The ADIC
Distributed AML Server

Message Filter Installation Guide

Advanced Digital Information Corp

Copyright Notice
© *Copyright* adic *1999*

The information contained in this document is subject to change without notice.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without prior written consent of adic.
adic shall not be liable for errors contained herein or for incidental or consequential damages (including lost profits) in connection with the furnishing, performance or use of this material whether based on warranty, contract, or other legal theory.

All trademarks are property of their respective owners.

Copyright Notice (Europe)
© *Copyright* adic Europe*1999*

All rights reserved. No part of this document may be copied or reproduced in any form or by any means, without prior written permission of adic Europe, ZAC des Basses Auges, 1 rue Alfres de Vigny, 78112 - Fourqueux, FRANCE.

adic Europe assumes no responsibility for any errors that may appear in this document, and retains the right to make changes to these specifications and descriptions at any time, without notice.

This publication may describe designs for which patents are pending, or have been granted. By publishing this information, adic Europe conveys no license under any patent or any other right.

adic Europe makes no representation or warranty with respect to the contents of this document and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, adic Europe reserves the right to revise or change this publication without obligation on the part of adic Europe to notify any person or organization of such revision of change.

Every effort has been made to acknowledge trademarks and their owners. Trademarked names are used solely for identification or exemplary purposes, any omission are made unintentionally.

adic and adic Europe are trademarks of Advanced Digital Information Corporation.

Advanced Digital Information Corporation
Telephone: (303) 705-3900
Fax: (303) 792-2465
Customer Assistance: 1-800-827-3822
World Wide Web: http://www.adic.com

ADIC Europe
ZAC des Basses Auges
1, rue Alfred de Vigny
78112 FOURQUEUX, FRANCE
Telephone: 33.1.3087.5300
Fax: 33.1.3087.5301

Document number: 600818-A
Published: 11 June 1999

Printed in the USA

Contents

1

Introduction

Overview 1-3
Intended Audience 1-3
Organization 1-3
Associated Documents 1-4
Explanation of Symbols and Notes 1-4
Hazard Alert Messages 1-4
Assistance 1-6
Federal Communication Commission Note 1-7

2

Message Filter

Overview 2-3
Message Filter Description 2-3

3

Installation and Configuration

Overview 3-3

Before Installing the Message Filter 3-3

Install the Message Filter 3-3
Configure the Message Filter 3-4

4

Administration

Overview 4-3

Configuration 4-3
Drive Statement 4-3
Line Statements 4-3

Start-up / Shutdown 4-6
Start Message Filter 4-6
Stop Message Filter 4-7

5

File Description

Overview 5-3

msgf 5-3

start_msgf 5-3
MSGF_MEDIA Mapping 5-4

stop_msgf 5-5

6

Message Type Interface Description

Overview 6-3

Init Function 6-3

Term Function 6-4

Reader Function 6-5

Filter Function 6-5

Utility Functions 6-6
- TOKEN *msg_parser (char *line text) 6-6
- TOKEN *find_token (TOKEN *token_chain, enum token_type token) 6-6
- void free_tokens (TOKEN *token_chain) 6-6
- void aci_driver (TOKEN *token_chain) 6-6
- void suspend_on_drive (struct drive_entry *drive, long timeout) 6-6
- void update_drive (struct drive_entry *drive, char *volser) 6-7
- struct drive_entry *find_drive (char *drive_name) 6-7
- enum aci_media convert_media (char *media) 6-7

The Drive Queue 6-7

Index

Figures

Figure 2-1 Signalflow Trough the Message Filter 2-4
Figure 4-1 Example of a Drive Statement 4-3
Figure 4-2 Example of the Line Syntax 4-4
Figure 4-3 Example of the Type Keyword 4-4
Figure 4-4 Example of the Standard ADSM Conifg File. 4-5
Figure 4-5 Example of an ADSM Mount Message 4-5
Figure 4-6 Example of an ADSM Dismount Message. 4-6
Figure 4-7 Example of Starting the Message Filter 4-7
Figure 5-1 Example of the Syntax to Start ADSM Client 5-5
Figure 6-1 Example of the Init Function 6-3
Figure 6-2 Example of the Term Function 6-4
Figure 6-3 Example of the Reader Function 6-5
Figure 6-4 Example of the Filter Function 6-5

