

NDMP

Network Data Management Protocol

Network Working Group
Internet Draft
Category: Informational

R. Stager, PDC
D. Hitz, Network Appliance
October 1996

Network Data Management Protocol

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Abstract

The Network Data Management Protocol (NDMP) addresses the user's need for centralized control of enterprise-wide network data management while minimizing network traffic. The design objective of the protocol is to make every network attached storage device "backup ready", enabling true plug-and-play backup operation. The user will not be required to install any additional software on an NDMP-compliant network storage device. With the NDMP approach, each network-attached file server ships with a "universal agent," which can be used by any NDMP-compliant backup administration application. For IS departments and system administrators, NDMP will ensure interoperability between different file servers and backup solutions, significantly simplifying the data management process. This same universal agent architecture is also for network-attached backup devices, such as a tape libraries.

Filename: <draft-stager-pdc-netapp-backup-00.txt>
Expires: May 1997
Document Version: 1.7.3
Last Update: 10/24/96

1. OVERVIEW	4
1.1 MOTIVATION.....	4
1.2 AUDIENCE.....	4
1.3 TERMINOLOGY	4
2. ARCHITECTURE	5
2.1 ARCHITECTURAL MODEL	5
2.2 COMPARISON ARCHITECTURES	7
2.3 STATE DESCRIPTION.....	8
2.3.1 <i>Idle State</i>	9
2.3.2 <i>Active State</i>	9
2.3.3 <i>Paused State</i>	9
2.3.4 <i>Halted State</i>	9
2.4 PROTOCOL INTERFACES.....	9
2.4.1 <i>NDMP Server Interfaces</i>	9
2.4.2 <i>NDMP Client Interfaces</i>	11
2.5 MESSAGING PROTOCOL	11
2.6 HEADER.....	11
2.7 ERROR.....	13
2.8 MESSAGE DEFINITIONS	15
3. NDMP SERVER INTERFACES	16
3.1 CONNECT INTERFACE	17
3.1.1 <i>Open Connection</i>	17
3.1.2 <i>Authorization</i>	17
3.1.3 <i>Close Connection</i>	18
3.2 CONFIG INTERFACE.....	19
3.2.1 <i>Get Host Info</i>	19
3.2.2 <i>Get backup Type Attribute</i>	19
3.3 SCSI INTERFACE.....	21
3.3.1 <i>Open SCSI Device</i>	21
3.3.2 <i>Close Device</i>	21
3.3.3 <i>Get SCSI State</i>	22
3.3.4 <i>Set SCSI Target</i>	23
3.3.5 <i>Reset Device</i>	23
3.3.6 <i>Reset Bus</i>	24
3.3.7 <i>Execute CDB</i>	24
3.4 TAPE INTERFACE.....	26
3.4.1 <i>Open Tape Device</i>	26
3.4.2 <i>Close Device</i>	27
3.4.3 <i>Get Tape State</i>	28
3.4.4 <i>MTIO</i>	29
3.4.5 <i>Write</i>	31
3.4.6 <i>Read</i>	32
3.4.7 <i>Set Record Size</i>	32
3.4.8 <i>Execute CDB</i>	33
3.5 DATA INTERFACE.....	33
3.5.1 <i>Get Data State</i>	33
3.5.2 <i>Backup</i>	36
3.5.3 <i>Recover</i>	37
3.5.4 <i>Abort</i>	39
3.5.5 <i>Stop</i>	39
3.5.6 <i>Continue</i>	40
3.5.7 <i>Get ENV</i>	40
4. NDMP CLIENT INTERFACES	42

4.1 NOTIFY INTERFACE.....	42
4.1.1 <i>Notify Paused</i>	42
4.1.2 <i>Notify Halted</i>	42
4.1.3 <i>Notify Connect</i>	43
4.2 LOGGING INTERFACE.....	43
4.2.1 <i>Log</i>	44
4.2.2 <i>Debug</i>	44
4.3 FILE HISTORY INTERFACE.....	46
4.3.1 <i>Add Unix Path</i>	46
4.3.2 <i>Add Unix Dir</i>	47
4.3.3 <i>Add Unix Node</i>	48
5. REFERENCES	49
6. SECURITY.....	49
7. AUTHORS	49
FIGURE 1. SIMPLE CONFIGURATION.....	5
FIGURE 2. TWO DRIVE CONFIGURATION.....	6
FIGURE 3. JUKEBOX CONFIGURATION	6
FIGURE 4 - BACKUP STATE DIAGRAM.....	8

1. Overview

1.1 Motivation

The purpose of this protocol is to allow a network backup application to control the backup of an NDMP compliant server using a universal agent without installing special software on the server.

This separation of control and data allows complete interoperability at a network level. The file system vendors need only be concerned with maintaining compatibility with one, well-defined protocol. The backup vendors can place their primary focus on the sophisticated central backup administration software.

This protocol is specifically intended to support tape drives. This protocol is targeted towards backup software and there are extensive references to the tasks of backup and restore. However, the protocol may be used for other applications in the future.

1.2 Audience

This document is intended for use by software developers to implement Network Data Management Protocol. The reader is assumed to be familiar with network protocol specifications and with the general operation of backup software. The user is not expected to have knowledge of internal backup software behavior.

1.3 Terminology

NDMP client

The application that controls the NDMP server.

NDMP host

The host which has a tape drive physically attached and can perform local backups to that tape drive using the NDMP.

NDMP server

The virtual state machine on the NDMP host that is controlled using the NDMP protocol. There is one of these for each connection to the NDMP host. This term is used independent of implementation.

2. Architecture

2.1 Architectural Model

The architecture is a client server model and backup software is considered a client to the NDMP server. For every connection between the client and the NDMP host there is a virtual state machine on the NDMP host that is controlled using the NDMP. This virtual state machine is referred to as the NDMP server. Each state machine controls at most one device used to perform backups. The protocol is a set of XDR encoded messages that are exchanged over a bi-directional TCP/IP connection and are used to control and monitor the state of the NDMP server and to collect detail information about the data that is backed up.

In the most simple configuration, NDMP client will backup the data from the NDMP host to a tape drive connected to the NDMP host.

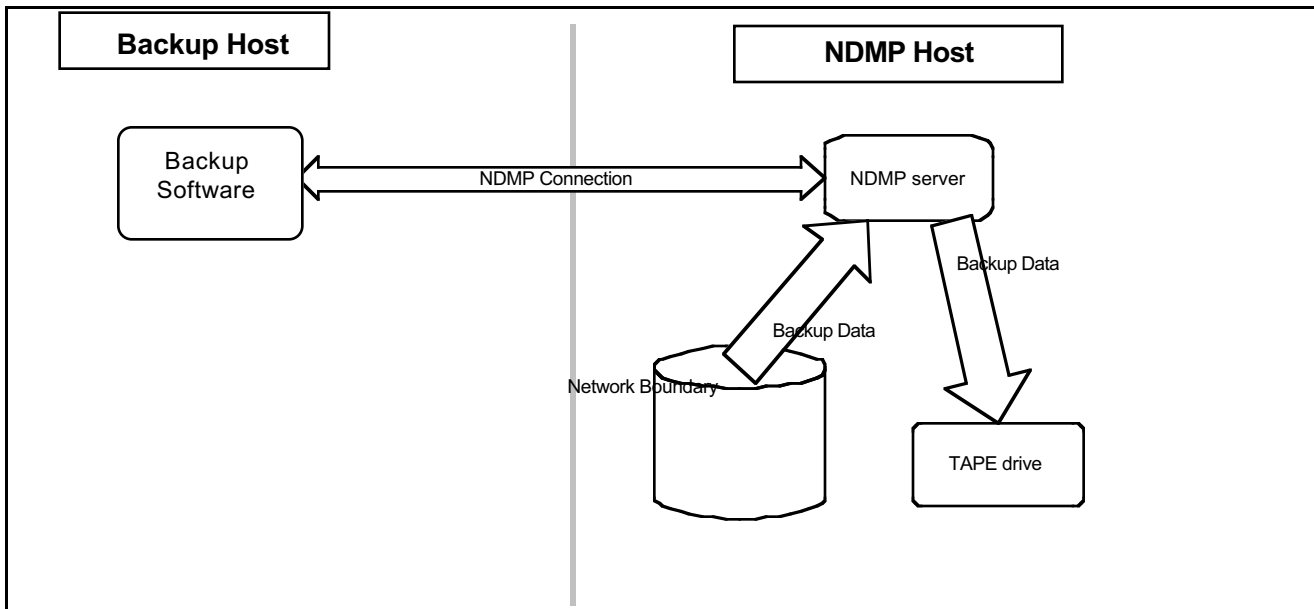


Figure 1. Simple configuration

It is also possible to use the NDMP to simultaneously backup to two tape drives physically attached to the NDMP host. In this configuration there are two instances of the NDMP server on the NDMP host.

