FS-TST: Forensic Software Testing Support Tools

Requirements, Design Notes and User Manual

Version 2.0
February 2005

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Abstract[†]

This document describes Release 2.0 of a software package developed to aid the testing of disk imaging tools typically used in forensic investigations. The package includes programs that initialize disk drives, detect changes in disk content, and compare pairs of disks. Most of the software was compiled using gcc 3.2.2. The software can be used in the Linux environment to setup disk drives for tests, measure the results of a test and aid in documenting test runs.

The intended audience for this document should be familiar with the Linux operating system, computer operation, computer hardware components such as hard drives, hard drive interfaces (e.g., IDE or SCSI), and computer forensics. A working knowledge of C is not necessary for understanding but would be helpful.

[†] **Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.**

Table of Figures

List of Tables

1 Introduction

The Computer Forensics Tool Testing project at the National Institute of Standards and Technology (NIST), an agency of the United States Department of Commerce, provides a measure of confidence in the software tools used in computer forensics investigations. This work stems from an agreement between NIST Office of Law Enforcement Standards (OLES), NIST Software Diagnostics and Conformance Testing Division, the National Institute of Justice, Federal Bureau of Investigation, DoD Computer Forensics Lab, United States Customs Service and the Technical Support Working Group (TSWG).

This document describes Release 2.0 of FS-TST, a software package that supports the testing of disk imaging tools. The package includes programs that initialize disk drives, detect changes in disk content, and compare pairs of disks. Most of the software was compiled using gcc 3.2.2. The software can be used in the Linux environment to set up disk drives for tests, measure the results of a test and aid in documenting test runs. One set of test cases for disk imaging tools is described in *Disk Imaging Tool Specification, Version 3.1.6* (see http://www.cftt.nist.gov/DI-spec-3-1-6.doc), another is set of test cases is described in *Digital Data Acquisition Tool Test Assertions and Test Plan, Version 1.0* (see http://www.cftt.nist.gov/DA-ATP-pc-01.pdf).

1.1 How to read this document

The intended audience for this document should be familiar with the Linux operating system, computer operation, computer hardware components such as hard drives, hard drive interfaces (e.g., IDE or SCSI) and computer forensics. A working knowledge of C is not necessary for understanding but would be helpful. We expect that the primary objectives of the reader to be some combination of the following:

1. The reader wants a general understanding of FS-TST.
2. The reader wants to use FS-TST as is.
3. The reader wants to use FS-TST after enhancement or modification.
4. The reader wants to verify or test the operation of FS-TST

The remainder of this section describes the capabilities required to test disk imaging tools and lists the components of FS-TST. The goal is to identify capabilities required for testing disk imaging tools and allocate the required capabilities to the programs of the FS-TST package.

Section 2 documents the requirements of each program. This section gives a concise description of what each program is supposed to do.

The program designs are discussed in **Section 3**. This section should be read in conjunction with the program listings. The objective of this section is to give the reader a high level view of the program design to help understand the source code.

Section 4 is a user manual and **Section 5** describes using the FS-TST tools on an example test case.

Not all of the document is of interest to every reader. The goals of the reader determine the relevant sections of the document. A bullet (●) in Table 1 indicates the relevant sections given the goals of the reader.

Table 1 Reader's Guide

Reader's Goal	Section				
	1	2	3	4	5
General understanding of FS-TST	●				●
Use FS-TST	●			●	●
Enhance or modify FS-TST	●		●	●	
Verify or test the operation of FS-TST	●	●		●	

1.2 Capabilities Required to Support Testing Disk Imaging Tools

The usual testing paradigm is a three step process.

1. Set up the test.
2. Execute the test
3. Measure the results.

This section establishes the capabilities required to set up and measure each test. This is accomplished by first determining what must be measured to confirm compliance with the tool requirements and second what conditions must be created to allow an accurate measurement of compliance.

The basic function of a disk imaging tool is to create a duplicate of either a hard disk drive or a partition from a hard disk drive. A disk imaging tool copies disk sectors from a source to a destination such that the destination is identical or nearly identical to the source. In the ideal situation, the destination is identical to the original. However, forensically valid differences do occur when the source and destination are not the same size or if partitions from the source disk must be relocated on the destination drive to keep the destination partitions accessible. The critical disk imaging requirements are the following:

- The tool shall not alter the original disk.
- The tool shall make a bit-stream duplicate or an image of an original disk or partition.
- The tool shall log errors (or other conditions of note).

1.2.1 Source Disk is Unchanged

The source disk must not be altered by running the disk imaging tool. One approach for detecting any changes to a disk is to compute a hash function of the entire contents of a disk both before and after the disk imaging tool is run. The likelihood that two disk drives with different content would produce the same hash value is related to the length of the hash function value. The secure hash algorithm (SHA-1) produces a 160 bit hash. This

gives a 1 in 2^{80} chance that any change to the source by the tool would be undetected. The hashing technique can also be used to verify that a block of sectors has not been changed by some operation.

1.2.2 Duplicate Creation

A program to compare the corresponding sectors of two disks can determine the accuracy or point of failure of the duplication process. If the destination is larger than the source, then there will be excess sectors on the destination. An analysis of the excess destination sectors is useful to determine tool behavior if the destination is larger than the source. However, to do such an analysis, the source and destination sectors need to be uniquely recognizable from the sector contents. If every sector of each disk used in a test is initialized to a known unique value then the task of determining the origin of each sector of the destination during the measurement of the results of a test run is simplified. When a destination sector is examined there are three possibilities for the contents of the sector: the original destination contents are unchanged by the test, the contents of a sector from the source disk is written to the destination or the imaging tool has generated new and unexpected contents for the sector.

Another factor to consider in duplicate creation is the accessibility of the duplicate. It may be desirable to ensure that the information copied to a disk with a different number of cylinders and heads can be accessed or booted. To create a bootable copy of the source, the partitions of the destination need to be aligned on cylinder boundaries. If there is more than one partition on the source then the destination partitions may be padded with extra sectors to create a bootable destination.

1.3 Summary of Required Capabilities

The minimum set of basic capabilities required to support testing a disk imaging tool is the following:

1. Initialize each sector of a disk to a known unique value.
2. Compute a SHA-1 hash of an entire disk or a block of disk sectors.
3. Compare two disks of either equal or unequal size.
4. Compare two disk partitions of either equal or unequal size.
5. Compare two disks where the destination disk has had partitions aligned to cylinder boundaries.

In addition to these basic capabilities there are several additional capabilities that are helpful. These include the following:

1. Compare two sectors.
2. Modify or examine the contents of a sector.
3. Relocate a block of sectors.
4. Examine a partition table.

1.4 Test Case Structure

Each disk imaging test case is designed to follow a sequence of steps that set up the test, execute the disk imaging tool under test and measure the results. The general structure of a test case is outlined in Figure 1. Usually a source drive is set up (steps 1-3) once and then used in more than one test case.

Figure 1 General Structure of a Test Case

Setup for the test case:

1. Record details of source disk setup.
2. Initialize the source disk to a known value.
3. Hash the source disk and save the hash value.
4. Record details of destination disk setup.
5. Initialize a destination disk.
6. If the test requires a partition on the destination, create and format a partition on the destination disk.
7. If the test uses an image file, partition and format a media disk.

Execute the tool being tested:

8. If the test requires an image file, use the tool to create an image file of the source on the media disk.
9. If the test requires a corrupted image file, corrupt the image file.
10. Use the disk imaging tool to create the destination disk either from a copy of the source disk or by restoring an image file of the source to the destination.

Measure the results:

11. Compare the source to the destination.
12. Compute a hash of the source disk and compare with the saved hash value.

1.5 Software Overview

To support the capabilities necessary for testing disk imaging tools the programs listed in Figure 2 were developed. The support software can be grouped into four categories: programs to set up something required for the test, programs to measure the results of a test, programs to document some aspect of a test, and utility programs to provide capabilities outside the scope of the usual test case.

Figure 2 List of Support Programs

Program	Category	Function
DISKWIPE	Setup	Write a unique pattern to each sector of a disk.
CORRUPT	Setup	Change one byte in a file (corrupt an image file).
ADJCMP	Measure	Compare two disks by partitions (allowing for cylinder alignment).
DISKCMP	Measure	Compare two disks.
PARTCMP	Measure	Compare two partitions.

Program	Category	Function
DISKHASH	Measure	Compute a SHA-1 for an entire disk.
SECHASH	Measure	Compute a SHA-1 for part of a disk.
LOGCASE	Document	Log information about the test run, e.g., disks used.
LOGSETUP	Document	Log information about source hard drive setup.
PARTAB	Document	Log partition tables.
DISKCHG	Utility	Change or examine the content of a single disk sector.
SECCMP	Utility	Compare two disk sectors.

2 Requirements

This section gives a concise description of what each program is supposed to do. The section is intended to serve as a reference for development of test cases to test the support programs. A reader with other goals could skip this section on first reading. The first subsection describes requirements for common functionality shared by most of the programs. The second subsection lists specific requirements for each program.

