NIST Special Database 10

Supplemental Fingerprint Card Data
for NIST Special Database 9

C.I.Watson

National Institute of Standards and Technology
Advanced Systems Division
Image Recognition Group
June 29, 1993

1.0 INTRODUCTION

This document describes the NIST fingerprint database, *NIST Special Database 10*. The database provides a large sample of patterns for transitional fingerprint classes and classes with a low natural frequency of occurrence in *NIST Special Database 9*. The 552 fingerprint cards in *NIST Special Database 10* are non-mated cards archived on a set of three CD-ROM's with the first CD-ROM containing 2160 fingerprint images and the last two CD-ROMs containing 1680 fingerprints each. All fingerprints are stored in NIST's IHead raster data format and compressed using a non-standard implementation of the JPEG lossless [1] compression algorithm. The prints are 832 (w) X 768 (h) pixels (see Appendix A). Image data stored on the first CD-ROM requires approximately 690 megabytes of storage. The second and third CD-ROMs require approximately 590 megabytes of storage. The average compression ratio for all the images is 1.9 : 1.

The data was collected by selecting non-mated fingerprint cards, from the FBI's Technical Master File, which contained the most occurrences of the desired fingerprint patterns. Since the entire card was digitized, there is a mix of other classes within the specific class groupings. The data also includes a significant number of referenced fingerprints. Appendix C shows the exact distribution of the classes and referencing for each major class group that was collected.

The specific classes being collected were Tented Arch, Arch, Low Ridge Count Loops, Central Pocket Whorls, Double Loop Whorls, Plain Whorls and Accidental Whorls. The fingerprints are classified using the National Crime Information Center (NCIC) classes assigned by the FBI [2]. All classes and references are stored in the NIST IHead **id** field of each file.

2.0 NON-STANDARD IMPLEMENTATION OF JPEG LOSSLESS COMPRESSION

The compression used was developed from techniques outlined in the WG10 "JPEG" (draft) standard [1] for 8-bit gray scale images with modifications to the compressed data format. This is the same code used in *NIST Special Database 4 and 9*. The NIST IHead format already contained most of the information needed in the decompression algorithm, so the JPEG compressed data format was modified to contain only the information needed when reconstructing the Huffman code tables and identifying the type of predictor used in the coding process. Codes used to compress and decompress the images are still developed per the draft standard, but only applied to 8-bit gray scale images.

The standard uses a differential coding scheme and allows for seven possible ways of predicting a pixel value. Tests showed that predictor number 4 provided the best compression on the fingerprint images; therefore, this predictor was used to compress all of the images.

3.0 DATABASE REFLECTANCE CALIBRATION

The reflectance values for the fingerprint database, *NIST Special Database 10,* were calibrated using a reflection step table [3]. A plot of the reflectance values obtained using this step table is shown in Appendix B. Also shown on the plot and below is an equation used to predict the reflectance of a given datapoint. The plot in Appendix B shows that this predicted reflectance closely follows the actual reflectance obtained using the reflection step table.

predicted % reflectance = -5.1 + (.36 * grayscale pixel value)

4.0 FINGERPRINT FILE FORMAT [4][5]

Image file formats and effective data compression and decompression are critical to the usefulness of image archives. Each fingerprint was digitized in 8-bit gray scale form at 19.6850 pixels/mm (500 pixels/inch), 2-dimensionally compressed using a modified JPEG lossless algorithm, and temporarily archived onto computer magnetic mass storage. Once all prints were digitized, the images were mastered and replicated onto ISO-9660 formatted CD-ROM discs for permanent archiving and distribution.

After digitization, certain attributes of an image are required to correctly interpret the 1-dimensional pixel data as a 2-dimensional image. Examples of such attributes are the pixel width and pixel height of the image. These attributes can be stored in a machine readable header prefixed to the raster bit stream. A program which manipulates the raster data of an image is able to first read the header and determine the proper interpretation of the data which follow it.

Numerous image formats exist, but most image formats are proprietary. Some are widely supported on small personal computers and others on larger workstations. A header format named IHead has been developed for use as a general purpose image interchange format. The IHead header is an open image format which can be universally implemented across heterogeneous computer architectures and environments. Both documentation and source code for the IHead format are publicly available and included with this database. IHead has been designed with an extensive set of attributes in order to adequately represent both binary and gray level images, to represent images captured from different scanners and cameras, and to satisfy the image requirements of diversified applications including, but not limited to, image archival/retrieval, character recognition, and fingerprint classification. Figure 1 illustrates the IHead format.

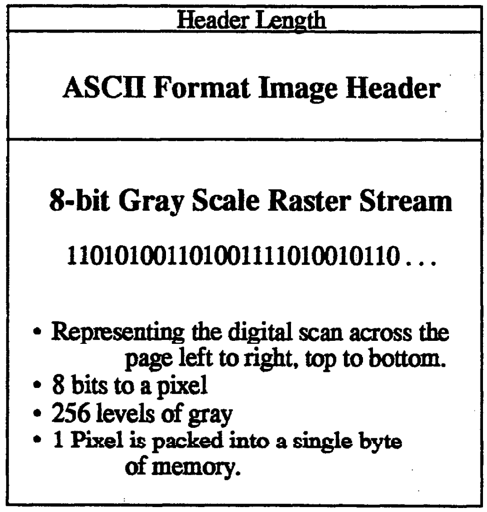


Figure 1: An illustration of the IHead raster file format.

