Application Storage Manager™ (ASM) for Unix

ASM Migration Toolkit Guide

Version 3.5.0

Part Number 313498601

Proprietary Information Statement

The information in this document is confidential and proprietary to Storage Technology Corporation and may be used only under the terms of the product license or nondisclosure agreement. The information in this document, including any associated software program, may not be disclosed, disseminated, or distributed in any manner without the written consent of Storage Technology Corporation.

Limitations on Warranties and Liability

This document neither extends nor creates warranties of any nature, expressed or implied. Storage Technology Corporation cannot accept any responsibility for your use of the information in this document or for your use of any associated software program. You are responsible for backing up your data. You should be careful to ensure that your use of the information complies with all applicable laws, rules, and regulations of the jurisdictions in which it is used**. Warning**: No part or portion of this document may be reproduced in any manner or in any form without the written permission of Storage Technology Corporation.

Restricted Rights

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software – Restricted Rights at 48 CFR 52.227-19, as applicable.

Application Storage Manager™ (ASM) for Unix Migration Tool Kit, Version 3.5.0, Edition 2, July 15, 2002, Part Number 313498601

This edition applies to the Application Storage Manager™ (ASM) for Unix product and to all modifications of that product until otherwise indicated in new editions or revisions pages. If there are changes in the product or improvements in the information about the product, this document will be revised and reissued.

Comments concerning the contents of the manual should be directed to:

ASM Product Management
Storage Technology Corporation
One StorageTek Drive, MS 2138
Louisville, CO 80028-2139

Copyright Statement

New Features

The *ASM Migration Toolkit Guide*, Part Number 313498601, supports the ASM and ASM QFS 3.5.0 releases running on the Solaris 2.6, 2.7, and 2.8 platforms. No new features were added to this release specifically to support ASM Remote, but the changes to the default file locations in the ASM and ASM QFS environments also affect the ASM Remote environment.

Record of Revision

Version	Description
3.3	January 1998. Original printing
3.5	July 2002. Document update

Table of Contents

Proprietary Information Statement ii
Limitations on Warranties and Liability ii
Restricted Rights ii
Application Storage Manager™ (ASM) for Unix Migration Tool Kit, Version 3.5.0, Edition 2, July 15, 2002, Part Number 313498601 ii
Copyright Statement ii

New Features iii

Record of Revision iv

Table of Contents v

About This Guide vii

Organization vii
StorageTek License vii
Conventions vii
StorageTek Publications viii

Chapter 1 - ASM Migration Toolkit Overview 1

ASM Migration Toolkit Library Calls 3

Migration Interface 3
ASM Stage Interface 3
Re-Archive Interface 3
ASM Restore Interface 4

Chapter 2 – Installing the Migration Toolkit 5

How To Install the ASM Migration Toolkit 5

Step 1: Verify Existing ASM Software 5
Step 2: License the ASM Migration Toolkit 5
Step 3: Read the Installation Medium 6
Step 4: Add the Package 6
Step 5: Write and Compile ASM Migration Programs 7
Step 6: Update the mcf File 8
Step 7: Shutdown ASM 8
Step 8: Restart ASM 9
Step 9: Check for ASM Migration Device Entries 10
Step 10: Run the ASM Migration Programs 10

Chapter 3 – Example Programs 11

ASM Migration Toolkit Example Programs - CD-ROM Format 11

mig_cd.c 12
mig_build_cd.c 12

mig_rearch.c 13
Makefile 13
Installing the ASM Migration Toolkit Example Programs - CD-ROM Format 13
Step 1: Compile the Example Programs 13
Step 2: Add Stranger Device to mcf File 14
Step 3: Start ASM and Mount Filesystems 14
Step 4: Insert and Mount CD-ROM 14
Step 5: Build the ASM Migration Entries 14
Step 6: Access the Stranger Data Files 15
Step 7: Re-archive the Stranger Data Files Under ASM 15
Example Tape ASM Migration Progam 15
Updating the ASM catalog 15
Example Program - Migrating Data from Tapes 16

About This Guide

This guide describes the ASM Migration Toolkit by StorageTek. The ASM Migration Toolkit is an application programming interface (API) used in conjunction with the Application Storage Manager™ (ASM) Filesystem release 3.5.0 or higher for importing data from stranger media to ASM.

The ASM Migration Toolkit is only available to StorageTek channel partners and authorized service providers (ASPs). The ASM Migration Toolkit enables channel partners to write conversion programs allowing ASM to read and use non-ASM data. It is assumed that the channel partner/ASP writing these conversion programs is an experienced C programmer, has a complete knowledge of the data storage formats being converted, and is familiar with the theory and operations of ASM.

Organization

This manual is organized as follows:

Chapter	Description
Chapter 1	Provides an overview of the ASM Migration Toolkit.
Chapter 2	Provides step-by-step installation instructions for the ASM Migration Toolkit.
Chapter 3	Describes the example ASM Migration Toolkit conversion programs supplied with the software.
Appendix A	Printed manual pages for the ASM Migration Toolkit library routines.