2.1 Requirements for Common Functionality

Most of the support programs share common functionality. For example, most support programs work with one or two disk drives and are required to log descriptive information about any disk drives used by the program. Rather than describe these common requirements with each program, they are described here once and then referenced as needed.

2.1.1 Program Execution Logging

These requirements specify what identifying information a program records in a log file each time the program is executed. A program required to do *program execution logging* must record:

1. The program name, version number, source file creation date & time, and compile date & time.
2. The support library name, version number, source file creation date & time, and compile date & time.
3. The header file name, version number, and source file creation date & time.
4. The command line (including command line options).
5. The date and time program execution begins and ends, and the elapsed time.
6. The test case ID.
7. The name of the computer where the program is executed.
8. A user supplied comment.

In addition, the program must provide command line options to:

9. Either start a new log file or append to an existing log file.
10. Either use the default log file name or use a user-provided log file name.
11. Print a summary of the program command line and command line options, then exit.

2.1.2 Disk Logging

Each hard disk drive used by a program is described in the program log file.
A program required to do *disk logging* must record the following information in the specified log file for the specified disk drive:

1. The disk geometry, i.e., maximum allowed cylinder value, maximum allowed head value, number of sectors per track and total number of sectors.
2. For IDE disk drives, record the model number and serial number as reported by the HDIO_GET_IDENTITY ioctl.
3. For SCSI disk drives, record the model number and serial number as reported by the SCSI INQUIRY command.
4. For disk drives that support LBA, record the *total number of user addressable sectors* as reported by the BLKGETSIZE ioctl.

2.1.3 Partition Table Logging

A program required to do *partition table logging* must record the following information for the partition table of the specified disk drive in the specified log file:

1. For each partition table entry in either the *master boot record* partition table and each partition table in any *extended partition*, print the following: starting LBA address, partition length, starting cylinder/head/sector address, ending cylinder/head/sector address, bootable flag and partition type code (in hexadecimal).
2. For common partition types (FAT12, FAT16, FAT32, extended, Linux EXT2, Linux swap, and NTFS) print a descriptive string, e.g., *Fat32* for type code 0x0B.

2.1.4 Comparison Logging

These requirements specify the information logged by a program required to do *comparison logging* when comparing a source to a destination. A source or destination is defined to be a block of contiguous disk sectors. A source or destination can be an entire disk, a disk partition or a block of sectors located between two partitions. The source (original) is assumed to have been initialized by **diskwipe** with the *source fill byte* and the destination is assumed to have been initialized by **diskwipe** with the *destination fill byte*.

1. Summarize corresponding sectors of the source and destination with counts of the sectors compared, sectors matching (have the same content), sectors differing and the total number of bytes that are different. Note that if large disk drives with few matching bytes are compared then the total number of matching bytes may exceed the maximum integer that can be represented by a single variable. In this case, overflow is permitted without notification.
2. If the source and destination are not the same size, log the size of each and the difference in size.
3. If the destination is larger than the source, categorize the excess sectors according to the following: zero fill (every byte is zero), **diskwipe** style fill, and other contents. The **diskwipe** style fill is actually three categories: source fill byte, destination fill byte and any other fill byte.

4. For each category, the first few (up to some arbitrary limit) sectors belonging to the category are logged. A contiguous block of sectors is logged as a hyphen separated pair of integers (*start sector - last sector*).

2.1.5 Error Reporting

The following requirements apply to all programs except as noted under each program. In general, the support programs should not encounter any I/O errors during a valid test run. If an I/O error occurs then the support program results are corrupt and cannot be used.

1. If the command line parameters are not valid, print an error message indicating the problem, print a summary of the program command line and command line options, then exit the program.
2. If any I/O operation fails, print a diagnostic message and then exit the program.
3. If any I/O operation fails, the content of the log file is undefined (the log file should be considered corrupt).

2.2 Individual Program Requirements

This section presents the specific requirements for each support program.

2.2.1 DISKWIPE

For a designated hard drive, the **diskwipe** program writes unique contents to each disk sector. The first 25 bytes of each disk sector contain an ASCII string representing both the C/H/S and LBA address of the sector. The remaining bytes of each disk sector are set to a specified hexadecimal fill byte value in the range 0x00 to 0xFF.

1. Write the specified content from Table 2 to each disk sector on the specified drive.
2. Allow specification of at least three log file names, one for a source disk, one for a destination disk, and one for a media disk.
3. Log the specified hard drive.
4. Log the program execution.
5. By default, use the number of heads obtained from the **ioctl** calls; however, optionally allow specification of the number of heads to override value from the **ioctl**.

Table 2 DISKWIPE Pattern Specification

Sector Contents	
Bytes	**Contents**
0-11	The C/H/S address of the sector %05d/%03d/%02d
12	Blank (space character)
13-24	LBA address of the sector: %012d
25	Null byte (zero)
26-511	Specified hexadecimal fill byte in the range 0x00 to 0xFF

2.2.2 CORRUPT

Corrupt is used to change a single byte of an image file.

1. Log the program execution.

2. Change a specified byte at a specified location in a specified file to a specified value.
3. Log the original value at the specified location.
4. Log the new value at the specified location.

2.2.3 ADJCMP

Adjcmp is used to compare two disks where the partitions copied to the destination are adjusted so that the copy is aligned on a cylinder boundary.

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the partition table for the specified hard drive.
5. For each disk, assign each sector to a contiguous block of sectors, called a *disk chunk*, such that each disk chunk is assigned to one of the following *chunk categories*: a sector contained within a partition, a sector contained within a partition boot track, the unallocated sectors between two partitions or unallocated sectors after the last partition on the disk.
6. Record the location of each disk chunk in the log file.
7. Allow specification of corresponding disk chunks between the source hard drive and the destination hard drive. (A disk chunk on the source drive is compared to the corresponding disk chunk on the destination drive.)
8. Log the correspondence between source disk chunks and destination disk chunks; i.e., for each disk chunk on the source drive, log the disk chunk on the destination that the source disk is to be compared to.
9. Log the comparison between each pair of corresponding disk chunks.
10. For any destination disk chunks that have no corresponding source chunk, categorize the sectors of the disk chunk according to the following: zero fill (every byte is zero), **diskwipe** style fill, and other contents. The **diskwipe** style fill is actually three categories: source fill byte, destination fill byte and any other fill byte. For each category, the first few (up to some arbitrary limit) sectors belonging to the category are logged. A contiguous block of sectors is logged as a hyphen separated pair of integers (*start sector - last sector*).
11. Log a summary as follows:
    - Number of boot tracks, total number of sectors assigned to boot tracks, and number of boot track sectors that do not compare equal.
    - Number of partitions, total number of sectors assigned to some partition, and number of corresponding partition sectors that do not compare equal.
    - Number of unallocated chunks with a corresponding unallocated chunk, number of sectors in this category and number of corresponding sectors that do not compare equal.
    - Number of excess sectors in destination chunks that have a corresponding source chunk, number of sectors that have every byte set to zero, and number of remaining sectors.
    - Number of sectors in destination chunks that do not have a corresponding source chunk, number of sectors that have every byte set to zero, and number of remaining sectors.
    - Total number of source sectors and total number of destination sectors.

2.2.4 DISKCMP

Diskcmp is used to evaluate the accuracy of a disk duplication operation.

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the comparison between the source drive and the destination drive.
5. If there is a read error the comparison results are undefined.
6. If there are any read errors, then continue scanning the disk and log a count of the number of tracks with read errors on each disk.

2.2.5 PARTCMP

Partcmp is used to evaluate the accuracy of a partition duplication operation.

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. Log the comparison between the source partition and the destination partition.

2.2.6 DISKHASH

For a designated hard drive, the **diskhash** program computes a SHA-1 hash on a specified hard disk drive.

1. Compute a SHA-1 hash for the designated hard drive.
2. Allow specification of at least two log file names, one for reference before a tool is run and one for comparison after a tool is run.
3. Log the specified hard drive.
4. Log the program execution.

2.2.7 SECHASH

For a designated hard drive, the **sechash** program computes a SHA-1 on a block of contiguous sectors from specified hard disk drive.

1. Compute a SHA-1 for a specified block of continuous sectors from the designated hard drive.
2. Log the computed hash value.
3. Allow specification of at least two log file names, one for reference before a tool is run and one for comparison after a tool is run.
4. Log the specified hard drive.
5. Log the program execution.

2.2.8 LOGCASE

Logcase is used to create a log file recording basic information about the test case.

1. Record the following: Test case ID, host computer, operator, source disk drive, destination disk drive, other disk drive, date and time.

2.2.9 LOGSETUP

Logsetup is used to log information about the setup of a source disk.

1. Record the following: disk label, host computer, operator, operating systems loaded, date and time.

2.2.10 PARTAB

Partab is used to print the partition table for a hard drive.