Tables

Table 1-1 Hazard Alert Message 1-5

Table 4-1 Message Types 4-5

Table 4-2 Parameters for Starting the Message Filter 4-7

Table 5-1 Scripts and Executable Associated with the Message Filter 5-3

Table 5-2 Environmental Variables 5-4

Table 5-3 MSGF_MEDIA Mapping 5-4

Table 6-1 Parameters for the Init Function 6-3

1

Introduction

Overview 1-3
Intended Audience 1-3
Organization 1-3
Associated Documents 1-4
Explanation of Symbols and Notes 1-4
Hazard Alert Messages 1-4
Assistance 1-6
Federal Communication Commission Note 1-7

Overview

This guide contains the programming interface for applications using the AML connectivity from Windows NT or Windows 95. The topics discussed in this chapter are:

- Overview
- Intended Audience
- Organization
- Associated Documents
- Explanation of Symbols and Notes
- Assistance

Intended Audience

This guide is intended for Windows NT or Windows 95 programmers and Administrators using the ACI for Windows NT and Windows 95.

Organization

This manual is divided into the following chapters:

Chapter 1	Introduction - Notes on the use of the manual
Chapter 2	Message Filter - Description of the message filter
Chapter 3	Installation and Configuration - Explanation of the tasks necessary for installation and configuration
Chapter 4	Administration - Information on the configuration, start-up and shutdown of the message filter
Chapter 5	File Description - Describes shell scripts and executable association with the message filter
Chapter 6	Message Type Interface Description - Explains different message types

Index

Associated Documents

You may wish to reference the following documents:

- DOC E00 003 AMU Installation Guide
- DOC E00 007 AMU Problem Determination Guide
- DOC E00 005 AMU Reference Guide
- 600817-A DAS Administration Guide

Explanation of Symbols and Notes

The following symbols and highlighted passages draw attention to important information.

Detailed explanations for the above symbols are provided in *Hazard Alert Messages*.

<1>+<2>	Press these keys simultaneously.
Italic	Headline, e.g., Chapter 3, *Safety* File name, e.g., *dasdata.ini*
Bold	Terms appearing on the operating panel Special Term, e.g., Utilities Commands with or without parameters, e.g., **INITIALIZE**
`Courier`	Command appearing on a console, e.g., `cd` Switch position, e.g., `ON`, `OFF`

Hazard Alert Messages

ADIC classifies hazards in several categories. Table 1-1 on page 1-5 shows the relationship of the symbols, signal words, actual hazards, and possible consequences.

Table 1-1 Hazard Alert Message

Symbol	Damage to ...	Signal Word	Definition	Consequence
	Persons	**DANGER**	Imminent hazardous situation	Death or serious injury
		WARNING	Potential hazardous situation	Possible death or serious injury
		CAUTION	Less hazardous situation	Possible minor or moderate injury
	Persons		Imminent hazardous electrical situation	Death or serious injury
	Persons	**Caution**	Less hazardous situation	Possible minor or moderate injury
	Material	**Attention**	Potential damaging situation	Possible damage to the product or environment
	Material	**Static Sensitive**	Potential electronic damaging situation	Possible damage to the product
		Note	Tips for operators	No hazardous or damaging consequences
			Important or useful information	No hazardous or damaging consequences

Specially emphasized paragraphs in this guide warn of danger or draw attention to important information. These paragraphs and their associated symbols include:

When used with the signal words, Danger or Warning, this symbol warns of a dangerous situation that threatens personnel with serious injury or death.
When used with the signal word Caution, the symbol warns of a hazardous situation that could result in minor injury.

The danger exists of a fatal electric shock. At places designated with this symbol, electrical current can be present. Before starting any work, always confirm that all electrical connections are free of electrical current.

Caution

This symbol indicates the presence of a laser.
Caution - use of controls or adjustments or performance of procedures other than those specified herein may result in hazardous radiation exposure.

Attention

This symbol means that specific regulations, rules, notices, and working procedures must be observed. Ignoring this symbol can lead to equipment damage or destruction or to other property damage.

STATIC SENSITIVE

This symbol indicates that the risk of equipment damage exist due to static discharge.

Note

This symbol draws attention to user tips. No dangerous or damaging consequences for personnel or property are associated with this symbol.

This symbol indicates important or useful information. No dangerous or damaging consequences for personnel or property are associated with this symbol.