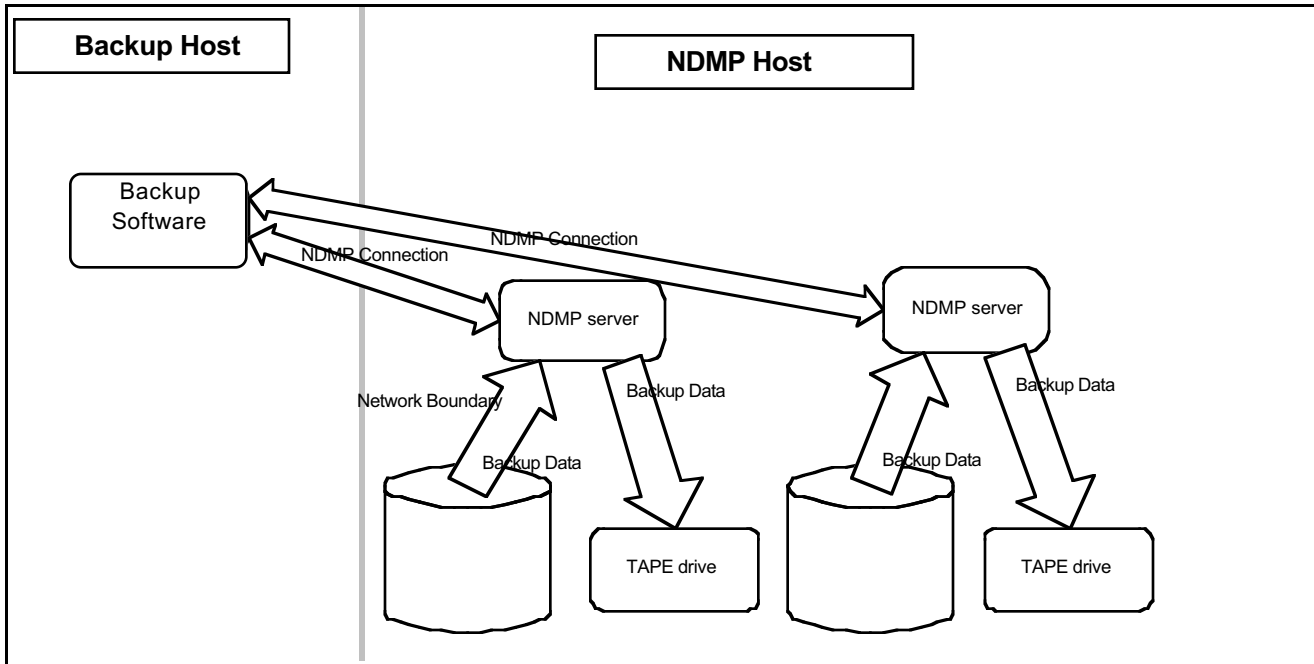


Figure 2. Two drive configuration

The NDMP can be used to backup data to a tape drive in a jukebox that is physically attached to the NDMP host. In this configuration, there is a separate instance of the NDMP server to control the robotic arm in the jukebox.

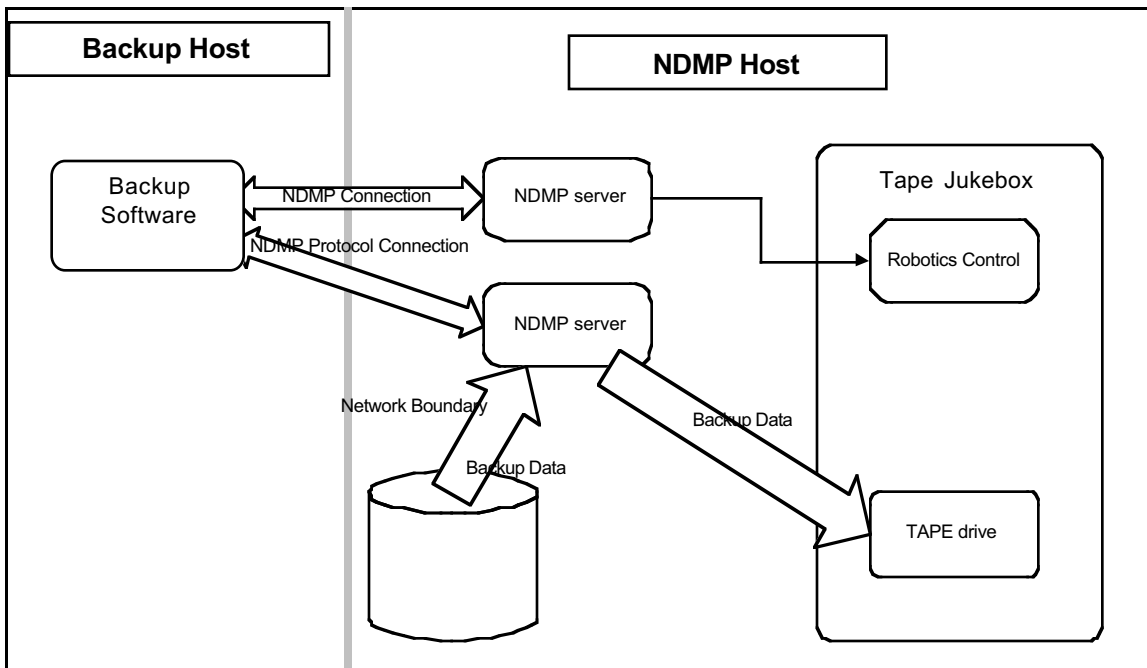


Figure 3. Jukebox configuration

2.2 Comparison Architectures

It is useful to compare the NDMP architecture to other architectures and note the similarities and differences.

rmt

The architecture is similar to the rmt architecture in that connection is made to a generic server and the server is instructed to open a specific tape drive device. The NDMP differs in that it uses a TCP/IP connection to a dedicated port whereas rmt uses the rsh demon to launch a server.

X11

The architecture is similar to the X11 architecture in that it uses a single connection to a TCP/IP port, however it differs in that the NDMP server is not assigned to a device until the client opens a device and that there is only one client per NDMPserver, whereas X11 is assigned to a display device before the first client connects and accepts connections from many clients.

RPC

The NDMP architecture is similar to RPC in that it uses XDR encoding. NDMP differs in that it is only defined for a TCP/IP connection and that it is not a call-return model, but rather a bi-directional asynchronous messaging model.

2.3 State Description

The following state diagram defines the DATA interface

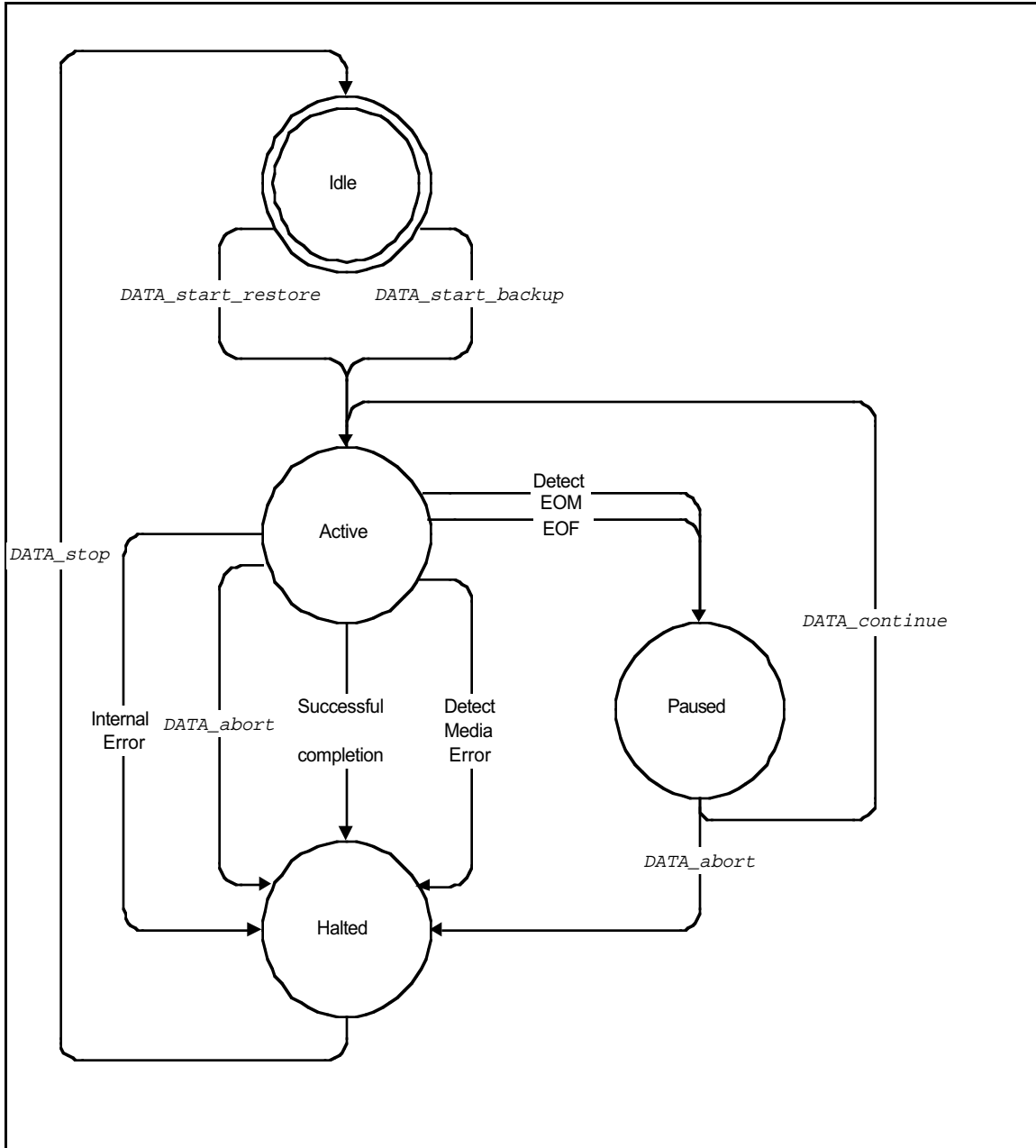


Figure 4 - Backup state diagram

The following defines the state machine for the data interface and the rules for transitions between states.