1. Log the specified hard drive.
2. Log the program execution.
3. Log the partition table for the specified hard drive.
4. Use a different log file name for each hard drive.
5. Log (optionally by command line control) a unique identification for each partition that can be used by the **partcmp** program to select partitions for comparison.
6. Log (optionally by command line control) empty partition table entries.

2.2.11 DISKCHG

Diskchg is used to setup hard disk drives for the testing of the other support programs. **Diskchg** can be used to set a single byte of a specified sector to a given value or to set an entire sector either to all zero bytes or to **diskwipe** style fill. **Diskchg** can also be used to examine the contents of a given sector.

1. Log the specified hard drive.
2. Log the program execution.
3. Allow specification of disk sector addresses in either CHS or LBA format.
4. Set every byte of a specified sector to zero.
5. For a specified sector, *s*, a specified address, *a*, (possibly not the same as the specified sector), a specified disk geometry and a specified fill value, fill sector *s* with the contents of a **diskwipe** style fill using *a* as the address value for the fill. In other words, set sector *s* to the contents that **diskwipe** would use for the sector at location *a* on a disk with the specified geometry using the specified fill value.
6. For a specified sector, a specified offset within the sector and a specified value, set the byte at the offset within the sector to the specified value.
7. For a specified hard drive, a specified sector, a specified offset within the sector and a specified count, log the contents *count* bytes from the specified sector starting at the specified offset.
8. Allow interactive examination of sector contents.
9. Use a different log file name for each function.

2.2.12 SECCMP

Seccmp is used to compare two disk sectors.

1. Log the specified source drive.
2. Log the specified destination drive.
3. Log the program execution.
4. If the sectors to compare are not **diskwipe** style filled or zero filled, log any differences between the source sector and the destination sector.
5. **Diskwipe** style filled sectors or zero filled sectors are logged with no need for comparison.
6. Allow specification of an alternate log file name.

3 Design Notes

This section describes the design of the support software. The logical structure of most of the support software is usually a simple loop. The program determines the number of

sectors in the selected object (disk or partition) and then performs some operation on each sector from the first to the last. The **ADJCMP** program is the most complex of all the support programs: the disk is divided into regions (partitions and unallocated space) that are each processed in turn via a nested loop (for each region do the operation for each sector of each region). A few programs only operate on a single sector or follow the linked list of a partition table.

The physical structure of the support programs is also simple. Except for **DISKHASH** and **SECHASH**, each program is compiled and linked from two source files and a header file. Each program has a **main** source file that contains the **main** procedure (e.g., **diskwipe.c**) and the essential functionality of the program. The second file, the library (**zbios.c**), contains functions that are commonly used by the support programs, such as functions to read or write disk sectors by absolute disk address. In general, the requirements listed in section 2.1 Requirements for Common Functionality, such as logging, are implemented in the library.

The remainder of this section is organized as follows: the next subsection discusses data structures defined in **zbios.h**, subsection 3.2 discusses the functions defined in **zbios.c**, and the individual programs are discussed in subsection 3.3.

3.1 Data Structures

The data structures defined in **zbios.h** are used to keep track of information about disk drives. Each data structure is defined in a **typedef** as a block of storage and as a pointer to a block of storage. There are three major data structures defined: data structures to describe a disk drive, data structures for handling partition tables and data structures for tracking lists of sectors with a common property.

3.1.1 Describing a disk drive

The **disk_control_block** is the data structure for describing a disk drive. Several other structures defined in **zbios.h** are components of the **disk_control_block**. These data structures are: **chs_addr, ide_rec, ide_ptr, physical_sector, and physical_track.**

Figure 3 Data Structures for Describing a Hard Drive

```
typedef struct {off_t cylinder,head,sector; } chs_addr;
typedef unsigned char physical_sector[BYTES_PER_SECTOR]; /* a sector of 512 bytes */
typedef physical_sector physical_track[DISK_MAX_SECTORS]; /* a track is an array of 63
sectors */

struct disk_struct {
       char           dev[NAME_LENGTH]; /* drive name */
       char           serial_no[21];    /* serial# */
       char           model_no[41];     /* Model# */
       int            fd;               /* reference to the hard drive */
       int            drive_type;       /* SCSI or IDE */
       off_t          n_sectors;
       chs_addr       disk_max;         /* number of cyl, number of head */
       int            geometry_is_real; /* 1 if able to find C/H/S; 0 if unable */
       physical_track buffer;           /* 63 sectors (1 track) buffer[sector][byte]*/
};
typedef struct disk_struct disk_control_block, *disk_control_ptr;
```

Any I/O to or from the disk takes place one track at a time. The data buffer is built up as an array of 63 sectors of 512 bytes per sector. The **dev** field contains the full path of the

drive's device. The **serial_no** and **model_no** fields are set by the **ioctl** command **HDIO_GET_IDENTITY** if the drive is IDE; otherwise, they are set from data returned by the **SCSI INQUIRY** command. The **fd** field is set when the device is opened. The **drive_type** field is determined by the device name. The **n_sectors** field is set to a value computed from values returned by the **ioctl** command **BLKGETSIZE**. The field **disk_max** contains the disk geometry returned by the **ioctl** command **HDIO_GETGEO**.

3.1.2 Describing a partition table

Partition tables are stored on a hard drive as a linked list with a root node in the master boot record of the drive. The layout of the master boot record is in the **mbr_sector** structure. The first 446 bytes are not part of the partition table. The four partition table entries are found in the **pe** field. The partition table is followed by a signature byte in the **sig** field. A partition table entry is described by the **partition_table_rec** data structure. It should be noted that the two high order bits of the 10 bit cylinder number are located in the two high order bits of the sector number field. After the partition table is read from the disk, the each partition table entry is placed in a linked list of **pte_rec** structures.

Figure 4 Data Structures for Describing a Partition Table

```
/*************************************************************************
Partition table entry layout on disk
*************************************************************************/

typedef struct pts { /* partition table layout */
       unsigned char    bootid,
                        start_head,
                        start_sector,
                        start_cylinder,
                        type_code,
                        end_head,
                        end_sector,
                        end_cylinder;
       unsigned int     starting_lba_sector, n_sectors;
} partition_table_rec, *partition_table_pointer;
/*************************************************************************
Layout of a partition table in a boot sector
*************************************************************************/

typedef struct { /* partition table in Master Boot Record */
        char                  fill[446]; /* MBR boot code */
        partition_table_rec   pe[4] PK;  /* master partition table */
        unsigned short        sig;       /* partition table signature word 0xAA55 */
}mbr_sector,*mbr_ptr;

/*************************************************************************
Data structure to keep partition table information
*************************************************************************/

typedef struct pte_struct pte_rec, *pte_ptr;
struct pte_struct {
        pte_ptr               next;
        unsigned char         is_boot,type;
        chs_addr              start,end;
        off_t                 lba_start,lba_length;
};
```

3.1.3 Describing a range of sectors

The compare programs examine each disk sector in LBA address sequence and assign each sector to a category, e.g., dst sector that differs from corresponding src sector, zero

filled, src filled, dst filled, etc. This data structure is used to track blocks of disk sector LBA addresses for sectors that are classified in the same category. A range is recorded as a pair, **from** and **to**. The **n** field counts the number of pairs in the **r** field. The **is_more** field is **true** if there are more ranges than **r** can contain.

Figure 5 Data Structures for Describing a Range of Sectors

```
/*****************************************************************************
Data structure to track ranges of integers. The compare programs examine each
disk sector in LBA address sequence and assign each sector to a catagory, e.g.,
dst sector that differs from corresponding src sector, zero filled, src filled,
dst filled, etc. This data structure is used to track blocks of
disk sector LBA addresses for sectors that are classified in the same catagory.
*****************************************************************************/

typedef struct {off_t from,to;} lba_range;
typedef struct { /* structure to keep a list of ranges */
       int     n;
       long    is_more; /* more than N_RANGE ranges present (i.e., some not recorded) */
       lba_range r[N_RANGE];
} range_list,*range_ptr;
```

3.2 Library Functions Designs

The library functions can be grouped into five categories: disk drive information, disk drive I/O, logging, partition tables, and range lists. The functions are usually short with the longest about 60 C code statements.

3.2.1 Disk Drive Information Functions

The functions that obtain information about disk drives use different **ioctl** functions to gather information.

The following functions obtain information about disk drives:

- **trim** This function removes excess blanks from the model number and serial number obtained by **probe_ide_adapter**.
- **open_disk** This function calls the other disk information functions to create a **disk_control_block** for a specified drive.

3.2.2 Disk Drive I/O Functions

These functions handle the actual I/O to the disk. All operations are performed on an entire track of 63 sectors at a time.

- **lba_to_chs** This is a utility function to convert an LBA address to CHS notation.
- **disk_write** This function writes an entire track to a specified disk.
- **disk_read** This function reads an entire track from the specified disk.
- **read_lba** This function returns a pointer to the disk sector at the specified address. If the requested sector is not in the **disk_control_block** buffer, then **disk_read** is called to read the appropriate track into the buffer.