Since the header is represented by the ASCII character set, IHead has been successfully ported and tested on several systems including UNIX workstations and servers, DOS personal computers, and VMS mainframes. All attribute fields in the IHead structure are of fixed length with all multiple character fields null-terminated, allowing the fields to be loaded into main memory in two distinct ways. The IHead attribute fields can be parsed as individual characters and null-terminated strings, an input/output format common in the 'C' programming language, or the header can be read into main memory using record-oriented input/output. A fixed-length field containing the size in bytes of the header is prefixed to the front of an IHead image file as shown in Figure 1.

```
/************************************************************
      File Name: IHead.h
      Package:  NIST Internal Image Header
      Author:   Michael D. Garris
      Date:   2/08/90
************************************************************/
/* Defines used by the ihead structure */
#define IHDR_SIZE         288     /* len of hdr record (always even bytes) */
#define SHORT_CHARS       8       /* # of ASCII chars to represent a short */
#define BUFSIZE           80      /* default buffer size */
#define DATELEN           26      /* character length of data string */

typedef struct ihead{
  char id[BUFSIZE];                  /* identification/comment field */
  char created[DATELEN];             /* date created */
  char width[SHORT_CHARS];           /* pixel width of image */
  char height[SHORT_CHARS];          /* pixel height of image */
  char depth[SHORT_CHARS];           /* bits per pixel */
  char density[SHORT_CHARS];         /* pixels per inch */
  char compress[SHORT_CHARS];        /* compression code */
  char complen[SHORT_CHARS];         /* compressed data length */
  char align[SHORT_CHARS];           /* scanline multiple: 8|16|32 */
  char unitsize[SHORT_CHARS];        /* bit size of image memory units */
  char sigbit;                       /* 0->sigbit first | 1->sigbit last */
  char byte_order;                   /* 0->highlow | 1->lowhigh*/
  char pix_offset[SHORT_CHARS];      /* pixel column offset */
  char whitepix[SHORT_CHARS];        /* intensity of white pixel */
  char issigned;                     /* 0->unsigned data | 1->signed data */
  char rm_cm;                        /* 0->row maj | 1->column maj */
  char tb_bt;                        /* 0->top2bottom | 1->bottom2top */
  char lr_rl;                        /* 0->left2right | 1->right2left */
  char parent[BUFSIZE];              /* parent image file */
  char par_x[SHORT_CHARS];           /* from x pixel in parent */
  char par_y[SHORT_CHARS];           /* from y pixel in parent */
}IHEAD;
```

Figure 2: The IHead 'C' programming language structure definition.

The IHead structure definition written in the 'C' programming language is listed in Figure 2. Figure 3 lists the header values from an IHead file corresponding to the structure members listed in Figure 2. This header information belongs to the database file **aa000001.pct** (see Figure A.1 in Appendix A). Referencing the structure members listed in Figure 2, the first attribute field of IHead is the identification field, **id**. This field uniquely identifies the image file, typically by a file name. The identification field in this example not only contains the image's file name, but also the sex of the individual, if the image was scanned from an inked or live scan printed image, and the NCIC classification of the fingerprint, with any references to another classification (see Figure 8 and Section 5.4 for an example of class referencing). This convention enables an image recognition system's hypothesized classification to be automatically scored against the actual classification.

IMAGE FILE HEADER

Identity	: aa000001.pct m 1 aa
Header Size	: 288 (bytes)
Date Created	: Tue Mar 23 03:41:03 1993
Width	: 832 (pixels)
Height	: 768 (pixels)
Bits per Pixel	: 8
Resolution	: 500 (ppi)
Compression	: 6 (code)
Compress Length	: 360295 (bytes)
Scan Alignment	: 8 (bits)
Image Data Unit	: 8 (bits)
Byte Order	: High-Low
MSBit	: First
Column Offset	: 0 (pixels)
White Pixel	: 255
Data Units	: Unsigned
Scan Order	: Row Major, Top to Bottom, Left to Right
Parent	: tape506.aa001.01 4096x1536
X Origin	: 0 (pixels)
Y Origin	: 0 (pixels)

Figure 3: The IHead values for the fingerprint data file **aa000001.pct**.

The attribute field, **created**, is the date on which NIST received the digitized image. The next three fields hold the image's pixel **width, height**, and **depth**. A binary image has a pixel depth of 1 whereas a gray scale image containing 256 possible shades of gray has a pixel depth of 8. The attribute field, **density**, contains the scan resolution of the image; in this case, 19.6850 pixels/mm (500 pixels/inch). The next two fields deal with compression.

In the IHead format, images may be compressed with virtually any algorithm. Whether the image is compressed or not, the IHead is always uncompressed. This enables header interpretation and manipulation without the overhead of decompression. The **compress** field is an integer flag which signifies which compression technique, if any, has been applied to the raster image data which follows the header. If the compression code is zero, then the image data is not compressed, and the data dimensions: width, height, and depth, are sufficient to load the image into main memory. However, if the compression code is nonzero, then the **complen** field must be used in addition to the image's pixel dimensions. For example, the images in this database have a compression code of 6 signifying that modified JPEG lossless compression has been applied to the image data prior to file creation. In order to load the compressed image data into main memory, the value in **complen** gives the size of the compressed block of image data.

Once the compressed image data has been loaded into memory, JPEG lossless decompression can be used to produce an image which has the pixel dimensions consistent with those stored in its header. Using JPEG lossless compression and this compression scheme on the images in this database, an average compression ratio of 1.9 to 1 was achieved.

The attribute field, **align**, stores the alignment boundary to which scan lines of pixels are padded. Pixel values of 8-bit gray scale images are stored 1 byte (or 8 bits) to a pixel, so the images will automatically align to an even byte boundary.

The next three attribute fields identify data interchanging issues among heterogeneous computer architectures and displays. The **unitsize** field specifies how many contiguous bits are bundled into a single unit by the digitizer. The **sigbit** field specifies the order in which bits of significance are stored within each unit; most significant bit first or least significant bit first. The last of these three fields is the **byte_order** field. If **unitsize** is a multiple of bytes, then this field specifies the order in which bytes occur within the unit. Given these three attributes, data incompatibilities across computer hardware and data format assumptions within application software can be identified and effectively dealt with.