2.3.1 Idle State

This is the start state of the state machine.

Transition to active state upon receipt of ndmp_data_start_backup message.

Transition to active state upon receipt of ndmp_data_start_restore message.

2.3.2 Active State

The NDMP server remains in this state while a backup or restore is active.

Transition to halted state upon detection of a backup/restore error.

Transition to halted state upon receipt of ndmp_data_abort message.

Transition to halted state upon completion of backup/restore.

Transition to paused state upon detection of EOM.

Transition to paused state upon detection of EOF.

2.3.3 Paused State

The NDMP server remains in this state while awaiting for a tape volume to be changed.

Transition to active state upon receipt of ndmp_data_continue message.

Transition to halted state upon receipt of ndmp_data_abort message.

2.3.4 Halted State

The NDMP server enters this state after a backup/restore has either completed or been aborted.

Transition to idle state upon receipt of ndmp_data_stop message.

2.4 Protocol interfaces

Messages are grouped together by functionality into several “interfaces”.

2.4.1 NDMP Server Interfaces

The NDMP server must implement the following interfaces

- CONNECT interface

This interface will be used when a client opens the communication to a NDMP server. This interface allows the NDMP server to authenticate the client and negotiate the version of protocol used.

- CONFIG interface

This interface allows NDMP client to discover the configuration and capabilities of the NDMP server.

- SCSI interface

This interface simply passes SCSI CDBs through to the SCSI device and returns the SCSI status. The NDMP client will use this interface to control a locally attached jukebox. The NDMP client will construct SCSI CDBs and will interpret the returned status and data. This interface can also be used to exploit special features of SCSI tape drives.

- TAPE interface

The TAPE interface will support both tape positioning and tape read/write operations. The NDMP client will use this interface to control the labeling and format of the tape. The NDMP client will also use this interface for positioning of the tape during backups and restores.

- DATA interface

This is the interface that actually deals with the format of the backup data. The NDMP client will initiate backups and restores using this interface. The NDMP client provides all of the parameters that may affect the backup or restore using this interface. The NDMP client should not place any constraints on the format of the backup data other than it must be a stream of data that can be written to the tape device.

2.4.2 NDMP Client Interfaces

The NDMP server's implementation may send the following messages to the NDMP client. All of the messages that the NDMP client accepts are asynchronous. None of these messages will generate a reply message.

- NOTIFY interface

This message is used by the NDMP server to notify NDMP client that the NDMP server requires attention.

- FILE HISTORY interface

These messages allows the NDMP server to make entries in the file history for the current backup. The File History will be used by NDMP client to select files for retrieval.

- LOGGING interface

These messages allows the NDMP server to make entries in the backup log. This is used by the operator to monitor the progress and successful completion of the backup. It is also used to diagnose problems.

2.5 Messaging Protocol

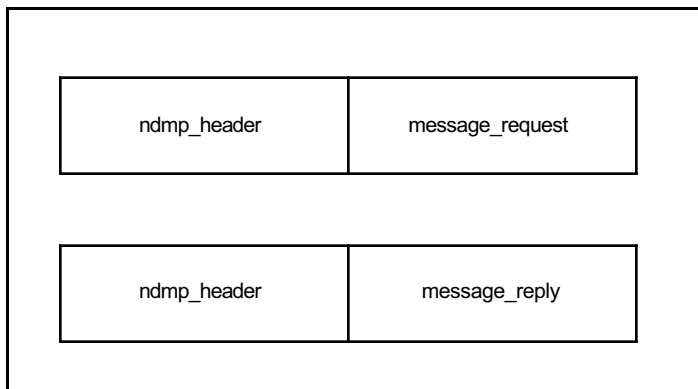
The NDMP protocol is based on XDR encoded messages transmitted over a TCP/IP connection.

The NDMP uses asynchronous messages and a message does not require a reply, however, many of the messages may result in a reply message. A message consists of a message header optionally followed by a message body. Each message is identified by a message number that is sent as part of the message header.

Messages that cannot be parsed or have invalid sequence information may be logged on the receiving host, but no response is returned to the sender.

2.6 Header

Each message will be preceded by a message header. The header will be used to identify the message, how to de-serialize the arguments.



The message headers are defined by the following XDR block

```
enum ndmp_message_type
{
    NDMP_REQUEST,
    NDMP_REPLY
};

struct ndmp_header
{
    u_long          sequence;
    u_long          time_stamp;
    ndmp_message_type message_type;
    ndmp_message    message;
    u_long          reply_sequence;
    ndmp_error      error;
};
```

sequence	The sequence number is a connection local counter that starts at 1 and increases by 1 for every message sent. The client and the server both start with 1 and increase independently.
time_stamp	The time_stamp identifies the time (in seconds since 00:00:00 GMT, Jan 1, 1970) that the message was sent.
message_type	The message_type enum identifies whether the message is a request or a reply message.
message	The message field identifies the message.
reply_sequence	The reply_sequence field is 0 in request message. In reply messages, the reply_sequence is the sequence number from the request message to which the reply is associated.
error	The error field is 0 in request messages. In reply messages, the error field identifies any problem that was incurred receiving or decoding the message. If the error value is non-zero, no message body will follow the message header.

2.7 Error

The following errors are defined:

```
enum ndmp_error
{
    NDMP_NO_ERR,
    NDMP_NOT_SUPPORTED_ERR,
    NDMP_DEVICE_BUSY_ERR,
    NDMP_DEVICE_OPENED_ERR,
    NDMP_NOT_AUTHORIZED_ERR,
    NDMP_PERMISSION_ERR,
    NDMP_DEV_NOT_OPEN_ERR,
    NDMP_IO_ERR,
    NDMP_TIMEOUT_ERR,
    NDMP_ILLEGAL_ARGS_ERR,
    NDMP_NO_TAPE_LOADED_ERR,
    NDMP_WRITE_PROTECT_ERR,
    NDMP_EOF_ERR,
    NDMP_EOM_ERR,
    NDMP_FILE_NOT_FOUND_ERR,
    NDMP_BAD_FILE_ERR,
    NDMP_NO_DEVICE_ERR,
    NDMP_NO_BUS_ERR,
    NDMP_XDR_DECODE_ERR,
    NDMP_ILLEGAL_STATE_ERR,
    NDMP_UNDEFINED_ERR,
    NDMP_XDR_ENCODE_ERR,
    NDMP_NO_MEM_ERR
};
```

NDMP_NO_ERR

No error.

NDMP_NOT_SUPPORTED_ERR

Specified message not supported. Some NDMP implementations may only support a subset of the NDMP protocol.

NDMP_DEVICE_BUSY_ERR

Specified device is in use. This error will be returned if an attempt is made to open a tape or SCSI device that is already in used.

NDMP_DEVICE_OPENED_ERR

A device is already open. NDMP connections are limited to having a single device opened at a time.

NDMP_NOT_AUTHORIZED_ERR

NDMP connection not yet authenticated. Prior to issuing most requests, the NDMP connection must first be authenticated via the connect_auth message. This error is returned if a message requiring connection authentication is received when the connection has not yet been authenticated.

NDMP_PERMISSION_ERR

The user that was used to authenticate the connection does not have the access permissions to execute this message.

NDMP_DEV_NOT_OPEN_ERR

Device not open. An attempt was made to access a device that was not open.

NDMP_IO_ERR

Device I/O error.

NDMP_TIMEOUT_ERR

Command timeout error.

NDMP_ILLEGAL_ARGS_ERR

Message received containing one or more invalid arguments.

NDMP_NO_TAPE_LOADED_ERR

Tape device could not be opened because no tape was loaded.

NDMP_WRITE_PROTECT_ERR

Tape device could not be opened in write mode because the tape is write protected.

NDMP_EOF_ERR

The tape command because end of file was encountered.

NDMP_EOM_ERR

The tape command because the end of media mark was encountered.

NDMP_FILE_NOT_FOUND_ERR

During a recover operation, a specified file was not found.

NDMP_BAD_FILE_ERR

Error due to invalid file descriptor.

NDMP_NO_DEVICE_ERR

Specified device device does not exist.

NDMP_NO_BUS_ERR

Specified SCSI controller does not exist.

NDMP_XDR_DECODE_ERR

Error decoding message.

NDMP_ILLEGAL_STATE_ERR

Message cannot be processed in the current state.

NDMP_UNDEFINED_ERR

Undefined error.