3.2.3 Logging Functions

These functions write selected information to a log file.

- **log_open** This function creates a log file and writes descriptive information about the program to the log.
- **log_close** This function calculates the elapsed program run time and writes the value to the log file.
- **log_disk** This function logs information about the selected disk drive to the log file.
- **feedback** This function provides progress feedback to the program operator for a program iterating through a loop. An estimate of the program run time remaining is printed to the console based on how much time has elapsed, the number of iterations executed so far and the total number of iteration to execute.

3.2.4 Partition Tables Functions

These functions fetch the partition table from a specified disk drive to create a linked list of partition table entries that can be examined or printed.

- **get_sub_part** This function is called to follow the linked list of partition table entries in an extended partition.
- **get_partition_table** This function reads the master boot record partition table and builds a linked list of partition table entries. If there are any extended partitions then **get_sub_part** is called.
- **print_partition_table** This function is called to print the partition table linked list created by **get_partition_table**. The function **partition_type** is called to map partition type codes to ASCII strings.
- **partition_type** This function maps common partition type codes to an ASCII string.

3.2.5 Range List Functions

The range list functions manage lists of ranges. Several of the programs categorize disk sectors according to content criteria and report on the number of sectors in each category. It is also useful to get a general idea of where and how each category is distributed on a disk. By tracking the first few contiguous groups of sectors in each category a general indication of the distribution can be obtained.

The **lba_range** data structure stores a range as two values, **from** and **to**, indicating the lower and upper end points of a range.

- **create_range_list** This function returns an empty range list.
- **add_to_range** This function adds a new value to a range. The list of ranges is searched and if a range is found such the new value is one step outside an existing range then that range is expanded by one. Otherwise a new range is created with the new value as the upper and lower end points. The **add_to_range** function assumes that the sectors are scanned in sequence so that a range can be built up as the scan progresses.
- **print_range_list** This function prints a range list.

3.3 Individual Program Designs

This section gives a general outline of each program. Most of the programs are small (less than 200 statements) and follow the same general plan (Figure 6). The **first_sector** and **last_sector** might encompass the entire disk, a single partition or only one sector depending on the actual program.

Figure 6 General Program Design for Support Software

```
main()
{
        decode command line parameters
        if (error) print message & quit
        open log file
        open disks
        log disks
        for (i = first_sector; i < last_sector; i++) {
                do something
                if (error) print message & quit
        }
        log results
        close log file
}
```

The programs **DISKWIPE, DISKCMP, DISKCHG, PARTCMP, PARTAB, ADJCMP, SECCMP,** and **CORRUPT** all follow the general support software design. The remaining subsections of this section discuss any deviation from the general program design and the *do something* aspect of the general program design in Figure 6.

3.3.1 DISKWIPE

The **diskwipe** program sets up the known unique content for each sector of a disk track and then when the contents for the last sector of a track is setup, writes the content for the given track to the disk.

3.3.2 CORRUPT

The **corrupt** program only operates on a single byte of the specified file. The byte, indicated by an offset from the beginning of the file, is read, logged and a new value is written back to the file.

3.3.3 ADJCMP

The **adjcmp** program adds an outer loop that indexes each disk chunk. The inner loop runs from the first sector to the last sector of each disk chunk.

3.3.4 DISKCMP

The **diskcmp** program reads corresponding sectors from the source and destination, checks for an exact match and tallies the result. If the destination is smaller than the source, no more sectors are examined, the results are logged and the program exits. If the destination is larger than the source, the excess destination sectors are classified, the results are logged and the program exits.

3.3.5 PARTCMP

The **partcmp** program reads corresponding sectors from the source and destination, checks for an exact match and tallies the result. If the destination is smaller than the source, no more sectors are examined, the results are logged and the program exits. If the destination is larger than the source, the excess destination sectors are classified, the results are logged and the program exits.

3.3.6 DISKHASH

The **diskhash** program reads each disk sector and calls a SHA-1 function to add each sector of data to the hash. After all the sectors have been added to the hash, the final SHA-1 hash value is extracted and logged.

3.3.7 SECHASH

The **sechash** program reads each disk sector from a range of sectors specified on the command line and calls a SHA-1 function to add each sector of data to the hash. After all the sectors have been added to the hash, the final SHA-1 hash value is extracted and logged.

3.3.8 LOGCASE

LOGCASE writes the command line parameters to a file to document disks used in a test case.

3.3.9 LOGSETUP

LOGSETUP writes the command line parameters to a file to document setup of a source disk.

3.3.10 PARTAB

Print the partition table from the boot sector. For each extended partition, follow the links to each extended table entry and print the entries.

3.3.11 DISKCHG

Decode the command line options and do the requested operation.

3.3.12 SECCMP

Read the two specified sectors, compare the data, and log the results.

4 User Manual

This section describes in detail how to use each program. The organization of this section is as follows. Section 4.1 describes the expected environment, Section 4.2 is a general discussion of the program log files, and Section 4.3 describes the format of error messages. Section 4.6 gives an overview of features common to the comparison programs. Each of the remaining sections describes one of the support programs. For each program there is a general description of the program followed by subsections devoted to *command line parameters, error messages*, and other topics as needed.

4.1 System Environment

The expected operating environment is an Intel X86 (or Pentium) architecture PC running RedHat Linux 9 with a floppy disk drive and at least two hard disk drives. The hard drives may use either an IDE or a SCSI interface; however, all disks must have 63 sectors per track.

4.2 Log files

Each program keeps a log file of results. The log files are named so that all the log files for a given test case can exist in the same file system directory. This is the rationale for some programs having options to set the log file name. For example, a test may use one, two or three disks. Each disk must be uniquely initialized. If each log file had the same name there would be a conflict. All support programs log the following information:

- Program version, date compiled, time compiled.
- Support library version, date compiled, time compiled.
- Test case ID (from command line)
- Host computer (from command line)
- A user supplied comment
- List of command line options specified
- Run start date and time
- Run finish date and time
- Elapsed time

The following information is logged for each disk used by the program:

- Drive number
- Geometry reported by **ioctl**
- Total sectors reported by **ioctl**
- Model and serial numbers and number of sectors accessible in LBA mode

Each program also logs any information specific to the program.

Table 3 Program Log Files

Program Log Files		
Program	**Logfile name**	**Comments**
diskwipe	wipedlog.txt	Default log name
diskwipe	wipeslog.txt	**-src** option selected. Used to wipe a src disk
diskwipe	wipemlog.txt	**-media** option selected. Used to wipe a media disk
corrupt	corlog.txt	Old and new values of the selected byte
adjcmp	cmpalog.txt	Compare two disk drives, allowing for cylinder alignment
diskcmp	cmplog.txt	Compare two disk drives (physical drives)
partcmp	cmpptlog.txt	Compare two partitions (logical drives)
diskhash	hashblog.txt	/**before** option selected – hash before running tool
diskhash	hashalog.txt	/**after** option selected – hash after running tool
sechash	hashbsec.txt	/**before** option selected
sechash	hashasec.txt	/**after** option selected
logcase	case.txt	Information about the test case

<table>
<tr><td>logsetup</td><td>setup.txt</td><td>Information about a disk setup</td></tr>
<tr><td>partab</td><td>pt-XXX-log.txt</td><td>XXX is replaced by the device name.</td></tr>
<tr><td>diskchg</td><td>cg-XXX-flog.txt</td><td>XXX is replaced by the device name. F is determined by the options selected.
<table>
<tr><th>Option</th><th>Logfile name</th></tr>
<tr><td>-read</td><td>cg-XXX-rlog.txt</td></tr>
<tr><td>-write</td><td>cg-XXX-wlog.txt</td></tr>
<tr><td>-fill</td><td>cg-XXX-flog.txt</td></tr>
<tr><td>-zero</td><td>cg-XXX-zlog.txt</td></tr>
<tr><td>-exam</td><td>cg-XXX-xlog.txt</td></tr>
</table></td></tr>
<tr><td>seccmp</td><td>seclog.txt</td><td>Default log file name. –log_name <name> overrides.</td></tr>
</table>

4.3 Error Messages

The major error messages for each program are noted. Location of information specific to the error instance is indicated by the C/printf code used to format the information. Examples of such specific information include *disk drive number* (%X), *status codes* (%d) and *disk logical block address* (%llu). The code %X indicates a hexadecimal value, %d indicates an integer, %llu indicates an unsigned long long (very large) integer and %s indicates a text string.

For the comparison programs, the same message may be possible for both the source and the destination. This is indicated by *[src/dst]* rather than repeating the explanation of the error message twice.

4.4 Disk initialization: DISKWIPE

DISKWIPE initializes a disk to a known value. DISKWIPE uses a two digit hexadecimal value as a fill byte on each sector of the disk. DISKWIPE writes the sector address in both CHS and LBA style to the first 25 bytes of each sector (see Table 2 DISKWIPE Pattern Specification).