StorageTek License

Conventions

The following conventions are used throughout this document:

Typeface	Meaning	Example
command	The fixed-space courier font denotes literal items such as commands, files, routines, path names, and messages	/etc/opt/LSCsamfs/mcf
Boldface Courier	The boldface courier font denotes text you enter at the shell prompt	server# **sls -D**
Italic Courier	Italics indicate variables in a command line. Replace variables with a real name or value.	# **mount** *mnt_pt*

StorageTek Publications

If you have comments about the technical accuracy, content, or organization of this document, please tell us. We value your comments and will respond to them promptly. You can contact us in any of the following ways:

- Send an electronic mail to David Smith, ASM for Unix Product Manager at David_Smith@storagetek.com
- Send a facsimile with your comments to the attention of David Smith at fax number: +1303-661-7949.
- Send your written comments to:

  Application Storage Manager (ASM) Product Management
  StorageTek.
  One StorageTek Drive, MS 2138
  Louisville, CO 80028-2138
  USA

To order additional manuals, please send us a written request using one of the methods above.

Chapter 1 - ASM Migration Toolkit Overview

The ASM Migration Toolkit from StorageTek provides a user exit interface to read and migrate data into the Application Storage Manager™ (ASM) File System from non-ASM media. This media (called *stranger media* throughout the rest of this document) is read by a program written by an expert who has a thorough understanding of the data format used to write to the media. Examples of stranger media include CD-ROMs, tapes written using another vendor's software application (including storage management systems other than ASM), or other media written in a predictable, consistent manner. The ASM Migration Toolkit supplies a user exit interface that allows you to write a program to restore data from stranger media into ASM.

The ASM Migration Toolkit requires the following:

- A storage server running the ASM 3.5.0 release or higher;
- The ASM Migration Toolkit software package (labeled **LSCmigkit**) supplied by StorageTek;
- The stranger media to be made available for reading and/or migrating data into ASM; and
- A migration interface program provided by an expert in the stranger media format in the form of a shared object library (suffixed by `.so`).

The ASM Migration Toolkit can be used in the following manner:

1) *Migration mode*, in which an application running in the ASM environment needs data residing on stranger media but wishes to migrate the data permanently to ASM. A stage request for a file is processed and the data is written to ASM disk cache. The data is re-archived to ASM media for future use, essentially migrating the data from stranger media to ASM.

2) *Stage-only mode*, in which an application running in the ASM environment needs data residing on stranger media, a stage request for a file is processed and the data is written to ASM disk cache. The application completes and the disk cache copy of the data is released. The permanent archive file remains on the stranger media.

Note that the ASM Migration Toolkit does not support the volume overflow feature in ASM. The `sam_mig_rearchive`(3) routine does not support spanning multiple volumes.

Figure 1-1 diagrams the flow for a ASM storage server using the ASM Migration Toolkit. Note the shading of the components indicating StorageTek or the reseller as the supplier of the programs.

The create and stage library calls are described in the following subsection under "ASM Restore Interface" and "ASM Stage Interface", respectively, as well as in

the supplied manpages. The ASM Migration Interface user exit shared library is described in the next subsection under "ASM Migration Interface".

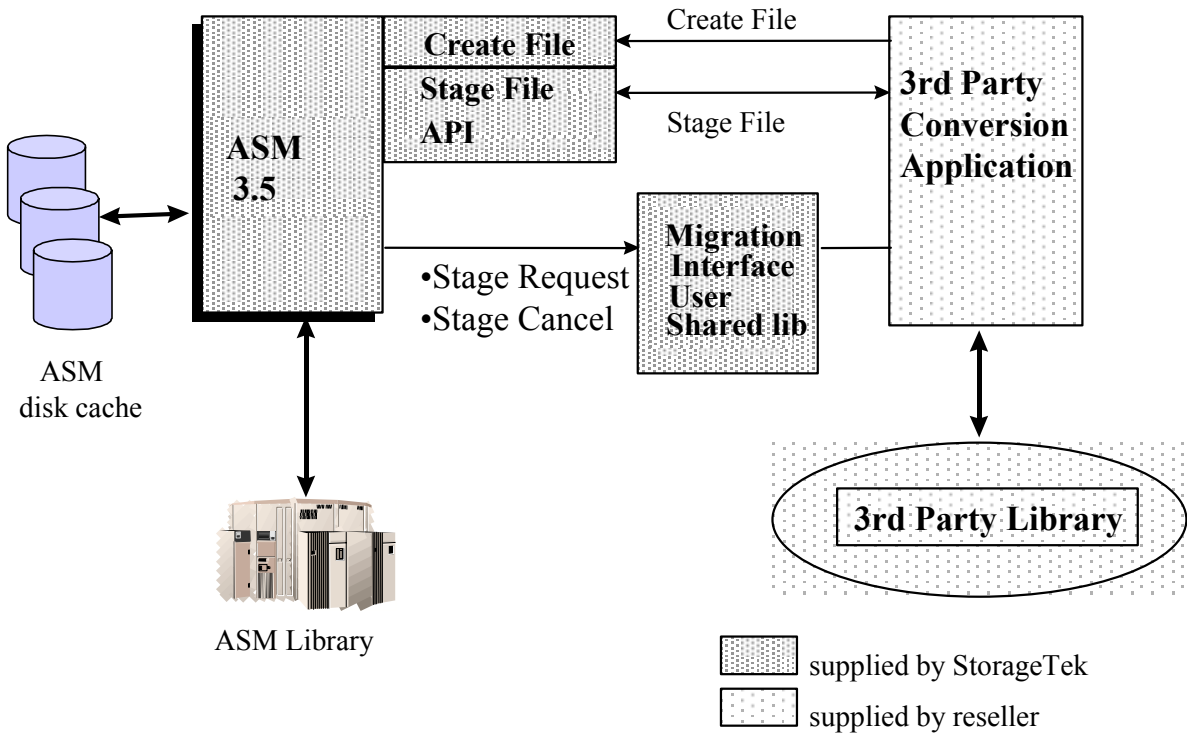

Figure 1-1. ASM Migration Toolkit Flow Diagram

ASM Migration Toolkit Library Calls

This section lists the library calls available with the ASM Migration Toolkit. For more details about each of the library calls, see the corresponding man page. Printed versions of the manpages are found in Appendix A.

