# Nist Special Database 8

# Machine Print Database Users' Guide

**R. A. Wilkinson**

National Institute of Standards and Technology

Advanced Systems Division

Image Recognition Group

October 1, 1992

**Users' Guide**

---

## Contents

## 1.0 INTRODUCTION

This report describes the NIST Machine Print Database, *NIST Special Database 8* (SD8), which contains 360 8-bit gray scale images of pages containing machine printed characters, and a corresponding binary version of each page, resulting in a total of 720 digitized pages. This database is being distributed as a common set of images for use in the development and testing of Optical Character Recognition (OCR) systems. This allows vendors to report results with respect to this common image set. Each disc in this three-disc set contains approximately 593 Megabytes of storage when the images are compressed. Uncompressed each disc contains 1.1 Gigabytes of data (1.85: 1 average compression ratio using JPEG[3] and CCITT Group 4[1]).

## 2.0 DATABASE IMAGES AND REFERENCE FILES

The 720 digitized pages in SD8 are broken down into four groups. The first group is the style type. Three styles are used in this database; normal, bold and italic. Each style is organized on a separate disc; disc1, disc2, and disc3 respectively. Within each style there are six fonts; Courier, Helvetica, New Century School Book, Optima, Palatino, and Times Roman. Each font has 10 point sizes, 4, 5, 6, 8, 10, 11, 12, 15, 17, and 20 point. Finally, there are two examples of each point size, a randomly ordered page and a sequentially ordered page. The random page evenly distributes each character represented while randomizing the character occurrence. The ordered page selects the characters in order.

SD8 contains two types of images, binary and gray scale. The gray scale images are compressed using the lossless JPEG described in Section 2.3. The binary images are compressed using the CCITT Group 4 compression.

Both the binary and gray scale images have ASCII reference files containing the text of what was scanned from each page in the database. The page reference files are the ASCII files of text that was printed to generate the original hard copy version that would later be scanned to produce the gray scale images. These images can be used as input to a recognition system and the ASCII output file compared to the reference file. If the reference file did not exist then an operator would have to look at each character and decide what it was. These files contain a varying number of rows and columns for the different point sizes. For example more 4 point font characters can be printed on a standard 21.25 x 27.50 centimeter page, 8.5 x 11 inch, than 20 point font characters.

Back to Contents

### 2.1 IMAGE FILE FORMATS

Image file formats and effective data compression and decompression are critical to the usefulness of image archives. Each ASCII reference page was printed and later scanned at 12 dots per millimeter (dpm) gray scale, compressed using JPEG, and temporarily archived onto computer magnetic mass storage. The gray scale images were converted to binary using a smart thresholder, **thresh2**, which is explained in section 2.2. The binary images produced by thresholding were then compressed using CCITT Group 4. Once the gray scale, binary images, and reference ASCII files were organized they were mastered and replicated onto ISO-9660 formatted CD-ROM discs for permanent archiving and distribution.

In this application, a raster image is a digital encoding of light reflected from discrete points on a scanned page. The 2-dimensional area of the page is divided into discrete locations according to the resolution of a specified grid. Each cell of this grid, which is called a pixel, is represented by a value between 0 and 255; 255 represents a white pixel, 0 represents a black pixel with the values in between being levels of gray. For a binary image a white has a pixel value of 0 and black has a pixel value of 1. This 2-dimensional sampling grid is then stored as a l-dimensional vector of pixel values in raster order, left to right, top to bottom. Successive scan lines (top to bottom) contain the values of a single row of pixels from the grid.

Certain attributes of a raster image are required to interpret the l-dimensional pixel data as a 2-dimensional image. Examples of such attributes are the pixel width and pixel height of the image. These attributes are stored in a machine readable header prefixed to the raster bit stream.