The **pix_offset** attribute defines a pixel displacement from the left edge of the raster image data to where a particular image's significant image information begins. The **whitepix** attribute defines the value assigned to the color white. For example, the gray scale image described in Figure 3 is gray print on a white background and the value of the white pixel is 255. This field is particularly useful to image display routines. The **issigned** field is required to specify whether the units of an image are signed or unsigned. This attribute determines whether an image with a pixel depth of 8, should have pixel values interpreted in the range of -128 to +127, or 0 to 255. The orientation of the raster scan may also vary among different digitizers. The attribute field, **rm_cm**, specifies whether the digitizer captured the image in row-major order or column-major order. Whether the scan lines of an image were accumulated from top to bottom, or bottom to top, is specified by the field, **tb_bt**, and whether left to right, or right to left, is specified by the field, **rl_lr**.

The final attributes in IHead provide a single historical link from the current image to its **parent** image. The images used in this database were renamed from their original filenames, given by the FBI, and the 'link' to the original filename was stored in the **parent** field as well as the size of the ten print image (columns x rows) before the individual fingerprints were segmented. The FBI filename consists of three fields separated by periods. The first field contains a tape number (i.e. tape501, tape502, ..., tape523), indicating the FBI streamer tape the file was stored on. The second field contains 5 characters. The first two characters in the second field indicate the specific fingerprint class group being collected. The next three characters are the card sequence number for that class grouping. The last field indicates the finger number of the file (01-10).

The **par_x** and **par_y** fields contain the origin, upper left hand corner pixel coordinate, from where the extraction took place from the parent image. These fields provide a historical thread through successive generations of images and subimages. We believe that the IHead image format contains the minimal amount of ancillary information required to successfully manage binary and gray scale images.

5.0 DATABASE CONTENT AND ORGANIZATION

NIST Special Database 10 contains 5520 8-bit gray scale fingerprint images which are distributed on three ISO-9660 formatted CD-ROMs and compressed using a non-standard implementation of the JPEG lossless compression algorithm [1]. Included with the fingerprint data are software and documentation.

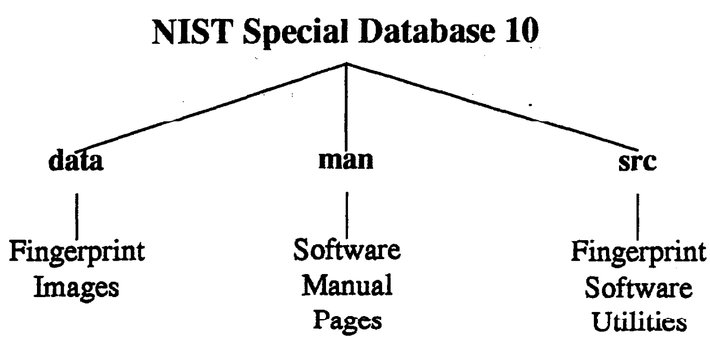


Figure 4: Top level directory tree for each CD-ROM in *NIST Special Data*base 10.

5.1 Database File Hierarchy

The top level of the file structure contains three directories **src**, **man**, and **data** (see Figure 4). The code needed to decompress and use the image data is contained in the **src** directory with man pages for the source code stored in the **man** directories. The **data** directory contains the fingerprint images stored in two levels of subdirectories for easier access and clarity (see Figure 5). The first level of subdirectories indicate the specific class collected within that group of data. The specific class groupings on each CD-ROM are: disc 1-> arches (**aa**), central pocket whorls (**cw**) and double loop whorls (**dw**), disc2 -> low ridge count loops (**sl**) and plain whorls (**pw**), and disc 3 -> tented arches (**tt**) and accidental whorls (**xw**). The next level has a subdirectory for each fingerprint card which contains the ten segmented fingerprint images for each card. The **aa**, **sl** and **tt** class groups have 120 cards each (card_001 - card_120)and the other class groups have 48 cards each.

Fingerprints are stored with filenames containing two letter, six digits and a ".pct" (picture) extension. The first two characters in the filename are the same as the specific class being collected (**aa**, **tt**, **sl**, **cw**, **dw**, **pw** or **xw**). The next six characters indicate the sequence number (000001 - 001200 for **aa**, **sl** and **tt** groups). The finger number is given by the last of the six digits, with zero representing digit ten on the fingerprint card (see Figure 6 for fingerprint card layout). Every ten prints in sequential order are a group of prints from the same card (i.e. digits 1-10).

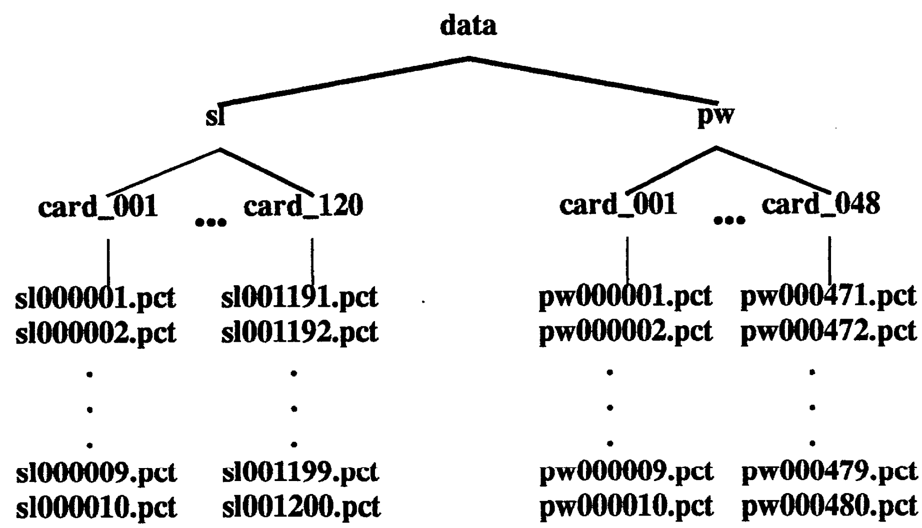


Figure 5: Arrangement of fingerprint files on second disc of *NIST Special Database 10*.

1. R. Thumb	2. R. Index	3. R. Middle	4. R. Ring	5. R. Little
6. L. Thumb	7. L. Index	8. L. Middle	9. L. Ring	10. L. Little

Figure 6: Layout of fingerprint card numbers.

