adic

The ADIC
Distributed AML Server

DAS V3.10E Interfacing Guide

Advanced Digital Information Corp

ADIC	ADIC Europe	ADIC Germany Beteiligungs GmbH, KG
Tel.: +1 303-705-3900	ZAC des Basses Auges	Eschenstraße 3
Fax: +1-303-792-2465	1, rue Alfred de Vigny	D-89558 Böhmenkirch, Germany
ATAC: 1-800-827-3822	78112 Fourqueux, France	Tel:+00.800.9999.3822
www.adic.com	Tel.: +33.1.3087.5300	
	Fax: +33.1.3087.5301	

Document number: 601626-B
Published: 21 Dec 2001

Printed in the USA

ADIC CORPORATE • 11431 WILLOWS ROAD, NE • REDMOND, WASHINGTON, USA • 1-800-336-1233
ADIC • 8560 UPLAND DRIVE • PARKER, COLORADO, USA • 1-800-827-3822
ADIC • 10 BROWN ROAD • ITHACA, NEW YORK, USA • 1-607-266-4000

Contents

Figures

Tables

1 Introduction

Overview

This guide contains information and instructions necessary to program an application for using the ADIC AML via the Distributed AML Server (DAS). 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 use by system programmers and administrators working with the DAS software. Knowledge of the UNIX and OS/2 operating systems is required.

Organization

This manual is divided into the following chapters:

Chapter 1	Introduction - Notes on the use of the manual
Chapter 2	DAS ACI - Overview of the DAS/2 software and information on AML Client Interface (ACI) services
Chapter 3	Safety - Describes the hazard symbols, messages, safety features, and operational considerations
Chapter 4	DAS ACI Functions - Information and descriptions of ACI functions and function callas
Chapter 5	ACI 3.0 Asynchronous Support Layer - Provides a description of the Asynchronous ACI calls
Appendix A	Important Information - Error recovery procedures and explanations of terms used throughout this document

Index

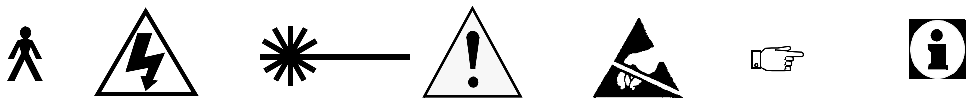
Associated Documents

You may wish to reference the following documents:

- 601324-A DAS V3.1 Release Guide
- 601625-A DAS V3.1 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* on page 3-3.

<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`

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

2

DAS ACI

Overview

This section contains an overview of the DAS/2 software and information on AML Client Interface (ACI) services.

DAS/2

DAS is a client/server software product designed to provide shared access to the family of ADIC AML systems (AMLs). The DAS software may be installed as a stand-alone AML connection or be configured to share an AML with MVS or other ADIC supported, host attachments. DAS may be requesting services via a client command line interface, or may be integrated with backup, tape management and/or HSM applications on the client to direct automated removable media activity through the DAS server to the automated media library. See Figure 2-1.

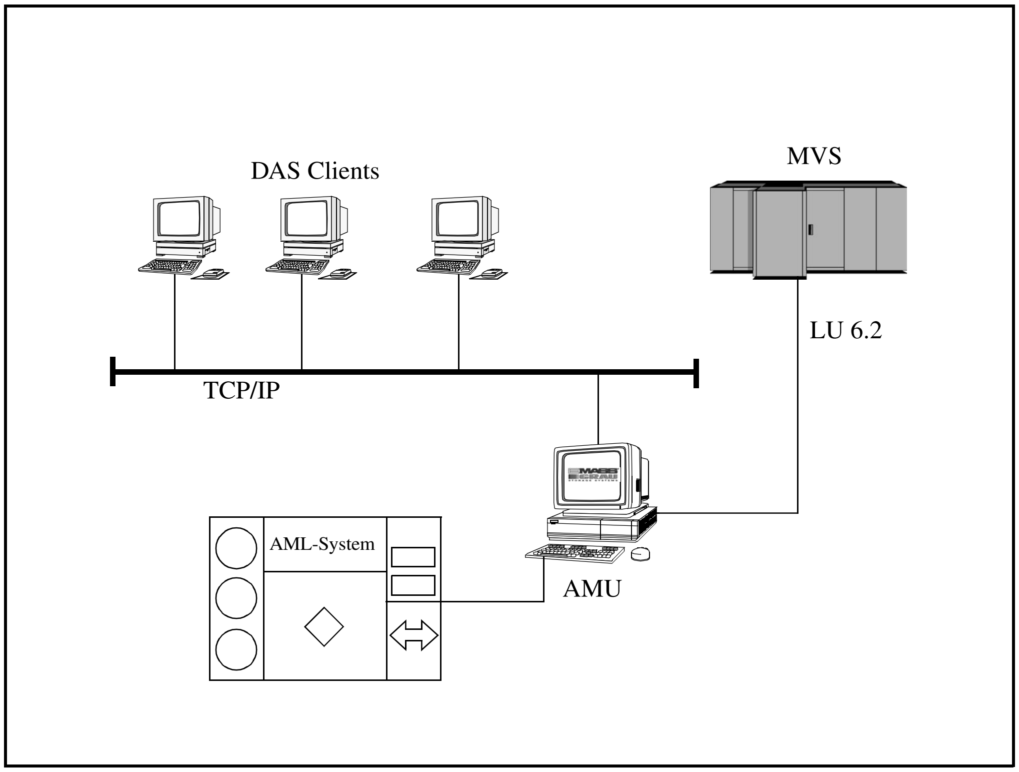


Figure 2-1 Logical Relationship Between AML Components

The DAS server component is an OS/2 program that runs within the AMU controller personal computer (PC). It converts DAS client requests into AMU AMS requests and sends them to the AMS for action. Fifty heterogeneous networked clients can be configured within the DAS server environment.

Clients requiring access to ADIC AML systems may use the DAS ACI software component to communicate library requests to the DAS server component. DAS clients make the necessary calls to the DAS server with remote procedure calls (RPC). The DAS ACI library component hides this interface from the client application and provides function calls to the library to request library operations. Clients may be granted complete or restricted access to the AML resources. DAS administrators control client access and privileges through client registration.

The DAS client ACI component provides a set of C function calls, available as static and/or dynamic libraries (depending on the operating system platform) for a range of operating system platforms. The libraries may be linked to application software, or be used with the command line administration command interface.

ACI Services

Since the DAS server is a network server, all communication with the server component use a network application level protocol. The ACI hides the network support, and enables the user to treat the functions as ‘stand-alone’, without the need for a supporting structure. The ACI operates synchronously. Once a request is made to the AML, the request process does not regain control until the operation has completed or has otherwise terminated. (An exception is the inventory call, which starts a physical inventory, and returns.)

The ACI uses RPCs to request DAS services and to receive replies over the network. The RPC port mapper assures that port numbers are routed correctly.

Client Services

The ACI provides two types of user services:

- Basic Services
- Complete Services

ACI Routines - Basic Services

Table 2-1 lists the routines that are available to an ACI client with basic service access rights.

Table 2-1 Routines Available with Basic Service Access

Routine	Explanation
aci_dismount	Dismount media from the drive.
aci_init	Initialize the AML for client use.
aci_initialize	Initialize ACI library for client use.
aci_mount	Mount specified media in drive.

ACI Routines - Complete Services

Table 2-2 lists the routines that are available to an ACI client with complete service access rights.

Table 2-2 Routines Available with Complete Service Access

Routine	Explanation
aci_barcode	Switching ON and Off Barcode Reading
aci_cancel	Cancel outstanding request.
aci_dismount	Dismount media from drive.
aci_cleandrive	Clean a drive (immediately)
aci_clientaccess	Change client access list.
aci_clientstatus	Query client access list.
aci_driveaccess	Change client drive status.
aci_drivestatus	Query client drive status (for compatibility only).
aci_drivestatus2	Query client drive status with additional information.
aci_drivestatus3	Query physical drive status.
aci_eject	Eject media from AML (for compatibility only).

Table 2-2 Routines Available with Complete Service Access

Routine	Explanation
aci_eject2	Eject media from AML, and keep database entry for future insert requests.
aci_ejectclean	Eject clean media from AML, and remove database entry.
aci_eject2_complete	Eject media from AML, and remove database entry.
aci_flip	Turn Optical Disk on in the drive.
aci_force	Force a dismount request from a specified drive.
aci_foreign	Add or delete foreign media.
aci_getvolsertodrive	Get the volsers, which are allowed to be mounted to a specified drive. (Volsers are configured in the DAS config file)
aci_getvolsertoside	Get the second volser of the Optical Disk (OD has two volser).
aci_init	Initialize AML for client use.
aci_initialize	Initialize ACI library for client use.
aci_insert	Insert media into AML.
aci_insert2	Insert media into AML (also for clean- and scratch media).
aci_inventory	Start physical inventory with AMU database update.
aci_list	List outstanding and currently operating DAS requests.
aci_mount	Mount selected media in drive.
aci_partial_inventory	Start a physical inventory of subsystem in the AML
aci_qversion	Query version of DAS and ACI.
aci_qvolsrange	List client accessible and physically present volsers within requested range.
aci_register	Temporarily allow client access to DAS.

Table 2-2 Routines Available with Complete Service Access

Routine	Explanation
aci_robhome	Set the AML-System to offline and move the robot to home position.
aci_robstat	Get Information about the AML-status and set the AML to on-line.
aci_scratch_get	Mount a scratch volume.
aci_scratch_info	Query scratch volume information.
aci_scratch_set	Add volume to scratch pool.
aci_scratch_unset	Remove volume from scratch pool.
aci_killamu	Shutdown AMU complete with OS/2.
aci_shutdown	Shut down DAS.
aci_switch	Switch between active and inactive AMU (DAS and AMS).
aci_unload	On an AML drive pressed one or more drive buttons (e.g. unload button).
aci_view	Query volume database entry.
aci_volseraccess	Set ownership of volser.
aci_volserstatus	Query ownership for volser.

Media Types

The DAS ACI supports a variety of media types. The media type is passed as a parameter to all ACI functions that require media operations. Each media type name is a member of an enumerated type *aci_media*, defined in the *aci.h* header file.

Table 2-3 lists the media types supported by DAS.

Table 2-3 Supported Media Types

ACI Media Name	Media Type
ACI_3480	3480/3490 cartridges
ACI_3590	3590/8590 cartridges (NTP)
ACI_4MM	DDS or DAT (4mm tape)

Table 2-3 Supported Media Types

ACI Media Name	Media Type
ACI_8MM	DDS 8mm tape (e.g. EXABYTE)
ACI_AUDIO_TAPE	standard audio tape cartridges
ACI_BETACAM	SONY BetaCAM cartridge
ACI_BETACAML	large BetaCAM cartridge
ACI_CD	CD-ROM (with CADDY)
ACI_D2	D2 (small and medium) tape cartridge
ACI_DECDLT	DLT (CompacTape) cartridge
ACI_DTF	SONY DTF cartridge
ACI_OD_THIN	Reflection optical disks
ACI_OD_THICK	512 MO/WORM optical disks
ACI_TRAVAN	TRAVAN cartridge
ACI_VHS	VHS cartridge
ACI_SONY_AIT	Sony AIT
ACI_LTO	LTO[a]

a. only IBM LTO media type is supported.

DAS Error Codes

The DAS ACI functions return either a successful or failed return code. In case of failure, a DAS error code *d_errno* is set describing the failure. The DAS error numbers are defined in the *derrno.h* header file, which is included by the *aci.h* header file. Refer to *Error Recovery Procedures* on page A-3.

3

Safety

Overview

Note
In addition to the safety instructions in this guide, local and professional safety rules apply.

Knowledge and observance of these instructions is imperative for the safe operation of the ADIC Storage Systems AML system.

Avoid danger when maintaining and operating the machine by

- behaving in a safety-conscious manner
- acting judiciously

Hazard Alert Messages

ADIC classifies hazards in several categories. Table 3-1 shows the relationship of the symbols, signal words, actual hazards, and possible consequences.

Table 3-1 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

Table 3-1 Hazard Alert Message

Symbol	Damage to ...	Signal Word	Definition	Consequence
⚠	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.

Validity

These instruction are valid for ADIC Storage Systems AML systems.

Supplementary safety provisions for any components used on the machine are not invalidated by these instructions.

Any other manufacturer's documentation forms part of the AML documentation.

4 DAS ACI Functions

Overview

All ACI function calls and ACI structures are defined in the *aci.h* header file. ACI functions return `0` or `-1` for successful and unsuccessful command execution respectively. A command failure sets the DAS error code variable *d_errno* to the specific error code. In such case, a text error message may be written to standard error by calling aci_perror, which accepts a user defined message string and attaches it to the DAS error message.

aci_barcode

The aci_barcode function switches the barcode reader for the volser on the robot on or off. See Figure 4-1.

```
#include "aci.h"
int aci_barcode(     char *cRobNum,
                     char *Action)
```

Figure 4-1 aci_barcode Function Call

See Table 4-1 for a description of the parameters for the aci_barcode function call.

Table 4-1 Parameters for the aci_barcode Function Call

Parameter	Description	
cRobNum	defined the number of the robot (only on AML/2 systems with 2 robots)	
	1	robot 1 (AML/E, AML/J and robot 1 of AML/2)
	2	robot 2 of AML/2
Action	new condition for the following mount and eject commands from this host	
	OFF	barcode on the cartridge will not be checked
	ON	barcode on the cartridge will be checked on each command

The Scalar 1000 does not support the aci_barcode command (barcode on Scalar 1000 will never read on mount and eject).

Use this command to switch the barcode reading after the command aci_mount, aci_cleandrive or any aci_eject ended with failure and derrno=EBARCODE. After aci_barcode completed, try the previous command again.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable *d_errno* is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EBADCLIENT`
- `EDASINT`
- `ETIMEOUT`
- `EAMUCOMM`
- `ESWITCHINPROG`
- `EDASINT`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`

Refer to Figure 4-2 on page 4-7 for an example of the aci_barcode function.

```
/* Switch the barcode reading off for robot 1 */
int rc = 0;
char *cRobNum = "1";
char *Action = "OFF";
rc = aci_barcode( cRobNum, Action );
if( rc )
{
         aci_perror( "Command failed: " );
}
else
{
         printf( "barcode reading switched off \n" );}
```

Figure 4-2 Example of the aci_barcode Function

aci_cancel

The aci_cancel function cancels a specific DAS request. See Figure 4-3.

```
#include "aci.h"
int aci_cancel( unsigned long request_id )
```

Figure 4-3 aci_cancel Function Call

See Table 4-2 for a description of the parameter for the aci_cancel function call.

Table 4-2 Parameter for the aci_cancel Function Call

Parameter	Description
request_id	DAS command sequence number, get information on the sequence number with the aci_list function

The aci_cancel function cancels a previously issued and not completed client request. Before using this function, use the aci_list function to get the *request_id* in order to cancel the correct request.

The cancel request cancels the command in the DAS server and the AML. If the request is being acted upon in the AML, DAS attempts to cancel it, however this is unlikely to succeed. Once the robotics are in motion, the request may not be canceled. DAS reports that the cancel was successful while the canceled request completes.

For additional information, Refer to *aci_list* on page 4-84.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable *d_errno* is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOREQ`
- `EDASINT`
- `ERETRYL`
- `ECANCELED`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`

See Figure 4-4 for an example of the aci_cancel function.

```
/* Cancel first client request for a client */
int rc = 0;
char *pszthis_client = "clientname";
struct req_entry *prequest = NULL;
rc = aci_list( pszthis_client, prequest );
if( !rc )
{
         rc = aci_cancel( prequest[0].request_id );
         if( rc )
         {
            aci_perror( "Command failed:" );
         }
}
```

Figure 4-4 Example of the aci_cancel Function

aci_cleandrive

The aci_cleandrive function mounts a cleaning cartridge to a specific drive. See Figure 4-5.

```
#include "aci.h"
int aci_cleandrive( char *drive)
```

Figure 4-5 aci_cleandrive Function Call

See Table 4-3 for a description of the parameter for the aci_cleandrive function call.

Table 4-3 Parameter for the aci_cleandrive Function Call

Parameter	Description
drive	name of the drive to be cleaned

Only clean the drives when they need to be cleaned. Unnecessary cleaning damages the drives.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable *d_errno* is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `ENODRIVE`
- `EDRVOCUPPIED`
- `EPROBVOL`
- `EAMU`
- `EROBOTCOMM`
- `EDASINT`
- `EDEVEMPTY`
- `ENOTAUTH`
- `EBADCLIENT`

- `ERERTRYL`
- `EINUSE`
- `ECANCELLED`
- `EDASINT`
- `ECLEANING`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `EBARCODE`
- `EINVALIDDEV`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `ENOPOOL`
- `EPROBDEV`
- `EBARCODE`
- `EAREAFULL`

aci_clientaccess

The aci_clientaccess function modifies the access lists of a client. See Figure 4-6.

```
#include "aci.h"
int aci_clientaccess( char *clientname,
                  enum aci_command action,
                  char *volser_range,
                  enum aci_media type,
                  char *drive_range )
```

Figure 4-6 aci_clientaccess Function Call

See Table 4-4 for a description of the parameters for the aci_clientaccess function call.

Table 4-4 Parameters for the aci_clientaccess Function Call

Parameter	Description	
clientname	name of the client which authorization is to be changed	
action	type of the activity (add or remove access rights)	
	ACI_ADD	add access rights
	ACI_DELETE	remove access rights
volser_range	• a single volser • multiple volsers, separated by commas • a range of volsers separated by a hyphen • set to empty ("" or '\0') if it is not to be changed	
type	media type of the previously defined volser range Refer to *Media Types* on page 2-7.	
drive_range	• a single drive • multiple drives, separated by commas • a range of drives separated by a hyphen, or set to empty ("" or '\0') if it is not to be changed	

This ACI function changes the access lists of a client named *clientname* by either adding or removing access to a *drive_range* and/or a *volser_range*. The aci_clientstatus function may be needed to find out the *drive_range* and *volser_range* used by the client.

The changes will be lost when the DAS software is shut down. Only use this command if, at the time, you do not have access to the *config* configuration file, or you cannot restart DAS. Otherwise, change the access privileges in the *config* file.

This ACI command allows the administrator to add new volume ranges or drive ranges to the AML system without shutting down the DAS server. The modifications created by using the aci_clientaccess function are only valid while DAS is running. When DAS is shut down, all access modifications set by aci_clientaccess are lost. When the DAS server is restarted, client access returns to the default settings in the DAS configuration file. To permanently register a client, the administrator must add the new *drive_range* or *volser_range* to the DAS configuration file.

For additional information, refer to *aci_clientaccess* on page 4-11.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable *d_errno* is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `EBADHOST`
- `ENOTAUTH`
- `EBADCLIENT`
- `ENOSPACE`
- `ENOTFOUND`
- `ETIMEOUT`
- `ESWITCHINPROG`

Refer to Figure 4-7 on page 4-13 for an example of the aci_clientaccess function.

```
/* Add a volser range to a clients access list */
int rc = 0;
enum aci_command action = ACI_ADD; /*default action*/
char *client = "SomeClient";
char *volser = "AAB000 - AAB999";
char *drive = "";
enum aci_media type = ACI_3590;
if ((rc = aci_clientaccess ( client,
                        action,
                        volser,
                        type,
                        drive ) ))
{
         aci_perror( "Command failed: " );
}
else
{
         if (*volser)
         {
         printf ( "Volser range %s for client %s
added.\n",
               volser, client );
         }
         if (*drive)
         {
         printf ( "Drive range %s added for client
%s.\n",
               drive, client );
         }
}
```

Figure 4-7 Example of the aci_clientaccess Function

aci_clientstatus

The aci_clientstatus function queries client access list configuration. See Figure 4-8.

```
#include "aci.h"
int aci_clientstatus( char *clientname,
            struct aci_client_entry *client )
```

Figure 4-8 aci_clientstatus Function Call

Query the current client access configuration status of a client named *clientname*. All configuration information is returned in the structure *aci_client_entry*. See Figure 4-9.