Migration Interface

The migration interface is a thread-safe shared object library consisting of three entry points. The migration API routines are made available through a vendor-supplied migration library. Table 1-1 lists the routines available through the migration API.

Table 1-1. Migration Library Routines

Library Call	Description
`mig_initialize`	Initializes the migration library libsam_mig.so
`mig_stage_file_req`	Processes a stage request allowing files to be staged to ASM disk cache.
`mig_cancel_stage_req`	Cancels a pending stage request.

ASM Stage Interface

The ASM stage interface is a thread-safe set of routines made available to `libsam_mig` when the stranger shared object library (`.so`) is loaded. These routines are supplied with the ASM Migration Toolkit. Table 1-2 lists the functions available through the stage API.

Table 1-2. ASM Stage Library Routines

Library Call	Description
sam_mig_stage_error	Passes a stage error to the file system.
sam_mig_stage_file	Prepares to stage data for a stage request.
sam_mig_stage_write	Writes data for a stage request from libsam_mig.so to a ASM file system.
sam_mig_stage_end	Completes a stage request.
sam_mig_mount_media	Queues a mount media request to a device.
sam_mig_release_device	Releases a device from a sam_mig_mount_media request.

Re-Archive Interface

The re-archive interface consists of a single thread-safe function, `sam_mig_rearchive`, that traverses a file system and marks archives residing on a VSN as needing to be re-archived. The `sam_mig_rearchive` routine allows a site to "migrate" files from stranger media to ASM media in a controlled manner, migrating a few VSNs at a time. Until a stranger file is re-archived, all access to this file would be through the ASM Migration Toolkit.

ASM Restore Interface

The ASM restore interface consists of single routine, `sam_mig_create_file`. This routine restores the name space for a file in a ASM file system, then creates an off-line ASM file with the stranger API information in any of the archive copy numbers.

Note that the program calling this function is responsible for creating all directories in the path *before* calling the function.

Chapter 2 – Installing the Migration Toolkit

This chapter shows how to install the ASM Migration Toolkit.

How To Install the ASM Migration Toolkit

The ASM Migration Toolkit is released as a separately licensed ASM package in Solaris `pkgadd(1M)` format. The package is named `LSCmigkit`.

Read through all of these instructions prior to installing the Toolkit. While you can configure ASM devices from which to migrate non-ASM data, you must also have your conversion programs and libraries completed and compiled. In these installation instructions, it is assumed that these programs or the sample programs in chapter 3, "Sample Datan ASM Migration Programs", are compiled and available.

All of the steps in this section assume that you are a super-user or are logged in as root.

Step 1: Verify Existing ASM Software

Verify that the server on which you are installing the ASM Migration Toolkit is running the ASM 3.5.0 release as follows:

```
server# pkginfo -l LSCsamfs
```

If you are not running ASM 3.5.0 or higher, you must upgrade ASM. See the ASM Administrator's Guide, for upgrade procedures.

Step 2: License the ASM Migration Toolkit

ASM reads a license key from the file `/etc/opt/LSCsamfs/LICENSE` at startup time. This key allows your system to use the ASM Migration Toolkit and must be updated if you were running ASM only without the ASM Migration Toolkit. If you need a license key that supports ASM and the ASM Migration Toolkit, contact your Authorized Service Provider with the following information:

- Company PO number
- Company name, address, phone, and contact
- Host ID on which the ASM Migration Toolkit software is to be licensed. (To display your machine's Host ID, use the `/usr/ucb/hostid` command.)
- Number of storage slots in your system
- The level of Solaris running on your system. (To display your machine's Solaris level, use the `uname -sr` command.)

Once you have your license key, place it , starting in column one, on the first and only line in `/etc/opt/LSCSsamfs/license3.5`. No other keywords, host ids, etc. may appear. The license becomes effective the next time `sam-init` is started.

Step 3: Read the Installation Medium

Copy the ASM Migration Toolkit files onto your system. The result is a `pkgadd`-format file. Create the directory in which the files will be copied:

```
server# rm -rf /tmp/migkit
server# mkdir /tmp/migkit
```

Using the Solaris Volume Manager, enter the following sequence of commands for each diskette that you receive.

```
Insert diskette
server# volcheck
server# cd /floppy/floppy0
server# cp * /tmp/migkit
server# cd /tmp/migkit
server# gunzip *.gz
server# eject
   Remove diskette
```

Step 4: Add the Package

The ASM Migration Toolkit uses the Solaris packaging utilities for adding and deleting software. As such, you must be logged in as superuser to make changes to software packages. `pkgadd`(1M) prompts you to confirm various actions necessary to install the ASM packages.

```
server# pkgadd -d /tmp/migkit/sammig
```

The ASM Migration Toolkit package consists of the following files:

File	Description
`/etc/opt/LSCsamfs/sam_migd`	ASM Migration Toolkit daemon binary.
`/opt/LSCsamfs/lib/libsam_mig.so`	ASM Migration Toolkit shared object library.