Assistance

If problems cannot be solved with the aid of this document or if recommended training is desired, contact the ADIC Technical Assistance Center (ATAC).

ADIC
10949 East Peakview Avenue
Englewood, CO 80112
U.S.A.

- United States 1-800-827-3822
- Europe and Africa 00-800-9999-3822

Federal Communication Commission Note

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

To meet FCC emissions limits, properly shielded and grounded cables and connectors must be used. The user accepts responsibility for radio or television interference caused by improperly shielded or grounded cables and connectors or by unauthorized modifications or changes to the equipment. Unauthorized modifications or changes could void the user's authority to operate the equipment.

2

Message Filter

Overview 2-3
Message Filter Description 2-3

Overview

The message filter is an independent component that reads messages for library actions and translates them to ACI calls to the DAS software. The messages can be generated from an application or operating system. These messages are designed to enable a human operator to mount and dismount media from a drive.

Message Filter Description

It is preferred for applications and operating systems to use the ACI calls directly, as this provides complete consistency between the library and the client. Use of the message filter is necessary only in situations where this is not possible. This does not mean that using the message filter is a last resort but that basing actions on messages is not consistent.

Each application or operating system has different messages to indicate library actions. The message types currently supported are:

- ADSM,
- AMASS (single drive only)
- UTS.

As the message filter is started, a message type is selected to be used during the life of the filter. The message filter as an interface between an application (e.g. ADSM) and the API from DAS (ACI). The message filter waits for messages from ADSM, AMASS or UTS. The messages are sent over a pipe (“ | “) to the program msgf. The message filter is split into

- a generic component
- multiple message type component
- shell scripts.

The generic component makes the ACI calls based on token strings. The type components build these token strings based on each message read. Some Shell-Scripts help start and stop the message filter and its message handling. Refer to Figure 2-1 on page 2-4.

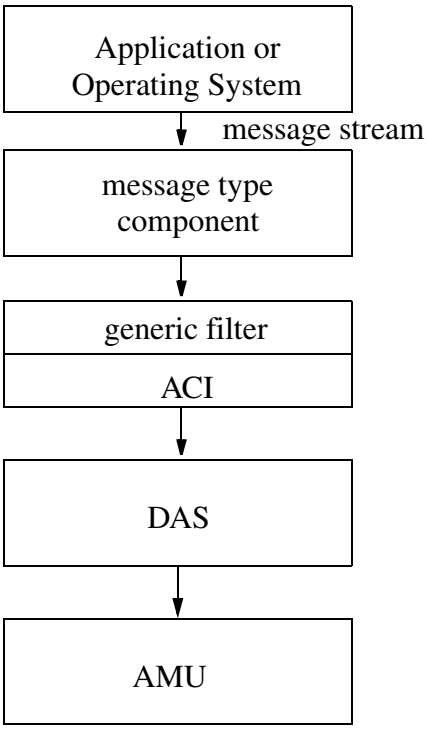

Figure 2-1 Signalflow Trough the Message Filter

3

Installation and Configuration

Overview 3-3

Before Installing the Message Filter 3-3

Install the Message Filter 3-3
Configure the Message Filter 3-4

Overview

This section contains information on installing and configuring the message filter.

Before Installing the Message Filter

Before installing the Message Filter, follow the steps below.

Step 1 Check which drives are supported.

Step 2 Check which application (ADSM, AMASS, UTS) sends the messages to the message filter.

Step 3 Check in which directory the message filter has to be install.

Install the Message Filter

To install the Message Filter, follow the steps below.

Step 1 Choose a parent directory where the message filter can be installed.
The directory that must be chosen dependent on the application that stands above the message filter.

For UNIX-DAS: Normally this would be `usr/local/das` on the DAS-Server. If this directory does not exist, it needs to be created.

For OS/2-DAS: Normally this would be `usr/local/aci` on the Client-Host. If this directory does not exist, it needs to be created.

In the correct directory make a directory with the name `msgf`.

Step 2 Place the distribution media in the drive. In the `msgf` directory use the follow command:
`tar -xvpf [special file of drive]`
For AIX: `tar -x -v -p -f [special file of drive]`

- For example the special file of drive could be
  - `/dev/rst4` for Sun Solaris
  - `dev/rmt/0m` for HP-UX
  - `/dev/rmt0` for AIX

Configure the Message Filter

The configuration depends on the drives which must be supported and on the application that sends messages to the message filter. Refer to *Configuration* on page 4-3.