For additional information, refer to *aci_clientaccess* on page 4-11.

```
struct aci_client_entry{
         char clientname[ACI_NAME_LEN];
         struct in_addr ip_addr;
         boolean avc;
         boolean complete_access;
         boolean dismount;
         char
volser_range[ACI_MAX_RANGES][ACI_RANGE_LEN];
         char drive_range[ACI_RANGE_LEN];
};
```

Figure 4-9 Returned Configuration Information

See Table 4-5 for a description of the parameters for the aci_clientstatus function call.

Table 4-5 Parameters for the aci_clientstatus Function Call

Parameter	Description	
clientname	name of the client which authorization is to be changed	
aci_client_entry	returned information to the requested client	
	clientname	requested client name
	in_addr	32-bit dotted decimal noted Internet Protocol (IP) address
	avc	(true or false) Parameter “avoid volume contention” defined the reaction during a mount of an already mounted cartridge
	complete_access	(true or false) Parameter authorized complete or basic command set (Refer to *Client Services* on page 2-4)
	dismount	(true or false) Parameter controls the automatic dismount if a next mount occur to an occupied drive
	volser_range	for the client configured volser range
	drive_range	for the client defined drives

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable *d_errno* is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOTAUTH`
- `EUPELSE`
- `EBADCLIENT`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EDASINT`

Refer to Figure 4-10 on page 4-16 for an example of the aci_clientstatus function.

```
/* Query some clients access list configuration */
int rc = 0;
char *client = "SomeClient";
struct aci_client_entry client_entry;
rc = aci_clientstatus ( client, &client_entry );
if( rc )
{
        aci_perror( "Command failed: " );
}
```

Figure 4-10 Example of the aci_clientstatus Function

aci_dismount

The aci_dismount function dismounts a volume. See Figure 4-11.

```
#include "aci.h"
int aci_dismount( char *volser,
            enum aci_media type )
```

Figure 4-11 aci_dismount Function Call

Dismount the volume *volser* of defined media type from its drive. The drive is identified by the DAS software.

For additional information, Refer to *aci_mount* on page 4-91.

Retries can be configured in the config file or in the AMS configuration.

See Table 4-6 for a description of the parameters for the aci_dismount function call.

Table 4-6 Parameters for the aci_dismount Function Call

Parameter	Description
volser	volser that is to be moved from a drive to the original position in the AML
type	media type of the named volser Refer to *Media Types* on page 2-7

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable *d_errno* is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `ENODRIVE`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `EDEVEMPTY`
- `ENOTAUTH`
- `EBADCLIENT`
- `ENOTMOUNTED`
- `ECANCELED`
- `EDASINT`
- `ENOMATCH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCODE`
- `EINVALIDDEV`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `EPROBDEV`
- `EBARCODE`
- `EAREAFULL`

Refer to Figure 4-12 on page 4-18 for an example of the aci_dismount function.

```
/* Dismount volume from drive */
int rc = 0;
enum aci_media type = ACI_3590;
char *volser = "AAB001";
if (( rc = aci_dismount( volser, type ) ))
{
         aci_perror( "Dismount command failed: " );
}
else
{
         printf( "Dismount of %s successful.\n", volser
);
}
```

Figure 4-12 Example of the aci_dismount Function

aci_driveaccess

The aci_driveaccess function modifies allocation status of a drive. See Figure 4-13.

```
#include "aci.h"
int aci_driveaccess( char *clientname,
               char *drive,
               enum aci_drive_status status );
```

Figure 4-13 aci_driveaccess Function Call

Modify allocation status of a drive for a specified client.

For additional information, Refer to *aci_drivestatus* on page 4-21.

See Table 4-7 for a description of the parameters for the aci_driveaccess function Call.

Table 4-7 Parameters for the aci_driveaccess Function Call

Parameter	Description	
clientname	client that has allocated the drive or wants to allocate the drive. Using SHARED_ACCESS as a key word for the client name when requesting a drive reservation causes that drive to be shared with other clients. All clients which are configured for that drive can access it.	
drive	one of the device names defined in the DAS configuration file for the specific client	
status	ACI_DRIVE_UP	normal reservation of the drive (other clients can change the reservation back with: ACI_DRIVE_DOWN, if the drive is empty
	ACI_DRIVE_DOWN	deleted reservation normal (only possible with an empty drive)
	ACI_DRIVE_FUP	force drive allocation (also possible, if the drive is occupied)
	ACI_DRIVE_FDOWN	force delete drive allocation (also possible, if the drive is occupied)
	ACI_DRIVE_EXUP	exclusive reservation of a drive, can only changed from the client self or the client named DAS-SUPERVISOR
	ACI_DRIVE_DOWN	deleted reservation normal (only possible with an empty drive)

The drive can only be put in the DOWN status by ACI_DRIVE_FDOWN if the drive is occupied.

A drive may only be available to a single client at a time. The drive status is defined to be either *UP* (active) or *DOWN* (inactive) to requesting clients. When the client sets the status to ACI_UP, it is exclusively available to that client. If another client already has the drive status set to `ACI_UP`, the request is returned with the *d_errno* set to EUPELSE. If the client indicates a status of `ACI_DOWN`, the drive is unavailable to the client requesting drive access.

A drive must be empty to modify the drive access parameter to `ACI_DOWN`.

Return Values

The aci_driveaccess returns the following values:

- 0: The call was successful.
- -1: The call failed.

The external variable *d_errno* is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENODRIVE`
- `EDRVOCCUPIED`
- `ENOTHAUTH`
- `EUPELSE`
- `EBADCLIENT`
- `ENOTAUTH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EEXUP`
- `EDASINT`

See Figure 4-14 for an example of the aci_driveaccess function.

```
/* Allocate a drive for client use */
int rc = 0;
char *client = "SomeClient";
char *drive = "Drive1";
enum aci_drive_status status;
status = ACI_DRIVE_UP;
if (( rc = aci_driveaccess( client, drive, status ) ))
{
         aci_perror( "Drive allocation failed: " );
}
else
{
         printf( "Allocation of %s for %s successful\n",
            drive, client );
}
```

Figure 4-14 Example of the aci_driveaccess Function

aci_drivestatus

The aci_drivestatus function queries status of up to 15 drives. See Figure 4-15.

```
#include "aci.h"
int aci_drivestatus( char *clientname,
         struct aci_drive_entry
*pstDriveEntry[ACI_MAX_DRIVE_ENTRIES])
```

Figure 4-15 aci_drivestatus Function Call

Return the status of drives which are set to *UP* (active) for the client with name clientname. If clientname is the NULL string, return status on all drives. The status is returned in *drive_entry*, an array of pointers to *aci_drive_entry* structures. See Figure 4-16.

For additional information, refer to
aci_driveaccess on page 4-18, and
aci_drivestatus on page 4-21.

```
struct aci_drive_entry {
         char drive_name[ACI_DRIVE_LEN];
         char amu_drive_name[ACI_AMU_DRIVE_LEN];
         enum aci_drive_status drive_state;
         char type;
         char system_id[ACI_NAME_LEN];
         char clientname[ACI_NAME_LEN];
         char volser [ACI_VOLSER_LEN];
         bool_t cleaning;
         short clean_count;
};
```

Figure 4-16 Returned Status

For an explanation of the aci_drive_entry function, refer to *aci_drivestatus* on page 4-21

The maximum number of drives displayed is 15.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable d_errno is set to one of the following DAS error codes:

- ERPC
- EINVALID
- ETIMEOUT
- ESWITCHINPROG
- EBADCLIENT

aci_drivestatus2

The aci_drivestatus2 function queries status of up to 250 drives. See Figure 4-17.

```
#include "aci.h"
int aci_drivestatus2( char *clientname,
struct aci_drive_entry *pstDriveEntry[ACI_MAX_DRIVE_ENTRIES2])
```

Figure 4-17 aci_drivestatus2 Function Call

Return the status of drives which are set to UP (active) for the client with name *clientname*. The status is returned in drive_entry, an array of pointers to aci_drive_entry structures. See Figure 4-18 on page 4-23.

For additional information, Refer to *aci_driveaccess* on page 4-18.

```
struct aci_drive_entry {
         char drive_name[ACI_DRIVE_LEN];
         char amu_drive_name[ACI_AMU_DRIVE_LEN];
         enum aci_drive_status drive_state;
         char type;
         char system_id[ACI_NAME_LEN];
         char clientname[ACI_NAME_LEN];
         char volser [ACI_VOLSER_LEN];
         bool_t cleaning;
         short clean_count;
};
```

Figure 4-18 Returned Status

See Table 4-8 for a description of the parameters for the aci_drivestatus2 function call.

Table 4-8 Parameters for the aci_drivestatus2 Function Call

Parameter	Description
clientname	name of the client that requested the status of the drives. If *clientname* is the NULL string, return status on all drives. Using SHARED_ACCESS as a key word for the client shows all drives which are allocated in the SHARED_ACCESS mode. Using EXUP as a key word for the clientname shows all drives which are allocated in the EXUP mode.

Table 4-8 Parameters for the aci_drivestatus2 Function Call

Parameter	Description	
aci_drive_entry	returned information about the status of the drives	
	drive_name	name of the drive (name used in DAS and AMS description)
	amu_drive_name	internal AMS drive name e.g. 03 or ZZ
	drive_state	UP or DOWN reservation of the drive Refer to *aci_driveaccess* on page 4-18
	type	type of the drive (internal AMS code, e.g. E for DLT drive) (See the *AMU Reference Guide*)
	system_id	empty, reserved for further use
	clientname	name of the client that the drive is presently allocated to
	volser	Volser, if the drive is currently occupied
	cleaning	true if the drive is presently occupied with a medium for cleaning
	clean_count	number of mounts until the next clean activity

The maximum number of drives displayed is 250.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable *d_errno* is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EBADCLIENT`

Refer to Figure 4-19 on page 4-26 for an example of the aci_drivestatus2 function.

```
/* Get drive status information for some client */
int rc = 0;
int i = 0;
char *client = "SomeClient";
struct aci_drive_entry
*drive_entry[ACI_MAX_DRIVE_ENTRIES2];
if       (( rc = aci_drivestatus2( client, drive_entry
) ))
{
         aci_perror( "listd failed" );
}
else
{
         printf("Drive status request for client:
               %s successful\n",client );
for (i = 0; i < ACI_MAX_DRIVE_ENTRIES2; i++)
         {
         if ( *drive_entry[i]->drive_name == '\0' )
            break;
         printf( "drive:%s amu drive:%s st:%s type:%c"
            "sysid:%s client:%s volser:%s cleaning:%d"
            "clean_count: %d\n",
            drive_entry[i]->drive_name,
            drive_entry[i]->amu_drive_name,
            drive_entry[i]->drive_state ==
                     ACI_DRIVE_UP ? "UP" : "DOWN",
            drive_entry[i]->type,
            drive_entry[i]->system_id,
            drive_entry[i]->clientname,
            drive_entry[i]->volser,
            drive_entry[i]->cleaning,
            drive_entry[i]->clean_count );
         }
}
```

Figure 4-19 Example of the aci_drivestatus2 Function

aci_drivestatus3

The aci_drivestatus3 function queries the physical status of up to 250 drives. See Figure 4-20.

```
#include "aci.h"
init aci_drivestatus3 ( char *clientname,
         struct aci_ext_drive_entry *pstDriveEntry[])
```

Figure 4-20 aci_drivestatus3 Function Call

Return the status of drives which are set to UP for the client with the name clientname. If clientname is the NULL string, the call returns status on all drives. The status is returned in an array of pointers to aci_ext_drive_entry structure. The array element can be maximal 250. See Figure 4-21.

```
struct aci_ext_drive_entry {
         char drive_name[ACI_DRIVE_LEN];
         char amu_drive_name[ACI_AMU_DRIVE_LEN];
         enum aci_drive_status drive_state;
         char type;
         char system_id[ACI_NAME_LEN];
         char volser[ACI_VOLSER_LEN];
         bool_t cleaning
         short clean_count
         int mount
         int keep
};
```

Figure 4-21 Returned Status

See Table 4-9 for an explanation of the parameters used for the aci_drivestatus3 function.

Table 4-9 Parameters for the aci_drivestatus3 Function Call

Parameter	Description
clientname	name of the client that requested the status of the drives. If *clientname* is the NULL string, return status on all drives. Using SHARED_ACCESS as a key word for the client shows all drives which are allocated in the SHARED_ACCESS mode. Using EXUP as a key word for the clientname shows all drives which are allocated in the EXUP mode.

Table 4-9 Parameters for the aci_drivestatus3 Function Call

Parameter	Description	
aci_drive_entry	returned information about the status of the drives	
	drive_name	name of the drive (name used in DAS and AMS description)
	amu_drive_name	internal AMS drive name e.g. 03 or ZZ
	drive_state	UP or DOWN reservation of the drive Refer to *aci_driveaccess* on page 4-18
	type	type of the drive (internal AMS code, e.g. E for DLT drive) (See the *AMU Reference Guide*)
	system_id	empty, reserved for further use
	clientname	name of the client that the drive is presently allocated to
	volser	Volser, if the drive is currently occupied
	mount	• drive logically occupied but the mount is physically not finished (mount=1, keep=0) • drive logically occupied and mount is physically finished (mount=0, keep=0)
	keep	• drive logically empty but the keep is physically not finished (mount=0, keep=1) • drive logically empty and the keep is physically finished (mount=0, keep=0)
	cleaning	true if the drive is presently occupied with a medium for cleaning
	clean_count	number of mounts until the next clean activity

Return Values

- 0: The call was successful
- -1: The call has failed

The external variable *d_errno* is set to one of the following DAS error codes:

- `EBADCLIENT`
- `ERPC`
- `EINVALID`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROGRESS`
- `EDASINT`

aci_drivestatus4

The aci_drivestatus4 function queries the physical status of up to 380 drives. See Figure 4-22.

```
#include "aci.h"
init aci_drivestatus4 ( char *clientname,
         char *drive
         struct aci_ext_drive_entry4 *pstDriveEntry[ACI_MAX_DRIV_ENTRIES4])
```

Figure 4-22 aci_drivestatus4 Function Call

Return the status of drives which are set to UP for the client with the name *clientname*. If *clientname* is the NULL string, the call returns status on all drives. If a value for the drive field is indicated, the information relates to that single drive. The status is returned in an array of pointers to the aci_ext_drive_entry structure. The array element can be maximal 380. See Figure 4-23.

The variable ACI_MAX_DRIVE_ENTRIES4 is equal to 380.

```
struct aci_ext_drive_entry4 {
         char drive_name[ACI_DRIVE_LEN];
         char amu_drive_name[ACI_AMU_DRIVE_LEN];
         enum aci_drive_status drive_state;
         char type;
         char system_id[ACI_NAME_LEN];
         char clientname[ACI_NAME_LEN];
         char volser[ACI_VOLSER_LEN];
         bool_t cleaning;
         short clean_count;
         int mount;
         int keep;
         char serial_number[ACI_SERIAL_NUMBER_LEN];
};
```

Figure 4-23 Returned Status

The variable ACI_SERIAL_NUMBER_LEN is equal to 51.

See Table 4-10 for an explanation of the parameters used for the aci_drivestatus4 function.

Table 4-10 Parameters for the aci_drivestatus4 Function Call

Parameter	Description	
clientname	name of the client that requested the status of the drives. If *clientname* is the NULL string, return status on all drives. Using SHARED_ACCESS as a key word for the client shows all drives which are allocated in the SHARED_ACCESS mode. Using EXUP as a key word for the clientname shows all drives which are allocated in the EXUP mode.	
drive	value associated with a single drive	
aci_drive_entry	returned information about the status of the drives	
	drive_name	name of the drive (name used in DAS and AMS description)
	amu_drive_name	internal AMS drive name e.g. 03 or ZZ

Table 4-10 Parameters for the aci_drivestatus4 Function Call

Parameter	Description	
	drive_state	UP or DOWN reservation of the drive Refer to *aci_driveaccess* on page 4-18
	type	type of the drive (internal AMS code, e.g. E for DLT drive) (See the *AMU Reference Guide*)
	system_id	empty, reserved for further use
	clientname	name of the client that the drive is presently allocated to
	volser	Volser, if the drive is currently occupied
	cleaning	true if the drive is presently occupied with a medium for cleaning
	clean_count	number of mounts until the next clean activity
	mount	• drive logically occupied but the mount is physically not finished (mount=1, keep=0) • drive logically occupied and mount is physically finished (mount=0, keep=0)
	keep	• drive logically empty but the keep is physically not finished (mount=0, keep=1) • drive logically empty and the keep is physically finished (mount=0, keep=0)
	serial_number	the serial number of the selected drive

Return Values

- 0: The call was successful
- -1: The call has failed

The external variable *d_errno* is set to one of the following DAS error codes:

- ERPC
- EINVALID
- ETIMEOUT
- ESWITCHINPROGRESS
- EBADCLIENT
- ENOTSUPPHCMD

aci_drivestatus_one

The aci_drivestatus_one function queries the physical status of up to 15 drives. See Figure 4-24.

```
#include "aci.h"
init aci_drivestatus_one ( char *clientname,
         char *drive
         struct aci_drive_entry *pstDriveEntry[ACI_MAX_DRIVE_ENTRIES])
```

Figure 4-24 aci_drivestatus_one Function Call

Return the status of drives which are set to UP for the client with the name clientname. If clientname is the NULL string, the call returns status on all drives. If a value for the drive field is indicated, the information relates to that single drive. The status is returned in an array of pointers to aci_drive_entry structure. The array element can be maximal 15. See Figure 4-25.

The value of ACI_MAX_DRIVES_ENTRIES is equal to 15.

```
struct aci_drive_entry {
         char drive_name[ACI_DRIVE_LEN];
         char amu_drive_name[ACI_AMU_DRIVE_LEN];
         enum aci_drive_status drive_state;
         char type;
         char system_id[ACI_NAME_LEN];
         char clientname
         char volser[ACI_VOLSER_LEN];
         bool_t cleaning
         short clean_count
};
```

Figure 4-25 Returned Status

See Table 4-11 for an explanation of the parameters used for the aci_drivestatus_one function.

Table 4-11 Parameters for the aci_drivestatus_one Function Call

Parameter	Description	
clientname	name of the client that requested the status of the drives. If *clientname* is the NULL string, return status on all drives.	
aci_drive_entry	returned information about the status of the drives	
	drive_name	name of the drive (name used in DAS and AMS description)
	amu_drive_name	internal AMS drive name e.g. 03 or ZZ
	drive_state	UP or DOWN reservation of the drive Refer to *aci_driveaccess* on page 4-18
	type	type of the drive (internal AMS code, e.g. E for DLT drive) (See the *AMU Reference Guide*)
	system_id	empty, reserved for further use
	clientname	name of the client that the drive is presently allocated to
	volser	Volser, if the drive is currently occupied
	cleaning	true if the drive is presently occupied with a medium for cleaning
	clean_count	number of mounts until the next clean activity

Return Values

- 0: The call was successful
- -1: The call has failed

The external variable *d_errno* is set to one of the following DAS error codes:

- ERPC
- EINVALID
- ETIMEOUT
- ESWITCHINPROGRESS
- EBADCLIENT
- ENOTSUPPHCMD

aci_drivestatus2_one

The aci_drivestatus2_one function queries the physical status of up to 250 drives. See Figure 4-26.

```
#include "aci.h"
init aci_drivestatus2_one ( char *clientname,
         char *drive
         struct aci_drive_entry *pstDriveEntry[ACI_MAX_DRIVE_ENTRIES2])
```

Figure 4-26 aci_drivestatus_2_one Function Call

Return the status of drives which are set to UP for the client with the name clientname. If clientname is the NULL string, the call returns status on all drives. If a value for the drive field is indicated, the information relates to that single drive. The status is returned in an array of pointers to aci_drive_entry structure. The array element can be maximal 250. See Figure 4-27.