DISKWIPE wipes all the user addressable sectors reported. DISKWIPE may wipe sectors beyond the last sector address reported by **ioctl**. DISKWIPE logs the number of sectors wiped and the hexadecimal fill value.

A disk used in testing disk imaging tools may be used in one of three roles: setup as a known source for copy or image operations, setup with a file system to contain an image file, or as setup as a destination for a copy or image restore operation. The role of the disk in the test case determines the selection of **-src**, **-media** or **-dst** options.

A suggested method for assigning hexadecimal fill values is to give each disk drive a unique label of two hexadecimal digits. The disk label can then be used as the fill value for any operations on the disk.

4.4.1 DISKWIPE command line

`diskwipe test-case host operator drive fill [-opts]`

Table 4 DISKWIPE Command Line Parameters

Parameter	Description
`test-case`	A test case identifier.
`host`	The name of the computer running the test.
`operator`	The initials of the person running the test.
`drive`	The drive's device name. Example: /dev/hda
`fill`	A two digit hex value, used to fill each sector.
`-comment "…"`	A comment for the test log file. If this option is not used, the program prompts for a log floppy to be inserted. If this option is used, then a log floppy is assumed to be already loaded.
`-src`	Set log file name to: wipeslog.txt (defaults to wipedlog.txt). Use this option if the disk is used as a source.
`-media`	Set log file name to: wipemlog.txt (defaults to wipedlog.txt). Use this option if the disk is used to contain an image file.
`-dst`	Use this option for disks used as a destination. Set log file name to: wipedlog.txt
`-heads nnn`	Override the number of heads reported by the BIOS. This only changes the pattern of addresses written to each sector.
`-noask`	Skip the confirmation dialog asking if you are really sure you want to wipe out a disk. This option is intended for running DISKWIPE from a script (unattended).
`-new_log`	Start a new log file rather than append to an existing log file.
`-log_name <name>`	Use alternate log file name.
`-h`	Print a summary of command options and exit

4.4.2 DISKWIPE error messages

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./diskwipe DI-01 lab joe /dev/hda B4 -comment "Wipeout disk labeled B4"
```

- **-log_name option requires a logfile name**

This option must be followed by the name of a log file to use instead of the default log file.

- **-heads option requires a value**

This option must be followed on the command line with a decimal integer (the number of heads per track).

- **could not access drive %s status code %d**

The specified disk drive could not be accessed. Most likely cause is the drive number does not refer to an attached drive or the indicated drive may not be properly installed.

- **Error: %i (%s) has occurred attempting to write to %s\n**

The disk has not been completely wiped. There is likely a hardware problem that must be resolved.

4.5 Corrupt an image file: CORRUPT

CORRUPT is used to corrupt an image file by changing a single selected byte in an image file. CORRUPT logs the original content of the selected byte, the replacement value, the index of the byte within the file and the file name.

4.5.1 CORRUPT command line

`corrupt test-case host operator image-file index value [-opts]`

Table 5 CORRUPT Command Line Parameters

Parameter	Description
`test-case`	A test case identifier.
`host`	The name of the computer running the test.
`operator`	The initials of the person running the test.
`image-file`	Full path name of the image file to be corrupted.
`index`	Relative index position of the byte to be changed.
`value`	A two digit hex value, used to replace the selected byte.
`-comment "…"`	A comment for the test log file. If this option is not used, the program prompts for a log floppy to be inserted. If this option is used, then a log floppy is assumed to be already loaded.
`-new_log`	Start a new log file rather than append to an existing log file.
`-log_name <name>`	Use alternate log file name.
`-h`	Print a summary of command options and exit.

4.5.2 CORRUPT error messages

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./corrupt DI-01 lab joe image 123456 AE -comment "Corrupt byte 123456"
```

- **-log_name option requires a logfile name**

This option must be followed by the name of a log file to use instead of the default log file.

- **Open %s failed with error %i (%s)**

CORRUPT could not open the specified file. The file name or path may be incorrect. The open would also fail if the file is marked read only.

- **Offset value (%s) is not valid**

CORRUPT could not convert the offset to an integer. A likely cause would be if CORRUPT tried to convert a string that contained characters other than digits.

- **Replacement value (%s) is not valid**

CORRUPT could not convert the replacement value to an integer. A likely cause would be if CORRUPT tried to convert a string that contained characters other than digits.

- **Offset %llu is not valid for %s**

CORRUPT could not seek to the specified offset location in the file. A likely cause would be if an offset value larger than the file size were specified.

- **Read failed\n**

CORRUPT failed to read the byte designated for replacement.

- **Seek to %llu on %s after read failed**

Should not happen unless the disk fails.

- **Write failed**

Should not happen unless the disk fails.

- **Seek to %llu on %s after write failed**

Should not happen unless the disk fails.

- **Verify read failed**

Should not happen unless the disk fails.

- **verify failed: char read back (0x%02X) should be (0x%02x)**

Should not happen unless the disk fails.

4.6 Comparison Software: ADJCMP, DISKCMP and PARTCMP

The comparison software is a group of five programs. The program selected depends on the requirements of the test case.

Table 6 Comparison Programs

Program	Test case requirements
DISKCMP	Compare two entire disks' sectors from the same address on each disk, sector by sector.
PARTCMP	Compare two partitions.
ADJCMP	Compare two entire disks when there has been cylinder alignment, i.e., the destination drive does not have the same number of heads as the source and partitions on the destination drive may have been shifted to cylinder boundaries.
SECCMP	Compare two sectors. This program is used to investigate, in detail for selected sectors, the results obtained from the other three compare programs. This utility is described in Section 4.13.

All the comparison software assumes the following preconditions:

1. DISKWIPE has been run with different fill values on each disk.
2. One disk, designated the source, has been copied by a disk imaging tool to the other disk, designated the destination. The destination may actually be created by either a copy or an image restore operation.

The programs DISKCMP, PARTCMP and ADJCMP compare the source to the destination and log the following:

- Relative and absolute size of source and destination
- Number of sectors compared
- Number of sectors that match (identical contents)
- Number of sectors that differ
- Total number of bytes that differ

If the destination is larger than the source then the following is also reported for destination sectors that do not correspond to any source sector:

- Number of sectors zero filled
- Number of sectors filled with the source disk fill byte
- Number of sectors filled with the destination disk fill byte
- Number of sectors filled with some other fill byte
- Number of sectors with some other content

The addresses of the first few sectors in each category are also listed for possible investigation by the SECCMP program. Contiguous sectors are reported as a range of sector addresses.

4.6.1 Compare two disks by partitions: ADJCMP

Some disk imaging tools optionally create a destination with partitions aligned to disk cylinder boundaries. All the content of the source disk is reproduced on the destination, but not at the same location as on the source disk. The ADJCMP program is used to compare corresponding sectors of two disks where the corresponding sectors are located at different addresses on the source disk and the destination disk.

ADJCMP divides the source and destination disk into *disk chunks*, where a *disk chunk* is defined as a group of contiguous sectors (as listed below). Corresponding disk chunks between the source and destination are identified, and sectors at the same location relative to the first sector of the corresponding chunks are compared. A disk chunk is one of the following:

- Partition defined in the partition table
- Partition boot track (the track just before the start of the partition)
- Unallocated sectors (a group of contiguous sectors not part of any partition)

ADJCMP automatically designates disk chunks found in the same order on the source and destination as corresponding. The user may optionally reassign the correspondence between pairs of disk chunks by an interactive dialog.

A comparison report is logged for each disk chunk and a summary report for the entire disk. An example summary report is as follows:

Figure 7 ADJCMP Summary Report Example

```
Summary
Boot tracks  4           252 diffs          4
Partitions   6       2241540 diffs          9
Unallocated  4        931392 diffs          1
Total src sectors    3173184
Partition excess       76671 zero      74189 non-zero       2482
```

```
Disk excess          9422595 zero    9408998 non-zero      13597
Total dst sectors   12672450
```

An example summary report is presented in Figure 7. The summary report groups the results into four categories: boot tracks, partitions, partition excess areas (added to the destination partition so that the destination partition fills out the last cylinder of the partition) and disk excess sectors (sectors not part of a partition remaining on the destination after all the source has been copied to the destination. For boot tracks and partitions, the critical value is the number of sectors that do not match (diffs). For the partition fill area and excess sectors, the values reported indicate number of sectors filled with null bytes (zero) or with come other content.

In the example, there are 4 boot tracks with 252 sectors. There are 4 sectors of the 252 boot track sectors that differ. There are a total of 2241540 sectors allocated to 6 partitions with 9 sectors that do not match. There are 931392 sectors not allocated to a partition (or boot track). There is one unallocated sector on the source drive that does not match the corresponding sector on the destination. This accounts for the 3173184 sectors of the source drive and the corresponding sectors on the destination drive.