4

Administration

Overview 4-3

Configuration 4-3
Drive Statement 4-3
Line Statements 4-3

Start-up / Shutdown 4-6
Start Message Filter 4-6
Stop Message Filter 4-7

Overview

This section contains information on the configuration and the start-up and shutdown of the message filter.

Configuration

Each message type depends on the application and has its own config file. There may be more files for one particular type, however only one may be active for the instance of the filter.

The configuration consists of two types of information:

- drive names which will be used in the library
- lines in the message stream where actions are to be taken.

Drive names must be equal to the drive description name of the AMU.

Drive Statement

This statement defines the drives in a library. A subset of the drives can be defined in the library. This is used by the filter to detect which messages to act on. In environments where there are other libraries or manual drives, it avoids attempting ACI calls for drives which do not exist in the library. The drive list is contained within parentheses, each drive name is separated by commas. See Figure 4-1.

```
drives = (TD414, TD415, TD416, TD417)
```

Figure 4-1 Example of a Drive Statement

Line Statements

Each line statement defines a message style which is to perform a single action (i.e. mount or dismount). The line syntax is show in Figure 4-2 on page 4-4.

```
line        type = (position, text, action type),
            volser = position,
            drive = position,
            media = position,
            side = position,
            ifs = "delimiter characters"
```

Figure 4-2 Example of the Line Syntax

The type keyword defines the single word text and its position on the message line. See Figure 4-3.

```
type = (1, ANR8326I, mount_type)
```

Figure 4-3 Example of the Type Keyword

This shows that if the parser finds the text `ANR8326I` at the first position in a message line, then this is a mount message.

The action type values can be:

- mount_type
- dismount_type
- force_type

It is possible to have multiple line statements for the same action type. For example, if these are two messages which indicate a dismount request.

The volser, drive, media and side keywords are all position keywords. That is, they define the relative position of the text where that value is to be found on the line.

In the case where volser = 4 and the line:

ANR8468I 4MM volume DT000001 dismounted from drive DRV1 (/dev/mtb0) in lib LIB4MM

The parser will assign the value `DT000001` to the token volser.

Only those values which are needed from the message line need to be defined on the line statement. Refer to Table 4-2 on page 4-7.

Table 4-1 Message Types

Message	Required Values
mount	• volser • media type (or use default) • drive (optional)
dismount	• volser • media type (or use default)
force	drive

Note
When using the tab character in the `ifs` string, the value must be typed as a `tab` not as \t. The escape sequence is not recognized by the config parser.

The `ifs` keyword indicates the input field separator. The value is a string enclosed in double quotes (`""`). Each character of the `ifs` is a delimiter character in the message stream. One or more of these characters determine the positioning on the line. Normally the space character is a delimiter with a ',' or tab character.

The line statements can be configured and not part of the filter code for the message type. A new version of a product can be quickly adapted if any messages are changed. A new config file could be delivered by ADIC to do this rather than providing new binaries. Another advantage of this is that it is platform independent. See Figure 4-4.

```
drives = (TD414, TD415, TD416, TD417)
```

Figure 4-4 Example of the Standard ADSM Conifg File

The drives statement defines the drives TD414, TD415, TD416 and TD417 as those to be recognized by the filter. Requests for other drives will be ignored.

```
line        type = (1, ANR8326I, mount-type),
            volser = 6,
            media = 4,
            drive = 10,
            ifs = " "
```

Figure 4-5 Example of an ADSM Mount Message

The line statement shows that where the word `ANR8326I` is found in the first position, this is to be treated as a mount message. The following volser, media and drive define the positions of each value in the mount message. The ifs is a space, that is the values on the line are only separated by a space.

Note how the line statement describes the objects needed for a mount request.

ANR8326I 003: Mount 4MM volume DT000007 R/W in drive TD416 (/dev/mtb3) of library LIB4MM within 60 minute(s).

See Figure 4-6.

```
line        type = (1, ANR8468I, dismount-type),
            volser = 4,
            media = 2,
            drive = 8,
            ifs = " "
```

Figure 4-6 Example of an ADSM Dismount Message

This line statement shows that the word `ANR8468I` in the first position indicates a dismount message. The volser, media and drive show the positioning of each value and again the ifs is a space.

Note how the line description matches the message.