The value of ACI_MAX_DRIVES_ENTRIES2 is equal to 250.

```
struct aci_drive_entry {
         char drive_name[ACI_DRIVE_LEN];
         char amu_drive_name[ACI_AMU_DRIVE_LEN];
         enum aci_drive_status drive_state;
         char type;
         char system_id[ACI_NAME_LEN];
         char clientname
         char volser[ACI_VOLSER_LEN];
         bool_t cleaning
         short clean_count
};
```

Figure 4-27 Returned Status

See Table 4-12 for an explanation of the parameters used for the aci_drivestatus2_one function.

Table 4-12 Parameters for the aci_drivestatus2_one Function Call

Parameter	Description	
clientname	name of the client that requested the status of the drives. If *clientname* is the NULL string, return status on all drives.	
aci_drive_entry	returned information about the status of the drives	
	drive_name	name of the drive (name used in DAS and AMS description)
	amu_drive_name	internal AMS drive name e.g. 03 or ZZ
	drive_state	UP or DOWN reservation of the drive Refer to *aci_driveaccess* on page 4-18
	type	type of the drive (internal AMS code, e.g. E for DLT drive) (See the *AMU Reference Guide*)
	system_id	empty, reserved for further use
	clientname	name of the client that the drive is presently allocated to
	volser	Volser, if the drive is currently occupied
	cleaning	true if the drive is presently occupied with a medium for cleaning
	clean_count	number of mounts until the next clean activity

Return Values

- 0: The call was successful
- -1: The call has failed

The external variable *d_errno* is set to one of the following DAS error codes:

- ERPC
- EINVALID
- ETIMEOUT
- ESWITCHINPROGRESS
- EBADCLIENT
- ENOTSUPPHCMD

aci_drivestatus3_one

The aci_drivestatus3_one function queries the physical status of up to 250 drives. See Figure 4-28.

```
#include "aci.h"
init aci_drivestatus3_one ( char *clientname,
         char *drive
         struct aci_drive_entry *pstDriveEntry[ACI_MAX_DRIVE_ENTRIES2])
```

Figure 4-28 aci_drivestatus_3_one Function Call

Return the status of drives which are set to UP for the client with the name clientname. If clientname is the NULL string, the call returns status on all drives. If a value for the drive field is indicated, the information relates to that single drive. The status is returned in an array of pointers to aci_drive_entry structure. The array element can be maximal 250. See Figure 4-29.

The value of ACI_MAX_DRIVES_ENTRIES2 is equal to 250.

```
struct aci_drive_entry {
         char drive_name[ACI_DRIVE_LEN];
         char amu_drive_name[ACI_AMU_DRIVE_LEN];
         enum aci_drive_status drive_state;
         char type;
         char system_id[ACI_NAME_LEN];
         char clientname
         char volser[ACI_VOLSER_LEN];
         bool_t cleaning
         short clean_count
};
```

Figure 4-29 Returned Status

See Table 4-13 for an explanation of the parameters used for the aci_drivestatus3_one function.

Table 4-13 Parameters for the aci_drivestatus3_one Function Call

Parameter	Description	
clientname	name of the client that requested the status of the drives. If *clientname* is the NULL string, return status on all drives.	
aci_drive_entry	returned information about the status of the drives	
	drive_name	name of the drive (name used in DAS and AMS description)
	amu_drive_name	internal AMS drive name e.g. 03 or ZZ
	drive_state	UP or DOWN reservation of the drive Refer to *aci_driveaccess* on page 4-18
	type	type of the drive (internal AMS code, e.g. E for DLT drive) (See the *AMU Reference Guide*)
	system_id	empty, reserved for further use
	clientname	name of the client that the drive is presently allocated to
	volser	Volser, if the drive is currently occupied
	cleaning	true if the drive is presently occupied with a medium for cleaning
	clean_count	number of mounts until the next clean activity

Return Values

- 0: The call was successful
- -1: The call has failed

The external variable *d_errno* is set to one of the following DAS error codes:

- ERPC
- EINVALID
- ETIMEOUT
- ESWITCHINPROGRESS
- EBADCLIENT
- ENOTSUPPHCMD

aci_eif_conf

The aci_eif_conf function queries status from up to 20 logical ranges. See Figure 4-30.

```
#include "aci.h"
int aci_eif_conf( struct aci_lora *lora_desc,
                  int *nCount)
```

Figure 4-30 aci_eif_conf Function Call

The call returns nCount and the staus of the logical range. The status is returned in an array of pointers to the aci_lora structure. The array can be a maximum of 20 elements. See Figure 4-31.

```
struct aci_lora {
          char lora_name[ACI_AREANAME_LEN];
          char StartCoord[ACI_COORD_LEN];
          char EndCoord[ACI_COORD_LEN];
          char ctype
          char description[ACI_DESCRIPTION_LEN];
};
```

Figure 4-31 Returned Status

For the parameters of the aci_lora structure, refer to Table 4-14 on page 4-39.

Table 4-14 Parameters for the aci_eif_conf Function Call

Parameter	Description	
aci_lorta	structure used to describe the import/export area and addressing	
	lora_name	the name of the import/export area
	StartCoord	the starting coordinates of the area
	EndCoord	the ending coordinates of the area
	ctype	the cartridge type associated with the area
	description	a description of the import.export area
nCount	the number of returned status (maximum of 300)	

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EBADCLIENT`
- `ENOTSUPPHCMD`

aci_eject

The aci_eject function ejects a range of volumes from the AML. See Figure 4-32.

```
#include "aci.h"
int aci_eject( char *eject_area,
               char *volser_range,
               enum aci_media type )
```

Figure 4-32 aci_eject Function Call

Eject the volumes in *volser_range* to the *eject_area*. The media type of the volumes must match that of the *eject_area*. Set *type* to the media type of the *volser_range*. See Table 4-15.

Table 4-15 Parameters for the aci_eject Function Call

Parameter	Description
clientname	Using SHARED_ACCESS as a key word for the client shows all drives which are allocated in the SHARED_ACCESS mode. Using EXUP as a key word for the clientname shows all drives which are allocated in the EXUP mode.
eject_area	Logical area defined in AML, where the cartridges should be ejected
volser_ranges	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen
types	media type of the named volser Refer to *Media Types* on page 2-7

The database entry for the volume is not deleted, and the position in the AML that the volume occupied remains reserved for insertion of a volume with a matching volser. This could be useful if the volume is temporarily ejected and will be inserted in the near future. In such case, the position remains reserved and the volume's location within the AML does not change.

The eject will stop, when the eject area is full. The request will either be cancelled or completed after the area is marked empty depending on the `DAS_EJECTAREAFULL` environment variable.

For additional information, refer to *aci_eject2* on page 4-42, and *aci_eject_ complete* on page 4-49.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`

- ENOVOLUME
- EPROBVOL
- EAMU
- EAMUCOMM
- EROBOTCOMM
- EDASINT
- ENOAREA
- ENOTAUTH
- EBADCLIENT
- ERETRYL
- EINUSE
- ECANCELED
- EDASINT
- ENOMATCH
- ETIMEOUT
- ESWITCHINPROG
- EHICAPINUSE
- ECOORDINATE
- EBARCODE
- EINVALIDDEV
- ENOROBOT
- EDATABASE
- ENOTSUPPHCMD
- EAREAEMPTY
- EBARCODE
- EAREAFULL

Refer to Figure 4-33 for an example of the aci_eject function.

```
int rc = 0;
rc = aci_eject( "E01", "AAB001", ACI_3590 );
if( rc )
{
         aci_perror( "Command failed: " );
}
else
{
         printf( "Volser AAB001 ejected to E01 \n" );
}
```

Figure 4-33 Example of the aci_eject Function

aci_eject2

The aci_eject2 function ejects a range of volumes from the AML. See Figure 4-34.

```
#include "aci.h"
int aci_eject2( char *eject_area,
               char *volser_range,
               enum aci_media type,
               int *pnActualCount,
               struct aci_ei_info *psteiInfo)
```

Figure 4-34 aci_eject2 Function Call

For the structure of the aci_ei_info function, refer to Figure 4-35.

```
struct aci_ei_info {
         char volser[ACI_VOLSER_LEN];
         char media_type[ACI_DRIVE_LEN];
         int errcode
};
```

Figure 4-35 Structure of the aci_ei_info function from aci.h

See Table 4-16 for a description of the parameters for the aci_eject2 function call.

Table 4-16 Parameters for the aci_eject2 Function Call

Parameter	Description
eject_area	logical area defined in the AMS, where the cartridges should be ejected
volser_range	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen • set to empty (““ or ‘\0’) if it is not to be changed
type	media type of the named volser range Refer to *Media Types* on page 2-7
pnActualCount	returned number of ejected volsers

Table 4-16 Parameters for the aci_eject2 Function Call

Parameter	Description	
aci_ei_info	returned information on each volser ejected	
	volser	volser of the ejected volume
	type	media type of the named volser Refer to *Media Types* on page 2-7
	errcodes	error code derrno

If the eject area is full, the system stops the eject procedure. Depending on the `DAS_EJECTAREAFULL` environment variable on the AMU, the command will be canceled, or the system starts with the next eject after the operator has removed cartridges from the eject area.

The `Attribute` of the Coordinate will be set to *ejected* in the AMU database. The compartment is now reserved for this volser, if the compartment needs to be used for other volsers, issue the `aci_eject_complete2` function.

Use the eject stop, when the eject area is full. Depending on the `DAS_EJECTAREAFULL` environment variable, the request will either be cancelled or completed after the area is marked empty.

For additional information, refer to *aci_insert* on page 4-75, and *aci_eject_ complete* on page 4-49.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ENOAREA`

- ENOTAUTH
- EBADCLIENT
- ERETRYL
- EINUSE
- ECANCELED
- EDASINT
- ENOMATCH
- ETIMEOUT
- ESWITCHINPROG
- EHICAPINUSE
- ECOORDINATE
- EBARCODE
- EINVALIDDEV
- ENOROBOT
- EDATABASE
- ENOTSUPPHCMD
- EAREAEMPTY
- EBARCODE
- EAREAFULL

See Figure 4-36 for an example of the aci_eject2 function.

```
/* Eject volume but reserve archive location */
int rc, i, pnActualCount;
char *volser_range = "000815 - 004711";
char *eject_area = "E02"
struct aci_ei_info ei_info[ACI_EI_MAX_RANGE];
rc = aci_eject2( eject_area, volser_range, ACI_3590
&pnActualCount, &ei_info );
if( rc )
{
         aci_perror( "Command failed: " );
}
else
{
         printf("Volume ejects request successful\n" );
for (i = 0; i < ACI_EI_MAX_RANGE; i++)
         {
         if ( *ei_info[i]->volser == '\0' )
            break;
         printf( "volser:%s media type:%s Error:%d\n",
            ei_info[i]->volser,
            ei_info[i]->media_type,
            ei_info[i]->errcode );
         }
```

Figure 4-36 Example of the aci_eject2 Function

aci_eject3

The aci_eject3 function ejects a range of volumes from the AML. See Figure 4-37.

```
#include "aci.h"
int aci_eject3( char *eject_area,
               char *volser_range,
               enum aci_media type,
               int *pnActualCount,
               struct aci_ei_info *psteiInfo)
```

Figure 4-37 aci_eject3 Function Call

The volser_range field has a maximum size of 512 bytes.

For the structure of the aci_ei_info function, refer to Figure 4-38.

```
struct aci_ei_info {
         char volser[ACI_VOLSER_LEN];
         char media_type[ACI_DRIVE_LEN];
         int errcode
};
```

Figure 4-38 Structure of the aci_ei_info function

See Table 4-17 for a description of the parameters for the aci_eject2 function call.

Table 4-17 Parameters for the aci_eject3 Function Call

Parameter	Description
eject_area	logical area defined in the AMS, where the cartridges should be ejected
volser_range	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen • set to empty ("" or '\0') if it is not to be changed
type	media type of the named volser range Refer to *Media Types* on page 2-7

Table 4-17 Parameters for the aci_eject3 Function Call

Parameter	Description	
pnActualCount	returned number of ejected volsers	
aci_ei_info	returned information on each volser ejected	
	volser	volser of the ejected volume
	type	media type of the named volser Refer to *Media Types* on page 2-7
	errcodes	error code derrno

If the eject area is full, the system stops the eject procedure. Depending on the `DAS_EJECTAREAFULL` environment variable on the AMU, the command will be canceled, or the system starts with the next eject after the operator has removed cartridges from the eject area.

The `Attribute` of the Coordinate will be set to *ejected* in the AMU database. The compartment is now reserved for this volser, if the compartment needs to be used for other volsers, issue the `aci_eject_complete2` function.

Use the eject stop, when the eject area is full. Depending on the `DAS_EJECTAREAFULL` environment variable, the request will either be cancelled or completed after the area is marked empty.

For additional information, refer to *aci_insert* on page 4-75, and *aci_eject_ complete* on page 4-49.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`

- ENOAREA
- ENOTAUTH
- EBADCLIENT
- ERETRYL
- EINUSE
- ECANCELED
- ENOMATCH
- ETIMEOUT
- ESWITCHINPROG
- EHICAPINUSE
- ECOORDINATE
- EBARCODE
- EINVALIDDEV
- ENOROBOT
- EDATABASE
- ENOTSUPPHCMD
- EAREAEMPTY
- EBARCODE
- EAREAFULL

aci_eject_ complete

The aci_eject_complete function ejects volumes and removes the database entries. See Figure 4-39.

```
#include "aci.h"
int aci_eject_complete( char *eject_area,
                        char *volser_range,
                        enum aci_media type)
```

Figure 4-39 aci_eject_complete function call

Eject the volumes in *volser_range* to the *eject_area*. The media type of the volumes must match that of the *eject_area*. Set *type* to the media type of the *volser_range.* The database entry for the volume is deleted, and the position in the AML that the volume occupied becomes free. This could be useful if the volume is to be placed in long-term archive storage or more space is needed in the AML. If the volume is re-inserted into the AML, it is stored in the next available position, and it probably will not occupy the previous position. The volume is re-inserted in the same position only if `aci_eject` is used. See Table 4-18 for a description of the parameters for the aci_eject_complete function call.

Table 4-18 Parameters for the aci_eject_complete Function Call

Parameter	Description
eject_area	Logical area defined in AMS, where the cartridges should be ejected
volser_range	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen
types	media types of the named volser Refer to *Media Types* on page 2-7

The eject will stop, when the eject area is full. Depending on the `DAS_EJECTAREAFULL` environment variable, the request will either be cancelled or completed after the area is marked empty.

For additional information, refer to *aci_eject* on page 4-39, and *aci_insert* on page 4-75.

Return Values

- 0: The call was successful.
- -1: The call failed.

The *external* variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ENOAREA`
- `ENOTAUTH`
- `EBADCLIENT`
- `ERETRYL`
- `EINUSE`
- `ECANCELED`
- `EDASINT`
- `ENOMATCH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCODE`
- `EINVALIDDEV`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `EBARCODE`
- `EAREAFULL`

Refer to Figure 4-40 on page 4-51 for an example of the aci_eject_complete function.

```
/* Eject volume from archive and free storage position.
*/
int rc = 0;
rc = aci_eject_complete("E01", "AAB001", ACI_3590 );
if( rc )
{
          aci_perror( "Command failed:" );
}
else
{
         printf( "Volser AAB001 ejected to E01 \n" );
}
```

Figure 4-40 Example of the aci_eject_complete Function

aci_eject2_complete

The aci_eject2_complete function ejects a range of volumes from the AML, and deletes the home coordinate. See Figure 4-41.

```
#include "aci.h"
int aci_eject2_complete( char *eject_area,
               char *volser_range,
               enum aci_media type,
               int *pnActualCount,
               struct aci_ei_info *psteiInfo)
```

Figure 4-41 aci_eject2_complete Function Call

See Figure 4-42 for the structure of the aci_ei_info function.

```
struct aci_ei_info {
         char volser[ACI_VOLSER_LEN];
         char media_type[ACI_DRIVE_LEN];
         int errcode
};
```

Figure 4-42 Structure of the aci_ei_info function

See Table 4-19 for a description of the parameters for the aci_eject2_complete function call.

Table 4-19 Parameters for the aci_eject2_complete Function Call

Parameter	Description	
eject_area	logical area defined in AML, where the cartridges should be ejected	
volser_range	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen • set to empty ("" or '\0') if it is not to be changed	
type	media type of the named volser range Refer to *Media Types* on page 2-7	
pnActualCount	returned number of ejected volsers	
aci_ei_info	returned information to each ejected volser	
	volser	volser of the ejected volume
	type	media type of the named volser Refer to *Media Types* on page 2-7
	errcode	Error code derrno

If the eject area is full the system stops the eject procedure. Depending on the `DAS_EJECTAREAFULL` environment variable on the AMU, the command will be canceled, or the system starts with the next eject after the operator has removed cartridges from the eject area.

The attribute of the coordinate will be set to empty in the AMU database. The volser will be set to 0000000000000000, but the type will remain unchanged. The compartment is ready for the other volsers.

For additional information, refer to *aci_insert* on page 4-75, and *aci_eject2* on page 4-42.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ENOAREA`
- `ENOTAUTH`
- `EBADCLIENT`
- `ERETRYL`
- `EINUSE`
- `ECANCELED`
- `EDASINT`
- `ENOMATCH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCODE`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `EBARCODE`
- `EAREAFULL`

Refer to Figure 4-43 on page 4-54 for an example of the aci_eject2_complete function.

```
/* Eject volume complete from AML system*/
int rc, i, pnActualCount;
char *volser_range = "000815 - 004711";
char *eject_area = "E02"
struct aci_ei_info ei_info[ACI_EI_MAX_RANGE] ;
rc = aci_eject2_complete( eject_area, volser_range,
         ACI_3590, & pActualCount, &ei_info );
if( rec )
{
         aci_perror( "Command failed: " );
}
else
{
         printf("Volume ejects request successful\n" );
for (i = 0; i < ACI_EI_MAX_RANGE; i++)
         {
         if ( *ei_info[i]->volser == '\0' )
            break;
         printf( "volser:%s media type:%s Error:%d\n",
            ei_info[i]->volser,
            ei_info[i]->media_type,
            ei_info[i]->errcode );
         }
}
```

Figure 4-43 Example of the aci_eject2_complete Function

aci_eject3_complete

The aci_eject3_complete function ejects a range of volumes from the AML, and deletes the home coordinate. See Figure 4-44.

```
#include "aci.h"
int aci_eject3_complete( char *eject_area,
               char *volser_range,
               enum aci_media type,
               int *pnActualCount,
               struct aci_ei_info *psteiInfo)
```

Figure 4-44 aci_eject3_complete Function Call

The volser_range field has a maximum size of 512 bytes.

See Figure 4-45 for the structure of the aci_ei_info function.

```
struct aci_ei_info {
         char volser[ACI_VOLSER_LEN];
         char media_type[ACI_DRIVE_LEN];
         int errcode
};
```

Figure 4-45 Structure of the aci_ei_info function

See Table 4-20 for a description of the parameters for the aci_eject3_complete function call.

Table 4-20 Parameters for the aci_eject3_complete Function Call

Parameter	Description	
eject_area	logical area defined in AML, where the cartridges should be ejected	
volser_range	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen • set to empty (““ or ‘\0’) if it is not to be changed	
type	media type of the named volser range Refer to *Media Types* on page 2-7	
pnActualCount	returned number of ejected volsers	
aci_ei_info	returned information to each ejected volser	
	volser	volser of the ejected volume
	type	media type of the named volser Refer to *Media Types* on page 2-7
	errcode	Error code derrno

If the eject area is full the system stops the eject procedure. Depending on the `DAS_EJECTAREAFULL` environment variable on the AMU, the command will be canceled, or the system starts with the next eject after the operator has removed cartridges from the eject area.

The attribute of the coordinate will be set to empty in the AMU database. The volser will be set to 0000000000000000, but the type will remain unchanged. The compartment is ready for the other volsers.