Sample ASM Migration programs are included with the ASM Migration Toolkit. These sample programs include:

AN ASM Migration Toolkit library for reading SunSolve CD-ROM data

A program that reads directories from the CD-ROM and creates new directories within ASM

A program for creating new archive copies of these files under ASM

`A Makefile used to build the examples`

Table 2-1 lists the file names and a description of the sample programs. All of the sample files are located in the /opt/LSCsamfs/migkit directory unless noted.

Table 2-1. ASM Migration Toolkit Sample Programs

File	Description
`Makefile`	The Makefile used to build `mig_rearch`, `mig_build_cd`, and the user ASM Migration library, `libusam_mig.so`.
`README`	A description and installation instructions for the example ASM Migration Toolkit programs.
`mig_build_cd.c`	Directory building source. Creates directories under ASM from the CD-ROM directory structure.
`mig_cd.c`	CD-ROM ASM Migration interface source.
`mig.h`	An include file used with the example programs. This file is located in `/opt/LSCsamfs/include`.
`mig_rearch.c`	Re-archiving program source.

Step 5: Write and Compile ASM Migration Programs

The ASM Migration programs are coded by an integrator knowledgeable in the data format of the stranger media. The sample programs supplied by ASM are discussed in chapter 3, "Example Programs". In order to get these programs to run, you must compile them on your system. Chapter 3 gives instructions on compiling and running these example programs.

The integrator will restore the file names into the ASM file system using their own program and the ASM restore API (see the previous chapter, "ASM Migration Toolkit Overview" for a listing of the restore API library calls). The restore API creates offline files with the information about the location of the data.

The restore interface needs to have the standard inode information such as file size, owner and group, etc. It also requires the following stranger media information, which will be stored in the ASM inode for each file:

media_type	This is the two-letter media type beginning with the letter "z", for example, "**za**". All media types that start with "z" are identified as stranger media.
creation_time	The time that the archive was created.
position_u, *position*	This is an 8 byte field set to any value the integrator needs. This field is a long_long. The information is passed on to the shared object library supplied by the programmer.

vsn This is a 32 byte field set to any value needed by the integrator. This information is passed on to the shared object library supplied by the programmer.

Thus there is a total of 40 bytes of information that can be placed into the archive information for a file. This information can be anything that is needed to identify the location of the data.

An example might be a key into a database. If there is not enough room within the archive record to completely identify the location of the off-line data, the restore program written by the integrator will need to build some type of database where the information may be stored using the position in the inode archive record as the key in to the database. The sample programs supplied with the ASM Migration Toolkit do just this.

Step 6: Update the mcf File

The ASM Migration Toolkit uses additional media types to identify stranger media. As with all ASM devices, the master configuration file (`/etc/opt/LSCsamfs/mcf`) defines the devices. The syntax for the stranger media entry follows:

pathname eq media_type

pathname is the path name the user ASM Migration library (called `/opt/LSCsamfs/lib/libusam_mig.so` in the supplied example programs).

eq is the equipment ordinal for the stranger media device type. *eq* is an integer from 1 to 16384 and must be unique for each `mcf` entry.

media_type is a two-character equipment type for the stranger media. The media type must start with the character "**z**" followed by a single character "**a**" through "**9**". You can have more than one stranger media type defined per system as long as the second character is unique for each stranger media and it matches the *media_type* used when restoring the file names (see `sam_mig_create_file(3X)`.

An example ASM Migration Toolkit mcf entry follows. This site is using the ASM Migration Toolkit to read CD-ROMs as in the example programs and for tapes created in an alternate data format. The `mcf` entries for these media types are as follows:

/opt/LSCsamfs/lib/libusam_mig.so 50 za

/opt/LSCsamfs/lib/libusam_mig.so 51 zb

Step 7: Shutdown ASM

To stop ASM you should make sure that no archive processes are writing to tape or staging files to/from the drives, unmount the file systems, and then kill the `sam-init` process.

To stop ASM enter the following:

1) To ensure that no archive or stage processes are active, idle all of the drives in the library. Enter one of the following:

   From `samu`(1M) enter the following, where *`eq`* is the ordinal the drive:

   :idle *eq*

   From `devicetool`(1M) select the drive in the devices panel. Select the "Change State" button, pull down the menu, and select "Idle".

   The drives will switch from "idle" to "off" when all I/O activity is completed.

2) Unmount any volumes in the drives. Enter one of the following:

   From `samu`(1M) you can unload the drive by enter the following command, where *`eq`* is the ordinal of the drive:

   :unload *eq*

   From `previewtool`(1M) select the drive in which the VSN is present. Select the "Unload" button. The robot unloads the medium from the drive and places it in to its slot.

3) Unmount all of the ASM file systems. Enter the following for each file system:

   # umount *samfs1*

4) Identify the `sam-init` process id, then kill the process id with an interrupt signal. Enter the following:

   # ps -ef | grep sam-init

   # kill -INT *sam-init-pid*

Check again for `sam-init`. Once it is gone, ASM is down.

Step 8: Restart ASM

To restart ASM enter the following:

1) To ensure that no archive or stage processes are active, idle all of the drives in the library. Enter one of the following:

   From `samu`(1M) enter the following, where *`eq`* is the ordinal the drive:

   :idle *eq*

   From `devicetool`(1M) select the drive in the devices panel. Select the "Change State" button, pull down the menu, and select "Idle".

   The drives will switch from "idle" to "off" when all I/O activity is completed.

2) Unmount any volumes in the drives. Enter one of the following:

   From `samu`(1M) you can unload the drive by entering the following command, where *`eq`* is the ordinal of the drive:

:unload *eq*

From `previewtool`(1M) select the drive in which the VSN is present. Select the "Unload" button. The robot unloads the medium from the drive and places it into its slot.