ANR8468I 4MM volume DT000007 dismounted from drive TD416 (/dev/mtb3) in library LIB4MM.

Start-up / Shutdown

This section describes starting and stopping the message filter.

Start Message Filter

The filter can be started with the following Refer to Figure 4-7 on page 4-7.

```
msgf -t [adsm | amass | uts]
         [-c config file]
         [-f message stream]
         [-l log file]
         [-d debug level]
```

Figure 4-7 Example of Starting the Message Filter

See Table 4-2 for an explanation of the parameters for starting the message filter.

Table 4-2 Parameters for Starting the Message Filter

Parameter	Type	Explanation
-t [adsm | amass | uts]	mandatory	selects the message type to be used. The message type defined other flags on the command line.
[-c config file]	optional	defined the name of the config file (default: [*message type*]_config, e.g. adsm_config)
[-f message stream]	optional	supply a file where the message will be written to by the application operating system Default output are a standard file, depending on the message type. • ADSM: stdin • AMASS: stdin • UTS: /dev/osm
[-l log file]	optional	The log file contains error meddages and debugging information. (default: {message type}_log e.g. adsm_log
[-d debug level]	optional	Valid values are 0, 1 and 2 (default: 0)

Stop Message Filter

To stop the filter send a TERM or INT signal to the parent msgf process with `kill`(1). If the msgf is running in a window, typing <CTL><C> (intr char) will stop it.

It is possible to run the filter in the background at start-up from an rc local file.

Two scripts are provided in the scripts directory. They are set-up to start and stop the filter for ADSM. However, it is simple to change this for UTS or AMASS.
Refer to *Overview* on page 5-3

5

File Description

Overview 5-3

msgf 5-3

start_msgf 5-3
MSGF_MEDIA Mapping 5-4

stop_msgf 5-5

Overview

This section describes shell scripts and an executable associated with the message filter. See *Scripts and Executable Associated with the Message Filter*Table 5-1.

Table 5-1 Scripts and Executable Associated with the Message Filter

Datei	Type	Location of the File
msgf	C-Program	installation directory
start_msgf	Shell-Script to start the message filter	“scripts” subdirectory
stop_msgf	Shell-Script to stop the message filter	“scripts” subdirectory
kill_children	Shell-Script for internal use	“scripts” subdirectory
*_log[a]	Error Message file	installation directory
*_config[b]	Configuration file	installation directory

a. * indicates the corresponding application
b. see note a

msgf

The msgf program is a compiled C-program that is used to start the message filter. This program reads in the configuration file, specifically hardware statements. After that, the program waits for messages from the application. When a message comes over the standard input, msgf check whether it is a MOUNT message.

start_msgf

This shell script starts and controls the program. The script represents an endless loop. In it the ADSM-, AMASS- or UTS-Client will be started and the output will be sent with a pipe (“ | “) to the message filter.

The `MSGF_HOME` variable defines the place where the msgf program will be found. This variable must be set correctly and is required.

Prior to running msgf, there are two environmental variables and one optional variable which are essential to set. Refer to the *DAS Administration Guide*.

Table 5-2 Environmental Variables

Variable	Type	Explanation
DAS_SERVER	mandatory	TCP/IP host name or ip-address of the AMU-PC (DAS server)
DAS_CLIENT	mandatory	Name of the client for authorization on the DAS server (name must be defined on the DAS server config file together with the TCP/IP name of the client). The Variable is case sensitive.
ACI_MEDIA_TYPE	optional	Media type for use of another default media type in the dasadmin commands (default = 3480)
MSGF_MEDIA	optional	Useful where a media type is needed for a mount or dismount operation but the media type is not provided on the message line. For example, a dismount from UTS only provides a volser, and drive. These values are case insensitive.

MSGF_MEDIA Mapping

For a regular end of the program, get the process number of the shell in the start mechanism. See Table 5-3.