For additional information, refer to *aci_insert* on page 4-75, and *aci_eject2* on page 4-42.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ENOAREA`
- `ENOTAUTH`
- `EBADCLIENT`
- `ERETRYL`
- `EINUSE`
- `ECANCELED`
- `EDASINT`
- `ENOMATCH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCODE`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `EBARCODE`
- `EAREAFULL`

aci_ejectclean

The aci_ejectclean function ejects all exhausted cleaning cartridges from one Cleanpool. See Figure 4-46.

```
#include "aci.h"
int aci_ejectclean( char *areaname,
               char *Cleanpoolname,
               int *pnActualCount,
               struct aci_ei_info *psteiInfo)
```

Figure 4-46 aci_ejectclean Function Call

See Figure 4-47 for the structure of the aci_ei_info function.

```
struct aci_ei_info {
         char volser[ACI_VOLSER_LEN];
         char media_type[ACI_DRIVE_LEN];
         int errcode
};
```

Figure 4-47 Structure of the aci_ei_info function from aci.h

See Table 4-21 for a description of the parameters for the aci_ejectclean function Call.

Table 4-21 Parameters for the aci_ejectclean Function Call

Parameter	Description
areaname	Logical area defined in AMS, where the cartridges should be ejected, e.g. E01
Cleanpoolname	Name of the group of cleaning cartridges, e.g. CLP01, defined in the AML configuration
pnActualCount	returned number of ejected volsers

Table 4-21 Parameters for the aci_ejectclean Function Call

Parameter	Description	
aci_ei_info	returned information to each ejected volser	
	volser	volser of the ejected volume
	type	media type of the named volser Refer to *Media Types* on page 2-7
	errcode	error code derrno

If the eject area is full the system stops the eject procedure. Depending on the `DAS_EJECTAREAFULL` environment variable on the AMU, the command will be canceled, or the system starts with the next eject after the operator has removed cartridges from the eject area.

The attribute of the coordinate will be set to empty in the AMU database. The volser will be set to 0000000000000000, and the type will be set to AMU-Dynamic. The compartment is ready for the other volsers.

The compartments for cleaning cartridges are not reserved. Organize enough compartments for the cleaning cartridges every time.

For additional information, refer to *aci_insert* on page 4-75, and *aci_eject2* on page 4-42.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ENOAREA`
- `ENOTAUTH`

- `EBADCLIENT`
- `ERETRYL`
- `EINUSE`
- `ECANCELED`
- `EDASINT`
- `ENOMATCH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `ENOPOOL`
- `EAREAFULL`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCODE`
- `EINVALIDDEV`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `EBARCODE`
- `EAREAFULL`

Refer to Figure 4-48 on page 4-60 for an example of the aci_ejectclean function.

```
/* Eject clean cartridges from AML system*/
int res, i, pnActualCount;
char *Cleanpoolname = "CLP01";
char *areaname = "E02"
struct aci_ei_info *ei_info[ACI_EI_MAX_RANGE];
res =       aci_ejectclean( areaname, Cleanpoolname,
            ei_info);
if( rec )
{
         aci_perror( "Command failed: " );
}
else
{
         printf("Volume ejects request successful\n" );
for (i = 0; i < ACI_EI_MAX_RANGE; i++)
         {
         if ( *ei_info[i]->volser == '\0' )
            break;
         printf( "volser:%s media type:%s Error:%d\n",
            ei_info[i]->volser,
            ei_info[i]->media_type,
            ei_info[i]->errcode );
         }
}
```

Figure 4-48 Example of the aci_ejectclean Function

aci_email

The aci_email function sends email messages. See Figure 4-49.

```
#include "aci.h"
int aci_email (char *Adr
            char *Msg)
```

Figure 4-49 aci_email Function Call

See Table 4-22.

Table 4-22 Parameter for the aci_email Function Call

Parameter	Description
Adr	A non-empty email address of no more than 64 characters
Msg	defines the email message of no more than 255 characters

Only supported by the Scalar DLC software.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EBADCLIENT`
- `ENOTSUPPHCMD`

aci_flip

The aci_flip function turns an Optical Disk (O. D.) in a drive. See Figure 4-50.

```
#include "aci.h"
int aci_flip( char *drive)
```

Figure 4-50 aci_flip Function Call

In the drive named *drive*, the dismounted O. D. will be turned 180° and mounted in the same drive. Refer to Table 4-23 for a description of the parameter for the aci_flip function.

For additional information, refer to *aci_mount* on page 4-91, and *aci_dismount* on page 4-16.

Table 4-23 Parameter for the aci_flip Function Call

Parameter	Description
drive	name of the optical disk drive for the flip operation

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `ENODRIVE`
- `EDRVOCCUPIED`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `EDEVEMPTY`
- `ENOTAUTH`
- `EBADCLIENT`
- `ENOTMOUNTED`
- `ECANCELED`
- `EDASINT`
- `ECLEANING`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCODE`
- `EINVALIDDEV`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`

- `EBARCODE`
- `EAREAFULL`

See Figure 4-51 for an example of the aci_flip function.

```
/* Flip a volume in a optical disk drive */
int rc = 0;
char *drive = "od01";
if (( rc = aci_flip( drive ) ))
{
         aci_perror( "Flip command failed: " );
}
else
{
         printf("Flip of in drive %s successful
         completed.\n", drive );
}
```

Figure 4-51 Example of the aci_flip Function

aci_force

The aci_force function dismounts any cartridge from a specific drive. See Figure 4-52.

```
#include "aci.h"
int aci_force( char *drive )
```

Figure 4-52 aci_force Function Call

Force a dismount from a drive named *drive*, regardless of which volser is in the drive. See Table 4-24.

For additional information, refer to *aci_dismount* on page 4-16, and *aci_mount* on page 4-91.

Table 4-24 Parameter for the aci_force Function Call

Parameter	Description
drive	name of the drive for the dismount

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `ENODRIVE`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `EDEVEMPTY`
- `ENOTAUTH`
- `EUPELSE`

- EBADCLIENT
- ENOTMOUNTED
- ECANCELED
- EDASINT
- ENOMATCH
- ETIMEOUT
- ESWITCHINPROG
- EHICAPINUSE
- ECOORDINATE
- EBARCODE
- EINVALIDDEV
- ENOROBOT
- EDATABASE
- ENOTSUPPHCMD
- EAREAEMPTY
- EBARCODE
- EAREAFULL

See Figure 4-53 for an example of the aci_force function.

```
/* Dismount any volume from drive */
int rc = 0;
rc = aci_force( "Drive1" );
if( rc )
{
          aci_perror( "Command failed: " );
}
else
{
         printf( "Dismount of Drive1 successful.\n" );
}
```

Figure 4-53 Example of the aci_force Function

aci_foreign

The aci_foreign function catalogs a foreign volume. See Figure 4-54.

```
#include "aci.h"
int aci_foreign( enum aci_command action,
               char *volser,
               enum aci_media type,
               char *coordinate,
               short position )
```

Figure 4-54 aci_foreign Function Call

When a foreign volume named *volser* of media *type* has been added to or removed from the AML system in a foreign mount area, DAS must be informed of the status. This function notifies DAS of the existence of the volume, and its coordinate. If added, the volume can be used simply by referencing this *volser* and *type* in a further `aci_mount`() or `aci_dismount`() request. See Table 4-25.

Table 4-25 Parameters for the aci_foreign Function Call

Parameter	Descriptions	
action	select the command for the foreign catalog procedure	
	ACI_ADD	new volser will be added to the foreign media area in DAS and AML database
	ACI_DELETE	volser entry for a foreign media that will be removed from the DAS and AMU database
volser	volser, that should be used in the application for the mount (volser with up to 16 digits and alphanumeric symbols, no special characters)	
type	media type of the named volser Refer to *Media Types* on page 2-7	
coordinate	10 digit logical coordinate of the compartment of the foreign volume in the AMS, e.g. E501010110 Refer to the *AMU Reference Guide*	

Table 4-25 Parameters for the aci_foreign Function Call

Parameter	Descriptions
position	reserved for compatibility, not used

This version does not have a command to display occupied symbolic volsers. Please note this assignment; the symbolic volser will be needed for the rmf command.

For additional information, refer to *aci_dismount* on page 4-16, and *aci_mount* on page 4-91.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EDASINT`
- `ENOTAUTH`
- `EBADCLIENT`
- `ENOSPACE`
- `EDASINT`
- `ENOMATCH`
- `ETIMEOUT`
- `ERSWITCHINPROG`
- `EINVALIDDEV`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`

Refer to Figure 4-55 on page 4-68 for an example of the aci_foreign function.

```
/* Add foreign media to DAS catalog */
int rc = 0;
enum aci_media type = ACI_3590;
char *volser = "FOR001";
char *coord = "E301010310";
short pos = 0;
if ((rc = aci_foreign(ACI_ADD, volser, type,
            coord, pos )))
{
         aci_perror("Foreign media location failed: ");
{
else
{
         printf("Foreign volser %s added.\n", volser);
}
```

Figure 4-55 Example of the aci_foreign Function.

aci_getvolsertodrive

The aci_getvolsertodrive function gets the configured relation between the specified drive and the volser. See Figure 4-56.

```
#include "aci.h"
int aci_getvolsertodrive ( char *Drive,int num
            struct aci_voltodrive_entry
            *Idrive_inf[ACI_DRIVE_NUMBERS] )
```

Figure 4-56 aci_getvolsertodrive Function Call

This call returns the configured relation between the specified drive and volsers. If 'Drive' is the Null String, the call returns the relation of all drives that are configured. The response is returned in InfoVolserToDrive, an array of pointers to an aci_voltodrive_entry structure. Refer to
Figure 4-57 on page 4-69.

```
struct aci_voltodrive_entry{
         char Drive[ACI_DRIVE_LEN];
         char VolserRange[ACI_RANGE_LEN];
}
```

Figure 4-57 aci_voltodrive_entry Structure

See Table 4-26 for a description of the parameters for the aci_getvolsertodrive function call.

Table 4-26 Parameters for the aci_getvolsertodrive Function Call

Parameter	Description	
Drive	name of the drive, that the reserved volsers requested	
num	number of reported entries	
aci_voltodrive_entry	information returned for the selected drive	
	Drive	name of drive
	VolserRange	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen

Return Values

- 0: The call was successful
- -1: The call failed

The external variable d_errno is set to one of the following DAS error codes

- `ERPC`
- `ENODRIVE`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`

Refer to Figure 4-58 on page 4-70 for an example of the aci_getvolsertodrive function.

```
/* Display volumes reserved for a drive*/
int rc, i, num;
char *Drive = "Drive1";
struct aci_voltodrive_entry
         *drive_inf[ACI_MAX_RANGES];
rc = aci_getvolsertodrive( Drive, num, drive_inf);
if( rc )
{
         aci_perror( "Command failed: " );
}
else
for (i = 0; i < ACI_MAX_RANGES; i++)
         {
         if ( *drive_inf[i]->volser == '\0' )
            break;
         printf( "drive:%s Volser Range%d\n",
            drive_inf[i]->Drive,
            drive_inf[i]->VolserRange );
         }
```

Figure 4-58 Example of the aci_getvolertodrive Function

aci_getVolserToSide

The aci_getvolsertoside function returns the second volser to an optical disk (a volume with two volsers). See Figure 4-59.

```
int aci_getVolserToSide (char *volser,
            struct aci_sideinfo
*sideinfo[ACI_SIDE_NUMBER])
```

Figure 4-59 aci_getVolserToSide Function Call

This function returns in the '*sideinfo*' parameter, the volser attached to one of a two sided volume. The '*volser*' parameter can be the A-side or the B-side volser. When a volser of a volume with one side is specified, an error ENODOUBLESIDE error is returns.

The *sideinfo*[0] parameter gives the attachment of the A-side, and *sideinfo*[1] gives the attachment of the B-side.

The media type for the volser in the AMS configuration must be Optical disk (O0 or O1). Refer to Figure 4-60 on page 4-71.

```
struct aci_sideinfo {
         char  cVolumeSide;
         char  szVolser[ACI_VOLSER_LEN];
}
```

Figure 4-60 Structure of Type aci_sideinfo

The define 'ACI_SIDE_NUMBER' parameter is set, in aci.h, to 2. See Table 4-27.

Table 4-27 Parameters for the aci_getvolsertoside Function Call

Parameter	Description	
volser	name of the volser, for which information is requested	
aci_sideinfo	information returned for the selected volser	
	CVolumeside	• A: for the Volser located on the top label • B: for the Volser located on the bottom label on the cartridge
	sZVolser	Volser from the AMU database

Return Values

- 0: The call was successful
- -1: The call failed

The external variable d_errno is set to one of the following DAS error codes

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `ENOTAUTH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `ENODOUBLESIDE`

Refer to Figure 4-61 on page 4-72 for an example of the aci_getvolsertoside function.

```
/* Display volsers for the optical disk*/
int rc = 0;
char *volser = "00815F";
struct aci_sideinfo *sideinfo[ACI_SIDE_NUMBER];
rc = aci_getvolsertoside( volser, sideinfo);
if( rc )
{
         aci_perror( "Command failed: " );
}
else
         printf( "Side:%s Volser:%d\n",
         side_info[0]->Side,
         side_info[0]->Volser,
         side_info[1]->Side,
         side_info[1]->Volser);
         }
```

Figure 4-61 Example of the aci_getvolsertoside Function

aci_init

The aci_init function initializes the AML for client use. See Figure 4-62.

```
#include "aci.h"
int aci_init( void )
```

Figure 4-62 aci_init Function Call

Most clients wish to start from a known status, the `aci_init` call is provided so that an initialization, or restart, of a client may request that any resources previously held are freed. Normally this call can be placed in the initialization of the client code. This function requests that the DAS server dismount any occupied drives defined to the requesting client.

For additional information, refer to *aci_initialize* on page 4-74.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `EDASINT`
- `EBADCLIENT`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`

See Figure 4-63 for an example of the aci_init function.

```
/* Initialize client and free drive resources */
int rc = 0;
rc = aci_init( );
if( rc )
{
        aci_perror( "ACI failed initialization:" );
}
```

Figure 4-63 Example of the aci_init Function

aci_initialize

The aci_initialize function initializes ACI library for client use. See Figure 4-64.

```
#include "aci.h"
int aci_initialize( void )
```

Figure 4-64 aci_initialize Function Call

This function initializes the DAS 3.01 ACI library. This function should be called as the first activity, before the use of any other functions from the ACI library.

For additional information, refer to *aci_init* on page 4-72.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EDASINT`
- `EDASINT`
- `ESWITCHINPROG`

See Figure 4-65 for an example of the aci_initialize function.

```
/* Initialize client, do not change drive state */
int rc = 0;
rc = aci_initialize( );
if( rc )
{
          aci_perror( "ACI failed initialization:" );
}
```

Figure 4-65 Example of the aci_initialize Function

aci_insert

The aci_insert function inserts volumes into the AML. See Figure 4-66.

```
#include "aci.h"
int aci_insert( char *insert_area,
               char *volser_ranges,
               enum aci_media type )
```

Figure 4-66 aci_insert Function Call

This function is for compatibility. Please use the aci_insertgen function.

The cartridges which are actually located in the insert area of the AML and registered by the AMU (automatically inventoried after closing the area) will be inserted.

- Volsers which already have a entry in the AMU database, will be moved to this position.
- Volsers which have no entry in the AMU database, will be moved to the first free slot (AMU database Attribute: "Empty" of this media type of the Type AMU-Dynamic)
- Cartridges with an invalid volser (barcode not readable) will moved to the problembox.

See Table 4-28 for a description of the parameters for the aci_insert function call.

Table 4-28 Parameters for the aci_insert Function Call

Parameter	Description
insert_area	Logical insert range of the Insert/Eject unit of the AML in the AMS (e.g. I03)
volser_range	The volumes found in the insert_area are returned in the volser_ranges array of strings, where each volser is separated by a comma. Each range is ACI_RANG_LEN long and there are up to ACI_MAX_RANGES ranges. An empty string indicates the end of the ranges
type	media type of the volser in the insert area Refer to *Media Types* on page 2-7

Use the insert2 function instead of this command. This function experiences difficulties with large I/O units with long volsers (16-digit) since the buffer for displaying the inserted volser is restricted. For compatibility reasons this function continues to be supported.

For additional information, refer to *aci_insert* on page 4-75, and *aci_eject2* on page 4-42.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ENOAREA`
- `ENOAUTH`
- `ERETRYL`
- `ECANCELLED`
- `EDASINT`
- `ENOMATCH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCOODE`
- `EINVALIDDEV`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `ENOPOOL`

- `EBARCODE`
- `EAREAFULL`

Refer to Figure 4-67 on page 4-77 for an example of the aci_insert function.

```
/* Insert media into AML */
int rc, i;
enum aci_media type = ACI_3590;
char *area_name = "I01";
char *volser_ranges[ACI_MAX_RANGES];
rc = aci_insert(area_name, volser_ranges, &type);
if( rc )
{         aci_perror("Command failed: ");
}
else
{
          for (i = 0; i < ACI_MAX_RANGES; i++)
          if (*volser_ranges[i] == '\0')
           break;
          else
           printf("%s\n", volser_ranges[i]);
}
```

Figure 4-67 Example of the aci_insert Function

aci_insert2

The aci_insert2 function inserts volumes, including clean media, into the AML. See Figure 4-68

```
#include "aci.h"
int aci_insert2( char *insertopt,
               char *areaname,
               char *cleanPoolname,
               int *pnActualCount,
               struct aci ei info *pstEiInfo)
```

Figure 4-68 aci_insert2 Function Call

The cartridges which are located in the Insert area of the AML and registered by the AMU (automatically inventoried after closing the area), will be inserted.

- Volsers which already have a entry in the AMU database, will be moved to this position.
- Volsers which have no entry in the AMU database, will be moved to the first free slot (AMU database Attribute: "Empty" of this media type of the Type AMU-Dynamic)
- With `insertopt -c` will move the cartridges to the first free slot (AMU database Attribute: "Empty") of this media type of the Type AMU-Dynamic. The attribute will automatic change to Clean
- Cartridges with an invalid volser (barcode not readable) will be moved to the problembox.

The function returned the inserted volser with the following structure. See Figure 4-69.

For additional information, refer to *aci_eject* on page 4-39.

```
struct aci_ei_info {
          char  volser[ACI_VOLSER_LEN];
          char  media_type[ACI_DRIVE_LEN;
          int errcode
}
```

Figure 4-69 aci_ei_info Structure from aci.h

See Table 4-29 for a description of the parameters for the aci_insert2 function call.

Table 4-29 Parameters for the aci_insert2 Function Call

Parameter	Description	
insertopt	defined the type of media for the insert operation	
	-n	normal (data cartridges)
	-c	clean (cleaning cartridges)
areaname	Logical insert range of the Insert/Eject unit of the AML in the AMS (e.g. I03)	
cleanPoolname	Name of the Cleanpool defined in AML (must be NULL, if insertopt= n)	
pnActualCount	Number of inserted volsers and pointer to an array of structure	

Table 4-29 Parameters for the aci_insert2 Function Call

Parameter	Description	
aci_ei_info	returned information on each volser ejected	
	volser	volser of the ejected volume
	type	media type of the named volser Refer to *Media Types* on page 2-7
	errcodes	error code derrno

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ENOAREA`
- `ENOAUTH`
- `ERETRYL`
- `ECANCELLED`
- `EDASINT`
- `ENOMATCH`
- `ECLEANING`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `ENOPOOL`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCOODE`
- `EINVALIDDEV`
- `ENOROBOT`

- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `ENOPOOL`
- `EBARCODE`
- `EAREAFULL`

See Figure 4-70 for an example of the aci_insert2 function.

```
/* insert clean volume in the AML system*/
int res, i, pnActualCount;
char *cleanPoolname = "CLP01";
char *areaname = "I02"
struct aci_ei_info *pstEiInfo[ACI_EI_MAX_RANGE];
res = aci_insert2( "-c", areaname, cleanPoolname,
            &pnActualCount, pstEiInfo;
if( rec )
{
         aci_perror( "Command failed: " );
}
else
{
         printf("Volume insert request successful\n" );
for (i = 0; i < ACI_EI_MAX_RANGE; i++)
         {
         if ( *ei_info[i]->volser == '\0' )
            break;
         printf( "volser:%s media type:%s Error:%d\n",
            ei_info[i]->volser,
            ei_info[i]->media_type,
            ei_info[i]->errcode );
         }
}
```

Figure 4-70 Example of the aci_insert2 Function

aci_inventory

The aci_inventory function performs a physical inventory of the AML. See Figure 4-71.

```
#include "aci.h"
int aci_inventory( void )
```

Figure 4-71 aci_inventory Function Call

This command instructs the robot to perform a physical inventory of the archive and to update the database. The inventory command is issued to the archive and the function will not wait for the completion of the inventory operation. The `aci_list()` function may be called to determine whether the inventory command is still active (result for a running inventory is `PINV`)

Attention

The inventory function is intended for testing and startup. An error function will be displayed in the AMU log during operation (and not returned to the calling process). The entire database will be overwritten with a symbolic volser "*Ixxxx" if the barcode reader malfunctions.