5.2 Low Count Loops

The main low count loops collected were ulnar loops (see statistics in Appendix C). The user can create low count radial loop data by flipping the ulnar loop image about a vertical axis centered on the image. This allows for more samples of both low count loop classes

5.3 NCIC Classifications

The classes stored in the NIST Ihead **id** field are the NCIC classes [2], including any references, that were assigned by the FBI Identification Division Automated System (IDAS). A listing of the possible class codes is given below. Note that the classification **ac** (approximate class) means that the classification immediately after the **ac** is the best classification that can be assigned to the print given the information shown in the image.

Classification name (**class code**)

Arch (**aa**)
Tented Arch (**tt**)
Ulnar Loop Ridge Counts (**01 - 49**)
Radial Loop Ridge Counts (**51 - 99**)
Plain Whorl Inner Ridge Tracing (**pi**)
Plain Whorl Outer Ridge Tracing (**po**)
Plain Whorl Meeting Ridge Tracing (**pm**)
Central Pocket Whorl Inner Ridge Tracing (**ci**)
Central Pocket Whorl Outer Ridge Tracing (**co**)
Central Pocket Whorl Meeting Ridge Tracing (**cm**)
Double Loop Whorl Inner Ridge Tracing (**di**)
Double Loop Whorl Outer Ridge Tracing (**do**)
Double Loop Whorl Meeting Ridge Tracing (**dm**)
Accidental Whorl Inner Ridge Tracing (**xi**)
Accidental Whorl Outer Ridge Tracing (**xo**)
Accidental Whorl Meeting Ridge Tracing (**xm**)
Approximate Class (**ac**) followed by a valid class
Amputation or Missing (**xx**)
Scar or Mutilation (**sr**)

Figure 7: Classification codes for *NIST Special Database* 10.

5.4 Class Referencing

The referencing of a fingerprint is caused by a variety of ambiguities such as a scar occurring in the fingerprint, the quality of the print rolling, or the print having ridge structures characteristic of two different classes. The referenced prints could easily cause a wrong classification when used in testing an automated classification system but could provide a challenge in the later stages of development.The **id** field of the prints in *NIST Special Database 10* contain the primary fingerprint class followed by any references. An example IHead header is shown in Figure 8 (**aa000042.pct**) and the corresponding print is shown in Figure A.2. The fingerprint is classified as an **aa** and referenced to a **tt**.

IMAGE FILE HEADER

~~~~~~~~~~~~~~~~~~~~

Identity	: aa000042.pct m i aa/tt
Header Size	: 288 (bytes)
Date Created	: Tue Mar 23 04:04:48 1993
Width	: 832 (pixels)
Height	: 768 (pixels)
Bits per Pixel	: 8
Resolution	: 500 (ppi)
Compression	: 6 (code)
Compress Length	: 296094 (bytes)
Scan Alignment	: 8 (bits)
Image Data Unit	: 8 (bits)
Byte Order	: High-Low
MSBit	: First
Column Offset	: 0 (pixels)
White Pixel	: 255
Data Units	: Unsigned
Scan Order	: Row Major, Top to Bottom, Left to Right
Parent	: tape506.aa005.02 4096x1536
X Origin	: 0 (pixels)
Y Origin	: 0 (pixels)

Figure 8: The IHead values for the fingerprint data file **aa000042.pct.**

5.5 Segmenting the Fingerprint Images

The individual fingerprint images were obtained by scanning all ten prints on a card into one large image (4096 X 1536 pixels) and then segmenting each individual image from that larger image. The individual images were segmented at the same exact points on all the larger (full card) images. The image size of 832 X 768 pixels was selected to allow the user the capability of reconstructing the fingerprint card image and resegmenting the individual fingerprint images if desired. The images overlap by 32 pixels with horizontally adjacent images and by 18 pixels with vertically adjacent images which must be accounted for when reconstructing the fingerprint card image.

5.6 Inked and Live Scan Printed

The fingerprints were scanned from two types of prints. The first type were fingerprint patterns created by rolling the individuals ink covered finger on the fingerprint card (denoted by an **i** in the **id** field). The second type were patterns taken with a live scanning device and then printed onto a fingerprint card (denoted by an **l** in the **id** field). This is important in that the quality of the image is affected by the resolution of the printer used to print the live scanned image onto the fingerprint card.
~~~~~~~~~~~~~~~~~~~~

6.0 SOFTWARE FOR ACCESSING DATABASE

Included with the fingerprint images are documentation and software written in the 'C' programming language. The software was developed on a SUN sparc station and has only been tested on that platform. Four programs are included in the **src** directory: **dumpihdr**, **ihdr2sun**, **sunalign**, and **dcplljpg**. These routines are provided as an example to software developers of how IHead images can be manipulated and used. Descriptions of these programs and their subroutines are given below as well as in the included man pages located in the **man** directory. Copies of the manual pages are also included in Appendix D.

6.1 Compilation

The CD-ROMs used for *NIST Special Database 10* are read only storage medium. The files in the src directory must be copied to a read-writable partition prior to compiling. After copying these files, executable binaries can be produced by invoking the UNIX utility **make** to execute the included makefile. An example of this command follows.

make -f **makefile.mak**

6.2 Dumpihdr <Ihead file>

Dumpihdr is a program which reads an image's IHead data from the given file and formats the header data into a report which is printed to standard output. The report shown in Figure 3 was generated using this utility. The main routine for **dumpihdr** is found in the file **dumpihdr.c** and calls the external function **readihdr()**.