NDMP_XDR_ENCODE_ERR

Error encoding reply message.

NDMP_NO_MEM_ERR

Memory allocation error.

2.8 Message Definitions

Each NDMP message body is described using a block of XDR specification in the following format:

Message XDR definition

```
struct message_name_request {
    type    dummy_request_argument1;
    ...
};
struct message_name_reply {
    enum    ndmp_error error;
    type    dummy_reply_argument1;
    ...
};
```

Not all messages contain arguments. For messages that do contain arguments, an XDR specification block defining the arguments is provided. If the request has no arguments, then an XDR specification for the request message body is omitted. If the message does not have a reply, then an XDR specification for the reply message is omitted.

The XDR specification section is followed by a detailed description of each request and reply message argument.

All reply messages contain an error argument. If the value of the returned error status is non-zero, then some of the reply arguments may be meaningless. A description of errors that may result from executing the request is provided. This list is not necessarily a complete list and does not list the generic errors (e.g. memory allocation error). The complete list of error codes and general descriptions is provided by the next section

The following is a list of valid message numbers grouped by interface

```
enum ndmp_message {
    /* Config Interface
    NDMP_CONFIG_GET_HOST_INFO      = 0x100,
    NDMP_CONFIG_GET_BUTYPE_ATTR,

    /* SCSI Interface */
    NDMP_SCSI_OPEN                  = 0x200,
    NDMP_SCSI_CLOSE,
    NDMP_SCSI_GET_STATE,
    NDMP_SCSI_SET_TARGET,
    NDMP_SCSI_RESET_DEVICE,
    NDMP_SCSI_RESET_BUS,
    NDMP_SCSI_EXECUTE_CDB,

    /* Tape Interface */
    NDMP_TAPE_OPEN                  = 0x300,
    NDMP_TAPE_CLOSE,
    NDMP_TAPE_GET_STATE,
    NDMP_TAPE_MTIO,
    NDMP_TAPE_WRITE,
    NDMP_TAPE_READ,
    NDMP_TAPE_SET_RECORD_SIZE,
    NDMP_TAPE_EXECUTE_CDB,

    /* Data Interface */
    NDMP_DATA_GET_STATE             = 0x400,
    NDMP_DATA_START_BACKUP,
    NDMP_DATA_START_RECOVER,
    NDMP_DATA_ABORT,
    NDMP_DATA_GET_ENV,
    NDMP_DATA_RESVD1,
    NDMP_DATA_RESVD2,
    NDMP_DATA_STOP,
    NDMP_DATA_CONTINUE,

    /* Notify Interface */
    NDMP_NOTIFY_PAUSED              = 0x500,
    NDMP_NOTIFY_HALTED,
    NDMP_NOTIFY_CONNECTED,

    /* Log Interface */
    NDMP_LOG_LOG                    = 0x600,
    NDMP_LOG_DEBUG,
    NDMP_LOG_FILE,

    /* File History Interface */
    NDMP_FH_ADD_UNIX                = 0x700,
    NDMP_FH_ADD_UNIX_DIR,
    NDMP_FH_ADD_UNIX_NODE,

    /* Connect Interface */
    NDMP_CONNECT_OPEN               = 0x900,
    NDMP_CONNECT_AUTH,
    NDMP_CONNECT_CLOSE
};
```

3. NDMP Server Interfaces

This section will define the interfaces. The defined interfaces and procedures are:

3.1 **CONNECT Interface**

This interface allows NDMP server to authenticate the client and negotiate the version of protocol which will be used.

The NDMP client first connects to a well known port (currently 10,000). The NDMP server accepts the connection and sends a NOTIFY_CONNECT message. The NDMP client then sends a CONNECT_OPEN message, usually followed by a CONNECT_AUTH message.

3.1.1 Open Connection

The connect interface allows an NDMP server to authenticate the client and negotiate the version of protocol that will be used.

Message XDR definition

```

struct ndmp_connect_open_request
{
    u_short protocol_version;
};

struct ndmp_connect_open_reply
{
    ndmp_error error;
};

```

Request Arguments

protocol_version	Protocol version suggested by the NDMP client.
------------------	--

Reply Errors

NDMP_NO_ERR	Protocol version suggested by the client is supported by the server.
NDMP_ILLEGAL_ARGS_ERR	Protocol version suggested by the client is not supported by the server. The client should retry the request with a lower protocol version number.

3.1.2 Authorization

Used to authenticate the NDMP connection. Some services may be denied if the connection is not successfully authorized.

NDMP servers must support at least one of the following authentication methods.

1. NONE: no authentication required.
2. TEXT: connection is authenticated using a user name and non-encrypted password.

Message XDR definition

```

enum ndmp_auth_type
{
    NDMP_AUTH_NONE,          /* No authentication */
    NDMP_AUTH_TEXT,         /* Clear text password authentication */
    NDMP_AUTH_RESVD
};

struct ndmp_auth_text
{
    string  user<>;
    string  password<>;
};

union ndmp_auth_data switch (enum ndmp_auth_type auth_type)
{
    case NDMP_AUTH_NONE:
        void;
    case NDMP_AUTH_TEXT:
        struct ndmp_auth_text auth_text;
};

struct ndmp_connect_auth_request
{
    ndmp_auth_data  auth_data;      /* Authorization data */
};

struct ndmp_connect_auth_reply
{
    ndmp_error  error;
};

```

Request Arguments

auth_data	Authentication data. NDMP servers must support at least one of the following authentication methods. NONE: no authentication required. TEXT: connection is authenticated using a user name and non-encrypted password.
-----------	--

Reply Errors

NDMP_NO_ERR	Connection successfully authenticated.
NDMP_NOT_AUTHORIZED_ERR	Incorrect authentication data.
NDMP_ILLEGAL_ARGS_ERR	Specified authentication method not supported.

3.1.3 Close Connection

This message is used when client wants to close the NDMP connection. This message should be sent by the NDMP client before shutting down the TCP/IP connection.

Message XDR definition

```
/* no request arguments */
/* no reply arguments */
```

3.2 CONFIG Interface

This interface allows NDMP client to query the configuration of the NDMP server.

3.2.1 Get Host Info

This request is used to get information about the NDMP server.

Message XDR definition

```
/* No request message body */

struct ndmp_config_get_host_info_reply
{
    ndmp_error      error;
    string          hostname<>;
    string          os_type<>;
    string          os_vers<>;
    string          hostid<>;
    ndmp_auth_type auth_types<>;
};
```

Request Arguments

This request does not have a message body.

Reply Arguments

error	Error code.
hostname	Host name of the NDMP server
os_type	Name of NDMP server operating system (i.e. Solaris).
os_vers	Version of NDMP server operating system (i.e. 2.5).
hostid	NDMP server host identifier.
auth_types	Connection authentication types supported by the NDMP server.

Reply Errors

NDMP_NO_ERR	Request successfully processed.
-------------	---------------------------------

3.2.2 Get backup Type Attribute

This message is used to query the capability of the supported backup type.

Message XDR definition

```

const NDMP_NO_BACKUP_FILELIST    = 0x0001;
const NDMP_NO_BACKUP_FHINFO     = 0x0002;
const NDMP_NO_RECOVER_FILELIST  = 0x0004;
const NDMP_NO_RECOVER_FHINFO    = 0x0008;
const NDMP_NO_RECOVER_RESVD     = 0x0010;
const NDMP_NO_RECOVER_INC_ONLY  = 0x0020;

struct ndmp_config_get_butype_attr_request
{
    string      name<>;
};

struct ndmp_config_get_butype_attr_reply
{
    ndmp_error  error;
    u_long      attrs;
};

```

Request Arguments

name Name of backup type for which attributes are being requested (e.g. dump, tar, cpio). Backup types are NDMP server implementation dependent.

Reply Arguments

error Error code.

attrs Backup attributes bit mask. The following attributes bits are defined:

The following attributes are defined:

NDMP_NO_BACKUP_FILELIST	NDMP server doesn't support archiving of selective files as specified by a file list. (i.e. only supports dumping the entire file system.)
NDMP_NO_BACKUP_FILEINFO	NDMP server doesn't support the file history.
NDMP_NO_RECOVER_FILELIST	NDMP server doesn't support restoration of individual files.
NDMP_NO_RECOVER_FHINFO	NDMP server doesn't support the direct access restore (i.e. positioning to the offset of a backup image and restore the specified file).
NDMP_NO_RECOVER_RESVD	this value is reserved.
NDMP_NO_RECOVER_INC_ONLY	NDMP server doesn't support incremental only restoration (i.e. a full restore must be performed prior to an incremental restore).

Reply Errors

NDMP_NO_ERR Attributes for specified backup type successfully returned.

NDMP_ILLEGAL_ARGS_ERR Specified backup type not supported.

3.3 SCSI Interface

The SCSI interface allows low level control of SCSI devices such as jukeboxes.