3) Unmount all of the ASM file systems. Enter the following for each file system:

   server# umount *samfs1*

4) Identify the `sam-init` process id, then kill the process id with an interrupt signal. Enter the following:

   server# ps -ef | grep sam-init

   server# kill -INT *sam-init-pid*

   Check again for `sam-init`. Once it is gone, ASM is down.

5) Start ASM using the standard startup procedure at your site.

Step 9: Check for ASM Migration Device Entries

Check to see if the ASM Migration device entries are recognized by ASM. Start `samu`(1M) and check the following:

- The "s" (status) display should show the newly-configured device entries. These devices should have a status of "on".
- The ASM log file (usually located in `/var/adm/sam-log` unless you've configured `/etc/syslog.conf` to point to another file) captures any ASM Migration Toolkit messages.

With the stranger media mounted, you should be ready to use the programs written for the stranger data formats to stage files using ASM.

Step 10: Run the ASM Migration Programs

You should be able to run the example ASM Migration programs or the programs written to recognize the stranger data format. See chapter 3, "Example Programs" for a description and instructions on how to compile and run the example programs.

Chapter 3 – Example Programs

This chapter provides a complete example of a migration interface developed for reading data organized in UNIX directories on a CD-ROM. An overview of the example programs, installation instructions, and a description of each program is presented.

ASM Migration Toolkit Example Programs - CD-ROM Format

This example uses a Sunsolve CD-ROM as the stranger media from which to stage data. The programmer wrote a stranger media API to migrate data from the CD-ROM into ASM.

The program creates off-line files from stranger media in the ASM file system using the ASM restore API (see the chapter 1, "ASM Migration Toolkit Overview" and the `sam_mig_create_file(3)` manpage). The restore API creates off-line files with the information about the location of the data.

The restore API needs to have the standard inode information such as file size, owner and group, etc. It also requires the following stranger media API information which will be stored in the ASM inode for each file:

media_type	This is the two-letter media type beginning with the letter "z", for example, "`za`". All media types that start with "z" are identified as stranger media.
creation_time	The time that the archive was created.
position_u, *position*	This is an 8 byte field set to any value the integrator needs. This field is a long_long. The information is passed on to the shared object library supplied by the programmer.
vsn	This is a 32 byte field set to any value needed by the integrator. This information is passed on to the shared object library supplied by the programmer.

Thus there is a total of 40 bytes of information that can be placed into the archive information for a file. This information can be anything that is needed to identify the location of the data.

An example might be a key into a database. If there is not enough room within the archive record to completely identify the location of the off-line data, the restore program written by the integrator will need to build some type of database where the information may be stored using the inode archive record as the key in to the database. The example programs supplied with the ASM Migration Toolkit do just this.

Two executables are created in this example. `mig_build_cd` restores the directory structure and inodes under ASM and builds a database to track the CD-ROM files. The archive record within each inode points to the database which in turn contains the information needed to find the data associated with each file. Once `mig_build_cd` is run, files can be accessed and staged using ASM.

`mig_rearch` archives the staged stranger media data to ASM media. `mig_rearch` actually sets the re-archive bit within each inodes so that it will be a candidate for archiving on the next pass of the archiver.

Unless noted otherwise, all of these files are located in the `/opt/LSCsamfs/migkit` directory. The example files include:

- `/opt/LSCsamfs/include/mig.h` - The include file for stranger media API.
- `mig_cd.c` - Source for the stranger media API.
- `mig_build_cd.c` - Source for the example migration program. The executable creates directories and files under ASM paralleling the directories on a CD-ROM.
- `mig_rearch.c` - Source for the re-archive program.
- `README` - Information on the programs.
- `Makefile` - A make file for setting up the example programs.
- `mig_mcf` - The mcf file used with the example.

mig_cd.c

This library module performs all of the work for retrieving stranger data. You need to compile and install this module before starting ASM. This module is called automatically.

mig_build_cd.c

This is the C source code for `mig_build_cd`. This program reads the UNIX directory structure from a CD-ROM and builds a corresponding directory structure under an ASM file system. After running this program, the data on the CD-ROM can be accessed from ASM and staged as needed.

Once compiled and made executable, `mig_build_cd` has two arguments as follows:

```
mig_build_cd  cd-pathname sam-pathname
```

where *`cd-pathname`* is the name of the CD-ROM pathname to duplicate and *`sam-pathname`* is the name of an ASM directory in which to recreate the CD-ROM paths.

For example, the following command builds entries in the /sam/migdata directory to access the data from the CD-ROM. The ASM entries will be marked as off-line, have an archive record of media-type za, on VSN cdrom0:

```
mig_build_cd  /cdrom/cdrom0  /sam/migdata
```

mig_rearch.c

This is the C source code for `mig_rearch`. This program sets the re-archive bit on the ASM files so that they will be re-archived to ASM controlled media. You would run this program if you wanted to re-archive all the stranger data to new media.

Once compiled and made executable, `mig_build_cd` has three arguments as follows:

```
mig_rearch  sam-mountpoint media-type vsn-list
```

where *`sam-mountpoint`* is the name of an ASM directory with files to re-archive, *`media-type`* is the stranger media type specified for each file, and *`vsn-list`* is the name of the CD-ROM.