Readihdr() is a function responsible for loading an image's IHead data from a file into main memory. This routine allocates, reads, and returns the header information from an open image file in an initialized IHead structure. This function is found in the file **ihead.c**. The IHead structure definition is listed in Figure 2 and is found in the file **ihead.h**

6.3 Ihdr2sun <Ihead file>

Ihdr2sun converts an image from NIST IHead format to Sun rasterfile format. **Ihdr2sun** loads an IHead formatted image from a file into main memory and writes the raster data to a new file appending the data to a Sun rasterfile header. The main routine for this program is found in the file **ihdr2sun.c** and calls the external function **ReadIheadRaster()** which is found in the file **rasterio.c.**

ReadIheadRaster() is the procedure responsible for loading an IHead image from a file into main memory. This routine reads the image's header data returning an initialized IHead structure by calling **readihdr()**. In addition, the image's raster data is returned to the caller uncompressed. The images in this database have been 2-dimensionally compressed using a modified JPEG lossless compression algorithm, therefore **ReadIheadRaster()** invokes the external procedure **jpglldcp()** which is responsible for decompressing the raster data. Upon completion, **ReadIheadRaster()**

returns an initialized IHead structure, the uncompressed raster data, the image's width and height in pixels, and pixel depth.

Jpglldcp() accepts image raster data compressed using the modified JPEG lossless compression algorithm and returns the uncompressed image raster data. **Jpglldcp()** was developed using techniques described in the WG10 "JPEG" (draft) standard [1] and adapted for use with this database. Source code for the algorithm is found in **jpglldcp.c**.

6.4 Dcplljpg <lossless JPEG compressed file>

Dcplljpg is a program which decompresses a fingerprint image file (approximately 10 seconds per image, for images from this database, on a scientific workstation) that was compressed using the modified JPEG compression routine. The routine accepts a compressed image in NIST IHead format and writes the uncompressed image to the same filename using the NIST IHead format. The main routine is found in **dcplljpg.c** and calls the external functions **ReadIheadRaster()** (see section 6.3 for ReadIheadRaster description) and **writeihdrfile()**.

Writeihdrfile() is a routine that writes an IHead image into a file. This routine opens the passed filename and writes the given IHead structure and corresponding data to the file. **Writeihdrfile()** is found in the src file **rasterio.c**.

References

[1] WG10 "JPEG", committee draft ISO/IEC CD 10198-1, "Digital Compression and Coding of Continuous-Tone Still Images," March 3, 1991.

[2] *The Science of Fingerprints.* U.S. Department of Justice, Washington, D.C., 1984.

[3] National Bureau of Standards, "Standard Reference Materials," Reflection Step Table 2601.

[4] M.D. Garris, "Design and Collection of a Handwriting Sample Image Database," Social Science Computing Journal, Vol. 10: 196-214, 1992.

[5] C.I. Watson and C.L. Wilson, "NIST Special Database 4, Fingerprint Database," National Institute of Standards and Technology, March 15, 1992.