3.3.1 Open SCSI Device

Opens the specified SCSI interface device. This operation is required before any other SCSI requests can be executed. The open must be an exclusive open. Only one NDMP server can open a SCSI device at a time. The NDMP server can only open one SCSI or tape device at a time. A `NDMPDeviceBusy` is returned if the NDMP server already has a tape or SCSI device opened.

```

struct ndmp_scsi_device
{
    string name<>;
};

struct ndmp_scsi_open_request
{
    ndmp_scsi_device    device;
};

struct ndmp_scsi_open_reply
{
    ndmp_error    error;
};

```

Request Arguments

name	Name of SCSI interface device to open. The usage of this argument is NDMP server implementation dependent. This argument may be used to specify the name of an actual SCSI device but more typically will be used to specify the name of a SCSI pass-through driver pseudo device. The specific device to be controlled is selected via the set SCSI target request.
------	--

Reply Arguments

error	Error code.
-------	-------------

Reply Errors

NDMP_NO_ERR	SCSI interface device successfully opened.
NDMP_DEVICE_OPENED_ERR	The connection already has a tape device or SCSI device open.
NDMP_NO_DEVICE_ERR	Invalid device specified.
NDMP_DEVICE_BUSY_ERR	Another NDMP connection currently has the specified device open.
NDMP_IO_ERR	IO error while opening SCSI device.

3.3.2 Close Device

Closes the currently open SCSI interface device. No further requests can be until another open request is successfully executed.

Message XDR definition

```
/* no request arguments */
struct ndmp_scsi_close_reply
{
    ndmp_error    error;
};
```

Request Arguments

This request does not have a message body.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR Device successfully closed.

NDMP_DEV_NOT_OPEN_ERR No device currently open by the connection.

3.3.3 Get SCSI State

Return the current state of the currently open SCSI interface. The target information provides information about which SCSI device is controlled by this interface.

Message XDR definition

```
/* No request message body. */
struct ndmp_scsi_get_state_reply
{
    ndmp_error    error;
    short         target_controller;
    short         target_id;
    short         target_lun;
};
```

Request Arguments

This request does not have a message body.

Reply Arguments

error Error code.

target_controller Identifier of the SCSI controller to which the currently targeted SCSI device is attached.

target_id SCSI target identifier. Specifies the SCSI bus address of the targeted device.

target_lun Logic unit number of the targeted device.

Reply Errors

NDMP_NO_ERR	Target device information successfully returned.
NDMP_DEV_NOT_OPEN_ERR	No SCSI device currently open by the connection.

3.3.4 Set SCSI Target

Selects or changes the SCSI target. When the SCSI interface is opened, we do not know if the NDMP server has opened a device file that can pass commands to a single SCSI target or to multiple SCSI targets. This part of protocol allows us to pass the information describing the SCSI target to which to send commands. Additionally, if the target can talk to multiple targets, this allows us to “scan” the SCSI bus on the NDMP host for diagnostics or the jukebox discovery.

Message XDR definition

```

struct ndmp_scsi_set_target_request
{
    ndmp_scsi_device    device;
    u_short             target_controller;
    u_short             target_id;
    u_short             target_lun;
};

struct ndmp_scsi_set_target_reply
{
    ndmp_error    error;
};

```

Request Arguments

device	SCSI device name. This argument is NDMP server implementation dependent. Some implementations may support the targeting of a device via a logical device name. If this argument is used, the following arguments may be ignored. If this argument is not specified or supported, then the following arguments must be specified.
target_controller	Identifier of the SCSI controller to which the targeted SCSI device is attached.
target_id	SCSI target identifier. Specifies the SCSI bus address of the targeted device.
target_lun	Logic unit number of the targeted device.

Reply Arguments

error	Error code.
-------	-------------

Reply Errors

NDMP_NO_ERR	Specified SCSI device successfully targeted.
NDMP_DEV_NOT_OPEN_ERR	No SCSI device currently open by the connection.

3.3.5 Reset Device

Send a SCSI device reset message to the SCSI device.

Message XDR definition

```
/* No request message body. */  
  
struct ndmp_scsi_reset_device_reply  
{  
    ndmp_error          error;  
}
```

Request Arguments

This request does not have a message body.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR SCSI device successfully reset.

NDMP_DEV_NOT_OPEN_ERR No SCSI device currently open by the connection

3.3.6 Reset Bus

Assert a SCSI bus reset on the SCSI bus to which the SCSI device is attached.

```
/* No request message body. */  
  
struct ndmp_scsi_reset_bus_reply  
{  
    ndmp_error          error;  
}
```

Request Arguments

This request does not have a message body.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR SCSI device successfully reset.

NDMP_DEV_NOT_OPEN_ERR No SCSI device currently open by the connection

3.3.7 Execute CDB

Send a SCSI Control Data Block to a SCSI device. If a check condition is generated, then the extended sense data is also retrieved. Data can be transferred to or from the SCSI device as part of the command.

The server selects the SCSI target. The cdb is sent to the SCSI device in command mode. If DATA_OUT is set in the flags, then the data_out is sent to the SCSI device in data mode. Sometimes, the host will disconnect from the target and waits for a re-select. If timeout is zero then the host will wait indefinitely for the target to re-select. If timeout is

non-zero then the host will wait `timeout` milliseconds for the target to reselect. If the reselect does not occur then an `NDMPTimeout` exception occurs. If the target re-selects and the status is `CHECK CONDITION`, then the server executes a `REQUEST SENSE` cdb. If the `DATA_IN` flag is set, then server gets data from the target in a data mode and `datain_len` and `dataout_len` are set to indicate the expected data length. The SCSI status, the data in and the extended sense data is returned.

`DATA_IN` and `DATA_OUT` are with reference to the host. They refer to data from the SCSI device into the host and data out of the host and to the SCSI device.

Message XDR definition

```

/* SCSI CDB flags */
const NDMP SCSI_DATA_IN = 0x00000001;
const NDMP SCSI_DATA_OUT = 0x00000002;

struct ndmp_execute_cdb_request
{
    u_long    flags;
    u_long    timeout;
    u_long    alloc_len;
    opaque    cdb<>;
    opaque    dataout<>;
};

struct ndmp_execute_cdb_reply
{
    ndmp_error error;
    u_char    status;
    u_long    dataout_len;
    opaque    datain<>;
    opaque    ext_sense<>;
};

```

Request Arguments

<code>flags</code>	Specifies the data transfer (if any) direction.
<code>timeout</code>	Number of milliseconds to wait if a re-select occurs. If <code>timeout</code> is zero then the host will wait indefinitely for the target to reselect.
<code>alloc_len</code>	If the data transfer direction is <code>DATA_IN</code> , the expected number of data bytes that will be received.
<code>cdb</code>	The SCSI command data block.
<code>dataout</code>	If the data transfer direction is <code>DATA_OUT</code> , the data to be transferred to the SCSI device.

Reply Arguments

<code>error</code>	Error code.
<code>status</code>	SCSI status byte.
<code>data_out</code>	If the data transfer direction is <code>DATA_OUT</code> , the number of data bytes transferred to the device.

datain	If the data transfer direction is DATA_IN, the data transferred from the SCSI device.
ext_sense	Extended SCSI sense data.

Reply Errors

NDMP_NO_ERR	Message successfully processed. Does not necessarily indicate that the SCSI command was successfully executed. The returned SCSI status byte must be used to determine if the command was successful.
NDMP_DEV_NOT_OPEN_ERR	No SCSI device currently open by the connection.
NDMP_NO_DEVICE_ERR	A SCSI device has not yet been targeted via the set SCSI target message.
NDMP_ILLEGAL_ARGS	Invalid argument in request message.
NDMP_TIMEOUT_ERR	The SCSI command timed out.
NDMP_NO_BUS_ERR	The SCSI currently specified controller (as set via the set SCSI target message) does not exist.

3.4 TAPE Interface

Provide complete control of a tape drive. If the tape drive is a SCSI tape drive, then this interface also provides low level CDB access to the tape drive. This interface is analogous to the rmt protocol. The physical device is assigned when the server is started.

3.4.1 Open Tape Device

Opens the tape device in the specified mode. This operation is required before any other tape requests can be executed. The device is opened exclusively. Each tape device may only be opened by one NDMP server at a time. Each NDMP server may only have one tape or SCSI device open at a time. If the drive does not have a tape loaded, an error is returned. If the loaded media is write protected, then the device may only be opened in read only mode.

Message XDR definition

```
enum ndmp_tape_open_mode
{
    NDMP_TAPE_READ_MODE,
    NDMP_TAPE_WRITE_MODE
};

struct ndmp_tape_open_request
{
    ndmp_tape_device    device;
    ndmp_tape_open_mode mode;
};

struct ndmp_tape_open_reply
{
    ndmp_error          error;
};
```

Request Arguments

device Name of tape device to open.

mode Tape open mode.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR Tape device successfully opened.

NDMP_DEVICE_OPENED_ERR The NDMP server already has a SCSI device or tape device open.

NDMP_NO_DEVICE_ERR The specified device does not exist.

NDMP_DEVICE_BUSY_ERR The device is already open by another NDMP server or system process.

NDMP_IO_ERR Device I/O error.

NDMP_WRITE_PROTECT_ERR Device can not be opened in write mode because the tape is write protected.

NDMP_NO_TAPE_LOADED_ERR No tape loaded in the tape device.