Table 5-3 MSGF_MEDIA Mapping

MSGF_MEDIA	ACI_MEDIA_TYPE	Explanation	AMU-Type
3480, 3490 or CARTRIDGE	3480	3480 and 3490 and 3490E cartridges	C0
3590	3590	3590/8590 cartridges	C2

Table 5-3 MSGF_MEDIA Mapping

MSGF_MEDIA	ACI_MEDIA_TYPE	Explanation	AMU-Type
OD-THIN	OD-THIN	Optical disk Reflection (9 mm)	O0
OD-THICK or OPTICAL	OD-THICK	Optical disk 512, MO/Worm (11 mm)	O1
---	CD	CD-ROM disk (CD-Caddy)	C6
---	TRAVAN	TRAVAN cartridges	V5
---	BETACAM	BETACAM cartridges	V8
DECDLT	DECDLT	TK-85 Digital Linear Tape (DLT)	C1
8MM	8MM	D8 cartridge (8 mm)	V1
4MM	4MM	DDS or DAT cartridge (4 mm) (Digital Data Storage)	V2
VHS	VHS	VHS cartridge	V0
D2	D2	D2 small and medium cartridge	V3 (V4)
---	DTF	DTF small and medium cartridges	V6 (V7)

In the endless loop the syntax to start, for example the ADSM-Client and send the output to the message filter is, for example Figure 5-1.

```
usr/lpp/adsm/bin/dsmadmc [-options ] |
                  $MSGF_HOME/msgf -t adsm [-options]
```

Figure 5-1 Example of the Syntax to Start ADSM Client

To start and correct the client refer to the corresponding reference guide of the application. To correctly start the message filter with its options. Refer to*Start-up / Shutdown* on page 4-6.

stop_msgf

This shell script ends the program normally. For that purpose `stop_msgf` must ends three process.

- ADSM-Client
- AMASS-Client
- UTS-Client
- msgf
- start_msgf.

`Stop_msgf` automatically looks for the process number that must be killed. Make sure that the `MSGF_HOME` variable is set correctly.

6

Message Type Interface Description

Overview 6-3

Init Function 6-3

Term Function 6-4

Reader Function 6-5

Filter Function 6-5

Utility Functions 6-6
- TOKEN *msg_parser (char *line text) 6-6
- TOKEN *find_token (TOKEN *token_chain, enum token_type token) 6-6
- void free_tokens (TOKEN *token_chain) 6-6
- void aci_driver (TOKEN *token_chain) 6-6
- void suspend_on_drive (struct drive_entry *drive, long timeout) 6-6
- void update_drive (struct drive_entry *drive, char *volser) 6-7
- struct drive_entry *find_drive (char *drive_name) 6-7
- enum aci_media convert_media (char *media) 6-7

The Drive Queue 6-7

Overview

For existing and new message types msgf has a standard interface. It primarily consists of four function calls:

- init
- term
- reader
- filter

Each one is to manage a specific part of the message type.

Use the other types of message types examples, if new ones are to be added.

Init Function

The init function is responsible for opening the message stream, config and log files. See Figure 6-1.

```
FILE *
{msg type}_init (int argc, char *argv[])
```

Figure 6-1 Example of the Init Function

See Table 6-1 for an explanation of the parameters for the init function.

Table 6-1 Parameters for the Init Function

Parameter	Explanation
-t	if getopts (3) is to be saved
[-c config file]	defined the name of the config file (default: {*message type*]_config, e.g. adsm_config_

Table 6-1 Parameters for the Init Function

Parameter	Explanation
[-f message stream]	supply a file where the message will be written to by the • application • operating system Default output are standard files, depending on the message type. • ADSM: stdin • AMASS: stdin • UTS: /dev/osm
[-l log file]	The log file contains error messages and debugging information. (default: {*message type*}_log e.g adsm_log
[-d debug level]	defines the debug level for the debug and vdebug values are 0, 1 and 2 (default: 0)

The message stream FILE pointer should be returned to the caller. If there is a problem found in the init, returning NULL terminates the msgf.

Set the FILE *log global value to the opened log file. This value is used in fprintf calls throughout the msgf.

Once the config file is open set the FILE *yyin to the file and call yyparse to parse the config file.

From the -d value the global value short debug_level should be set.

Term Function

The term function is responsible for closing the message stream, config and log files. See Figure 6-2.

```
void
{msg type}_term (FILE *msg_stream)
```

Figure 6-2 Example of the Term Function

Reader Function

The reader function is responsible for reading the message stream, config and log files. See Figure 6-3.

```
void
{msg type}_reader (FILE *msg_stream)
```

Figure 6-3 Example of the Reader Function

Read from the message stream. When one or more lines from the stream make up an action request (e.g. mount or dismount), call the filter function with the line. The filter function expects the text line to be one single null terminated string with the new line characters removed.