Appendix A: Database Fingerprint Image Samples

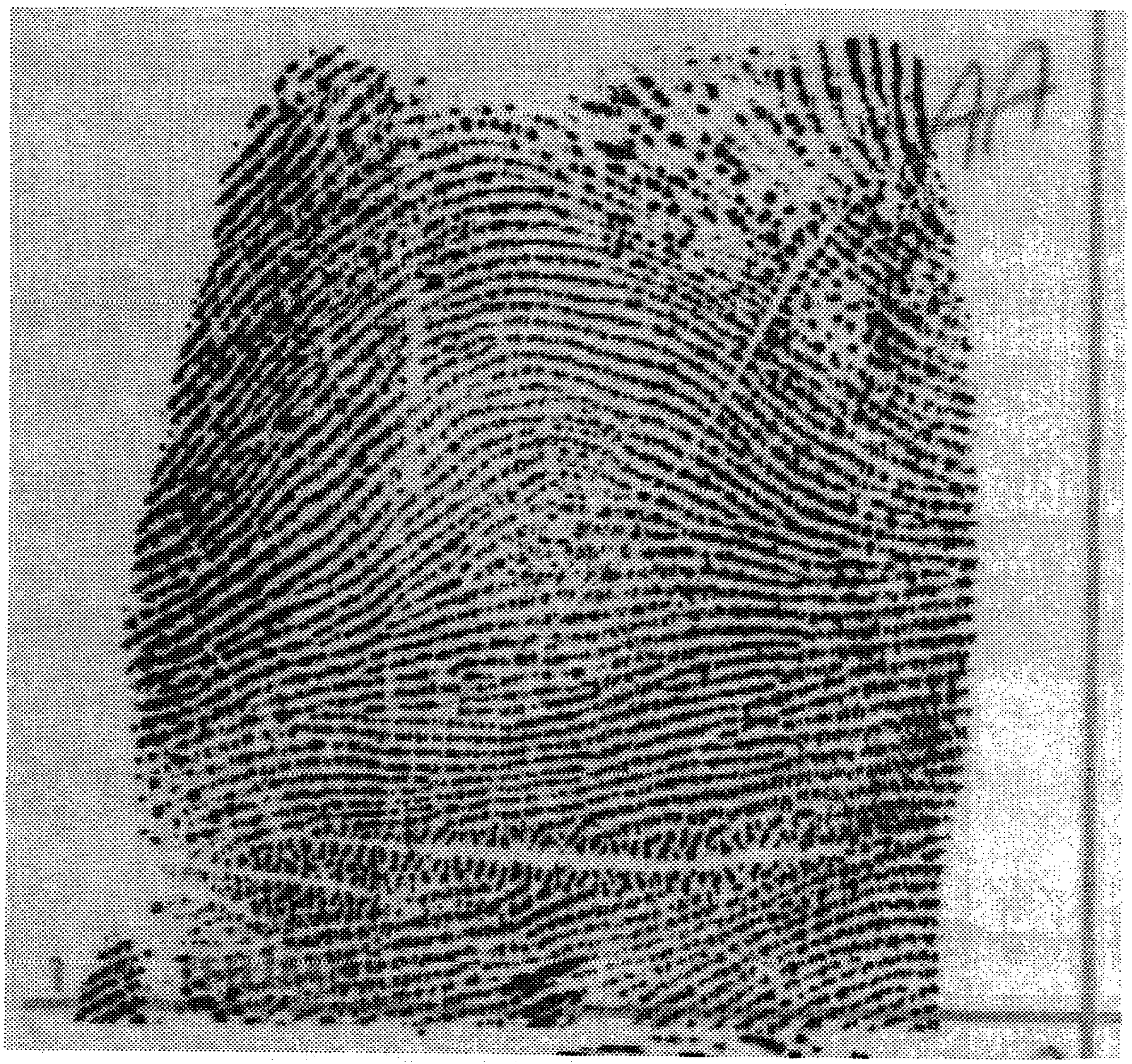
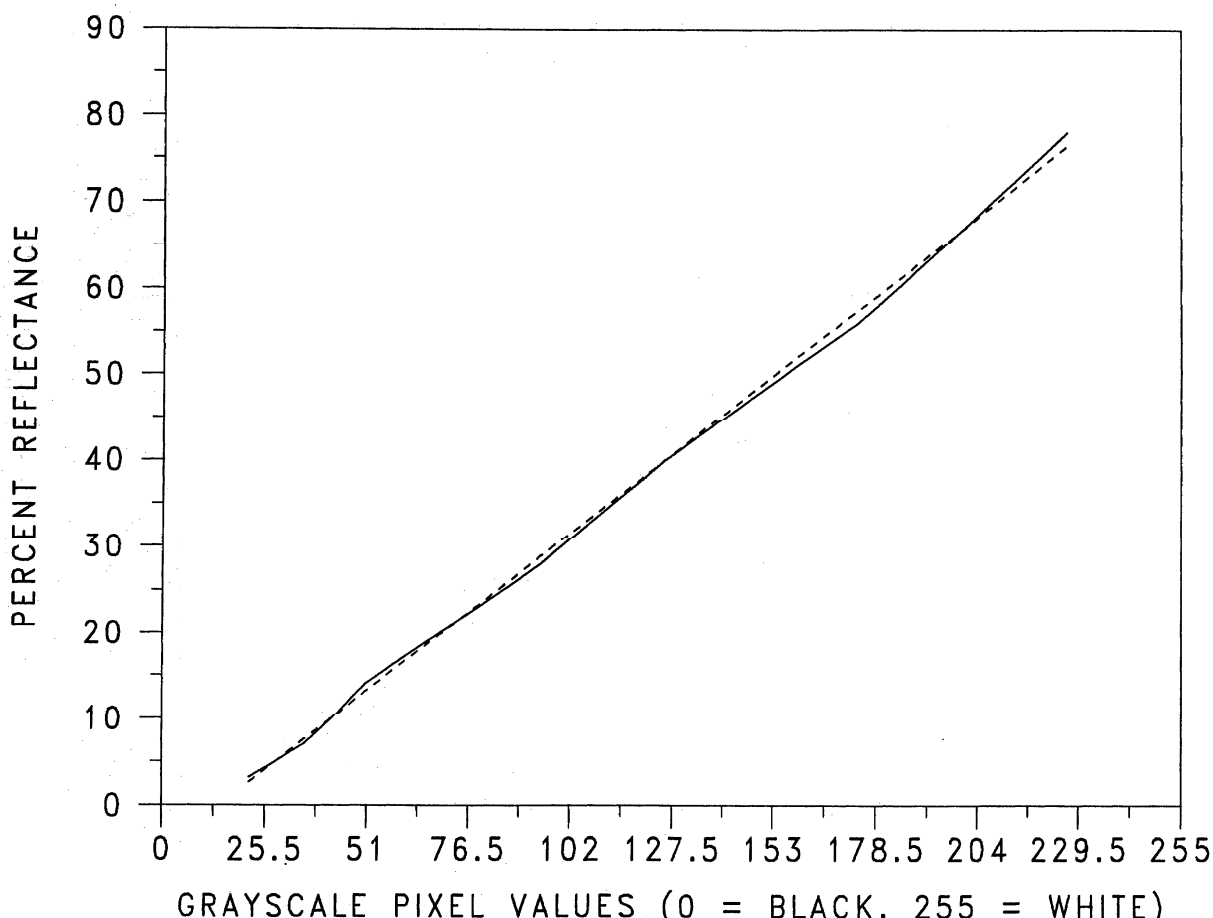


Figure A.1: Fingerprint file **aa000001.pct** from *NIST Special Database 10*.

Figure A.2: Fingerprint file **aa000042.pct** from *NIST Special Database 10*.

Appendix B: Database Reflectance and Resolution Calibration

FINGERPRINT DATABASE REFLECTANCE VALUES



KEY: SOLID: SCANNED DASHED: PREDICTED

PREDICTED % REFLECTANCE = −5.1 + (.36 * GRAYSCALE PIXEL VALUE)

Database Resolution Calibration

The resolution of the scanner used to create the database was calibrated using a NBS 1010A resolution chart[1]. A portion of this image is shown below (Figure B.2) after being magnified for viewing (Note: Printing has significantly reduced the quality of this image.). The scan of the table showed that the vertical resolution was approximately 10.5 line pairs/mm and the horizontal resolution was approximately 11.0 line pairs/mm.

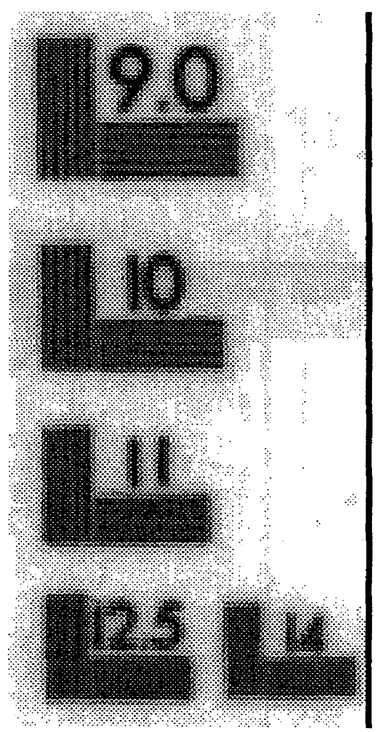


Figure B.2: Scan of Resolution Calibration Table.

1. National Bureau of Standards, Microcopy Resolution Test Chart, Standard Reference Material 1010a, ANSI and ISO test chart No. 2.