The following example causes ASM to re-archive all data in /sam/migdata that has a media type of za and resides on VSN cdrom0:

```
mig_rearch  /sam  za  cdrom0
```

Makefile

This is the `make`(1) file for the example programs. It contains seven makefile targets: all, install, clean, libusam_mig.so, mig_cd.o, mig_rearch, and mig_build_cd.

Make the following change to `Makefile` before running the `make`(1) command:

1) Change the "CC" makefile variable to reflect the compiler that you want to use on your system.

Installing the ASM Migration Toolkit Example Programs - CD-ROM Format

Once you've completed chapter 2, "Installing the ASM Migration Toolkit", you can use the example programs for reading a CD-ROM. The following steps should be run as super-user.

Step 1: Compile the Example Programs

Compile the programs using `make(1)`. Enter the following:

```
server# cd /opt/LSCsamfs/migkit
server# make clean
server# make
server# make install
```

The program executables `mig_build_cd` and `mig_rearch` are created. If you encounter errors, you may need to use a compiler other than the one specified in the Makefile.

Step 2: Add Stranger Device to mcf File

Add the following entry to your `/etc/opt/LSCsamfs/mcf` file:

/opt/LSCsamfs/lib/libusam_mig.so 200 za

The first field tells where the shared object library is that supports the ASM Migration Toolkit API calls for retrieving data. The second field is the equipment ordinal for this device. If 200 is already in use pick another unique integer. The third field defines an ASM Migration Toolkit media type of "za".

Step 3: Start ASM and Mount Filesystems

Start ASM and mount the file systems following your normal site procedure.

Step 4: Insert and Mount CD-ROM

Insert a Sunsolve CD-ROM in to the CD-ROM drive. Mount the CD-ROM by entering the following:

```
server# volcheck
```

Step 5: Build the ASM Migration Entries

You need to build the migration entries by executing the example program `mig_build_cd`. The following example builds entries in the `/sam/migdata` directory (it is assumed that you have already created this directory in an ASM file system) to access the data from the CD-ROM.

```
server# /opt/LSCsamfs/migkit/mig_build_cd /cdrom/cdrom0 /sam
```

The ASM entries will be marked as off-line, have an archive record of media type "**za**", on VSN cdrom0. The following is an example listing of a stranger data file accessed with ASM. Note the that the creation time, attributes, and residence fields are set to "none".

```
server# cd /opt/LSCsamfs/migkit
server# sls -D samrev.2.5
samrev.2.5:
mode: -rw-r--r--  links:   1  owner: 18621     group: 3900
length:   1717248  inode:     1195
offline;  archdone;
copy 1: ---- Dec 29 15:40         e.0    za cdrom0
access:      Nov 18 17:01  modification: Nov 18 17:01
changed:     Nov 18 17:01  attributes:   none
creation:    none          residence:    none
```

Step 6: Access the Stranger Data Files

The stranger data files are now accessible using ASM. You can stage files to disk cache as you would with ASM. The following example uses an octal dump to stage off-line stranger media files to disk:

```
server# cd /sam/migtest
server# od filename
```

Step 7: Re-archive the Stranger Data Files Under ASM

The mig_rearch program will set the re-archive bit on files and subsequently re-archive them, assuming that you have set up your `archiver.cmd` file properly. To re-archive the example data files, enter the following:

```
server# /opt/LSCsamfs/migkit/mig_rearch /sam za cdrom0
```

Example Tape ASM Migration Progam

This section describes a scenario for importing stranger tapes to your ASM system, updating the catalog for the stranger tapes, and shows an example program for reading data from a stranger tape.

Given the fact that stranger tapes probably already exist within your media library, how does ASM know when to access data from a stranger tape as opposed to an existing ASM tape? This example program shows how you can use `sam_mig_mount_media(3)` and `sam_mig_release_device(3)` to cause ASM to mount stranger media.

An additional example program, named `mig_tp.c`, mounts stranger media and reads a data file DLT tape. The data that is written to this DLT tape is simply a file copy using the Solaris `dd(1)` utility.

Updating the ASM catalog

The ASM library catalog must be updated to reflect that fact that stranger tapes are present in the library. This can be accomplished using one of the following methods.

1) Use the `import(1M)` command. The import command allows you to import stranger media using the "-n" option. When the medium is imported to the robot, the catalog will be updated to indicate that a stranger medium has been loaded.

2) The `chmed(1M)` command is used to change media attributes in the catalog. You can use `chmed` to set or clear the stranger status on the catalog entry for a medium. See the "+N" and "-N" parameters on the chmed(1M) manpage for details.

   For example, you have just imported a stranger DLT tape (media type "lt") in a library (equipment number 30) to slot number 5. Since it probably will not have an ANSI standard label, you will have a catalog

entry which shows "nolabel". To set the VSN and media type in the catalog, enter the following:

```
server# chmed -vsn lt 5 30
```

```
Then, to set the stranger attribute on this medium
enter the following:
```

```
server# chmed +N 5 0
```

3) The `build_cat(1M)` command can be used to load many pieces of media in to a catalog.

   To create entries in the robot catalog for the strange media, you must first `dump_cat(1M)` the existing catalog. Entries which correspond to strange media should either be added or modified so that the media type begins with "z".

   Then, run `build_cat(1M)`, supplying the "-t <media-type>" option. The media-type you specify must be the physical media type, for example "lt". Do *not* use a "z" media type here.

   Each of the entries in the newly-built catalog which have "z" media types in the input file will be marked as strange tapes and will have that media type replaced with the physical media type supplied in the "-t *media-type*" option.