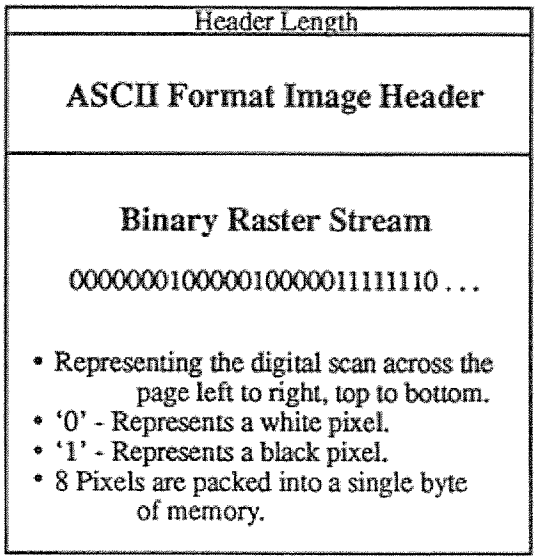



FIGURE 1. An illustration of the IHead binary raster file format.

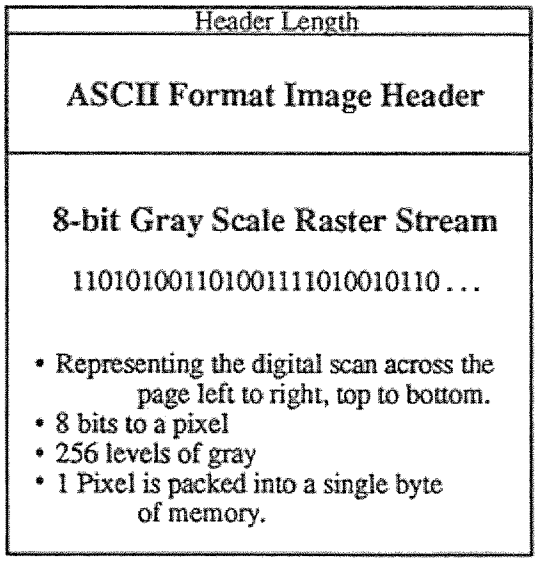



FIGURE 2. An illustration of the IHEAD grey scale raster file format

Numerous image formats exist, but most are proprietary. Some are widely supported on small personal computers and others are supported 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 scale 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. Figures 1 and 2 illustrate the IHead format for binary and gray scale images.

