error is returned by **mrd_map_os_ error(3mrd)** when the *os_status* is EINVAL. This typically occurs during **mrd_ startup(3mrd)** when the special file is not a SCSI device: for example, /dev/tty.

3. **MRD_STATUS_STARTUP_ERROR**

   This error is returned by **mrd_ map_os_error(3mrd)** when the *os_status* is ENODEV. This typically occurs during **mrd_startup(3mrd)** when the special file is not a SCSI device; /dev/null.

4. **MRD_STATUS_NO_SUCH_DEVICE**

   This error occurs when a UNIX system call returns ENXIO, to indicate that the device corresponding to the special device does not exist.

5. **MRD_STATUS_EBUSY**

   This error occurs when a UNIX system call returns EBUSY, to indicate that some other process is using that medium-changer device.

6. **MRD_STATUS_EINTR**

   This error occurs when a UNIX system call returns EINTR. This error corresponds to an interrupted system call, but also occurs when the SCSI CAM Layered Components Medium-Changer driver is not configured into the running system.

7. **MRD_STATUS_EIO**

   This error occurs when a UNIX system call returns EIO to indicate that there was an I/O error. In most cases an I/O error on a SCSI medium-changer indicates a SCSI error which be translated to another MRD error.

8. **MRD_STATUS_ENOENT**

   This error occurs when a UNIX system call returns ENOENT to indicate that a special device file doesn't exist.

9. **MRD_STATUS_EACCES**

   This error occurs when a UNIX system call returns EACCES to indicate that the caller does not have sufficient permission to open the special device file corresponding to the medium-changer. MRD expects to have read permission on the special device file.

10. **MRD_STATUS_OS_ERROR**

This error occurs when a UNIX system call returns an error that is not among those previously mentioned. The routine strerror(3) will be used to translate the error code into a standard text message which will be copied to *log_info*.

11. **MRD_STATUS_ROBOT_NOT_READY**

    Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

    - 0x80 - When the ASCQ is not zero (0).
    - 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
    - 0x04 - Logical unit not ready
    - 0x3E - Logical unit has not been self configured
    - 0x40 - Diagnostic failure; ASCQ indicates component
    - 0x42 - Power-on self test failure
    - 0x44 - Internal target failure
    - 0x46 - Unsuccessful soft reset
    - 0x4C - Logical unit failed self-configuration

    This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

12. **MRD_STATUS_ROBOT_CMD_ABORTED**

    This error is returned by **mrd_scsi_decode(3mrd)** when the *asc* is zero and the *ascq* is six, or when the *asc* and *ascq* are zero and the *key* is eleven (11).

25.3.2.4. OpenVMS Codes

The codes in this section apply only to the OpenVMS operating system variant of the Media Robot Driver.

1. **MRD_STATUS_DEVICE_INVALID**

   This error is returned when the element address for a drive is less than zero or greater than the number of drives. This error code is used when an OpenVMS system service fails with the status SS $_NOSUCHDEV or SS$_IVDEVNAM. This will typically occur in **mrd_ startup(3mrd)** when the caller tries to open a device which doesn't exist or uses an invalid device name.

   This error also occurs when the routine is called on behalf of a device controlled by the JU driver. The Media Robot Utility no longer uses the JU driver.

2. **MRD_STATUS_ROBOT_NOT_READY**

   Under OpenVMS and Tru64 UNIX, this error occurs as the result of a SCSI command failure, when the ASC is set to one of:

   - 0x80 - When the ASCQ is not zero (0).

- 0x81 - Vendor unique; gripper errors on the TL82X and TL81X
- 0x04 - Logical unit not ready
- 0x3E - Logical unit has not been self configured
- 0x40 - Diagnostic failure; ASCQ indicates component
- 0x42 - Power-on self test failure
- 0x44 - Internal target failure
- 0x46 - Unsuccessful soft reset
- 0x4C - Logical unit failed self-configuration

This status is also returned when the ASC and ASCQ are zero, but the key is two (2).

3. **MRD_STATUS_ROBOT_CMD_ABORTED**

   This error code is used when an OpenVMS system service fails with the status SS$_ABORT.

25.3.3. Related Information

Functions:

- **mrd_move_medium**(3mrd)
- **mrd_read_element_status**(3mrd)
- **mrd_startup**(3mrd)
- **mrd_position_to_element**(3mrd)
- **mrd_initialize_element**(3mrd)
- **mrd_ready**(3mrd)
- **mrd_prevent_allow**(3mrd)