For additional information, refer to *aci_qvolsrange* on page 4-98, and *aci_view* on page 4-123.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `EPC`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ERETRYL`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`

- EHICAPINUSE
- ECOORDINATE
- EBARCODE

See Figure 4-72 for an example of the aci_inventory function.

```
/* Inventory the AML */
int rc = 0;
rc = aci_inventory( );
if( rc )
{
          aci_perror( "Command failed:" );
}
```

Figure 4-72 Example of the aci_inventory Function

aci_killamu

The aci_killamu function homes the robots and shuts down all programs running on the AMU PC (including OS/2). See Figure 4-73.

```
#include "aci.h"
int aci_killamu(void)
```

Figure 4-73 aci_killamu Function Call

The call of this function shuts down the complete AMU-PC. DAS sends a response to the client. Wait 5 minutes before powering off.

Attention

Inform all administrators also using the AML system before starting the command. The command may cause disruption to their operation.

DAS sends a positive acknowledgment, before the process is complete. Wait at least 5 minutes following the positive acknowledgment before switching off the power supply to the machine. Switching off the power supply to the AMU PC too soon can lead to loss of data.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ENOTAUTH`
- `ERPC`
- `EINVALID`
- `EDASINT`
- `ETIMEOUT`
- `EAMUCOMM`
- `EHICAPINUSE`
- `ESWITCHINPROG`

See Figure 4-74 for an example of the aci_killamu function.

```
/* Shutdown of the AMU */
int rc = 0;
rc = aci_killamu( );
if( rc )
{
          aci_perror( "Shutdown of AMU PC failed:" );
}
```

Figure 4-74 Example of the aci_killamu Function

aci_list

The aci_list function lists outstanding requests for a client. See Figure 4-75.

```
#include "aci.h"
int aci_list( char *clientname,
         struct aci_req_entry
            *request_list[ACI_MAX_REQ_ENTRIES])
```

Figure 4-75 aci_list Function Call

List outstanding requests belonging to the client *clientname*. Return the requests in the *request_list* array of `aci_req_entry` *request_list* structures. The structure's request types are defined in the header file "aci_das.h". See Figure 4-76.

For additional information, refer to *aci_cancel* on page 4-7.

```
struct aci_req_entry {
         u_long request_no;
         u_long individ_no;
         char clientname[ACI_NAME_LEN];
         char req_type[ACI_REQTYPE_LEN + 1];
};
```

Figure 4-76 Example of the Returned Structure

Refer to Table 4-30 for a description of the parameters for the aci_list function call.

Table 4-30 Parameters for the aci_list Function Call

Parameter	Description	
clientnamet	Name of the client that active request are reported to.	
aci_req_entry	Returned information about outstanding commands	
	request_no	DAS command sequence serial number
	individ_no	reserved, not used
	clientname	name of the requesting client
	req_type	type of the outstanding command Refer to Table 4-31

See Table 4-31 for an explanation of the req_types.

Table 4-31 Explanation of the Req_types

req_types	Explanation
BACO	switch barcode reading on or off (aci_barcode)
EJCL	eject of cleaning cartridges (aci_ejectclean)
INCL	insert media from I/O unit with aci_insertgen
INVT	insert media from I/O with aci_insert
KEEP	keep media from drive (aci_dismount)
MOUNT	mount media to a drive (aci_mount)
MOVE	eject media to I/O unit (aci_eject, aci_eject_complete)
PINV	Complete archive inventory and database update (aci_inventory)
PRGE	Delete a command from the command list (aci_cancel)
SHUT	Shutdown off the complete AMU (aci_killamu)

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`

See Figure 4-77 for an example of the aci_list function.

```
/* List outstanding client requests */
int rc, i;
char *client = "SomeClient";
struct aci_req_entry *requests[ACI_MAX_REQ_ENTRIES];
if ( rc = aci_list( client, requests ) )
{
         aci_perror( "Command failed: " );
}
else
{
         printf ("List for client %s successful\n",
               client);
for (i=0;(i<ACI_MAX_REQ_ENTRIES)&&(requests[i]-
>request_no!=0);i++)
{
         printf("client = %s\n\trequest =
%u\n\tindivid_no = %ld\n"
             "\ttype = %s\n",
             requests[i]->clientname,
             requests[i]->request_no,
             requests[i]->individ_no,
             requests[i]->req_type);
}
```

Figure 4-77 Example of the aci_list Function

aci_list2

The aci_list2 function lists executing requests for a client. See Figure 4-78.

```
#include "aci.h"
int aci_list2( char *clientname,
            struct aci_req_entry2
            *req_status[])
```

Figure 4-78 aci_list2 Function Call

List executing requests belonging to the client *clientname*. Return the requests in the array of `aci_req_entry2` structures. The array element can be maximum size of 15. See Figure 4-79.

```
struct aci_req_entry2 {
         u_long request_no;
         u_long individ_no;
         char clientname[ACI_NAME_LEN];
         char req_type[ACI_REQTYPE_LEN2 + 1];
         char Volser[ACI_VOLSER_LEN];
         char Drive[ACI_DRIVE_LEN];
         char AreaName[ACI_AREANAME_LEN];
         char PoolName[ACI_POOLNAME_LEN];
         char VolserRange[ACI_RANGE_LEN];
         char StartCoord[ACI_COORD_LEN];
         char EndCoord[ACI_COORD_LEN];
};
```

Figure 4-79 Example of the Returned Structure

Refer to Table 4-32 for a description of the parameters for the aci_list2 function call.

Table 4-32 Parameters for the aci_list2 Function Call

Parameter	Description	
clientname	Name of the client that receives the active request information.	
aci_req_entry2	Returned information about executing commands	
	request_no	DAS command sequence serial number
	individ_no	reserved, not used
	clientname	name of the requesting client
	req_type	type of the outstanding command Refer to Table 4-31
	Volser	the volser number
	Drive	the drive number
	AreaName	the import or export area
	PoolName	the pool name (scratch, clean)
	VolserRange	the range of the volser
	StartCoord	starting coordinates of the range
	EndCoord	the ending coordinates of the range

See Table 4-33 for an explanation of the req_types.

Table 4-33 Explanation of the Req_types

req_types	Explanation
BACO	switch barcode reading on or off (aci_barcode)
EJCL	eject of cleaning cartridges (aci_ejectclean)
INCL	insert media from I/O unit with aci_insertgen
INVT	insert media from I/O with aci_insert
KEEP	keep media from drive (aci_dismount)
MOUNT	mount media to a drive (aci_mount)
MOVE	eject media to I/O unit (aci_eject, aci_eject_complete)

Table 4-33 Explanation of the Req_types

req_types	Explanation
PINV	Complete archive inventory and database update (aci_inventory)
PRGE	Delete a command from the command list (aci_cancel)
SHUT	Shutdown off the complete AMU (aci_killamu)

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `ENOTSUPPHCMD`

aci_list_foreign

The aci_list_foreign function queries status from up to 300 foreign volsers for a client. See Figure 4-80.

```
#include "aci.h"
int aci_list_foreign( char *volser,
                      int *nCount,
                      aci_foreign_desc *pszForeign)
```

Figure 4-80 aci_list_foreign Function Call

Returns the status of the foreign volser. If the volser is the NULL string, return nCount status on all foreign volsers. The status is returned in an array of pointers to aci_foreign_desc structure. The array element can be maximum size of 300. See Figure 4-81.

```
struct aci_foreign_desc {
         char volser[ACI_VOLSER_LEN];
         char coord[ACI_COORD_LEN];
         enum aci_media_type eType;
         char attrib
};
```

Figure 4-81 Example of the Returned Structure

Refer to Table 4-34 for a description of the parameters for the aci_list_foreign function call.

Table 4-34 Parameters for the aci_list_foreign Function Call

Parameter	Description	
volser	the volser number	
coord	coordinates of the foreign volser	
aci_media_type	eType	the medium type of the volser as listed in the *DAS Administration Guide*
attrib	the attributes of the foreign volser	

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EBADCLIENT`
- `ENOTSUPPHCMD`

aci_mount

The aci_mount function mounts a volume in a drive. See Figure 4-82

```
#include "aci.h"
int aci_mount( char *volser,
               enum aci_media type,
               char *drive )
```

Figure 4-82 aci_mount Function Call

Mount the volume named *volser* of media type *type* in the drive named *drive*. The volume will then be available to the requesting client. See Table 4-35.

For additional information, refer to *aci_dismount* on page 4-16, and *aci_scratch_set* on page 4-114, and *aci_view* on page 4-123.

Table 4-35 Parameters for the aci_mount Function Call

Parameter	Description
volser	Volser which should be mounted (also foreign volser possible after the cataloging with carf)
type	Media type of the named volser Refer to *Media Types* on page 2-7
drive	Name of the drive which should be mounted. If the drive is set to the NULL string, the drive is automatically selected from the list of available drives to the client. The client host must have Automatic Volume Recognition (AVR) active to detect the mount.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `ENOVOLUME`
- `ENODRIVE`

- EDRVOCCUPIED
- EPROBVOL
- EAMU
- EAMUCOMM
- EROBOTCOMM
- EDASINT
- EDEVEMPTY
- ENOTAUTH
- EUPELSE
- EBADCLIENT
- ERETRYL
- EINUSE
- ENOTFOUND
- ECANCELLED
- EDASINT
- ENOMATCH
- ECLEANING
- ETIMEOUT
- ESWITCHINPROG
- EHICAPINUSE
- ECOORDINATE
- EBARCODE
- EINVALIDDEV
- ENOROBOT
- EDATABASE
- ENOTSUPPHCMD
- ECOORDINATE
- EPROBDEV
- EBARCODE
- EAREAFULL

Refer to Figure 4-83 on page 4-93 for an example of the aci_mount function.

```
/* Mount volume in specific drive */
int rc = 0;
rc = aci_mount( "AAB001", ACI_3590, "Drive2" );
if( rc )
{
          aci_perror( "Command failed:" );
}
else
{
         printf( "Mount of %s successful.\n", volser );
}
```

Figure 4-83 Example of the aci_mount Function

aci_partial_inventory

The aci_partial_inventory function inventories part of the AML (only in one device). See Figure 4-84.

```
#include "aci.h"
int aci_partial_inventory  (char *SourceCoor,
                            char *TargetCoor)
```

Figure 4-84 aci_partial_inventory Function Call

Start the compare between physical AML (barcode of the Volser or database of the Scalar 1000) with the AMU database and update the AMU database.

When aci_partial_invenotry is called with NULL stings for SourceCoor and TargetCoor parameters, a complete inventory will be invoked. If the archive is large, it could take several hours for the inventory to complete.

Refer to Table 4-36 on page 4-94 for a description of the parameters for the aci_partial_inventory function call.

Table 4-36 Parameters for the aci_partial_inventory Function Call

Parameter	Description
SourCoor	10 digit logical coordinate of the device for the start of the inventory (defined in the AMU database), e.g. L501010101
TargetCoor	10 digit logical coordinate of the device for the end of the inventory (defined in the AMU database), e.g. L501011310 Source and targeted must be in the same device.

⚠ **Attention**

The aci_partial_inventory function is intended for testing and startup. An error function will be displayed in the AMU log during operation (and not returned to the calling process). The database segments that are defined in the function cell will be overwritten with a symbolic volser "*Ixxxx" if the barcode reader malfunctions.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ERETRYL`
- `ENOTFOUND`
- `ECANCELLED`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCODE`

See Figure 4-85 for an example of the aci_partial_inventory function.

```
/* Partial Inventory in AML */
int rc = 0;
rc =aci_partial_inventory("L501010101", L501011310");

if( rc )
{
          aci_perror( "Command failed:" );
}
else
{
         printf( "Inventory started successful.\n");
}
```

Figure 4-85 Example of the aci_partial_inventory Function

aci_perror

The aci_perror function writes DAS error text to standard error. See Figure 4-86.

```
#include "aci.h"
int aci_perror( char *error_prefix )
```

Figure 4-86 aci_perror Function Call

The `aci_perror` function writes an error message to the standard error device, describing the last error encountered. The error message is either returned by the DAS server, or if not available, the error message is selected according to the value stored in `d_errno`. The parameter *error_prefix* is attached to the message. See Table 4-37.

Table 4-37 Parameter for the aci_perror Function Call

Parameter	Description
Error_prefix	message (error, warning or information) stored in d_errno

See Figure 4-87 for an example of the aci_perror function.

```
/* Write DAS error message with usr text prepended */
if ( d_errno )
{
        aci_perror( "This text is prepended: " );
}
```

Figure 4-87 Example of the aci_perror Function

aci_qversion

The aci_qversion function queries the version string of ACI and DAS component. See Figure 4-88.

```
#include "aci.h"
int aci_qversion( char *aciver,
               char *dasver )
```

Figure 4-88 aci_qversion Function Call

Query the version of the installed ACI and DAS component. This function allows the client to determine that the correct software component prerequisite is installed. See Table 4-38

Table 4-38 Parameters for the aci_qversion Function Call

Parameter	Description
aciver	displays the release of the used ACI (function library), e.g. 3.01
dasver	displays the release of the used DAS software on the AMU, e.g. 3.01

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ETIMEOUT`
- `ESWITCHINPROG`

Refer to Figure 4-89 on page 4-98 for an example of the aci_qversion function.

```
/* Check DAS and ACI version */
int  rc = 0;
char szACIVersion[ ACI_MAX_VERSION_LEN ] = "";
char szDASVersion[ ACI_MAX_VERSION_LEN ] = "";
if( ( rc = aci_qversion( (char *) &szACIVersion,
         (char *) &szDASVersion ) ) )
{
         aci_perror("Version request failed: ");
}
else
{
         printf("DAS-Version : %s\n", szDASVersion );
}
printf("ACI-Version : %s\n", szACIVersion ); /*always OK */
```

Figure 4-89 Example of the aci_qversion Function

aci_qvolsrange

The aci_qvolsrange queries the list of available volsers. See Figure 4-90.

```
#include "aci.h"
int aci_qvolsrange( char *startvolser,
               char *endvolser,
               int num_of_requested_volsers,
               char *client_name,
               int *number_of_returned_volsers,
               struct aci_volserinfo *list )
```

Figure 4-90 aci_qvolsrange Function Call

The function returns a list of volsers from the archive catalog. Figure 4-91 on page 4-99 shows the relation of the configuration to the amount of returned volsers.

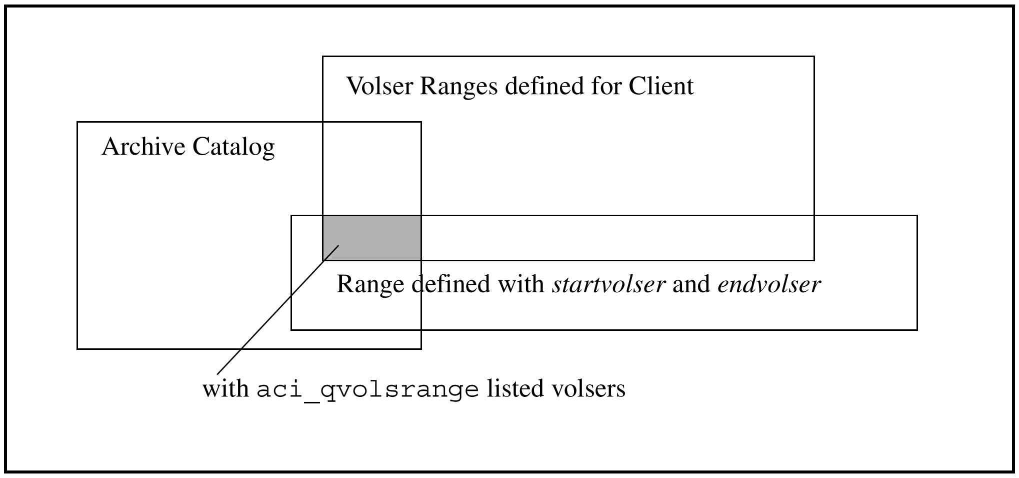

Volser Ranges defined for Client
Archive Catalog
Range defined with *startvolser* and *endvolser*
with aci_qvolsrange listed volsers

Figure 4-91 Amount of Listed Volsers

See Figure 4-92 for the aci_volserinfo function structure.

```
struct aci_volserinfo {
         char volser[ACI_VOLSER_LEN];
         enum aci_media media_type;
         char attrib;
};
```

Figure 4-92 Structure for the aci_volserinfo

See Table 4-39 for a description of the parameters for the aci_qvolsrange function call.

Table 4-39 Parameters for the aci_qvolsrange Function Call

Parameter	Description
startvolser	first volser in the range of displayed volsers
endvolser	last volser in the range of displayed volsers
num_of_requested_volser	number of volsers to be displayed
client_name	optional parameter to specify the volsers for a client other than the local one

Table 4-39 Parameters for the aci_qvolsrange Function Call

Parameter		Description
number_of_returned_volser		number of the volsers actually found in the range
aci_volserinfo	volser	barcode number of the volser
	media_type	media type of the volser
	attrib	status of the volser

If the *startvolser* and *endvolser* parameters are set to the NULL string ("" or `'\0'`), a search will start at the smallest database entry defined for the client and end at the largest database entry defined for the client. However, the maximum string length of *startvolser* must be the string length of `ACI_VOLSER_LEN`

Refer to Table 4-40 on page 4-100 for an explanation of the attrib values.

For further information, refer to *aci_view* on page 4-123.

Table 4-40 Explanation of the Attrib Values

Name	attrib	Explanation
ACI_VOLSER_ATTRIB_ MOUNTED	M	Mounted (slot empty, medium has been placed in a drive)
ACI_VOLSER_ATTRIB_ EJECTED	E	ejected (slot empty, medium has been placed in the I/O unit)
ACI_VOLSER_ATTRIB_ OCCUPIED	O	occupied (slot occupied, medium is in its home position)
ACI_VOLSER_ATTRIB_ UNDEFINED	U	undefined (special attribute used by HCC/MVS)
ACI_VOLSER_ATTRIB_ EMPTY	Y	empty (slot empty, no medium defined for the slot)
ACI_VOLSER_ATTRIB_ REVERSESIDEMOUNTED	R	reverse side mounted (slot empty, optical disk has been placed in the jukebox)
ACI_VOLSER ATTRIB_ JUKEBOX	J	in jukebox (slot empty, optical disk has been placed in the jukebox)
ACI_VOLSER_ATTRIB_ INITIAL	I	initial (attribute not used)

Table 4-40 Explanation of the Attrib Values

Name	attrib	Explanation
ACI_VOLSER_ATTRIB_ TEMP_HERE	T	temp here (slot occupied, medium in the problem box)
ACI_VOLSER_ATTRIB_ TEMP_AWAY	A	temp away (medium temporarily not at the specified coordinates, in transit on AML/2 with double robotic controller systems)

Return Values

- 0: The call was successful.
- 1: More data is available.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `EINVALID`
- `EDASINT`
- `ENOVOLUME`
- `EBADCLIENT`
- `ERPC`
- `ETIMEOUT`
- `ESWITCHINPROG`

See Figure 4-93 for an example of the aci_qvolsrange function.

```
/* Query volser range */
int  rc = 0;
char szBeginVolser[ ACI_VOLSER_LEN ] = "";
char szEndVolser  [ ACI_VOLSER_LEN ] = "";
char szClient     [ ACI_NAME_LEN ]   = "";
int  nCount = ACI_MAX_QUERY_VOLSRANGE;
int  nActualCount = 0;
int  nFor;
struct aci_volserinfo stVolsRange[ nCount ];
rc = aci_qvolsrange((char *) &szBeginVolser,
            (char *) &szEndVolser,
            nCount,
            (char *) &szClient,
            &nActualCount,
            (struct aci_volserinfo *) &stVolsRange );
if( rc == -1 )
         aci_perror("Command failed: ");
else
{
         printf( "\nnext volser %s", szBeginVolser );
         printf( "\ncount %ld", nActualCount );
         printf( "\n%smore
data",d_errno==EMOREDATA?"":"no" );
         for( nFor = 0; nFor < nActualCount; nFor++ )
         printf( "\nvolser %s media %ld attrib %c",
         stVolsRange[ nFor ].volser,
         stVolsRange[ nFor ].media_type,
         stVolsRange[ nFor ].attrib       );
}
```

Figure 4-93 Example of the aci_qvolsrange Function

aci_register

The aci_register function registers a client. See Figure 4-94.

```
#include "aci.h"
int aci_register( char *client,
               char *host,
               enum aci_command action,
               bool_t avc,
               bool_t complete,
               bool_t dismount,
```

Figure 4-94 aci_register Function Call

Clients are defined for the DAS server by the DAS configuration file. However, clients may be registered temporarily with the `aci_register` function. This function registers client information and serves three purposes:

- create a new client profile
- change an existing client profile
- remove a client profile

When the DAS server is shut down, the change is lost. To permanently create, modify or delete a client, the administrator must edit the definition in the DAS configuration file. See Table 4-41.