Example Program - Migrating Data from Tapes

The following example reads data from a stranger tape. This stranger tape is simply a DLT tape with no label that simply has data written using the `dd(1)` utility.

This example uses `sam_mig_mount_media(3)` and `sam_mig_release_device(3)` to cause ASM to mount stranger media. This example is rather limited, as it always reads the first bytes from a fixed VSN. You will need to do a similar sequence of function calls, and replace the "mt -f ... rewind" section with code to correctly position the tape.

It is very important to carefully call `sam_mig_release_device(3)` to free up the media drive returned by a successful call to `sam_mig_mount_media(3)`. `sam_mig_release_device(3)` is called only after you've finished all your positioning, reading, etc.

Note that you must use the physical type of the media as the media type passed to `sam_mig_mount_media(3)`.

```
/*
 * mig_tp.c
 *
 * Library routines to handle processing of third-party data from a CD-ROM.
 *
*/
```

```
/*
* ASM_disclaimer_begin
*
* Copyright (c) 1996-1998  StorageTek. All rights reserved.
*
* This file is a product of StorageTek and is provided for unrestricted use
* provided that this header is included on all media and as a part of the
* software program in whole or part.  Users may copy, modify or distribute
* this file at will.
*
* This file is provided with no support and without any obligation on the part
* StorageTek to assist in its use, correction, modification or
* enhancement.
*
* THIS FILE IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND
INCLUDING THE
* WARRANTIES OF DESIGN, MERCHANTIBILITY AND FITNESS FOR A
PARTICULAR
* PURPOSE, OR ARISING FROM A COURSE OF DEALING, USAGE OR
TRADE PRACTICE.
*
* STORAGETEK  SHALL HAVE NO LIABILITY WITH RESPECT TO THE
INFRINGEMENT OF
 * COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY THIS FILE OR
ANY PART THEREOF.
*
* IN NO EVENT WILL  STORAGETEK  BE LIABLE FOR ANY LOST
REVENUE OR PROFITS OR OTHER
* SPECIAL, INDIRECT AND CONSEQUENTIAL DAMAGES, EVEN IF
THEY HAVE BEEN
* ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
*
* StorageTek
*
* ASM_disclaimer_end
*/

#ifndef lint
static char     rcs_id[] = "@(#)$Id: mig_tp.c,v 1.4 1998/01/16 21:00:22 jlh Dev $";
#endif                                  /* lint */
#pragma ident "$Id: mig_tp.c,v 1.4 1998/01/16 21:00:22 jlh Dev $"

#include <thread.h>
#include <synch.h>
#include <stdio.h>
#include <signal.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <syslog.h>
#include <ndbm.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/syscall.h>

#include "mig.h"

#define  DATA_XFER_SIZE   (64 * 1024)  /* units of I/O to read */

/* Function prototypes */
void          *local_stage_processor(void *);
void           local_stage_file(tp_stage_t *);

/* list of stage requests */
typedef struct {
mutex_t        mutex;
cond_t         cond;
int            count;
tp_stage_t     *next;
tp_stage_t     *last;
}
              local_stage_list_t;

local_stage_list_t local_stage_list;
char          *current_database = NULL;
DBM           *current_db = NULL;

/*
* sam_mig_initialize
*
* Called by the thirdparty "device" to allow the interface to initialize any
* local structs, threads, etc.
*/

int
usam_mig_initialize(int stage_count)
{
```

```
/* Initialize the stage list to USYNC_THREAD (all zero) */
memset(&local_stage_list, 0, sizeof(local_stage_list));

if (thr_create(NULL, 0, local_stage_processor, (void *) NULL,
      (THR_BOUND | THR_DETACHED | THR_NEW_LWP), NULL)) {
syslog(LOG_INFO, "Unable to start local_stage_processor: %m");
return (-1);
}
return (0);
}

/*
* sam_mig_stage_file_req
*
* Called by the thirdparty "device" to inform the interface that the file
* system has requested a stage.
*
* For the simple case, we well just link this request onto our list of stage
* requests.  For access to "sequential media" (tape), it would be a better
* idea to keep multiple lists based on physical media and ordered by
* position.  This would allow the media to be read in a less random mode.
* This is not a requirement but it does speed up staging many files from one
* tape.
*/

int
usam_mig_stage_file_req(tp_stage_t * stage_req)
{

syslog(LOG_INFO, "in usam_mig_stage_file_req");

/* Use the private data region as the next pointer for the list */
stage_req->tp_data = NULL;

/*
* Link the request onto the list of stage requests. Must get the
* list mutex first to insure that the other threads are not using
* the list.
*/
mutex_lock(&local_stage_list.mutex);

/* Link this stage request onto the list */
if (local_stage_list.count++ == 0)      /* no entries on the list */
local_stage_list.next = stage_req;
else                            /* put it on the end */
local_stage_list.last->tp_data = stage_req;
```

```
/* Adjust last to point to the new entry */
local_stage_list.last = stage_req;


/* Wake up local_stage_processor */
cond_signal(&local_stage_list.cond);
mutex_unlock(&local_stage_list.mutex);

return (0);
}

/*
* sam_mig_cancel_stage_req
*
* Called by the thirdparty "device" to inform the interface that the file
* system has canceled a stage request.  Most likely the user has terminated
* their request.
*/

int
usam_mig_cancel_stage_req(tp_stage_t * stage_req)
{
    tp_stage_t     *sr_p, *last_sr_p = NULL;

mutex_lock(&local_stage_list.mutex);
for (sr_p = local_stage_list.next;
    sr_p != NULL;
    sr_p = (tp_stage_t *) sr_p->tp_data) {
     if (sr_p == stage_req)       /* Found it */
          break;
          else
          last_sr_p = sr_p;         /* keep last pointer */
     }

          if (sr_p == NULL) {   /* not found */
                    mutex_unlock(&local_stage_list.mutex);
                    return (-1);      /* cannot cancel (can't find it) */
          }
          if (last_sr_p == NULL)          /* looks like the head of the list */
                    local_stage_list.next = (tp_stage_t *) sr_p->tp_data;