3.4.2 Close Device

Close the tape drive. Drive can be opened by others. No further requests can be processed until another open request is successfully executed.

Message XDR definition

```
/* No request message body. */  
  
struct ndmp_tape_close_reply  
{  
    ndmp_error      error;  
};
```

Request Arguments

This message does not have a message body.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR Tape device successfully closed.

NDMP_DEV_NOT_OPEN_ERR No tape device currently open by the connection.

NDMP_IO_ERR Device I/O error.

3.4.3 Get Tape State

Return the state of the tape drive interface.

Message XDR definition

```

/* No request message body. */

struct u_quadl
{
    u_long  high;
    u_long  low;
};

/* flags */
const NDMP_TAPE_NOREWIND    = 0x0008;
const NDMP_TAPE_WR_PROT    = 0x0010;
const NDMP_TAPE_ERROR      = 0x0020;
const NDMP_TAPE_UNLOAD     = 0x0040;

struct ndmp_tape_get_state_reply
{
    ndmp_error      error;
    u_long          flags;
    u_long          file_num;
    u_long          record_num;
    u_long          record_size;
    u_long          soft_errors;
    u_long          block_size;
    u_long          blockno;
    u_quadl        total_space;
    u_quadl        space_remain;
};

```

Request Arguments

This message does not have a message body.

Reply Arguments

error Error code.

Flags Bitmask of the following tape device mode flags:

NDMP_TAPE_NOREWIND Upon device close, the tape will not be rewound.

NDMP_TAPE_WR_PROT The loaded tape is write-protected.

NDMP_TAPE_ERROR A media error was detected during the previous tape operation. This bit is cleared at the start of each tape operation.

NDMP_TAPE_UNLOAD The currently loaded tape will automatically be unloaded when the device is closed. Only applies to media changer devices such as tape stackers and jukeboxes.

file_num	Current file position. First file on the tape is file number 0.
record_num	Current record position. First record in a file is record number 0.
record_size	Tape record size in bytes.
soft_errors	Total number of soft media errors detected since the device was opened.
block_size	Tape block size in bytes.
blockno	Current tape block number. First tape block is block number 0.
total_space	Total tape capacity in bytes. 0 if this feature not supported by the NDMP server implementation.
space_remain	Total remaining tape capacity in bytes. 0 if this feature not supported by the NDMP server implementation.

Reply Errors

NDMP_NO_ERR	Tape state successfully returned.
NDMP_DEV_NOT_OPEN_ERR	No tape device currently open by the connection.
NDMP_IO_ERR	Device I/O error.

3.4.4 MTIO

Provides access to the standard magnetic tape I/O operations. When spacing forward over a record, the tape head is positioned in the tape gap between the record just skipped and the next record. When spacing forward over file marks, the tape head is positioned in the tape gap between the next file mark and the record that follows it. When spacing backward over a record data, the tape head is positioned in the tape gap immediately preceding the tape record where the tape head is currently positioned. When spacing backward over file marks, the tape head is positioned in the tape gap preceding the file mark the next read would fetch the EOF.

Message XDR definition

```

enum ndmp_tape_mtio_op
{
    NDMP_MTIO_FSF,
    NDMP_MTIO_BSF,
    NDMP_MTIO_FSR,
    NDMP_MTIO_BSR,
    NDMP_MTIO_REW,
    NDMP_MTIO_EOF,
    NDMP_MTIO_OFF
};

struct ndmp_tape_mtio_request
{
    ndmp_tape_mtio_op    tape_op;
    u_long               count;
};

struct ndmp_tape_mtio_reply
{
    ndmp_error           error;
    u_long               resid_count;
};

```

Request Arguments

tape_op	One of the following tape operations:
NDMP_MTIO_FSF	Forward space over file marks. The tape head is positioned in the tape gap between the file mark and the record that follows it.
NDMP_MTIO_BSF	Backward space over file marks. The tape head is positioned in the tape gap preceding the file mark such that the next read encounters EOF.
NDMP_MTIO_FSR	Forward space over tape records. The tape head is positioned in the tape gap between the record just skipped and the next record.
NDMP_MTIO_BSR	Backward space over tape records. The tape head is positioned in the tape gap preceding the tape record just skipped.
NDMP_MTIO_REW	Rewind the tape.
NDMP_MTIO_EOF	Write end of file marks.
NDMP_MTIO_OFF	Eject the tape from the device.
count	Number of operations to perform.

Reply Arguments

error	Error code.
resid_count	Residual operation count. Represents the number of operations that were not able to be performed due to encountering beginning of tape, end of tape, end of written media, or a tape error.

Reply Errors

NDMP_NO_ERR	Tape operation successfully completed.
NDMP_DEV_NOT_OPEN_ERR	No tape device currently open by the connection.
NDMP_IO_ERR	Device I/O error.
NDMP_ILLEGAL_ARGS_ERR	Invalid tape operation specified.
NDMP_WRITE_PROTECT_ERR	Tape is write protected.

3.4.5 Write

Writes data to the tape device. The NDMP server performs fixed size record tape writes. This size is set via the set tape record size message. If necessary, the NDMP server will pad the tape data to make a complete record.

Message XDR definition

```

struct ndmp_tape_write_request
{
    opaque          data_out<>;
};

struct ndmp_tape_write_reply
{
    ndmp_error      error;
    u_long          count;
};

```

Request Arguments

data_out	The data to be written to the tape device.
----------	--

Reply Arguments

error	Error code.
count	Number of data bytes written. If the size of data_out is not a multiple of the tape record size, this count will reflect the total amount of data written including the pad data.

Reply Errors

NDMP_NO_ERR	All data successfully written to the tape device.
NDMP_DEV_NOT_OPEN_ERR	No tape device currently open by the connection.
NDMP_IO_ERR	Device I/O error.
NDMP_WRITE_PROTECT_ERR	Tape is write protected.
NDMP_EOM_ERR	End of tape was encountered while writing.

3.4.6 Read

Reads data from the tape drive. The NDMP server always reads a complete record. If the specified number of bytes to read is not a multiple of the tape record size, then the NDMP server discards the bytes from the end of the record. The next read will return bytes starting from the beginning of the next record. To perform contiguous reads, the number of bytes read must be a multiple of the tape record size.

Message XDR definition

```

struct ndmp_tape_read_request
{
    u_long          count;
};

struct ndmp_tape_read_reply
{
    ndmp_error      error;
    opaque          data_in<>;
};

```

Request Arguments

count	Number of bytes to read.
-------	--------------------------

Reply Arguments

error	Error code.
data_in	The data read from the tape drive.

Reply Errors

NDMP_NO_ERR	Requested number of bytes successfully read from the tape device.
NDMP_DEV_NOT_OPEN_ERR	No tape device currently open by the connection.
NDMP_IO_ERR	Device I/O error during read.
NDMP_EOF_ERR	End of file was encountered while reading. The number of returned data bytes may be less than the number of bytes requested.

3.4.7 Set Record Size

Sets the tape record size. The NDMP server performs all tape read and write operations in multiples of the tape record size.

Message XDR definition

```

struct ndmp_tape_set_record_size_request
{
    u_long          len;
};

struct ndmp_tape_set_record_size_reply
{
    ndmp_error      error;
}

```

Request Arguments

len Tape record size in bytes.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR Tape record size successfully set.

NDMP_ILLEGAL_ARGS_ERR Invalid tape record size. Some NDMP server implementations may limit the tape record size. Implementations may also limit the record size to a multiple of the tape device block size.

NDMP_ILLEGAL_STATE_ERR Tape record size cannot be set in the current state. Some NDMP server implementations may limit the tape record size to being set only when the tape is positioned at the beginning of tape.

3.4.8 Execute CDB

This message behaves in exactly the same way as the SCSI_EXECUTE_CDB except that it sends the cdb to the tape device. This request can't be used to change the state of the tape device.

Message XDR definition

```

typedef scsi_execute_cdb_request tape_execute_cdb_request;
typedef scsi_execute_cdb_reply tape_execute_cdb_reply;

```

3.5 DATA Interface

Selects and formats data for backup. Extracts files from backup for retrieval.

3.5.1 Get Data State

Returns data state information that may be used for monitoring the progress of the current data operation.

Message XDR definition

```

/* No request message body. */

enum    ndmp_data_operation
{
    NDMP_DATA_OP_NOACTION,
    NDMP_DATA_OP_BACKUP,
    NDMP_DATA_OP_RESTORE
};

enum    ndmp_data_state
{
    NDMP_DATA_STATE_IDLE,
    NDMP_DATA_STATE_ACTIVE,
    NDMP_DATA_STATE_PAUSED,
    NDMP_DATA_STATE_HALTED
};

enum    ndmp_data_halt_reason
{
    NDMP_HALT_NA,
    NDMP_HALT_SUCCESSFUL,
    NDMP_HALT_ABORTED,
    NDMP_HALT_MEDIA_ERROR,
    NDMP_HALT_INTERNAL_ERROR,
};

enum    ndmp_data_pause_reason
{
    NDMP_PAUSE_NA,
    NDMP_PAUSE_EOM,
    NDMP_PAUSE_EOF,
    NDMP_PAUSE_RESVD
};

struct ndmp_data_get_state_reply
{
    ndmp_error            error;
    ndmp_data_operation  operation;
    ndmp_data_state      state;
    ndmp_data_halt_reason halt_reason;
    ndmp_data_pause_reason pause_reason;
    u_quadl              resvd1;
    u_quadl              bytes_processed;
    u_quadl              est_bytes_remain;
    u_long               est_time_remain;
    u_quadl              resvd2;
    u_quadl              resvd3;
};

```

Request Arguments

This message does not have a message body.