Table 4-41 Parameters for the aci_register Function Call

Parameter	Description	
client	name of the client which configuration should be changed	
host	either the hostname or IP address where the client resides	
action	action with the client	
	ACI_ADD	new temporary client
	ACI_MODIFY	change an existing client
	ACI_DELETE	delete a client

Table 4-41 Parameters for the aci_register Function Call

Parameter	Description	
avc	true or false	option avoid volume contention (Refer to the *DAS Administration Guide)*
complete	true or false	selection between basic or complete command set (Refer to *Client Services* on page 2-4)
dismount	true or false	optional dismount; for configuring an automatic dismount without DAS command (Refer to the *DAS Administration Guide*)

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `EDASINT`
- `EBADHOST`
- `ENOSPACE`
- `ETIMEOUT`
- `ESWITCHINPROG`

Refer to Figure 4-95 on page 4-105 for an example the aci_register function.

```
/* Modify existing client configuration */
int rc =0;
char *client = "SomeClient";
char * ipname = "CLIENT1";
enum aci_command action;
short avc;
short c;
short dism;
int i, j;
struct aci_client_entry client_entry;
rc = aci_clientstatus (client, &client_entry);
if ( !rc )
{
         action = ACI_MODIFY;
         avc = client_entry.avc;
         dism= client_entry.dismount;
         c = FALSE;

rc=aci_register(client,ipname,action,avc,c,dism);
         if( rc )
            aci_perror("Command failed: ");
         else
            printf ("Client %s updated.\n", client);
}
```

Figure 4-95 Example of the aci_register Function

aci_robhome

The aci_robhome function sets the robot (accessor in Scalar 1000) in the AML to off-line and moves it to the home position. See Figure 4-96.

```
#include "aci.h"
int aci_robhome(char *szRobot)
```

Figure 4-96 aci_robhome Function Call

The call of this function sends the robot with number ‘szRobot’ to the home position. See Table 4-42.

Table 4-42 Parameter for the aci_robhome Function Call

Parameter	Description	
szRobot	the defined robot of a twin AML system	
	R1	Robot 1 of the AML (must also be used for all single AML systems)
	R2	Robot 2 of the twin AML

Return Values

- 0: The call was successful
- -1: The call failed

The external variable d_errno is set to one of the following DAS error codes

- `ERPC`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ERETRYL`
- `ECANCELLED`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`

- `EHICAPINUSE`
- `ECOORDINATE`
- `EDATABASE`
- `ENOTSUPPHCMD`

See Figure 4-97 for an example of the aci_robhome function.

```
/* Robot Homing */
int rc = 0;
rc =aci_robhome("R1");
if( rc )
{
          aci_perror( "Command failed:" );
}
else
{
         printf( "robot is now home.\n");
}
```

Figure 4-97 Example of the aci_robhome Function

aci_robstat

The aci_robstat function starts the robot or gets the status of the robot. See Figure 4-98.

```
#include "aci.h"
int aci_robstat(char *szRobot , char *szAction)
```

Figure 4-98 aci_robstat Function Call

See Table 4-43 for a description the parameters for the aci_robstat function call.

Table 4-43 Parameters for the aci_robstat Function Call

Parameter	Description	
szRobot	the defined robot of a twin AML system	
	R1	Robot 1 of AML (must also be used for all single AML system)
	R2	Robot 2 of the twin AML
szAction		
	START or start	in szRobot, set the defined Robot to on-line
	STAT or stat	ask the AMS for the status of the robots, parameter szRobot must set to NULL

Return Values

- 0: The call was successful
- -1: The call failed

The external variable d_errno is set to one of the following DAS error codes

- `ERPC`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ERETRYL`
- `ECANCELLED`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `ECOORDINATE`
- `EDATABASE`
- `ENOTSUPPHCMD`

Refer to Figure 4-99 on page 4-109 for an example of the aci_robstat function.

```
/* Check robot status */
int  rc;
char szRobot[ 3 ] = "R1";
char szAction[ 5 ] = "STAT";
if( ( rc = aci_robstat(szRobot, szAction)))
{
         aci_perror("Version request failed: ");
}
else
{
         printf("Robstat 1 : %s\n", szSourceCoord );
         printf (“robstat sucessful\n)
}
```

Figure 4-99 Example of the aci_robstat Function

aci_scratch_get

The aci_scratch_get function gets a scratch volume. See Figure 4-100.

```
#include "aci.h"
int aci_scratch_get( char *subpool,
                  enum aci_media type
                  char *volser,)
```

Figure 4-100 aci_scratch_get Function Call

Get a scratch volume from *subpool* and return the volume name in *volser*. The displayed volume is after the command in the database automatically set to unscratch. Refer to Table 4-44 on page 4-110.

For additional information, refer to *aci_scratch_set* on page 4-114, and *aci_scratch_unset* on page 4-116.

Table 4-44 Parameters for the aci_scratch_get Function Call

Parameter	Description
subpool	stored name of the pool for scratch media in the AMU database If the volume is to be taken from a default subpool of a media type, set type to the media type of the volser and set subpool to the NULL string ("" or '\0')
type	type of media in the named subpool (Refer to *Media Types* on page 2-7)
volser	first scratch volser of the named subpool found in the AMU database, independent of the status of the volser (ejected, mounted, etc.)

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `EINVALID`
- `EDASINT`
- `ENOPOOL`
- `ERETRYL`
- `ERPC`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EDATABASE`
- `ENOTSUPPHCMD`

Refer to Figure 4-101 on page 4-111 for an example of the aci_scratch_get function.

```
/* Get volser of new scratch media */
int  rc;
char *pszPoolName = "";
enum aci_media type = ACI_3590;
char szVolser[ACI_VOLSER_LEN];
rc = aci_scratch_get (pszPoolName, type, pszVolser);
if( rc )
{
   aci_perror ("Command failed: ");
}
else
{
   printf ("Scratch volser %s found.\n", szVolser);
}
```

Figure 4-101 Example of the aci_scratch_get Function

aci_scratch_info

The aci_scratch_info function gets information about a scratch pool. See Figure 4-102.

```
#include "aci.h"
int aci_scratch_info( char *subpool,
                  enum aci_media type,
                  long *volcount,
                  long *scratchcount )
```

Figure 4-102 aci_scratch_info Function Call

Query information regarding the scratch pool with the name *subpool*, or the default scratch pool, if the subpool name is not specified. The information returns the number of volsers defined in the pool and the number of volsers that have been selected as scratch. Refer to Table 4-45 on page 4-112.

For additional information, refer to *aci_scratch_unset* on page 4-116.

Table 4-45 Parameters for the aci_scratch_info Function Call

Parameter	Description
subpool	stored name of the pool for scratch media. If the information is to be taken from a default subpool of a media type, set type to the media type of the volser and seat subpool to the NULL string (““ or ‘\0’)
type	type of the media in the subpool (Refer to *Media Types* on page 2-7)
volcount	number of volsers in the named pool (independent of the status scratch or not defined)
scratchcount	number of the volser with the status scratch in the named subpool, independent of the status of the volser (eject, mounted, etc.)

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `EINVALID`
- `EDASINT`
- `ENOPOOL`
- `ERPC`
- `ETIMEOUT`
- `ESWITCHINPROG`

Refer to Figure 4-103 on page 4-113 for an example of the aci_scratch_info function.

```
/* List scratch pool information */
int  rc;
char *pszPoolName = "";
enum aci_media type = ACI_3590;
long lVolserCount, lScratchCount;
if ((rc =
               aci_scratch_info (pszPoolName, type,
                 &lVolserCount, &lScratchCount)) != 0
)
{
         aci_perror ("Command failed: ");
}
else
{
         if (strcmp (pszPoolName, "") == 0)
         {
            printf ("DEFAULT_POOL:VolserCount:%d,
                 ScratchCount:%d\n",
                 lVolserCount, lScratchCount);
         }
         else
         {
         printf ("%s: VolserCount: %d,
               ScratchCount: %d \n",
               pszPoolName, lVolserCount,
               lScratchCount);
         }
}
```

Figure 4-103 Example of the aci_scratch_info Function

aci_scratch_set

The aci_scratch_set function sets volume status to scratch. See Figure 4-104.

```
#include "aci.h"
int aci_scratch_set( char *subpool,
                  enum aci_media type,
                  char *volser )
```

Figure 4-104 aci_scratch_set Function Call

Set volume named *volser* of media type *type* to `scratch` status in the scratch pool named s*ubpool*.

The subpool is created if it does not exist. If a subpool is not specified and set to the NULL string ("" or `'\0'`), the default pool will be used.

If the volume is already defined to another pool, an error (`EOTHERPOOL`) will be returned. If this is a new scratch volume, *subpool* will be the pool from which the volume can be selected using the `aci_get_scratch` function. See Table 4-46.

Table 4-46 Parameters for the aci_scratch_set Function Call

Parameter	Description
subpool	the stored name of the pool for scratch media in the AMU database. If the information is to be taken from a default subpool of a media type set type to the media type of the volser and set subpool to the NULL string (““ or ‘\0’)
type	type of the media in the subpool (Refer to *Media Types* on page 2-7)
volser	set volser to scratch in the named subpool in the AMU database (only possible, if the volser is already defined in the AMU database but the status of the volser (ejected, mounted, etc.) is not relevant).

Attention

Data stored on the media may be lost. The command automatically sets the specified medium as a scratch medium (without a confirmation prompt). The data on the medium is overwritten by the next scratch mount command.

The command will be rejected with the message EOTHERPOOL if the medium already exists in another scratch pool.

For additional information, refer to *aci_scratch_unset* on page 4-116.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `EINVALID`
- `EDASINT`
- `ENOPOOL`
- `ERPC`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EDATABASE`
- `ENOTSUPPHCMD`

See Figure 4-105 for an example of the aci_scratch_set function.

```
/* Add volume to default scratch pool */
int  rc;
char *pszPoolName = "";
char *pszVolser   = "VOL001";
enum aci_media type = ACI_3590;
rc = aci_scratch_set(pszPoolName, type, pszVolser);
if( rc )
{
         aci_perror ("aci_scratch_set failed");
}
else
{
          printf("Scratch volume %s set.\n ",
pszVolser);
}
```

Figure 4-105 Example of the aci_scratch_set Function

aci_scratch_unset

The aci_scratch_unset function resets the scratch status of a volume. See Figure 4-106.

```
#include "aci.h"
int aci_scratch_unset( char *subpool,
                  enum aci_media type,
                  char *volser )
```

Figure 4-106 aci_scratch_unset Function Call

Reset the status of volume named *volser* in the scratch pool named *subpool* from scratch to non-scratch media and remove the *volser* from *subpool*. If *subpool* is set to the NULL string ("" or `'\0'`) the default pool will be used. If the requested volume is the last volume in the pool, the pool will be deleted. See Table 4-47.

For additional information, refer to *aci_scratch_set* on page 4-114.

Table 4-47 Parameters for the aci_scratch_unset Function Call

Parameter	Description
subpool	the stored name of the pool for scratch media in the AMU database. If the information is to be taken from a default subpool of a media type set type to the media type of the volser and set subpool to the NULL string ("" or '\0')
type	type of the media in the subpool (Refer to *Media Types* on page 2-7)
volser	set volser to scratch in the named subpool in the AMU database (only possible, if the volser is already defined in the AMU database but the status of the volser (ejected, mounted, etc.) is not relevant).

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `EINVALID`
- `EDASINT`
- `ENOPOOL`
- `ERPC`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EDATABASE`
- `ENOTSUPPHCMD`

See Figure 4-107 for an example of the aci_scratch_unset function.

```
/* Change volume’s scratch status */
int  rc;
char *pszPoolName = "";
enum aci_media type = ACI_3590;
char *pszVolser   = "VOL001"

rc = aci_scratch_unset(pszPoolName,type,pszVolser);
if( rc )
{
         aci_perror ("aci_scratch_unset failed");
}
else
{
         printf ("Volser %s removed from
scratchpool.\n",
               pszVolser);
}
```

Figure 4-107 Example of the aci_scratch_unset Function

aci_shutdown

The aci_shutdown function shuts down the DAS software. See Figure 4-108.

```
#include "aci.h"
int aci_shutdown (char *now)
```

Figure 4-108 aci_shutdown Function Call

This function shuts down the DAS server. Once the request has been accepted, no other request will be accepted. A restart of the DAS server is necessary to resume DAS operations. See Table 4-48.

Table 4-48 Parameter for the aci_shutdown Function Call

Parameter	Description
now	By default, DAS waits for outstanding requests to be completed before termination. It however, the now parameter or the now string is specified, the shut down is immediate.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EDASINT`
- `ESWITCHINPROG`

Refer to Figure 4-109 on page 4-119 for an example of the aci_shutdown function.

```
/* Shut down DAS operation */
int rc;
char *now = NULL;
if ((rc = aci_shutdown(now)))
{
         aci_perror("Shut down failed:");
}
else
{
         printf("Shut down successful.\n");
}
```

Figure 4-109 Example of the aci_shutdown Function

aci_snmp

The aci_snmp function sends SNMP messages. See Figure 4-110.

```
#include "aci.h"
int aci_snmp (char *Msg)
```

Figure 4-110 aci_snmp Function Call

See Table 4-49.

Table 4-49 Parameter for the aci_snmp Function Call

Parameter	Description
Msg	defines the SNMP message

Only supported by the Scalar DLC software.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ERPC`
- `EINVALID`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EBADCLIENT`
- `ENOTSUPPHCMD`

aci_switch

The aci_switch function switches the passive AMU to active. See Figure 4-111.

```
#include "aci.h"
int aci_switch (char *Switch)
```

Figure 4-111 aci_switch Function Call

Allow the second AMU to be used without manual intervention, if the first AMU-PS is down. An installed and running Dual-AMU is necessary for this function. See Table 4-50.

Table 4-50 Parameter for the aci_switch Function Call

Parameter	Description	
Switch	defines the priority of the switch	
	-n	normal switch, allowed to complete all running commands, and synchronization of the database before switching will start
	-f	force switch, immediately without any synchronization to the second AMU, but necessary if the first AMU is already down

Caution

Only use the -f option if the -n option is no longer functioning. There is a risk that anomalies may be caused in the AMU database.

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `ENOTAUTH`
- `ERPC`
- `EINVALID`
- `EDASINT`
- `ETIMEOUT`
- `EAMUCOMM`
- `EHICAPINUSE`
- `ESWITCHINPROG`
- `ENOTSUPPHCMD`

See Figure 4-112 for an example of the aci_switch function.

```
/* Switch to the Dual AMU */
int rc;
char *Switch = "-n";
if ((rc = aci_switch(Switch)))
{
         aci_perror("Swicht actual not possible:");
}
else
{
         printf("Now the Dual AMU is active.\n");
}
```

Figure 4-112 Example of the aci_switch Function

aci_unload

The aci_unload function presses one or more buttons on a drive in the AML. See Figure 4-113.

```
#include "aci.h"
int aci_unload(char *szDrive)
```

Figure 4-113 aci_unload Function Call

See Table 4-51 for an description of the parameter for the aci_unload function.

Table 4-51 Parameter for the aci_unload Function Call

Parameter	Description
szDrive	Name of the drive for unload, defined in Description in the AMS configuration

Return Values

- 0: The call was successful
- 1: The call failed

The external variable d_errno is set to one of the following DAS error codes

- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EPROBDEV`
- `ERPC`
- `EINVALID`
- `EDASINT`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EINVALIDDEV`
- `ENOROBOT`
- `EDATABASE`
- `ENOTSUPPHCMD`
- `EAREAEMPTY`
- `EPROBDEV`

aci_view

The aci_view function is used to view a database entry for a volume. See Figure 4-114.

```
#include "aci.h"
int aci_view( char *volser,
            enum aci_media type,
            struct aci_volume_desc *desc )
```

Figure 4-114 aci_view Function Call

Return the database entry for the volume named *volser* of media type *type*. The database entry is returned in the `aci_volume_desc` structure. See Figure 4-115.

For additional information, refer to *aci_qvolsrange* on page 4-98, and *aci_inventory* on page 4-81.

```
struct aci_volume_desc {
 char coordinate[ACI_COORD_LEN];
 char owner;
 char attrib;
 char type;
 char volser[ACI_VOLSER_LEN]
 char vol_owner;
 int use_count;
 int crash_count;
};
```

Figure 4-115 Volser Information Contained in the aci_volume_desc Structure

The structure returns the AMU archive catalog, which robot owns the volume, a volume attribute of `mounted`, `ejected`, `occupied`, or `undefined`, and the volser media type. Refer to Table 4-51 on page 4-122.

Table 4-52 Parameters for the aci_view Function Call

Parameter	Description	
volser	volser specifying the medium for which information is being queried	
type	media type to which the volser belongs Refer to *Media Types* on page 2-7	
aci_volume_desc	structure with the returned data from the AMU database table COORDINATES	
	coordinate	10-digit logical coordinate specifying the slot (Refer to the *AMU Reference Manual*)
	owner	number of the robot which can access the home coordinate of the named volser (1 = robot 1, 2 = robot 2, 3= both robots)
	attrib	current status of the slot (attributes) (Refer to Table 4-54 on page 4-125)
	type	type of slot, but not the media type (coordinate in the archive) (See Table 4-53)
	volser	queried volser (search criterion in the AMU database)
	vol_owner	not used
	use_count	number of accesses to the slot (not volser) by the robotic controller
	crash_count	not used

See Table 4-53 for an explanation of the table types.

Table 4-53 Table Types

attrib	Explanation
A	AMU Dynamic (dynamic storage locations in the AML system)
S	storage (dynamic storage locations in the AML system)
N	clean (cleaning media storage location)

See Table 4-54 for an explanation of the table attributes.

Table 4-54 Table Attributes

attrib	Explanation
O	occupied (slot occupied, medium is in its home position)
E	ejected (slot empty, medium has been placed in the I/O unit)
M	mounted (slot empty, medium has been placed in a drive)
I	initial (attribute not used)
J	in jukebox (slot empty, optical disk has been placed in the jukebox)
R	reverse side mounted (slot empty, optical disk has been placed in a drive)
Y	empty (slot empty, no medium defined for the slot)
U	undefined (special attribute, used by HCC/MVS)
T	temp here (slot occupied, medium in the problem box)
A	temp away (medium temporarily not at specified coordinated, in transit on AML/2 with double robotic controller systems)

Return Values

- 0: The call was successful.
- -1: The call failed.