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 a 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 Figures 1 and 2.

```
/*****************************************************************
      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 3. IHead C language definition.

Back to Contents

The IHead structure definition as written in the `C' programming language is listed in Figure 3. Referencing the structure members listed in this figure, the first attribute field of IHead is the indentification field, **id**. This field uniquely identifies the image file, typically by a file name. The attribute field, **created**, is the date on which the image was captured or digitized. 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 the case of this database, 12 dots per millimeter which is equivalent to 300 dots per inch. The next two fields deal with compression.

In the IHead format, images may be compressed with virtually any algorithm. The IHead header is always uncompressed, even if the image data is compressed. 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 that 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. In order to load the compressed image data into main memory, the value in **complen** is used to load the compressed block of data into main memory. Once the compressed image data has been loaded into memory, the appropriate decompression technique can be used to produce an image which has the pixel dimensions consistent with those stored in its header. The gray scale images in SD8 have been compressed using the JPEG compression described in section 2.3. The binary images have been compressed using CCITT Group compression.

The attribute field, **align**, stores the alignment boundary to which scan lines of pixels are padded. Pixel values of binary images are stored 8 pixels (or bits) to a byte. Most images, however, are not an even multiple of 8 pixels in width. In order to minimize the overhead of ending a previous scan line and beginning the next scan line within the same byte, a number of padded pixels are provided in order to extend the previous scan line to an even byte boundary. Some digitizers extend this padding of pixels out to an even multiple of 8 pixels, other digitizers extend this padding of pixels out to an even multiple of 16 pixels. This field stores the image's pixel alignment value used in padding out the ends of raster scan lines.

The next three attribute fields identify binary interchanging issues among heterogeneous computer architectures and displays. The **unitsize** field specifies how many contiguous pixel values 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, binary incompatibilities across computer hardware and binary 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 binary images in this database are black text on a white background and the value of the white pixels is 0. 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 one from which the current image was derived or extracted. The **parent** field typically contains the full path name to the image from which the current image was extracted. The **par_x** and **par_y** fields contain the origin point (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. If the image has no parent, these three fields contain a null string. The IHead image format contains the minimal amount of ancillary information required to successfully manage binary and gray scale images.

Back to Contents

## 2.2 BINARIZATION

The gray scale images in SD8 were converted to binary using **thresh2**. The program, **thresh2**, takes three parameters, an outer window, an inner window, and a thresholds level. For the point sizes 4, 5, 6, 8, and I0 an outer window of 6 and inner window of 5 are used. For the other 5 point sizes 11, 12, 15, 17, and 20 an outer window of 8 and inner window of 5 are used. On the normal page style, disc 1, and italics, disc3, discs, a thresholds level of 0.25 was used while on the bold style disc, disc2, 0.40 was used.

The outer window determines the size of the region that the weighted sum will be applied to. The inner window is used to determine the weight that will be applied to the value at a position in the outer window. The weights are generated using the Gaussian equation (1) where *i* is the row position, *j* is the column position, and *p* is the size of the inner window.

$$w_{i,j} = e^{\frac{-(i^2+j^2)}{p^2}} \qquad (1)$$