Reply Arguments

error	Error code.
operation	Data operation currently in progress.

NDMP_DATA_OP_NOACTION	No data operation currently in progress.
NDMP_DATA_OP_BACKUP	Backup operation currently in progress.
NDMP_DATA_OP_RESTORE	Restore operation currently in progress.
state	Current state of the NDMP server.
NDMP_DATA_STATE_IDLE	No active data operation.
NDMP_DATA_STATE_ACTIVE	Data operation in progress.
NDMP_DATA_STATE_PAUSED	Data operation paused awaiting operator attention.
NDMP_DATA_STATE_HALTED	Data operation completed.
halt_reason	Reason the data operation is halted.
NDMP_HALT_NA	Data operation not in progress or not in the halt state.
NDMP_HALT_SUCCESSFUL	Data operation completed successfully.
NDMP_HALT_ABORTED	Data operation aborted by the backup software.
NDMP_HALT_MEDIA_ERROR	Data operation halted due to unrecoverable media error.
NDMP_HALT_INTERNAL_ERROR	Data operation halted due to unrecoverable error incurred by the NDMP server data backup/recover software.
NDMP_HALT_NO_SPLIT	Data operation halted because the data backup/recover method does not support multi-volume backup/recover and the end of a volume was reached prior to completing the backup/recover operation.
pause_reason	Reason the data operation is paused.
NDMP_PAUSE_NA	Data operation not in progress or not in the pause state.
NDMP_PAUSE_EOM	Data operation encountered end of media. Backup software attention required.
NDMP_PAUSE_EOF	Data operation encountered end of file. Backup software attention required.
NDMP_PAUSE_RESVD	Reserved value.
resvd1	Reserved field.
bytes_processed	Total number of bytes processed by the data operation.
est_bytes_remain	Estimated number of bytes processed remaining to be processed by the data operation. May be set to 0 to indicate that this feature is not supported by the NDMP server.
est_time_remain	Estimated number of seconds until the data operation to completes. May be set to 0 to indicate that this feature is not supported by the NDMP server.
resvd2	Reserved field.

resvd3 Reserved field.

Reply Errors

NDMP_NO_ERR Data state successfully returned.

3.5.2 Backup

Begins a backup. The id identifies the object to be backed up. The meaning of id is implementation dependent. The type of backup is also implementation dependent. The env is a list of parameters that may affect the behavior of the backup. The env returned by the DATA_GET_ENV will be saved and made available to the retrieval process.

The backup is allowed to write to tape but cannot reposition the tape. It must enter a paused state and notify the NDMP client if it encounters an EOM. It must enter a halted state and notify the NDMP client if an IO error is detected on the tape.

Message XDR definition

```

struct pval
{
    string  name<>;
    string  value<>;
};

struct ndmp_data_start_backup_request
{
    string          bu_type<>; /* Backup method to use */
    pval            env<>;     /* Parameters that may modify backup
*/
};

struct ndmp_data_start_backup_reply
{
    ndmp_error      error;
};

```

The following environmental variables are defined by NDMP client.

Variable Name	Meaning	Value
TYPE	the type of backup	the value could be different from the bu_type passed to NDMP server.
ID	identifies the object to be backed up	implementation dependent
HIST	a flag to maintain file history	y/n

The NDMP server may require other variables depending on the type of backup:

For example, the following environmental variables may be required by a dump type of backup.

Variable Name	Meaning	Value
PREFIX	the prefix path for this request	path name

LEVEL	dump level	0 - 9
filesystem	device or file system name to be backed up	file system or device name (e.g. /dev/rsd0a)

The following environmental variables may be required by a tar type of backup.

Variable Name	Meaning	Value
files	a list of files to be backed up	e.g. /* /*.c /*h

Request Arguments

bu_type	The name of the backup method. Backup methods are NDMP server implementation dependent.
env	List of parameter names and values for configuring the backup method. Backup method parameters are NDMP server implementation dependent.

Reply Arguments

error	Error code.
-------	-------------

Reply Errors

NDMP_NO_ERR	Backup operation successfully started.
NDMP_ILLEGAL_STATE_ERR	A data operation is already in progress. Only one data operation per connection is allowed to be executing at a time.
NDMP_ILLEGAL_ARGS_ERR	Invalid backup method, invalid backup method parameter, or invalid backup method parameter value specified.
NDMP_DEV_NOT_OPEN_ERR	No tape device is currently open by the connection.
NDMP_WRITE_PROTECT_ERR	The tape is write protected.

3.5.3 Recover

Recover the files specified in `nlist` from the backup. The `env` is the list of parameters and values saved at the end of the backup.

The recovery can reposition the tape as long as it does not position outside of the current tape file. Any repositioning of the tape should be reflected in the tape status. If the recovery encounters an EOF, it should enter a paused state and notify the NDMP client to load the next tape.

Message XDR definition

```

struct ndmp_name
{
    string          name<>;
    string          dest<>;
    u_short        resvd;
    u_quadl        fh_info;
};

struct ndmp_data_start_recover_request
{
    pval           env<>;
    ndmp_name      nlist<>;
    string         bu_type<>;
};

struct ndmp_data_start_recover_reply
{
    ndmp_error     error;
};

```

Request Arguments

env	The backup environment that was returned from a data get environment request made prior to notifying the NDMP server that the backup was complete via a data stop message.
nlist	List of files to be recovered and the location each file is to be recovered to. Definition of list entry:
name	Name of a file/directory to be recovered. The name is the original backed up path name and is relative to the backup root directory.
dest	Full destination pathname to be used when recovering the file.
resvd	Reserved
fh_info	File history tape positioning data recorded when the file was backed up. This data may be used by the restore method to perform tape positioning for fast data retrieval. The positioning data is NDMP server dependent. Typically it will be the byte or record offset from the beginning of the tape of the file to be recovered. This field is ignored by data method implementations that do not support this feature.
bu_type	Name of the recover method. Recover methods are NDMP server implementation dependent.

Reply Arguments

error	Error code.
-------	-------------

Reply Errors

NDMP_NO_ERR	Recover operation successfully started.
NDMP_ILLEGAL_STATE_ERR	A data operation is already in progress. Only one data operation per connection is allowed to be executing at a time.

NDMP_ILLEGAL_ARGS_ERR	Invalid recover method, invalid recover method parameter, invalid recover method parameter value, or invalid name list entry specified.
NDMP_DEV_NOT_OPEN_ERR	No tape device is currently open by the connection.

3.5.4 Abort

Send a message to abort the current backup or restore. The operation should be terminated as soon as possible.

Message XDR definition

```
/* No request message body. */
struct ndmp_data_abort_reply
{
    ndmp_error          error;
};
```

Request Arguments

This message does not have a request body.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR Data operation successfully terminated.

NDMP_ILLEGAL_STATE_ERR Illegal state. The NDMP server is not in the the active or paused state.

3.5.5 Stop

Send a message to inform NDMP server that the current backup is complete. NDMP server will change to idle state and be ready for process the other request.

Message XDR definition

```
/* No request message body. */
struct ndmp_data_stop_reply
{
    ndmp_error          error;
};
```

Request Arguments

This message does not have a message body.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR Message successfully processed.

NDMP_ILLEGAL_STATE_ERR Illegal state. NDMP server not in the halted state.

3.5.6 Continue

Send a message to resume the backup due to EOM.

Message XDR definition

```
/* No request message body. */  
  
struct ndmp_data_continue_reply  
{  
    ndmp_error          error;  
};
```

Request Arguments

This message does not have a message body.

Reply Arguments

error Error code.

Reply Errors

NDMP_NO_ERR Message successfully processed.

NDMP_ILLEGAL_STATE_ERR Illegal state. NDMP server not in the halted state.

3.5.7 Get ENV

Returns the backup method environment parameters and values. The environment is initially set when a backup operation is started. The NDMP server may modify existing parameter values and/or add new environment parameters/values.

Message XDR definition

```
/* No request message body. */  
  
struct ndmp_data_get_env_reply  
{  
    ndmp_error          error;  
    pval                env<>;  
};
```

Request Arguments

This message does not have a message body.

Reply Arguments

error

Error code.

env

The backup method environment parameters and values.

Reply Errors

NDMP_NO_ERR

Environment successfully returned.

NDMP_ILLEGAL_STATE_ERR

Illegal state. A data operation is not currently in progress.

4. NDMP Client Interfaces

These interfaces are available to the NDMP server. Only the requests that are intended for use by the NDMP server are available. Many of the interfaces require initialization which will be performed by NDMP client before the NDMP server is called.

4.1 NOTIFY Interface

This interface is used by the NDMP server to let NDMP client know that the NDMP server requires attention.