The external variable `d_errno` is set to one of the following DAS error codes:

- `EAMU`
- `EAMUCOMM`
- `EINVALID`
- `EDASINT`
- `ENOVOLUME`
- `ERPC`
- `ETIMEOUT`
- `ESWITCHINPROG`

Refer to Figure 4-116 on page 4-126 for an example of the aci_view function.

```
/* Get volume information */
int rc;
enum aci_media type = ACI_3590;
char *volser = "";
struct aci_vol_desc desc;
if ( rc = aci_view(volser, type, &desc ))
{
  aci_perror("view failed");
}
else
{
  printf("Volser=%s Type=%c Attrib=%c\n\tcoordinate
=%s\n"
            "\tuse count = %d\n\",
            desc.volser,
            desc.type,
            desc.attrib,
            desc.coord,
            desc.use_count;
}
```

Figure 4-116 Example of the aci_view Function

aci_volser_inventory

The aci_volser_inventory function inventories the volsers within the AML. See Figure 4-117.

```
#include "aci.h"
int aci_volser_inventory(char *pszVolser, int status)
```

Figure 4-117 aci_volser_inventory Function Call

Start the comparison between the physical barcode volser and the AMU database and update the AMU database as necessary. Refer to Table 4-55 on page 4-127.

Table 4-55 Parameters for the aci_volser_inventory Function Call

Parameter	Description	
pszVolser	Pointer to an individual volser	
status	Indicator to determine manipulate of the database	
	ACI_INVT_NOTUPDT	0 = do not update the database
	ACI_INVT_UPDT	0 = update the database

Return Values

- 0: The call was successful
- -1: The call failed

The external variable d_errno is set to one of the following DAS error codes

- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EPROBVOL`
- `EAMU`
- `EAMUCOMM`
- `EROBOTCOMM`
- `EDASINT`
- `ERETRYL`
- `ENOTFOUND`
- `ECANCELLED`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EHICAPINUSE`
- `ECOORDINATE`
- `EBARCODE`
- `ENOTSUPPHCMD`

aci_volseraccess

The aci_volseraccess function sets ownership of a volser or range of volsers. See Figure 4-118.

```
#include "aci.h"
int aci_volseraccess(char *ClientName ,
                     char *VolserRange,
                     enum aci_volser_status status)
```

Figure 4-118 aci_volseraccess Function Call

Modify allocation status of a volser for a specified client. Refer to Table 4-56 .

Table 4-56 Parameters for the aci_volseraccess Function Call

Parameter	Description	
clientName	name of the client which authorization of a volser range should be changed	
VolserRange	Range of volser for the modification in one of the following form: • a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen	
status	new authorization status of the drive	
	ACI_UP	reservation of named volser for the client
	ACI_DOWN	Deleted reservation of named volser for the client. Only complete ranges can be deleted not subranges from another range

Return Values

- 0: The call was successful
- -1: The call failed

The external variable d_errno is set to one of the following DAS error codes

- `EUPELSE`
- `ERPC`
- `EINVALID`
- `ENOVOLUME`
- `EDASINT`
- `EBADCLIENT`
- `ENOTAUTH`
- `ETIMEOUT`
- `ESWITCHINPROG`
- `EUPOWN`

aci_volserstatus

The aci_volserstatus function queries ownership of the volser. See Figure 4-119.

```
#include "aci.h"
int aci_volserstatus  char *ClientName,
         int *num
         char *VolserRange,
         struct aci_volser_entry *VolserEntry[ACI_MAX_VOLSER_ENTRIES]
```

Figure 4-119 aci_volserstatus Function Call

Returns the status of volsers which are set to UP for the client with name clientname. If clientname is the NULL string, the call return status on all volsers specified in VolserRange. The status is returned in VolserEntry, an array of pointers to aci_volser_entry structure.

The structure aci_volser_entry is stated, in *aci.h*, see Figure 4-120

```
struct aci_volser_entry {
         char ClientName[ACI_NAME_LEN];
         char VolserStatus[ACI_VOLSERSSTATUS_LEN];
         char volser[ACI_VOLSER_LEN];
         enum aci_media media_type;
}
```

Figure 4-120 aci_volser_entry Structure

See Table 4-57 for a description of the parameters for the aci_volserstatus function call.

Table 4-57 Parameters for the aci_volserstatus Function Call

Parameter	Description	
ClientName	name of the client which allocated volser should be displayed	
num	number of volser ranges	
VolserRange	Range of the allocated Volser in on of the following forms: • a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen	
aci_volser_entry	details to the allocated range	
	ClientName	Owner of the volser
	VolserStatus	UP
	volser	allocated volser range
	media_type	media type to which the volser belongs Refer to *Media Types* on page 2-7

Return Values

- 0: The call was successful
- -1: The call failed

The external variable d_errno is set to one of the following DAS error codes

- `EBADCLIENT`
- `ERPC`
- `EINVALID`
- `EDASINT`
- `ETIMEOUT`
- `EHICAPINUSE`
- `ESWITCHINPROG`

5

DAS ACI 3.0 Asynchronous Support Layer

Overview

Asynchronous Support Layer library works as a filter between existing ACI libraries (version 3 or later) and the client application, which wants to issue asynchronous ACI calls. Since the native ACI interface is synchronous, a client had to wait for the completion of the request. Now using special programming techniques, developers who want to use ACI in an asynchronous manner can use the Asynchronous Support Layer library of this purpose.

How It Works

In early ACI releases, all programmers using the ACI library, had to consider that any ACI call issued by the application would wait until the work is complete. Now, each incoming request is stored into the shared memory area and each request is executed in forked process.

After the application issues a request, it is given a request ID. The request ID is actually the process ID of the forked child process. The operating system does not allow two processes with the same ID. The request ID ensures the uniqueness of each ID.

After the request is completed, it stores the results in the appropriate location in that shared table and terminates. To retrieve the information back to the client, define a special function that terminates (SIGCHLD) a child process. This routine could also transmit the data into an application internal data buffer and free the entry in the table. Doing this ensures that the entry (process) ID will never be repeated in the shared memory table.

Sadmin Sample Application

For the convenience and ease of understanding the programming techniques required to write the asynchronous ACI applications, the sadmin sample application is provided. This application represents a cut version of the standard dasadmin application. It the following commands:

- mount: mount a volser
- dismount: dismount the volser or force the drive to dismount
- insert: insert volsers

- eject: eject/elect complete volsers
- allocv: allocate volser
- allocd: allocate drive
- script: run the commands from a script file

This application can run in both single command and scripting modes. To stop this application, press <Ctrl>+<C> or use the **KILL** command.

Sadmin Syntax

The syntax of the sadmin sample application has two general forms:

- sadmin <command> [-h] | [[<arg1>] [<arg2>] ... [<argN>]]
- sadmin script < [<script file name>]

In the first case, sadmin is running in single command mode. In the second case, it is executing the specifically prepared script. The script contains all usual sadmin commands, written in same syntax. For example:

```
mount -t DECDLT 000030 DE02
eject -t OD-Thick P702010101-P702010122 E02
dism -d DE02
```

Each line must be EOL terminated. Sadmin responds with a help message when run without arguments. The script command isn’t included in the message.

Contents of the Async Support Layer Library

Currently the Asynchronous Support Layer library supports only the following commands:

- MOUNT
- DISMOUNT
- FORCE
- INSERT
- EJECT
- EJECT COMPLETE

The asynchronous API consists of the following four functions:

aci_async_add -	adds an entry in the shared memory area
aci_async_create -	creates and initializes the shared memory area
aci_async_find -	finds required shared memory table entry
aci_async_free -	frees required shared memory table entry

There is also a header file for C developers, *aci_async.h*, defining all the required structures and macros for development of software using the ACI asynchronous support.

aci_async_add()

The aci_asyn_add() function adds a new aci_async_entry element to the shared memory array. See Figure 5-1.

```
#include "aci_async.h"
#include "aci_xdr.h"
aci_async_entry* aci_async_add(int aci_func, ...);
```

Figure 5-1 Example of a Generic aci_async_add() Function

Parameters

The aci_func parameters are as follows:

- das_mount
- das_dismount
- das_force
- das_insert
- das_eject
- das_eject_complete

das_mount

The aci_async_add function with the das_mount parameter mounts a volume in a drive. See Figure 5-2

```
aci_asycn_add (das_mount, *volser, *type, *drive)
```

Figure 5-2 aci_async_add Function with the das_mount Parameter

Mount the volume named *volser* of media type *type* in the drive named *drive*. The volume will then be available to the requesting client. See Table 5-1.

Table 5-1 Parameters for the das_mount Parameter

Parameter	Description
volser	Volser which should be mounted (also foreign volser possible after the cataloging with carf)
type	Media type of the named volser Refer to *Media Types* on page 2-7
drive	Name of the drive which should be mounted. If the drive is set to the NULL string, the drive is automatically selected from the list of available drives to the client. The client host must have Automatic Volume Recognition (AVR) active to detect the mount.

Refer to Figure 5-3 on page 5-6 for an example of the aci_async_add function with the das_mount parameter.

```
aci_async_add (DAS_MOUNT, “000030”, C3480, “DE02”)
```

Figure 5-3 Example of the aci_async_add Function with the das_mount Parameter

das_dismount

The aci_async_add function with the das_dismount parameter dismounts a volume. See Figure 5-4.

```
aci_async_add (das_dismount, *volser, *type)
```

Figure 5-4 aci_async_add Function with the das_dismount Parameter

Dismount the volume *volser* of defined media type from its drive. The drive is identified by the DAS software.

For additional information, refer to *das_mount* on page 5-6.

Retries can be configured in the config file or in the AMS configuration.

See Table 5-2 for a description of the parameters for the aci_async_add function with the das_dismount parameter.

Table 5-2 Parameters for the das_dismount Parameter

Parameter	Description
volser	volser that is to be moved from a drive to the original position in the AML
type	media type of the named volser Refer to *Media Types* on page 2-7

das_force

The aci_async_add function with the das_force parameter dismounts any cartridge from a specific drive. See Figure 5-5.

```
aci_async_add (das_force, *drive)
```

Figure 5-5 aci_async_add Function with the das_force Parameter

Force a dismount from a drive named *drive*, regardless of which volser is in the drive. Refer to Table 5-3 on page 5-8.

For additional information, refer to *das_dismount* on page 5-7, and *das_mount* on page 5-6.

Table 5-3 Parameter for the das_force Parameter

Parameter	Description
drive	name of the drive for the dismount

das_insert

The aci_async_add function with the das_insert parameter inserts volumes into the AML. See Figure 5-6.

```
aci_async_add (das_insert, *insert_area, *volser_range,
*type)
```

Figure 5-6 aci_async_add Function with the das_insert Parameter

This function is for compatibility. Please use the aci_insertgen function.

The cartridges which are actually located in the insert area of the AML and registered by the AMU (automatically inventoried after closing the area) will be inserted.

- Volsers which already have a entry in the AMU database, will be moved to this position.
- Volsers which have no entry in the AMU database, will be moved to the first free slot (AMU database Attribute: "Empty" of this media type of the Type AMU-Dynamic)
- Cartridges with an invalid volser (barcode not readable) will moved to the problembox.

See Table 5-4 for a description of the parameters for the aci_insert function call.

Table 5-4 Parameters for the das_insert Parameter

Parameter	Description
insert_area	Logical insert range of the Insert/Eject unit of the AML in the AMS (e.g. I03)
volser_range	The volumes found in the insert_area are returned in the volser_ranges array of strings, where each volser is separated by a comma. Each range is ACI_RANG_LEN long and there are up to ACI_MAX_RANGES ranges. An empty string indicates the end of the ranges
type	media type of the volser in the insert area Refer to *Media Types* on page 2-7

Use the insert2 function instead of this command. This function experiences difficulties with large I/O units with long volsers (16-digit) since the buffer for displaying the inserted volser is restricted. For compatibility reasons this function continues to be supported.

das_eject

The aci_async_add function with the das_eject parameter ejects a range of volumes from the AML. See Figure 5-7.

```
aci_async_add (das_eject, *eject_area, *volser_ranges,
*type)
```

Figure 5-7 aci_async_add Function with the das_eject Parameter

Eject the volumes in *volser_range* to the *eject_area*. The media type of the volumes must match that of the *eject_area*. Set *type* to the media type of the *volser_range*. See Table 5-5.

Table 5-5 Parameters for the das_eject Parameter

Parameter	Description
eject_area	Logical area defined in AML, where the cartridges should be ejected
volser_ranges	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen
types	media type of the named volser Refer to *Media Types* on page 2-7

The database entry for the volume is not deleted, and the position in the AML that the volume occupied remains reserved for insertion of a volume with a matching volser. This could be useful if the volume is temporarily ejected and will be inserted in the near future. In such case, the position remains reserved and the volume's location within the AML does not change.

The eject will stop, when the eject area is full. The request will either be cancelled or completed after the area is marked empty depending on the `DAS_EJECTAREAFULL` environment variable.

das_eject_complete

The asi_async_add function with the das_eject_complete parameter ejects volumes and removes the database entries. See Figure 5-8.

```
aci_async_add (das_eject_complete, eject_area,
volser_range, type)
```

Figure 5-8 aci_async_add Function with the das_eject_complete Parameter

Eject the volumes in *volser_range* to the *eject_area*. The media type of the volumes must match that of the *eject_area*. Set *type* to the media type of the *volser_range.* The database entry for the volume is deleted, and the position in the AML that the volume occupied becomes free. This could be useful if the volume is to be placed in long-term archive storage or more space is needed in the AML. If the volume is re-inserted into the AML, it is stored in the next available position, and it probably will not occupy the previous position. The volume is re-inserted in the same position only if `das_eject` is used. See Table 5-6 for a description of the parameters for the das_eject_complete parameter.

Table 5-6 Parameters for the das_eject_complete Parameter

Parameter	Description
eject_area	Logical area defined in AMS, where the cartridges should be ejected
volser_range	• a single volser • multiple volsers separated by commas • a range of volsers separated by a hyphen
types	media types of the named volser Refer to *Media Types* on page 2-7

The eject will stop, when the eject area is full. Depending on the `DAS_EJECTAREAFULL` environment variable, the request will either be cancelled or completed after the area is marked empty.

For additional information, refer to *das_eject* on page 5-9, and *das_insert* on page 5-8.

Return Values

The call was successful if a pointer to the newly added entry is returned.

The call failed if zero is returned.

Refer to Figure 5-9 on page 5-12 for an example of the aci_async function call.

```
. . .
/*
 * sadmin sample application, dasadmin.c file
 * aci_async.h, ACI_INSERT macro
 */
aci_async_entry *async_entry;
if ((async_entry = aci_async_add(
DAS_INSERT,
insert_area,
volser_ranges,
type ) ) != 0)
{
  if ((async_entry->pid = fork()) == 0)
  {
    /* map the current process virtual memory to the
shared buffer */
    if ((int)(async_table =
(void*)shmat(async_table_desc, 0, 0)) == -1)
    {
      d_errno = ENOSHARED;
      strcpy(d_text, "shmat failed");
    }

    res = aci_insert(insert_area, volser_ranges, type);

    async_entry->d_errno = d_errno;
    strcpy(async_entry->d_text, d_text);

    shmdt(async_table);
    exit(res);
  }
}
. . .
```

Figure 5-9 Example of an aci_async_add Function

aci_async_create()

The aci_async_create function creates a shared memory array of the *entry_num* size, that will hold the requests sent. See Figure 5-10

```
#include "aci_async.h"
aci_async_entry* aci_async_create(long entry_num);
```

Figure 5-10 aci_async_create Function Call

See Table 5-7 for a description of the parameter for the aci_async_create function.

Table 5-7 Parameter for the aci_async_create Function

Parameter	Description
entry_num	entry_num - number of entries allocated to hold the request information.

Return Values

The call was successful if a pointer to the start of shared memory array mapped into virtual address space of the process.

The call failed zero is returned.

The entry_num value and pointer returned are stored globally in the Asynchronous Layer Library for later use. You must call this function only once. Repeated calls will destroy global values and will not free previously allocated shared memory arrays.

Refer to Figure 5-11 on page 5-14 for an example of the aci_async_create function.

```
. . .
/*
 * sadmin sample application, dasadmin.c file
 * creating a table in shared memory - 50 entries
 */
if( ( async_entry = aci_async_create( 50L ) ) == 0 )
{
  aci_perror( "create async failed" );
  return 1;
}
. . .
```

Figure 5-11 Example if the aci_async_create Function

aci_async_find()

The aci_async_find function looks for the appropriate aci_async_entry in shared memory array. If an entry has the same value in the *pid* field as *pid* parameter it is returned. See

```
#include "aci_async.h"
aci_async_entry* aci_async_find(pid_t pid);
```

Figure 5-12 aci_async_find Function Call

See Table 5-8 for a description of the parameter for the aci_async_find function.

Table 5-8 Parameter for the aci_async_find Function

Parameter	Description
pid	identifies the request entry in the shared memory array

Return Values

The call was successful if a pointer to the found value is returned.

The call failed if zero is returned

The structure of the aci_async_entry uses a pid_t type member pid as a unique identifier of an entry. When calling several asynchronous calls at a time the system can assign the same process id to a new process when a previous call has terminated. This could cause duplicate values the in pid fields of shared memory array entries. Refer to *Response Technique* on page 5-21.

See Figure 5-13 for an example of the aci_async_find function.

```
. . .
/*
 * sadmin sample application. wait_for_child.c file
 */
if ((async_entry = aci_async_find(pid)) != 0)
{
  printf("results: getting results...\n");
  switch(async_entry->aci_func)
  {
    case DAS_MOUNT:
            . . .
        }

  aci_async_free(async_entry);
  printf("results: results done...\n");
}
. . .
```

Figure 5-13 Example of the aci_async_find Function

aci_async_free()

The aci_async_free function clears the async_table entry. See Figure 5-14.

```
#include "aci_async.h"
void aci_async_free(aci_async_entry* async_entry);
```

Figure 5-14 aci_async_free Function Call

Table 5-9 Parameters for the aci_async_free Function

Parameter	Description
async_entry	pointer to an aci_async_entry that should be cleaned

Return Values

None

This call only clears the pid field in the appropriate aci_async_entry structure.

Macros

The *aci_async.h* header file contains macros that substitute all the techniques, required to issue an asynchronous ACI call. However, they require some additional conditions provided by programmer.

aci_mount

The aci_mount function mounts a volume in a drive. See Figure 5-15

```
#include "aci_async.h"
ACI_MOUNT(volser, type, drive_name)
```

Figure 5-15 Example of aci_mount Asynchronous ACI Call

All the parameters must comply with the rules for aci_mount function call.

The local variables *int res* must be defined before using this macro.

Return Values

The d_errno and d_text globals are copied to d_errno and d_text fields of shared memory array entry. The process is terminated by the *exit()* call with the *res* exit code. If there are problems with shared memory attachment, d_errno global will be set to ENOSHARED.

aci_dismount

The aci_dismount function dismounts a volume. See Figure 5-16.

```
#include "aci_async.h"
ACI_DISMOUNT(volser, type)
```

Figure 5-16 aci_dismount Asynchronous ACI Call

All the parameters must comply with the rules for the aci_dismount function call.

The local variable *int res* must be defined before using this macro.

Return Value

The d_errno and d_text globals are copied to d_errno and d_text fields of shared memory array entry. The process is terminated by the *exit* call with the *res exit* code. If there are problems with shared memory attachment, d_errno global will be set to ENOSHARED.

aci_force

The aci_force function dismounts any cartridge from a specific drive. Refer to Figure 5-17.

```
#include "aci_async.h"
ACI_FORCE(drive)
```

Figure 5-17 aci_force Asynchronous ACI Call

All the parameters must comply with the rules for the aci_force function call.

The local variable *int res* must be defined before using this macro.

Return Values

The d_errno and d_text globals are copied to d_errno and d_text fields of shared memory array entry. The process is terminated by the *exit* call with the *res exit* code. If there are problems with shared memory attachment, d_errno global will be set to ENOSHARED.

aci_insert

The aci_insert function inserts volumes into the AML. See Figure 5-18.

```
#include "aci_async.h"
ACI_INSERT(insert_area, volser_ranges, type)
```

Figure 5-18 aci_insert Asynchronous Function Call

All the parameters must comply with the rules for the aci_insert function call.

The local variable *int res* must be defined before using this macro.