Filter Function

Below is a basic description of the of what this function does for the other message type. Each message type will have to accommodate the special features of the application in the filter function. See Figure 6-4.

```
void
{msg type}_filter (char *line)
```

Figure 6-4 Example of the Filter Function

Call the `msg_parser()` function - this tokens the line based on the line statements in the config file.

Check whether the drive token is one in the drive list - with `find_drive()`.

Call `aci_driver()` function with the token chain.

Call `free_tokens()` functions to free the storage used in the token chain.

Utility Functions

The utility functions are there to used as required.

TOKEN *msg_parser (char *line text)

Takes the line and it compares against the line statements in the config file. When it finds a match of the line type (that is the action word matches at the position on the line) it tokens the line. It only creates tokens which are present on the line statement.

It returns this chain of tokens. The first token in the chain is always the REQUEST_TYPE token.

TOKEN *find_token (TOKEN *token_chain, enum token_type token)

This finds the token on the chain, if it exists. If not it returns NULL.

void free_tokens (TOKEN *token_chain)

Frees the memory used in the token chain.

void aci_driver (TOKEN *token_chain)

Makes an ACI call based in the token chain.
The `d_errno` and `d_text` global values will be set after returning from `aci_driver()`. If the request was successful d_errno will be set to `EOK`.

void suspend_on_drive (struct drive_entry *drive, long timeout)

This function waits for the requested drive to become free. If a time-out value is > 0 the function returns after that time whether or not it has the drive. If time-out is set to zero it waits until the drive is freed. A drive is freed by the volser member of the drive entry struct being set to a null string.

void update_drive (struct drive_entry *drive, char *volser)

The update_drive() function updates the drive_entry with the new volser value. The volser can be an actual volser to show the drive is occupied or a null string (“”) to free the drive. To ensure synchronization the drive queue is locked using a semaphore before making the update.

struct drive_entry *find_drive (char *drive_name)

This function scans the drive queue looking for a match on the drive name. If found, it returns a pointer to the drive_entry for the drive. If not, it returns NULL.

enum aci_media convert_media (char *media)

This function converts a media type in text form, for example, 4mm or VHS to its corresponding ACI media value. A new media type or text form of an existing media can be added following the structure of the function.

The Drive Queue

During the generic init function, a shared memory segment is allocated to hold a drive entry queue. The queue is made up of contiguous drive_entry structures. Each drive entry represents one drive defined in the config file. The only values in the structure are drive name and volser. At present, only the UTS message type uses the volser member. It is used as a mechanism to show the drive is occupied, and with which volume, or free.

The queue is pointed to be global value struct drive_entry *drive_area. The queue is terminated by a null entry with the struct member endflag set to TRUE. Review the find_drive() function to see how the queue is scanned.

Index

- A -

Administration 4-3
Assistance 1-6
Associated Documents 1-4

- C -

Configuration 4-3
Configure the Message Filter 3-4

- D -

directory 3-3
dismount type 4-4
Document Organization 1-3
Drive statement 4-3

- E -

enum aci_media convert_media 6-7

- F -

Federal Communication Commission Note . 1-7
Fieles
    msgf 5-3
Files
    Overview 5-3
    start_msgf 5-3
    stop_msgf 5-5
Filter function 6-5
force type 4-4

- H -

Hazard Alert Messages 1-4

- I -

Init function 6-3
Install the Message Filter 3-3
Installation and Configuration 3-3
Installation Overview 3-3
Intended Audience 1-3

- L -

Line Statement 4-3
lines 4-3

- M -

Media Types 5-4
Message type interface description 6-3
mount type 4-4
msgf 5-3
MSGF_MEDIA 5-4

- O -

Overview of Message Filter 2-3

- R -

Reader function 6-5

- S -

Signalflow trough message filter 2-4
Start Message Filter 4-6
start_msgf 5-3
Start-up/Shutdown 4-6
Stop Message Filter 4-7
stop_msgf 5-5

struct drive_entry *find_drive 6-7
Symbols
    Hazard Alert Messages 1-4
symbols
    formats 1-4
    information/note 1-4

- T -

tar 3-3
Term function 6-4
The drive queue 6-7
TOKEN *find_token 6-6
TOKEN *msg_parser 6-6

- U -

Utility function 6-6
Utility functions 6-6

- V -

void aci_driver 6-6
void free_tokens 6-6
void suspend_on_drive 6-6
void update_drive 6-7