                    else
                    last_sr_p->tp_data = sr_p->tp_data;

          return (0);
}
```

```
/*
 * local_stage_processor
 *
 * Wait for stage request to arrive on the list and process them one at a time
 * off the top of the list.
 */

void          *
local_stage_processor(void *noarg)
{
        tp_stage_t     *stage_req;

        /* Loop forever waiting for a request */
        while (1) {
                mutex_lock(&local_stage_list.mutex);
                /* Wait for the count to go non zero */
                while (local_stage_list.count == 0)    /* wait for something */
                        cond_wait(&local_stage_list.cond,
&local_stage_list.mutex);

                /* Pull the entry off the list, decrement the count */
                stage_req = local_stage_list.next;
                local_stage_list.count--;
                local_stage_list.next = (tp_stage_t *) stage_req->tp_data;

                /* Release the mutex */
                mutex_unlock(&local_stage_list.mutex);

                /* process the stage */
                local_stage_file(stage_req);
        }
}

/*
* local_stage_file
*
* Find the file in the database and do the stage.
 */

void
local_stage_file(tp_stage_t * stage_req)
{
        int           read_fd, position = stage_req->position;
        char          *file_data = NULL;
```

```
char          *ent_pnt = "local_stage_file";
offset_t       offset;
int           left;
datum          db_key;
datum          db_data;
char          *s;
char           buf[256];

file_data = malloc(DATA_XFER_SIZE);

/* Since media needs to be mounted, use the sam_mig_mount_media() API */

syslog(LOG_INFO, "%s: about to mount lt:XXX", ent_pnt);

s = sam_mig_mount_media("XXX", "lt");

syslog(LOG_INFO, "%s: s_m_m_m returns %s, errno %d: %m", ent_pnt,
      s ? s : "NULL", errno);

sprintf(buf, "/usr/bin/mt -f %s rewind", s ? s : "NULL");
syslog(LOG_INFO, "%s: about to %s", buf);
system(buf);

if ((read_fd = open(s, O_RDONLY)) < 0) {

     syslog(LOG_INFO, "%s: open(%s,O_RDONLY) failed: errno %d: %m",
          ent_pnt, s ? s : "NULL", errno);
     sam_mig_release_device(s);

} else {

     if (sam_mig_stage_file(stage_req)) {
               /*
                * The file system refused the stage request.  This
                * usually happens if the stage requests was
                          canceled
                * (ECANCELED) or there is not enough space to stage
                * the file (or the segment if stage never) (ENOSPC).
                */
               syslog(LOG_INFO,
          "%s: sam_mig_stage_file returned error: %m", ent_pnt);

               /* Free resources */
               if (file_data)
                         free(file_data);
               close(read_fd);
```

```
                sam_mig_release_device(s);
                return;
        }
        left = stage_req->size;/* amount of data left to xfer */
        offset = 0;       /* offset for our writes */

        /*
         * Continue the read, stage_write cycle until all requested
         * data has been sent.
         */
        while (left > 0) {
                int             amt_read, amt_sent, read_size;
                char            *buffer = file_data;

                /*
                 * Only read the smaller of whats left or the
                 * transfer size
                 */
                read_size = left > DATA_XFER_SIZE ?
DATA_XFER_SIZE : left;
        amt_read = read(read_fd, file_data, read_size);
        if (amt_read < 0) {     /* read error */
                int    hold_err = errno;/* syslog destroys errno */

                syslog(LOG_INFO,
                                "%s: Read error %s: %m", ent_pnt,
db_data.dptr);
                /* Free resources */
                close(read_fd);
                free(file_data);

                /* End the stage with the error */
                sam_mig_stage_end(stage_req, hold_err);
                sam_mig_release_device(s);
                return;
        }
                /* Loop sending the data to the file system */

                while (amt_read > 0) {
                        amt_sent = sam_mig_stage_write(stage_req, buffer,
              amt_read, offset);
                        if (amt_sent <= 0) {
                                int    hold_err = errno;/* syslog destroys
errno */
```

```
                                        syslog(LOG_INFO, "%s:
sam_mig_stage_write %s: %m", ent_pnt,
                                            db_data.dptr);
                                        /* Free resources */
                                        close(read_fd);
                                        free(file_data);

                                        /* End the stage with the error */
                                        sam_mig_stage_end(stage_req, hold_err);
                                        sam_mig_release_device(s);
                                        return;
                                    }
                                    buffer += amt_sent;    /* adjust data pointer */
                                    offset += amt_sent;    /* adjust data offset */
                                    amt_read -= amt_sent;          /* amount left to send
* this buffer */
                                    left -= amt_sent;        /* amount left to send * for
file */
                                }
                        }

                        /* File has been sent, free resources and clean up messages */
                        free(file_data);
                        close(read_fd);

                        /*
                         * Inform file system that this stage request finished
                         * without error.
                         */
                        sam_mig_stage_end(stage_req, 0);
                        sam_mig_release_device(s);
                }

}
```