The weighted sum is divided by the sum of the weights to obtain a weighted average. This weighted average is then compared to the threshold level. The threshold level is a value between 0 and 255. The value for the threshold level is calculated by 256 times percentage threshold. If the weighted average is less than the threshold level then the value is set to 0, otherwise it is set to 1.

### 2.3 MODIFIED JPEG LOSSLESS COMPRESSION

The compression used was developed from techniques outlined in the WG10 "JPEG" (draft) standard for 8-bit gray scale images with modifications to the compressed data format. 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. The IHead image format is described in section 2.1. Codes used to compress and decompress the images are still developed per the draft standard, but only with respect to 8-bit gray scale sca 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 up to 99.9% of fingerprint images; therefore, this same predictor was used to compress all of the character images.

Back to Contents

## 3.0 DATABASE CONTENT AND ORGANIZATION

In additon to the image and reference files both the binary and grey scale images have ASCII reference files which are the ASCII format of what was scanned. SD8 contains statistical logs, source code for file and image manipulation, and UNIX-style manual pages for software documentation.

### 3.1 Database Statistics

This database is a three disc set. Each volume contains font style: normal, bold, or italic. In each style there are six font types; Courier, Helvetica, New Century Schoolbook, Optima, Palatino, and I Times Roman. For each font there are 10 point sizes: 4, 5, 6,8, 10, 11, 12, 15, 17, and 20. Two " pages of each point size were generated, one random order and one sequential order. The random page has the character sampling randomly ordered and equally distributed while the sequential page uses the sequenial order of the character samples.

There are 94 different characters represented on each page. These characters include 26 upper case alphas, 26 lower case alphas, 10 numerics, and 22 special characters. The total number of examples is

3,063,168 characters at an average of 8509 characters per page. Specific information for each page is in the file **stats.log** in the **doc** directory.

## 3.2 Database Hierarchy

Figure 4 illustrates the top level directory tree in the database. This directory structure is the same for each disc. The **doc, man** and **src** directories are the same on each disc. The directory **doc** contains the statistical logs described in Section 3.1. The directories **man** and **src**, contain documentation and utilities necessary to manipulate the files and image data on the CDs discussed in Section 4. The **data** directory contains the image files and the ASCII page reference files described in Section 2. The organization of the **data** directory is illustrated in Figure 5.

## 3.3 File Naming Conventions

File names have been derived so that the names accurately describe the data being represented. All file names are of the format **xxx J _z.ddd**. The first three characters (**xxx**) represent the font type and style. The values that can be used are *cou* for Courier, *hel* for Helvetica, *new* for New Century School Book, *opt* for Optima, *pal* for palatino, and *tim* for Times-Roman. The style is specified by the third position of this field, if it is unchanged the style is normal, for bold this position is a b and for italics it is an i. The **y** field designates whether the reference file was generated randomly or not. An **r** in this field means it was randomly generated where an **n** means it was not. The **z** field holds the point size and contain the values 4,5,6,8, 10, 11, 12, 15, 17, or 20. The **ddd** extension describes what t ype of file this actually is, it can have one of three values, *grey* for gray scale images, *bin* for binary images, and *txt* for ASCII reference files. The file name **cob_r _15.gry** means the file is Courier, bold, 15 point font, randomly generated, and gray scale.

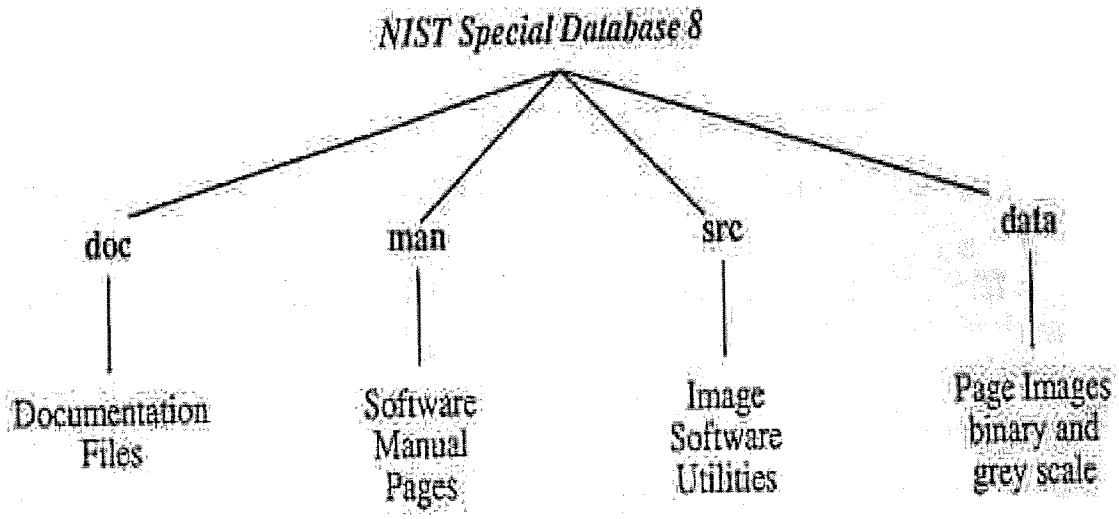



FIGURE 4. The top level directory tree of a disc in the database.

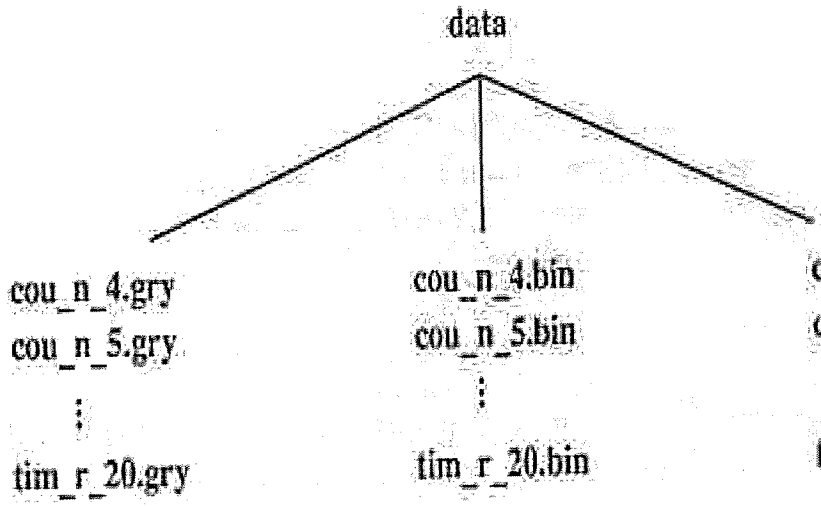



FIGURE 5. The data directory hierarchy. This example is from the normal disc, disc1.

Back to Contents