Returned values

The d_errno and d_text globals are copied to the d_errno and d_text fields of the shared memory array entry. The process is terminated by the *exit* call with the *res exit* code.

aci_eject

The aci_eject function ejects a range of volumes from the AML. See Figure 5-19.

```
#include "aci_async.h"
ACI_EJECT(eject_area, volser_range, type)
```

Figure 5-19 aci_eject Asynchronous Function Call

All the parameters must comply with the rules for the aci_eject function call.

The local variable *int res* must be defined before using this macro.

Return Values

The d_errno and d_text globals are copied to the d_errno and d_text fields of the shared memory array entry. The process is terminated by the *exit* call with the *res exit* code. If there are problems with the shared memory attachment, the d_errno global will be set to ENOSHARED.

aci_eject_complete

The aci_eject_complete function ejects volumes and removes the database entries. See Figure 5-20.

```
#include "aci_async.h"
ACI_EJECT_COMPLETE(eject_area, volser_range, type)
```

Figure 5-20 aci_eject_complete Asynchronous Function Call

All the parameters must comply with the rules for the aci_eject_complete function call.

The local variable *int res* must be defined before using this macro.

Return Values

The d_errno and d_text globals are copied to the d_errno and d_text fields of the shared memory array entry. The process is terminated by the *exit* call with the *res exit* code. If there are problems with the shared memory attachment, the d_errno global will be set to ENOSHARED.

Response Technique

This section describes the most desirable response processing technique.

Setup

The following code should be executed upon startup. This code installs a signal handler, which will be automatically run every time one of the child processes terminates. At this point retrieve the results, place them into internal data structures and free the shared memory array entry. See Figure 5-21.

```
. . .
/* dasadmin sample application, dasadmin.c file */
int  ( *func )();
struct sigaction action;
aci_async_entry *async_entry;

action.sa_handler = wait_for_child;
sigemptyset(&action.sa_mask);
action.sa_flags = 0;

if (sigaction(SIGCHLD, &action, (struct sigaction*)0)
== (int)SIG_ERR)
{
  fprintf( stderr, "Unable to set signal handler for
SIGCHLD\n");
  return( 1 );
}
. . .
```

Figure 5-21 Setup Signal Handler

Signal Handler Routine

The following code processes the result of child process work. At the point this function is run, the results are placed in the shared memory array entry, and developer should program the logic that will take the results from there and place them somewhere else.

The dasadmin sample application puts all the data into the standard output. See Figure 5-22.

```
/* dasadmin sample application, wait_for_child.c file */
void wait_for_child(int sig_no)
{
  pid_t pid;
  aci_async_entry* async_entry;
  int exit_code;
  int i;

  pid = wait(&exit_code);

  if ((async_entry = aci_async_find(pid)) != 0)
  {
    printf("results: getting results...\n");
            . . .
  }
}
```

Figure 5-22 dasadmin Sample Application

Data structures

This section provides and overview of the data structures used in asynchronous ACI data interchange. The shared memory array consists of several entries of type aci_async_entry. The exact number should be set in the aci_async_create call. This consists of three sections:

- Common information (pid, aci_func, d_errno, d_text member variables)
- Parameter data (parms structure)
- Response data (response structure)

aci_async_entry

Refer to Figure 5-23 on page 5-24 for the common structure, that a shared memory array contains.

```
struct _aci_async_entry
  {
    pid_t pid;                      /* process id, -1
means slot is empty    */
    int aci_func;                   /* DAS_MOUNT,
DAS_DISMOUNT, DAS_FORCE,
                                DAS_INSERT, DAS_EJECT,
                        DAS_EJECT_COMPLETE
*/
  int d_errno;           /* DAS error code
*/
  char d_text[DAS_SZ_MSG_LEN];  /* error message
*/

    union _parms                                  /*
parameters data         */
    {
      async_drive_parms st_drive_parms; /* mount,
dismount, force parameters */
     async_ei_parms st_ei_parms;      /* insert, eject,
eject_complete
                             parameters
*/
    } parms;

    union _response                               /*
response data           */
    {
      async_mount_parms st_mount_parms;           /*
mount command response  */
      async_response st_response;                 /*
dismount, force, eject,
                                        eject_complete
response */
      async_insert_response st_insert_response;   /*
insert command response */
    } response;

} aci_async_entry;
```

Figure 5-23 Common Structure for aci_async_entry

Parameter Data (Parms Structure)

This union contains several structures that hold the data required for making an appropriate ACI call. All the constants, starting with XDR, are defined in *aci_xdr.h* file.

Some of the structures contain unusable data fields. This is done for better compatibility with RPC structures. Developers who do not use the macros, may need to fill in the structures. If the macros are used, the developer only needs to take note of what the macros do.

See Figure 5-24 for the structure of the async_drive_parms.

```
struct async_drive_parms {
 char volser[XDR_VOLSER_LEN];  /* volser name
*/
 enum media type;         /* media type
*/
 char drive[XDR_DRIVE_LEN];   /* drive name
*/
  enum drive_status drive_state; /* drive status - not
used here               */
  async_common common;           /* common data - not
used here                */
};
```

Figure 5-24 async_drive_parms Structure

See Figure 5-25 for the structure of async_ei_parms.

```
struct async_ei_parms {
 char area_name[XDR_AREANAME_LEN]; /* I/E area name
*/
 char volser_range[XDR_RANGE_LEN]; /* volser range
*/
 enum media type;           /* media type
*/
  struct common common;            /* common data - not
used here             */
};
```

Figure 5-25 async_ei_parms Structure

Response Data (Structure)

This union contains several structures where the asynchronous call results are stored after the call is completed.

st_response

This structure is reserved for possible future use. Asynchronous ACI developer could use it for storing data. See Figure 5-26.

```
struct async_response {
  int code;
  char text[XDR_TEXT_LEN];
};
```

Figure 5-26 async_response Structure

st_mount_parms

This structure is reserved for possible future use. Asynchronous ACI developer could use it for storing data. See Figure 5-27.

```
struct async_mount_parms {
  async_response stResponse;
  char volser[XDR_VOLSER_LEN];
  enum media type;
  char drive[XDR_DRIVE_LEN];
  enum drive_status drive_state;
  char drvmedia[XDR_TEXT_LEN];
};
```

Figure 5-27 async_mount_parms Structure

st_insert_response

This structure is supported by ACI_INSERT macro, described above. If you do not use macros, you could fill it with appropriate data after aci_insert() call returns. ACI_INSERT macro also fills the member volser_ranges_len with the actual size of volser_ranges_val array, which could be less that XDR_NO_RANGES in most cases. See Figure 5-28.

```
struct async_insert_response {
 async_response resp;      /* not used
*/
  struct {                     /* this structure returns
the volsers inserted */
    u_int volser_ranges_len;   /* number of
volser_ranges_val array entries,
                                  which contain inserted
volsers               */
    char
volser_ranges_val[XDR_NO_RANGES][XDR_RANGE_LEN]; /*
this array contains
                    all the volsers inserted
*/
  } volser_ranges;
 enum media mediatype;    /* media type, not used
*/
 char szMediaType[XDR_AREANAME_LEN]; /* not used
*/
};
```

Figure 5-28 async_insert_responce Structure

A

Application Notes

Overview

This section contains information on error recovery procedures and explanations of terms used throughout this document.

Error Recovery Procedures

This section contains information on error recovery procedures. See Table A-1.

Table A-1 Error Code Reactions

Number	d_error Name	Recommended Reactions to the Error Code
0	EOK	No error.
1	ERPC	Communication problem, test the TCP/IP to the AMU PC and, if possible, try an alternate route.
2	EINVALID	A parameter in the command is wrong, try it with other parameters.
3	ENOVOLUME	Volser from the command with the media type is not found in the AMU database. Check the media type in the command. If a mismatch is detected between the archive and the database, update the AMU database with the inventory command. Otherwise try the command with another volser
4	ENODRIVE	The drive parameter in the command does not match any drives in the AML. Change the parameter in the command.
5	EDRVOCUPPIED	The drive named is already used for another cartridge. • Send an unload command to the drive. • Send a dismount command to the DAS software. • Wait until the previous mount (also possible for cleaning) is completed.

Table A-1 Error Code Reactions

Number	d_error Name	Recommended Reactions to the Error Code
6	EPROBVOL	The AMS returned a error code from robot control or information about an unrecoverable situation in the AML. Stop the command that returned this error. Refer to the AMU-Log for more information.
7	EAMU	The AMS returned an unexpected error. Stop the command that returned this error. Refer to the AMU-Log for more information.
8	EAMUCOMM	An internal error was detected in the AMS software (AMS error code 1001 in the AMU-log), wait a moment, then enter this, or another test command again. Otherwise, the AMS software will need to be stopped and restarted.
9	EROBOT	not used
10	EROBOTCOMM	There is a problem in the communication between AMU and robot control or the robot is switched logically, or physically to not ready. Check the status with aci_robstat and, if possible, set the status to ready. Try the command again.
11	ENODAS	On ACI Version 1.2 mapped to the Error EDASINT (Refer to *EDASINT* on page A-7).
12	EDEVEMPTY	AMS returned the error code 1094, the drive named is in the AMU database with the attribute Empty. If there is a cartridge in the drive (previous error with database mismatch or manual intervention), the AMU database must be updated. Use the AMS screen or remote SQL-commands to set the drive coordinate: • Volser: volser in the drive • Attribute: ’O’ (Occupied) • set the home coordinate of the Volser • Attribute: ’M’ (Mounted) • if the drive is an Optical Disk drive the second site of the OD must also be set. • Attribute ’R’ (Reverse side Mounted)
13	ENOTREG	not used

Table A-1 Error Code Reactions

Number	d_error Name	Recommended Reactions to the Error Code
14	EBADHOST	DAS was unable to resolve the IP-name to an address, or the address is invalid. Check the TCP/IP command. Set the IP-configuration and change configuration data in AMU or the environment.
15	ENOAREA	The named insert or eject area was not found in the configuration. Try the command with another area name, or change the AMS configuration.
16	ENOTAUTH	Access privilege limitations for the requesting client are defined in the config file parameters. Change the configuration. This can be temporarily done by aci_register. For a longer term change, use another client or change the config file.
18	EUPELSE	DAS has found that the drive is reserved by another client. • Check which client has reserved the drive and the status of the drive (occupied or empty) with aci_drivestatus2. • Set the drive for the other client down with aci_driveaccess (if occupied use FDOWN). • If the drive is occupied, send an unload command to the drive and to DAS with aci_dismount. • Set the drive for the Client UP with aci_driveaccess.
19	EBADCLIENT	The Client only BASIC rights. Change the Clientname on the ACI to a Client with Complete rights or change the Configuration file.
20	EBADDYN	not used
21	ENOREQ	The aci_cancel command was used with a invalid sequence number. • check the command queue for the correct sequence number with aci_list • try the aci_cancel command with the correct sequence number

Table A-1 Error Code Reactions

Number	d_error Name	Recommended Reactions to the Error Code
22	ERERTRYL	The maximum number of automatic retries for recovering has been exceeded. There is a problem in the AMS. Check the AMU-Log for more information: • <0420> Cartridge not ejected. • The dismount manager is not configured correctly (Refer to the *AMU Reference Guide*). • The drive has not received the unload command (send unload via data path to the drive or to DAS aci_unload. • The drive has a physical problem (call Service). • <1191> Tower not available. The tower can be set back to the ready status with the status command, if the tower does not have a physical problem (call Service). • <1123> Too many commands; possible AMS overload. • Restart the AMU. Do not send so many commands at one time, or request AMU hardware with more performance. • <1290> Command Canceled from AMS. Internal problem in the AMS software, try a restart of the AMU.
23	ENOTMOUNTED	AMS returned the error code 1162, the volser named is not in the AMU database. • Check the Volser in the command. • If there is a cartridge in the drive (previous error with database mismatch or manual intervention), the AMU database must be updated.
24	EINUSE	The requested volser is already in use. Wait until the usage of the cartridge is completed or send a unload and a dismount of the drive, that is using the requested volser.
25	ENOSPACE	Problem with the aci_register, aci_clientaccess and aci_foreign commands. The limits for configured ranges have been exceeded. Remove old ranges, before registering something new.

Table A-1 Error Code Reactions

Number	d_error Name	Recommended Reactions to the Error Code
26	ENOTFOUND	Problem with parameters in the command. Check the command and confirm the parameters with the AMS configuration.
27	ECANCELLED	A command was canceled with the aci_cancel command or on the AMS.
28	EDASINT	An internal DAS error has occurred. Restart DAS. Save the Log and inform ADIC support about the error situation.
29	EACIINT	An internal ACI error has occurred. Restart DAS. Save the Log and inform ADIC support about the error situation.
30	EMOREDATA	Return code from aci_qvolsrange command, if the number of volsers in the range exceed the number of displayed volsers. Try the aci_qvolsrange again with the last volser in the first range in the parameter of the first volser.
31	ENOMATCH	Parameters in the command are not correct confirm, that the parameter type (Mediatype) is correct for the command. Confirm the AMS configuration with the parameter in the command.
32	EOTHERPOOL	Return code to aci_scratch_set, if the volser is already defined in a pool with another name. Use the command aci_scratch_unset for the other pool and issue the aci_scratch_set command again.
33	ECLEANING	The command can not be executed, because the drive or the volser is being used for the drive cleaning. Wait until the cleaning process is complete, and try the command again.
34	ETIMEOUT	The maximum wait time for the command has been exceed. Check the life of the DAS software with aci_qversion and the life of the robot wit aci_robstat. Check the communication with ping command. If you have very high load in the system, you have to change the time-out parameters in the configuration (Refer to the *Environment Variables* in the *DAS Administration Guide*).

Table A-1 Error Code Reactions

Number	d_error Name	Recommended Reactions to the Error Code
35	ESWITCHINPROG	The commands can not be executed. DAS and AMS are being switched to the other AMU. Wait until the switching process is completed and try again.
36	ENOPOOL	There has been an attempt to insert or eject cleaning cartridges to a cleanpool that has not been configured. Confirm the cleanpool name in the command with the cleanpool names in the AMS configuration.
37	EAREAFULL	The aci_ejectclean command returns that the Eject area is full. Empty the area and try the command again.
38	EHICAPINUSE	The commands can not be executed, because the HICAP is open and the robot is not ready. Close the HICAP, press the start button on the controller and try the command again.
39	ENODOUBLESIDE	The volser in the aci_getvolsertoside command was not from type optical disk (AMU mediatype O0 and O1) Check the AMS configuration and AMU database.
40	EEXUP	The aci_driveacess command returned that the requested drive is exclusively used by a other client. Request the other client to release the drive or use the Client DAS_SUPERVISOR for the release.
41	EPROBDEV	AMS returned <1033> (position not in AMU database). Check the AMU database, use AMS command update device for actualizing the database.
42	ECOORDINATE	AMS returned <1142> (the logical coordinate is not in the AMS configuration). Compare the AMS configuration with the DAS parameter in the command.
43	EAREAEMPTY	Returned AMS <1156> (Insert area empty) during insert. Open and close the I/O unit, the AMS will start the automatic inventory of the range.
44	EBARCODE	Switch the bracode reading off with aci_barcode and try the command again

Table A-1 Error Code Reactions

Number	d_error Name	Recommended Reactions to the Error Code
45	EUPOWN	The Client tried to allocate volsers that are already allocated.
46	ENOTSUPPHCMD	The AMU has a command exclusion feature that can be used to configure which DAS commands are supported. The command that wes sent to the AMU is configured as a not supported host command. Refer to the *AMU Administration Guide* if this needs to be changed.
47	EDATABASE	Check the AMU log for a more detailed message. Also check the AMU error message.
48	ENOROBOT	Check the AMU configuration
49	EINVALIDDEV	Check the device parameter and correct it if necessary.
50	NO_ECODES	Number of error codes in header file increment when adding new codes to the end of the list

Terms

This section contains explanations of terms used throughout this document.

ACI	**A**ML **C**lient **I**nterface Application Program Interface for the AML
AML	**A**utomated **M**ixed-Media **L**ibrary; AML software and physical archive. • /2 stands for **2**nd version • /E stands for **E**ntry • **/**J stands for **J**unior
AMS	**A**ML (**A**rchive) **M**anagement **S**oftware The complete software package which controls the AML
AMU	**A**ML **(Archive) M**anagement **U**nit Central intelligence of the DAS system. Consists of hardware and software.

API	**A**pplication **P**rogram **I**nterface A program residing on the client's platform used to interpret the client's requests and to provide all the network communication compatible with the interface requirements.
Archive	The archive consists of: • physical archive • logical archive. The physical archive consists of storage segments for tape cartridges and optical disks (= media). The logical archive (archive catalog) is the list of volsers assigned to the compartments in the physical archive.
Archive catalog	An OS/2 database with the logical archive. Contains the assignment of volsers to the compartments in the physical archive as well as additional vital information about the media and the drives.
Bar code	An array of rectangular bars and spaces in a predetermined pattern (e.g., UPC symbol.)
Bin	A single medium storage location. Also referred to as a slot in some archives.
Cassette	A shell having two co-planar hubs, designed to hold magnetic recording tape. Used loosely, the same as Volume.
Cartridge	One or more physical volumes, bound in a transportable package with a human- readable external label.
Client	A volume server user that may be an application program.
Console	A human interface mechanism for controlling and monitoring system operation.
DAS	**D**istributed **A**ML **S**erver
Data	This term refers to information transferred over the network not including requests and operation responses.
dismount	The robotic action to remove media from a drive to storage.

Drive	A device used to read and write data on a medium.
eject	The physical action of removing a medium from an archive. For a robotic archive, the medium is robotically moved to the unload port for removal by the operator.
Eject area	The logical location within the I/O unit that accepts ejected media.
Ethernet	Interface standard defined by IEEE Standard 802.3
File	An individual collection of related data (e.g.; a letter, a table, a digitized photograph).
Foreign (non-system) media	Cartridges not listed with a volser in the archive catalog. They are processed by the DAS system via the I/O unit.
ID	Identifier. In DAS the ID is usually referring to the volser, which is the identifier for a volume.
insert	The action of physically entering a medium into an archive. For a robotic archive, the operator places the medium in the archive's load port from which the robot places the medium in the assigned bin.
Insert area	Logical location within the I/O unit that accepts inserted media.
I/O unit	A mechanical device into which an operator places media which are to be entered or removed from a robotic archive.
inventory	A physical action by the archive’s robot to determine the storage contents of the AML.
Media	More than one medium.
Medium	A storage object that, when mounted in a drive/ recorder/reproducer, is available for read and write operations, but also for clean. Types include magnetic tape, magnetic disk, optical tape, and optical disk, cleaning tape.
mount	The robotic action to move media from storage to a drive.

Network	The physical and logical connection of computers and peripheral devices that allows communication and data sharing.
Network Protocols	A set of rules defining the physical and logical connection.
OS/2	Operating system (multitasking, single user) that is used on the AMU controller PC.
PC	**P**ersonal **C**omputer.
RAM	**R**andom **A**ccess **M**emory
Robotic archive	A storage system featuring one or more robots for media handling.
RPC	**R**emote **P**rocedure **C**all - with XDR, RPC is the Session Layer (layer 5) and XDR is the Presentation Layer (layer 6) of the ISO/OSI layered client interface.
Scratch media	Media that has no client data and is free for use and reclassification.
Scratch pool	A collection of scratch media of the same media type.
Shelf archive	An identifiable set of contiguous bins for storing media.
Slot	A single medium storage location. Also referred to as a bin in some archives.
Stage	A type of media storage area containing no assignable bin/slot locations.
System Administrator (SA)	The primary human controller of a computer system. The SA configures each archive, issues restricted commands, and generates reports appropriate to efficient management of the overall system.
Terabyte (TB)	10^{12} bytes, or one million megabytes.
User	Also known as the client or the client system.
Volume	A removable entity of tape media.

Volser, VSN	**Vol**ume **Ser**ial **N**umber An up to sixteen-digit alphanumeric designation. It identifies one medium (cartridge, optical disk) in the archive. Exception: optical disk has one logical compartment but two volsers (A and B side). The volser is attached to the rear of the medium on a barcode label and can be read by the handling unit.

Index