The excess sectors, i.e., destination sectors that do not correspond to any source sector, are analyzed in the latter part of the summary report. In this example, 76671 sectors are used to pad out partitions for cylinder alignment. Of these 76671 sectors, 74189 are filled with zeros and 2482 have other content. There are 9422595 sectors on the destination beyond the last partition. Of these sectors, 9408998 are zero filled and 13597 have other content.

4.6.1.1 ADJCMP command line

`adjcmp test-case host operator src-drv src-fill dst-drv dst-fill [-opts]`

Table 7ADJCMP Command Line Parameters

Parameter	Description
`test-case`	A test case identifier.
`host`	The name of the computer running the test.
`operator`	The initials of the person running the test.
`src-drv`	The drive's device name. Example: /dev/hda
`src-fill`	A two digit hex value, used to fill each sector by **diskwipe**.
`dst-drv`	The drive's device name. Example: /dev/hdb
`dst-fill`	A two digit hex value, used to fill each sector by **diskwipe**.
`-comment "…"`	A comment for the test log file. If this option is not used, the program prompts for a log floppy to be inserted. If this option is used, then a log floppy is assumed to be already loaded.
`-assign`	Engage the user in a dialog to assign corresponding regions of the source and destination disks for comparison.
`-new_log`	Start a new log file rather than append to an existing log file.
`-log_name <name>`	Use alternate log file name.

`-h`	Print a summary of command options and exit.

4.6.1.2 ADJCMP error messages

- **missing parameters**

One or more of the six required command line parameters is missing.

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./adjcmp DI-01 lab joe /dev/hda B4 /dev/hdb CC -comment "Compare B4 to CC"
```

- **-log_name option requires a logfile name**

This option must be followed by the name of a log file to use instead of the default log file.

- **could not access [src/dst] drive %s status code %d**

The specified disk drive could not be accessed. Most likely cause is the drive number does not refer to an attached drive or the indicated drive may not be properly installed.

- **[src/dst] read error 0x%02X on track starting at lba %llu**

The disk may have bad sectors. There is likely a hardware problem that must be resolved.

- **-select requires two parameters: src partition index\n and dst partition index**

This option must be followed by two integer values.

- **could not find [src/dst] partition index %d**

The partition index specified does not exist on the selected disk. Try running without the **/select** option or use the PARTAB program to get the valid index values.

4.6.2 Compare two disks ignoring disk layout: DISKCMP

The DISKCMP program compares sectors located at the same address from the source and destination disk. Disk partition tables or other content is irrelevant to the program; everything is compared. DISKCMP also tracks disk read errors, reports the location of the first few (about 10) and prints a count of the number of read errors from the source disk and from the destination disk. If there are any read errors, then the comparison log is not valid.

4.6.2.1 DISKCMP command line

`diskcmp test-case host operator src-drv src-fill dst-drv dst-fill [-opts]`

Table 8 DISKCMP Command Line Parameters

Parameter	Description
`test-case`	A test case identifier.
`host`	The name of the computer running the test.
`operator`	The initials of the person running the test.
`src-drv`	The drive's device name. Example: /dev/hda

src-fill	A two digit hex value, used to fill each sector by **diskwipe**.
dst-drv	The drive's device name. Example: /dev/hdb
dst-fill	A two digit hex value, used to fill each sector by **diskwipe**.
-comment "…"	A comment for the test log file. If this option is not used, the program prompts for a log floppy to be inserted. If this option is used, then a log floppy is assumed to be already loaded.
-new_log	Start a new log file rather than append to an existing log file.
-log_name <name>	Use alternate log file name.
-h	Print a summary of command options and exit.

4.6.2.2 DISKCMP error messages

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./diskcmp DI-01 lab joe /dev/hda B4 /dev/hdb B9 -comment "Compare B4
with B9"
```

- **-log_name option requires a logfile name**

This option must be followed by the name of a log file to use instead of the default log file.

- **could not access [src/dst] drive %s status code %d**

The specified disk drive could not be accessed. Most likely cause is the drive number does not refer to an attached drive or the indicated drive may not be properly installed.

- **[src/dst] read error 0x%02X on track starting at lba %llu**

The disk may have bad sectors. There is likely a hardware problem that must be resolved.

4.6.3 Compare two disk partitions: PARTCMP

The PARTCMP program compares sectors located at the same address relative to the beginning of the respective partitions. The source and destination must both have valid partition tables. The partitions to compare are specified by an index into the partition table. The possible index values can be determined in two ways. Either run the program in interactive mode (do not specify the **-select** option) or examine the output of the PARTAB program for each disk.

4.6.3.1 PARTCMP command line

partcmp test-case host operator src-drv src-fill dst-drv dst-fill [-opts]

Table 9 PARTCMP Command Line Parameters

Parameter	Description
test-case	A test case identifier.
host	The name of the computer running the test.
operator	The initials of the person running the test.
src-drv	The drive's device name. Example: /dev/hda

`src-fill`	A two digit hex value, used to fill each sector by **diskwipe**.
`dst-drv`	The drive's device name. Example: /dev/hdb
`dst-fill`	A two digit hex value, used to fill each sector by **diskwipe**.
`-comment "…"`	A comment for the test log file. If this option is not used, the program prompts for a log floppy to be inserted. If this option is used, then a log floppy is assumed to be already loaded.
`-select sx dx`	Set the value of the source (sx) and destination (dx) partition index. If this option is not given then the program presents the user with a list of partition table entries (and index values) for selection.
`-new_log`	Start a new log file rather than append to an existing log file.
`-log_name <name>`	Use alternate log file name.
`-boot`	Include the partition boot track in the comparison.
`-h`	Print a summary of command options and exit.

4.6.3.2 PARTCMP error messages

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./partcmp DI-01 lab joe /dev/hda B4 /dev/hdb AA -comment "Compare
partition: B4 & AA"
```

- **-log_name option requires a logfile name**

This option must be followed by the name of a log file to use instead of the default log file.

- **could not access [src/dst] drive %s status code %d**

The specified disk drive could not be accessed. Most likely cause is the drive number does not refer to an attached drive or the indicated drive may not be properly installed.

- **[src/dst] read error 0x%02X on track starting at lba %llu**

The disk may have bad sectors. There is likely a hardware problem that must be resolved.

- **-select requires two parameters: src partition index\n and dst partition index**

This option must be followed by two integer values.

- **could not find [src/dst] partition index %d**

The partition index specified does not exist on the selected disk. Try running without the **-select** option or use the PARTAB program to get the valid index values.

- **Source and destination drives must be different**

Source and destination partitions cannot be on the same drive.

4.7 Verify no changes to a disk: DISKHASH

DISKHASH is used to detect changes to a disk. By default, DISKHASH calls the SHA-1 computing program **sha1sum**. An alternate hash program such as **md5sum** may be specified with the **–hash** option. DISKHASH logs the hash value and the number of sectors hashed. DISKHASH should be run once as soon as a source disk has been set up and *before* any tests are run. Then DISKHASH is run *after* each test case is run. Comparing the hash values from the *before log* with the *after log* determines if executing

a test case has changed the source disk. If the hash values agree then the source disk has not changed.

4.7.1 DISKHASH command line

`diskhash.csh case host user drive label [-opts]`

Table 10 DISKHASH Command Line Parameters

Parameter	Description
`Case`	A test case identifier
`Host`	The name of the computer running the test
`User`	The initials of the person running the test
`Drive`	The drive's device name. Example: /dev/hda
`Label`	The name of the disk
`-comment "…"`	A comment for the test log file. If this option is not used, the program prompts for a comment to be entered.
`-before`	Name the log file hashblog.txt (Used if DISKHASH is run before a test case is executed).
`-after`	Name the log file hashalog.txt (Used if DISKHASH is run after a test case is executed).
`-hash <prog_name>`	Use <prog_name> to compute the hash. Default is sha1sum.
`-new_log`	Start a new log file rather than append to an existing log file.
`-log_name <name>`	Use alternate log file name.
`-help`	Print a summary of command options and exit

4.7.2 DISKHASH log file

The name of the log file is determined by the setting of a command line option. The log file name is either hashblog.txt if -**before** option is selected, hashalog.txt if -**after** option is selected, or <name> if the **-log_name** option is selected.

4.7.3 DISKHASH error messages

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./diskhash.csh DI-01 lab joe /dev/hda B4 -comment "Compute hash of B4"
-before
```

- **must select –before, -after, or –log_name <name>**

The command line must include one of these options.

- **-log_name option requires a log filename**

This option must be followed by the desired name of the log file.

- **Invalid parameter %s**

The specified parameter is invalid.

- **At least one required parameter is missing**

There were too few parameters on the command line.

4.8 Verify no changes to a block of sectors: SECHASH

SECHASH is used to detect changes to a contiguous range of disk sectors. By default, SECHASH calls the SHA-1 computing program **sha1sum**. An alternate hash program such as **md5sum** may be specified with the **–hash** option. SECHASH logs the hash value and the number of sectors hashed. SECHASH should be run twice: once as soon as the area of the disk to be monitored has been set up and *before* any tests are run; then a second time *after* the test case is run. Comparing the hash values from the *before log* with the *after log* determines if executing a test case has changed the area of the disk being monitored. If the hash values agree then no change has occurred.