Appendix C: Fingerprint Class Distribution Statistics

Class Distribution

The data in Figure C.1 shows the exact class distribution of *NIST Special Database 10*. Since the data was collected by storing all ten fingerprints from a card some of the specific class groupings contain numerous fingerprints from other classes.

Note:

?L (1-5 ridge count loops)	All whorl ridge tracings (I,M,O) for
?M (6-30 ridge count loops)	a whorl class are counted together
?H (>30 ridge count loops)	(i.e. PI,PM,PO are counted in PW).

Specific Class Groupings

Actual Classes Collected	aa	sl	tt	cw	dw	pw	xw	total
AA	1199		88					1287
UL		843	149			2		994
UM		164	86			71		321
RL		189	23					212
RM			10					10
TT			842					842
PW				153	138	228	250	769
CW				308		94	7	409
DW				19	342	79	124	564
XW						6	99	105
SR	1	4	2					7
Total	1200	1200	1200	480	480	480	480	5520

Figure C.1: Class distribution statistics for *NIST Special Database 10*.

Class Referencing

The data in Figure C.2 shows the class referencing for *NIST Special Database 10*. The referencing statistics are given for each specific class grouping. All primary classes without references (UL, UM, ...) show the number of unreferenced prints for that particular class. See the note on page 21 for an explanation of the L, M, and H as used with the ulnar and radial loops. If a primary class has more than one reference the ordering is insignificant, meaning UL/AA/TT is the same as UL/TT/AA and both would be counted as UL/AA/TT.

Figure C.2: Class referencing statistics for *NIST Special Database 10*.

	Specific Class Groupings							
Class/Reference	aa	sl	tt	cw	dw	pw	xw	total
UL		812	178			69		1059
UL/UM		5	5					10
UL/TT		184	48					232
UL/UM/TT		1	3					4
UL/PW/DW			1					1
UL/CW/TT		1						1
UL/AA/TT		2						2
UM								
UM/UL		3	3					6
UM/CW						3		3
UM/TT			3					3
UM/UL/TT			1					1
UM/CW/DW						1		1
RL		115	18					133
RL/TT		70	9					79
RL/RM/TT			1					1
RM								
RM/RH			1					1
RM/CW			1					1
RM/TT			2					2
RM/PW/CW			1					1
RM/DW/XW			1					1
PW				145	120	198	220	683
PW/CW				8		9	4	21
PW/DW					18	21	26	65

Specific Class Groupings

Class/Reference	aa	sl	tt	cw	dw	pw	xw	total
CW				216		81	1	298
CW/UM				10		5	2	17
CW/RM				6			1	7
CW/PW				70		8	1	79
CW/DW				2			1	3
CW/PW/DW				1			1	2
CW/PW/XW				1				1
CW/DW/XW				2				2
DW				11	268	67	105	451
DW/UM				1	3	1		5
DW/PW				1	65	10	16	92
DW/CW				5	4	1	1	11
DW/XW							1	1
DW/PW/CW					2		1	3
XW						5	77	82
XW/RM							1	1
XW/PW							3	3
XW/CW							2	2
XW/DW						1	13	14
XW/PW/DW							3	3
AA	1160		68					1228
AA/TT	32		20					52
AA/UL/TT	1							1
TT			341					341
TT/UL			310					310
TT/UM			13					13
TT/RL			55					55
TT/RM			4					4
TT/CW			1					1
TT/AA			77					77
TT/UL/UM			6					6
TT/UL/RL			3					3
TT/UL/AA			22					22
TT/RL/RM			1					1
TT/RL/AA			2					2
Total scars	7	15	10	1				33

Appendix D: Manual Pages for Database Source Code

NAME

dcplljpg - non-standard JPEG lossless decompression for Ihead 8 bit gray scale images

SYNOPSIS

dcplljpg _ihdrfile_

DESCRIPTION

Dcplljpg takes an 8 bit gray scale ihead image, which was compressed using jpegcomp4, and decompresses it using techniques from the committee draft ISO/IEC CD 10198-1 for "Digital Compression and Coding of Continuous-tone Still images" with modifications to the draft image header.

NOTE: dcplljpg does not allow more than 8 bits/pixel input precision.

OPTIONS

ihdrfile
: Any 8 bit gray scale ihead raster image (previously compressed using jpegcomp4).

EXAMPLES

dcplljpg foo.pct

FILES

ihead.h	NIST's raster header include file
jpeg.h	Include file for jpeg algorithm

SEE ALSO

dumpihdr(1), ihdr2sun(1), ReadIheadRaster(3), writeihdrfile(3)

DIAGNOSTICS

dcplljpg exits with a status of -1 if an error occurs.

BUGS

dcplljpg only handles gray scale images up to 8 bits per pixel precision.

NAME

dumpihdr - takes a NIST IHead image file and prints its header content to stdout

SYNOPSIS

dumpihdr <u>ihdrfile</u>

DESCRIPTION

Dumpihdr opens a NIST IHead rasterfile and formats and prints its header contents to stdout.

OPTIONS

<u>ihdrfile</u>
any NIST IHead image file name

EXAMPLES

dumpihdr foo.pct

FILES

ihead.h NIST's raster header include file

SEE ALSO

ihdr2sun(1), ReadIheadRaster(3), writeihdrfile(3), writeihdr(3), readihdr(3), printihdr(3)

DIAGNOSTICS

Dumpihdr exits with a status of -1 if opening ihdrfile fails.

BUGS

NAME

ihdr2sun - takes a NIST ihead image and converts it to a Sun rasterfile

SYNOPSIS

ihdr2sun [-o outfile] ihdrfile [mapfile]

DESCRIPTION

Ihdr2sun converts a NIST ihead rasterfile to a Sun rasterfile. If the optional argument mapfile is included on the command line and the input image is multiple bitplane, the colormap in mapfile will be inserted into the Sun rasterfile, otherwise a default colormap gray.map will be used when necessary. The Sun image file created will have the root name of ihdrfile with the extension .ras appended, unless an alternate outfile is specified.