4.1.1 Notify Paused

This message is used to notify NDMP client that the NDMP server has paused.

Message XDR definition

```
struct ndmp_notify_paused_request
{
    ndmp_data_pause_reason  reason;
    u_quadl                 resvd;
};
/* No reply */
```

Request Arguments

reason	Reason code identifying why the NDMP server pasued the data operation.
resvd	reserved value

Reply Arguments

No reply is sent in response to this message.

4.1.2 Notify Halted

This message is used to notify NDMP client that the NDMP server has halted.

Message XDR definition

```
struct ndmp_notify_halted_request
{
    ndmp_data_halt_reason  reason;
    string                 text_reason<>;
};
/* No reply */
```

Request Arguments

reason	Reason code identifying why the NDMP server halted the data operation.
text_reason	Text string for conveying diagnostic information related to why the data operation was halted.

Reply Arguments

No reply is sent in response to this message.

4.1.3 Notify Connect

The message is sent to the NDMP client immediately after connection establishment. This message is also sent prior to gracefully closing the NDMP connection.

Message XDR definition

```
enum    ndmp_connect_reason
{
    NDMP_CONNECTED,
    NDMP_SHUTDOWN,
    NDMP_REFUSED
};

struct  ndmp_notify_connected_request
{
    ndmp_connect_reason    reason;
    u_short                protocol_version;
    string                 text_reason<>;
};
/* No reply */
```

Request Arguments

reason	Reason code describing the current connection state.
NDMP_CONNECTED	NDMP connection successfully established. This code will be returned in a message sent immediately after successful connection establishment.
NDMP_SHUTDOWN	The NDMP server is shutting down the NDMP connection. Will typically used when shutting down the NDMP host to gracefully close down the NDMP connection.
NDMP_REFUSED	NDMP connection refused by the NDMP server. This code will be returned in a message sent immediately after a connection establishment attempt to notify the NDMP client that the NDMP server is not able to accept the connection at the current time. This will typically be used if the NDMP server implementation limits the total number of concurrent NDMP connections, when NDMP services on the NDMP host are disabled, or when the NDMP host is in the process of shutting down.

Reply Arguments

No reply is sent in response to this message.

4.2 LOGGING Interface

The logging interface is used by the NDMP server to send informational and diagnostic data to the NDMP client. This data is used by the client to monitor the progress of the currently running data operation and to diagnose problems.

4.2.1 Log

Send an informational message to the NDMP client. Typically used to send log messages generated by the backup or recover method.

Message XDR definition

```
struct ndmp_log_log_request
{
    string          entry<>;
};
/* No reply */
```

Request Arguments

entry Text message.

Reply Arguments

No reply is sent in response to this message.

4.2.2 Debug

Send a diagnostic message to the NDMP client. This message typically used to diagnose NDMP server problems. The mechanism used to enable/disable diagnostic messages is NDMP server dependent. This feature is primarily intended to be used during software development and when troubleshooting.

Message XDR definition

```
enum ndmp_debug_level
{
    NDMP_DBG_USER_INFO,
    NDMP_DBG_USER_SUMMARY,
    NDMP_DBG_USER_DETAIL,
    NDMP_DBG_DIAG_INFO,
    NDMP_DBG_DIAG_SUMMARY,
    NDMP_DBG_DIAG_DETAIL,
    NDMP_DBG_PROG_INFO,
    NDMP_DBG_PROG_SUMMARY,
    NDMP_DBG_PROG_DETAIL
};

struct ndmp_log_debug_request
{
    ndmp_debug_level    level;
    string              message<>;
};
/* No reply */
```

Request Arguments

level The level is divided into two components. The first component is the intended audience. The audience can be the end user (user), the technical support personnel (diag), or the development engineer (prog). The second component is the level of detail requested. The level of detail is specified as info, summary, and detail. There are no specific guidelines on the use of level of detail, but a message that typically is encountered less than 10

times during a backup should be an info level. While a message that is encountered more than 100 times should be at a detail level.

message

Diagnostic text message.

Reply Arguments

No reply is sent in response to this message.

4.3 FILE HISTORY Interface

The NDMP server uses this interface to send file history entries to the NDMP client. The file history entries provide a file by file record of every file backed up by the backup method. The file history data is defined using a UNIX filesystem compatible format. There are two sets of messages for sending file history data. The first set consisting of just the add path message is for use by filename based backup methods (such as the UNIX tar and cpio commands) for which the full pathname and file attributes are available at the time each file is backed up. The second set consisting of the add directory and add node messages is for use by inode based backup methods (such as the UNIX dump command) for which the full pathname is not necessarily available at the time each file is backed up. Some backup methods may not support the sending of file history data.

4.3.1 Add Unix Path

Sends several pathname based file history entries.

Message XDR definition

```
typedef string ndmp_unix_path<>;
enum ndmp_unix_file_type
{
    NDMP_FILE_DIR,
    NDMP_FILE_FIFO,
    NDMP_FILE_CSPEC,
    NDMP_FILE_BSPEC,
    NDMP_FILE_REG,
    NDMP_FILE_SLINK,
    NDMP_FILE_SOCKET
};

struct ndmp_unix_file_stat
{
    ndmp_unix_file_type    ftype;
    u_long                 mtime;
    u_long                 atime;
    u_long                 ctime;
    u_long                 uid;
    u_long                 gid;
    u_long                 mode;
    u_quadl                size;
    u_quadl                fh_info;
};

struct ndmp_unix_fh_entry
{
    ndmp_unix_path         name;
    ndmp_unix_file_stat    fstat;
};

struct ndmp_fh_add_unix_request
{
    ndmp_unix_fh_entry     unix_fh_entries<>;
};
/* No reply */
```

Request Arguments

unix_fh_entries	Array of file history entries. Each entry contains:
name	The full pathname of the backed up file relative to the backup root directory.
fstat	File attribute data consisting of:
ftype	File type.
mtime	Time the file was last modified (in seconds since 00:00:00 GMT, Jan 1, 1970).
atime	Time the file was last accessed (in seconds since 00:00:00 GMT, Jan 1, 1970).
ctime	Time the file status was last modified (in seconds since 00:00:00 GMT, Jan 1, 1970). Indicates the last time that either the file data or the file attributes were modified.
uid	File owner identifier.
gid	File group identifier.
mode	File mode flags.
size	File size.
fh_info	File history tape positioning data representing the tape position at the time the file was written to tape. This data may be used by the restore method to perform tape positioning for fast data retrieval. The positioning data is NDMP server dependent. Typically it will be the byte or record offset from the beginning of the tape of the file to be recovered. This field is ignored by data method implementations that do not support this feature.

Reply Arguments

No reply is sent in response to this message.

4.3.2 Add Unix Dir

This message is used to support directory/inode types of backup formats. The `node` number can be any unique number that matches a corresponding `fh_add_unix_node` message.

Message XDR definition

```

struct ndmp_unix_dir_ent
{
    ndmp_unix_path      name;
    u_long              node;
    u_long              parent;
};

struct ndmp_fh_add_unix_dir_request
{
    ndmp_unix_dir_ent   dirs<>;
};

/* No reply */

```

Request Arguments

dirs	Array of directory entries. Each entry contains:
name	Node file name. This is not a full pathname; just the basename relative to the node's parent directory.
node	Node identifier that matches a node in a corresponding add node message. NDMP server implementation dependent but will typically be the inode number of the file.
parent	Node identifier of the node's parent directory. NDMP server implementation dependent but will typically be the inode number of the file.

Reply Arguments

No reply is sent in response to this message.

4.3.3 Add Unix Node

Add a list of attribute information to a directory/inode type of file history. These entries must match a corresponding node number in a add directory message. For each file, this message must be sent after the corresponding fh_add_unix_dir message.

Message XDR definition

```

struct ndmp_unix_node
{
    ndmp_unix_file_stat  fstat;
    u_long              node;
};

struct ndmp_fh_add_unix_node_request
{
    ndmp_unix_node       nlist<>;
};

/* No reply */

```

Request Arguments

nlist	Array of file history node entries. Each entry contains:
fstat	File attribute data.

node Node identifier that matches a node in a corresponding add directory message. NDMP server implementation dependent but will typically be the inode number of the file.

Reply Arguments

No reply is sent in response to this message.

5. References

[1] RFC 1832 , "XDR: External Data Representation Standard", R. Srinivasan, Sun Microsystems, August 1995

6. Security

The NDMP client is normally authenticated by the NDMP server, however the NDMP server can optionally permit access without authentication. Once authenticated, privileges are not specified by the NDMP protocol, but it is expected that NDMP server implementations will permit data to be transferred to and from tape using the protocol.

File history information is transferred to the NDMP client through a TCP/IP connection.

7. Authors

D. Hitz
Network Appliance
319 North Bernardo Ave.
Mountain View, CA 94043
USA
Tel: 415-428-5100
Fax: 415-428-5151
email: hitz@netapp.com
<http://www.netapp.com>

R. Stager
PDC
111C Lindbergh Ave
Livermore, CA 94550
USA
Tel: 510-449-6881
Fax: 415-428-5151
email: rstager@pdc.com
<http://www.pdc.com>

Expires: May 1997