The usual use of SECHASH is to determine if executing an imaging tool changes the excess destination sectors of a partition copy operation.

4.8.1 SECHASH command line

```
sechash.csh case host user drive label [-opts]
```

Table 11 SECHASH Command Line Parameters

Parameter	Description
`case`	A test case identifier
`host`	The name of the computer running the test
`User`	The initials of the person running the test
`drive`	The drive's device name. Example: /dev/hda
`Label`	The name of the disk.
`-comment "…"`	A comment for the test log file. If this option is not used, the program prompts for a comment to be entered.
`-before`	Name the log file hashbsec.txt (Used if SECHASH is run before a test case is executed.)
`-after`	Name the log file hashasec.txt (Used if SECHASH is run after a test case is executed.)
`-first n`	Start hashing with the sector *n*. If this option is not specified, hashing starts at the first sector of the drive.
`-last n`	Stop hashing with the sector *n*. If this option is not specified, hashing stops with the last sector of the drive.
`-hash <prog_name>`	Use <prog_name> to compute the hash. Default is sha1sum.
`-new_log`	Start a new log file rather than append to an existing log file.
`-log_name <name>`	Use alternate log file name.
`-help`	Print a summary of command options and exit

4.8.2 SECHASH log file

The name of the log file is determined by the setting of a command line option. The log file name is either hashbsec.txt if **-before** option is selected or hashasec.txt if **-after** option is selected.

4.8.3 SECHASH error messages

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./sechash.csh DI-01 lab joe /dev/hda B4 -comment "Compute hash of B4"
-before
```

- **Must select –before, -after, or –log_name**

The command line must include one of these options.

- **[-first|-last] option requires a sector number**

These options must be followed by an integer. This message indicates that nothing followed the option.

- **-log_name option requires a log filename**

This option must be followed by the desired name of the log file.

- **At least one required parameter is missing**

There were too few parameters on the command line.

- **Last sector (<sector LBA>) is before first sector (<sector LBA>)**

The first sector needs to be less than or equal to the last sector specified.

4.9 Log a test case: LOGCASE

Log administrative information about a test case run in the file **case.txt**.

4.9.1 LOGCASE command line

```
logcase test-case host operator src dst media
```

Table 12 LOGCASE Command Line Parameters

Parameter	Description
`test-case`	A test case identifier.
`host`	The name of the computer running the test.
`operator`	The operator setting up the disk.
`src`	The source disk for the test.
`dst`	The destination disk for the test.
`media`	The media disk for the test.

4.10 Log Disk Setup: LOGSETUP

Log administrative information about the setup of a disk for use as a source disk for a group of test cases in the file **setup.txt**.

4.10.1 LOGSETUP command line

```
logsetup disk host operator OS
```

Table 13 LOGSETUP Command Line Parameters

Parameter	Description
`disk`	The disk to be setup.
`host`	The name of the computer running the test.
`operator`	The operator setting up the disk.
`OS`	The Operating System loaded to the disk.

4.11 Print a partition table: PARTAB

The PARTAB program logs the partition table of the specified disk. The name of the log file is pt-XXX-log.txt, where XXX is replaced by the device name. This program can be used to determine partition indices for the PARTCMP program.

4.11.1 PARTAB command line

`partab test-case host operator drive [-opts]`

Table 14 PARTAB Command Line Parameters

Parameter	Description
`test-case`	A test case identifier.
`host`	The name of the computer running the test.
`operator`	The initials of the person running the test.
`drive`	The drive's device name. Example: /dev/hda
`label`	The external drive label. The first two characters of the drive label are hex (0-9, A-F) digits.
`-comment "…"`	A comment for the test log file. If this option is not used, the program prompts for a log floppy to be inserted. If this option is used, then a log floppy is assumed to be already loaded.
`-all`	Print empty table entries and extended partition entries.
`-new_log`	Start a new log file rather than append to an existing log file.
`-log_name <name>`	Use alternate log file name.
`-h`	Print a summary of command options and exit.

4.11.2 PARTAB error messages

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./partab DI-01 lab joe /dev/hda -comment "Partition table for disk
labeled B4"
```

- **-log_name option requires a logfile name**

This option must be followed by the name of a log file to use instead of the default log file.

- **could not access drive %s status code %d**

The specified disk drive could not be accessed. Most likely cause is the drive number does not refer to an attached drive or the indicated drive may not be properly installed.

- **No partition table signature**

The signature byte after the partition table did not have the correct value (0xAA55).

- **Error reading partition table, code %d**

There was a disk read error while accessing the partition table. This is usually caused by a corrupt partition table entry that points beyond the end of the disk.

4.12 Examine or Change a Disk: DISKCHG

DISKCHG is intended as a tool to aid testing the other support programs. DISKCHG has the following functions:

- Dump part of a sector in hexadecimal
- Change a single byte of a sector
- Zero the bytes of a sector
- Fill a sector in DISKWIPE style

4.12.1 DISKCHG command line

`diskchg test-case host operator drive [-opts]`

Table 15 DISKCHG Command Line Parameters

Parameter	Description
`test-case`	A test case identifier.
`host`	The name of the computer running the test.
`operator`	The initials of the person running the test.
`drive`	The drive's device name. Example: /dev/hda
`label`	The external drive label. The first two characters of the drive label are hex (0-9, A-F) digits.
`-exam`	Prompt the user for sectors to examine.
`-read addr offset length`	Print <length> bytes starting at <offset> from sector at <addr>.
`-write addr offset new_value`	Replace byte at <offset> in sector at <addr> with <new_value> (in hex).
`-fill addr fill_addr heads new_value`	Fill sector at <addr> in DISKWIPE style for address <fill_addr> using a disk geometry of <heads> heads with fill byte of <new_value> (in hex) if <heads> is zero, then number of heads on disk is used.
`-zero addr`	Set all bytes of sector at <addr> to zero. <addr> can be specified as either an LBA address (an integer) or as a cylinder/head/sector (three slash separated integers) value.
`-comment`	A comment for the test log file. If this option is not used, the program prompts for a log floppy to be inserted. If this option is used, then a log floppy is assumed to be already loaded.

-new_log	Start a new log file rather than append to an existing log file.
-log_name <name>	Use alternate log file name.
-h	Print a summary of command options and exit.

4.13 Compare two disk sectors: SECCMP

The SECCMP program is used to compare two sectors. SECCMP is not a required part of any test. It is used to investigate unexpected test results.

4.13.1 SECCMP command line

seccmp case host operator src-drv src-fill dst-drv dst-fill [-opts]

Table 16 SECCMP Command Line Parameters

Parameter	Description
case	A test case identifier.
host	The name of the computer running the test.
operator	The initials of the person running the test.
src-drv	The drive's device name. Example: /dev/hda
src-fill	A two digit hex value, used to fill each sector by **diskwipe**.
dst-drv	The drive's device name. Example: /dev/hdb
dst-fill	A two digit hex value, used to fill each sector by **diskwipe**.
-comment "…"	A comment for the test log file. If this option is not used, the program prompts for a log floppy to be inserted. If this option is used, then a log floppy is assumed to be already loaded.
-sector src dst	The LBA addresses of two sectors to compare. If this option is omitted, the program asks for the addresses.
-new_log	Start a new log file rather than append to an existing log file.
-log_name <name>	Use alternate log file name.
-h	Print a summary of command options and exit.

4.13.2 SECCMP error messages

- **-comment option requires a comment**

This option must be followed by the text of the comment on the command line. If the comment includes any space characters then the entire comment must begin and end with a double quote character (**"**). For example,

```
./seccmp DI-01 lab joe /dev/hda B4 /dev/hdb AE -comment "Compare sector on B4 to AE"
```

- **-log_name option requires a logfile name**

This option must be followed by the name of a log file to use instead of the default log file.

- **[src/dst] drive %s is not valid**

The drive number is not valid. The full path of the drive's device must be specified.

- **[src/dst] read error 0x%02X at lba %llu**

The disk may have bad sectors. There is likely a hardware problem that must be resolved.

- **-sector option requires src and dst sector LBA addresses**

This option must be followed by two integer values indicating the absolute LBA address of the source and destination sectors.

- **Enter src (base offset) dst (base offset):**

If the **–sector** option is not specified, **seccmp** prompts the user for souce and destination sectors relative to a partition offset. If sectors values are absolute (no partition offset), enter 0 for base offset.

- **Source and destination drives must be different**

Source and destination partitions cannot be on the same drive.

5 Example Test Case

This section illustrates using the software on an example test case. This example demonstrates how to use FS-TST to test a disk imaging tool. This example is **not** a tutorial on investigative procedure. The activities of *testing* a disk imaging tool have subtle differences from *proper usage* in an investigation of a disk imaging tool.