OPTIONS

ihdrfile
  any ihead raster image

mapfile
  optional colormap file

EXAMPLES

ihdr2sun foo.pct gray.map

FILES

/usr/include/rasterfile.h
  sun's raster header include file

ihead.h NIST's raster header include file

SEE ALSO

dumpidhr(1), sunalign(1), rasterfile(5)

DIAGNOSTICS

Ihdr2sun exits with a status of -1 if opening ihdrfile fails.

BUGS

Ihdr2sun does not currently support multiple bit levels per pixel other than depth 8.

NAME

sunalign - takes a sun rasterfile and word aligns its scan-lines

SYNOPSIS

sunalign <u>sunrasterfile</u>

DESCRIPTION

Sunalign takes the file sunrasterfile and determines if the stored scan lines in the file require word alignment. If so, the command overwrites the image data making scan lines word aligned. This command is useful when taking clipped images from the HP Scan Jet and importing them into Frame Maker.

OPTIONS

<u>sunrasterfile</u>
any sun rasterfile image

EXAMPLES

sunalign foo.ras

FILES

/usr/include/rasterfile.h
sun's raster header include file

SEE ALSO

rasterfile(5)

DIAGNOSTICS

Sunalign exits with a status of -1 if opening sunrasterfile fails.

BUGS

NAME

jpglldcp - takes a JPEG lossless compressed input data buffer (with modified data header) and writes the uncompressed data to the passed output buffer

SYNOPSIS

```
void jpglldcp(indata, width, height, depth, outbuffer)
unsigned char *indata, *outbuffer;
int width, height, depth;
```

DESCRIPTION

jpglldcp() takes the input buffer indata and decompresses it writing the uncompressed data into the output buffer outbuffer with length equal to the original image dimensions given. This procedure was developed using techniques from the committe draft ISO/IEC CD 10198-1 for "Digital Compression and Coding of Continuous-tone Still Images" with modifications to the draft image header. The source is found in the source code file jpglldcp.c.

indata
- the compressed data input buffer

width
- the pixel width of the image from which the input data came

height
- the pixel height of the image from which the input data came

depth
- the pixel depth of the image from which the input data came

outbuffer
- the output buffer in which the uncompressed data is to be returned

SEE ALSO

dcplljpg(1), ReadIheadRaster(3), writeihdrfile(3)

BUGS

NOTE: jpglldcp will only work with gray-scale images that were compressed using a modified data header (not the standard lossless JPEG data header).

NAME

printihdr - prints an ihead structure to the passed file pointer

SYNOPSIS

```
#include <ihead.h>

printihdr(head, fp)
IHEAD *ihead;
FILE *fp;
```

DESCRIPTION

Printihdr() takes a pointer to an ihead structure and prints the ihead structure to the file pointed to by fp. The source is found in the source code file ihead.c.

fp - an open file pointer

ihead
- a pointer to an initialized ihead structure

SEE ALSO

writeihdrfile(3), writeihdr(3), readihdr(3), ReadIheadRaster(3), dumpihdr(1)

BUGS

NAME

readihdr - allocates and reads header information into an ihead structure and returns the initialized structure

SYNOPSIS

```
#include <ihead.h>

readihdr(fp)
FILE *fp;
```

DESCRIPTION

Readihdr() takes a file pointer to an ihead structured file. Then allocates and reads the header information from the file into an ihead structure. The source is found in the source code file ihead.c.

fp - an open file pointer

SEE ALSO

ReadIheadRaster(3), writeihdrfile(3), printihdr(3), writeihdr(3), dumpihdr(1)

BUGS

NAME

ReadIheadRaster - loads into memory an ihead structure and corresponding image data from a file

SYNOPSIS

```
#include <ihead.h>

ReadIheadRaster(file, head, data, width, height, depth)
char *file;
IHEAD **head;
unsigned char **data;
int *width, *height, *depth;
```

DESCRIPTION

ReadIheadRaster() opens a file named file and allocates and loads into memory an ihead structure and its corresponding raster image data. If the image data is compressed, ReadIheadRaster will uncompress the data before returning the data buffer. This routine also returns several integers converted from their corresponding ASCII entries found in the header. The source is found in the source code file rasterio.c.

file - the name of the file to be read from

head - a pointer to where an ihead structure is to be allocated and loaded

data - a pointer to where the array of binary raster image data is to be allocated and loaded

width
- integer pointer containing the image's pixel width upon return

height
- integer pointer containing the image's pixel height upon return

depth
- integer pointer containing the image's Bits Per Pixel upon return

SEE ALSO

printihdr(3), readihdr(3), writeihdrfile(3), writeihdr(3), dumpihdr(1)

DIAGNOSTICS

ReadIheadRaster() exits with -1 when opening file fails.

BUGS

NAME

writeihdrfile - writes an ihead structure and corresponding image data to a file

SYNOPSIS

```
#include <ihead.h>

writeihdrfile(file, head, data)
char *file;
IHEAD *head;
unsigned char *data;
```

DESCRIPTION

Writeihdrfile() opens a file name file and writes an ihead structure and its corresponding image data to it. The source is found in the source code file rasterio.c.

file - the name of the file to be created

head - a pointer to an initialized ihead structure

data - the array of raster image data

SEE ALSO

writeihdr(3), printihdr(3), ReadIheadRaster(3), readihdr(3), dumpihdr(1)

DIAGNOSTICS

Writeihdrfile() exits with -1 when opening file fails.

BUGS

NAME

writeihdr - writes an ihead structure to an open file

SYNOPSIS

```
#include <ihead.h>

writeihdr(fp, ihead)
FILE *fp;
IHEAD *ihead;
```

DESCRIPTION

Writeihdr() takes a pointer to an ihead structure and writes it to the open file pointed to by fp. The source is found in the source code file ihead.c.

fp - an open file pointer

ihead
- a pointer to an initialized ihead structure

SEE ALSO

writeihdrfile(3), printihdr(3), readihdr(3), ReadIheadRaster(3), dumpihdr(1)

BUGS