In this example, the tool must demonstrate compliance with each of the four test assertions listed under *Test items* in Table 17. The first assertion requires the tool to create a copy of a source disk on another disk of the same size. The copy, however, is not done directly. First, an image file is created and the destination is created from the image file. An *image file* is a file that contains all the information required to recreate an exact duplicate of the original. In the simplest case, an image file is a bit for bit copy of the original. However, some imaging tools include additional information in an image file to allow for verification of file integrity or even error correction. A separate disk drive, called the media disk, is used to contain the image file. The second assertion requires the tool to give notice of any I/O errors during the duplication process. The third assertion specifies treatment of the destination if an I/O error has occurred. The last assertion is critical for maintaining the integrity of any evidence on the source disk.

The test case is executed in three phases: setup disks, execute tool, and measure results. The setup phase creates the source, destination and media disks. After the source disk is created, a SHA-1 is computed for later reference. The execute phase executes the tool. The measure results phase compares the source to the destination and computes a SHA-1 of the source disk. The SHA-1 is compared to the reference hash to verify that the source has not been changed.

Table 17 Example Test Case Specification

Test case ID	EX-01
Test items	1. If a duplicate destination disk is created from an image file of a source disk with the same geometry, then the disks must compare equal.

	2. If the tool is able to create a destination from an image file that contains read errors, the destination sectors corresponding to the unreadable data must be treated as fill sectors (the tool may allow a specified action or may fill the sectors with zeroes). 3. The contents of the source disk must not be changed by the tool.
Hardware Environment	
PCs	One PC named Lab.
Disks	Three disks: source disk labeled AA, destination disk labeled BB and media disk labeled CC.
Partitions	One Fat 32 partition.
Output Specifications (expected results)	1. The source and destination disks compare equal except for a block of sectors around the bad sector. 2. The tool under test logs an I/O error message. 3. A SHA-1 hash of the source disk, computed before running the tool under test, is the same as a SHA-1 hash computed after running the tool under test.

5.1 Pre-test preparations

These programs are intended to support testing of a disk imaging tool. For this example, suppose that we are testing an imaginary imaging tool called NSIT (No Such Imaging Tool). There is not such a tool but it serves to illustrate using the support tools. We need the following:

1. A test platform (pc to run the test) with three disk drives. In our test scenario the PC is a forensic Lab machine and the disk on /dev/hda has been seized from a suspect. We want to make an image file on /dev/hdc and make a duplicate on disk /dev/hdb of the original disk that is installed as /dev/hda. The name of the PC is lab.forensics-lab.agency.gov (lab for short).
2. Two of the disk drives (/dev/hda and /dev/hdb) have the same geometry.
3. A *forensic Linux boot disk*. i.e., the boot floppy that an investigator would use.
4. The support software and NSIT loaded to a CD, called the *tool CD*, which is mounted on /mnt/cdrom on the PC when booted from the Linux boot disk.
5. A floppy disk for keeping log files. This is referred to as the *log floppy*.

5.2 Setup

The test case setup steps from Figure 1 General Structure of a Test Case are repeated in Table 18 along with the support software (or other) program required for the step. Note that for each execution of the **diskwipe** program a different command line option is indicated because of the different type of disk initialized in each step.

Table 18 Commands Used During Test Setup

Step from Figure 1	Program to run
1. Record details of source disk setup.	`logsetup`

2. Initialize the source disk to a known value.	`diskwipe … -src`
3. Hash the source disk and save the hash value.	`sha1sum /dev/hda > srcbhash.txt`
4. Record details of test case setup.	`logcase`
5. Initialize a destination disk.	`diskwipe … -dst`
6. If the test requires a partition on the destination, create and format a partition on the destination disk.	`mkfs or similar tool`
7. If the test uses an image file, partition and format a media disk.	`diskwipe … -media`

During the setup phase the tester acts in multiple roles. Three disks have to be setup: a source, a destination and a media disk to contain the image file. When we are setting up the source disk we are acting in the role of the suspect. When setting up the destination and media disk we are in the role of the forensic investigator. The first part of the setup uses the **diskwipe** program to write a known pattern unique to each disk. This is not what the owner of the seized disk or a forensic investigator would do but it is important for testing the imaging tool to be able to determine easily the origin of any data on the destination disk. We will wipe the source disk with 0xAA, the destination disk with 0xBB and the media disk with 0xCC. After the source is setup the **sha1sum** program computes a SHA-1 hash that can be used later to verify that the source disk has not been changed (compute the hash again later and if it is different then the source has been changed). Table 19 presents the commands issued in the setup phase of the test case. The column labeled **Step** indicates the step number from the general test case description presented in Figure 1. The **Command** column shows the actual command that would be typed. A description of what the command is accomplishing is in the **Comments** column. Steps 1-3 set up a source disk. Step 1 records that Joe, the operator, had disk labeled AA installed on device /dev/hda. In step 2, disk AA is erased by writing a unique pattern to each sector of the drive. After **diskwipe** is run, a FAT32 partition is created and Windows NT is loaded to the disk (the actions to do this are not shown in the table). In step 3, a SHA-1 is computed for the disk. The SHA-1 hash can be used to verify that disk AA has not been changed by some later action, e.g. by running an imaging tool.

Table 19 Example Test Case Setup

Setup Actions		
Step	**Command**	**Comments**
1	`./logsetup /dev/hda:aa lab joe “windows nt”`	Record information about a source disk setup. The test operator, Joe, is setting up disk AA as the source disk (with Windows NT) for the test. The disk labeled AA is installed on /dev/hda. Boot from Linux boot floppy, insert tool CD, replace floppy with log floppy, run

Setup Actions		
Step	**Command**	**Comments**
		logsetup to record source disk setup details.
2	`./diskwipe EX-01 lab joe /dev/hda -src`	Write a unique pattern to each sector of the source disk. Run **diskwipe.** After running **diskwipe**, create a FAT32 partition and load Windows NT to the FAT32 partition (only the **diskwipe** command is shown).
3	`sha1sum /dev/hda > srcbhash.txt`	Compute a reference SHA-1 hash of the source disk. Boot from Linux boot floppy, insert tool CD, replace floppy with log floppy, and run **sha1sum.** The **srcbhash.txt** file contains the reference hash for the source drive.
4	`./logcase EX-01 lab joe aa bb cc`	Record information about the test case. The destination disk is BB and disk CC is used to contain an image.
5	`./diskwipe EX-01 lab joe /dev/hdb bb -dst`	Write a unique pattern to each sector of the destination disk. This step erases the destination disk by writing a pattern to the destination that is different from the one written to the source disk.
6	`No partition required on the destination.`	No partition is required since the test case calls for an entire disk copy.
7	`./diskwipe EX-01 lab joe /dev/hdc cc -media`	Write a unique pattern to the media disk. After wiping the media disk, a partition must be created to hold the image (not shown).

5.3 Execute

The test case execute steps from Figure 1 General Structure of a Test Case are repeated in Table 20 along with the support software (or other) program required for the step.

Table 20 Commands Used During Tool Execution

Step from Figure 1	Program to run
8. If the test requires an image file, use the tool to create an image file of the source on the media disk.	the tool being tested
9. If the test requires a corrupted image file, corrupt the image file.	`corrupt`
10. Use the disk imaging tool to create the destination disk either from a copy of the source disk or by restoring an image file of the source to the destination.	the tool being tested

During the execute phase we run the tool (NSIT).

Table 21 Example Test Case Execution

Execute Actions		
Step	**Command**	**Comments**
11	`/mnt/cdrom/nsit -image case-17.img`	Run the imaging tool to create an image that is stored on the media disk as case-17.img
12	`Nothing required here.`	
13	`/mnt/cdrom/nsit -restore case-17.img`	Boot from Linux boot floppy, insert tool CD, replace floppy with log floppy, and run the disk imaging tool to restore a copy of the source (from the image file) to the destination.

5.4 Measure

The test case measurement steps from Figure 1 General Structure of a Test Case are repeated in Table 22 along with the support software (or other) program required for the step.

Table 22 Commands Used to Measure Test Results

Step from Figure 1	Program to run
11. Compare the source to the destination.	One of the three compare programs: disks: `diskcmp` partitions: `partcmp` aligned disk: `adjcmp`
12. Compute a hash of the source disk, compare with the saved hash value.	`sha1sum … > srcahash.txt`

The measure actions are where we determine if the disk imaging tool yields the expected results.

Table 23 Example Test Case Measurement

Execute Actions		
Step	**Command**	**Comments**
12	`./diskcmp EX-01 lab joe /dev/hda aa /dev/hdb bb`	Compare the source to the destination. The log file, cmplog.txt, should indicate that most sectors match except for a small number of sectors including the simulated bad sector. In the ideal result, the selected sector is the only sector that does not match.
13	`sha1sum /dev/hda > srcahash.txt`	Verify that the source disk has not changed. Boot from Linux boot floppy, insert tool CD, replace floppy with log floppy, and run sha1sum. The hash in the two hash log files should match.