## 4.0 SOURCE CODE FOR DATABASE ACCESS

SD8 contains software written in the 'C' programming language on a UNIX serial workstation. Source code for 7 different programs: **decomp, dumpihdr, ihdr2sun, sunalign, dcplljpg, thresh 1,** and **thresh2** are included within the top level database directory **src**. These programs, their primary supporting subroutines, and associated file names are described below. These routines are provided as an example to software developers of how IHead images may be manipulated. Manual pages are included in Appendix C and are located in the top level database directory **man**.

### 4.1 Compilation

CD-ROM is a read-only storage medium; this requires the files located in the directory **src** to be copied to a read-writable disk partition prior to compilation. Once these files have been copied, executable binaries can be produced by invoking the UNIX utility **make**. A command-line example follows.

# make -f **makefile.mak**

### 4.2 decomp <IHead file in><IHead file out>

The program **decomp** decompresses an image in IHead format. The output file specified will be an image in IHead format with its image data uncompressed. The main routine for **decomp** is found in **decomp.c** and calls the external functions **readihdrfile()** and **writeihdrfile()**.

The procedure **readihdrfile()** is responsible for loading an IHead image from a file into main memory and is found in the file **loadihdr.c**. This routine reads the image's header data returning a 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 CCITT Group 4, therefore **readihdrfile()** invokes the external procedure **grp4decomp()** which decompresses the raster data. Upon completion, **readihdrfile()** returns an initialized IHead structure, the uncompressed raster data, and the image's width and height in pixels. **Grp4de- comp()** was developed by the CALS Test Network[2] and adapted by NIST for use with this database and is found in the file **g4decomp.c**.

The function **readihdr()** is 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**.

### 4.3 dumpihdr <IHead file>

The program **dumpihdr** 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 main routine for **dumpihdr** is found in the file **dumpihdr.c** and calls the external function **readihdr()**.

Back to Contents

### 4.4 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 **jpgIldcp()** 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.

**JpgIldcp()** accepts image raster data compressed using the modified JPEG lossless compression algorithm and returns the uncompressed image raster data. **JpgIldcp()** was developed using techniques described in

the WGIO "JPEG" (draft) standard [3] and adapted for use with this database. Source code for the algorithm is found in **jpglldcp.c**.

### 4.5 dcplljpg <lossless JPEG compressed IHEAD file>

**Dcplljpg** is a program which decompresses a gray scale image file (approximately 90 seconds per image 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 4.4 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**.

### 4.6 sunalign <Sun rasterfile>

The program sunalign is a program which ensures the Sun rasterfile input has scanlines of length equal to a even multiple of 16 bits. It has been found that some Sun rasterfile applications assume scanlines which end on an even word boundary. IHead images may contain scanlines which do not conform to this assumption. Therefore, it may be necessary to run **sunalign** on an image which has been converted using **ihdr2sun**. The main routine for this program is found in the file **sunalign.c**.

### 4.7 threshl <8-bit gray image><1-bit resulting image><percentage threshold>

The program **threshl** is a simplistic gray scale to binary thresholder that sets all values below a threshold to 0 and all others to 1. The threshold level is determined by 256 times the **percentage threshold** parameter. If the **percentage threshold** parameter is 0.50 then the threshold level would be 128.

### 4.8 thresh2 <8-bit gray image><1-bit resulting image><outer window><inner window><percentage threshold>

**Thresh2** is a gray scale to binary thresholder that uses a weighted average over a region to determine the value for a pixel to be compared against the thresholding level.

### References

[1] CCITT, "Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus, Facsimile VII.3 - Rec. T.6," 1984.

[2] Department of Defense, "Military Specification - Raster Graphics Representation in Binary Format, Requirements for, MIL-R-28002," 20 Dec 1988.

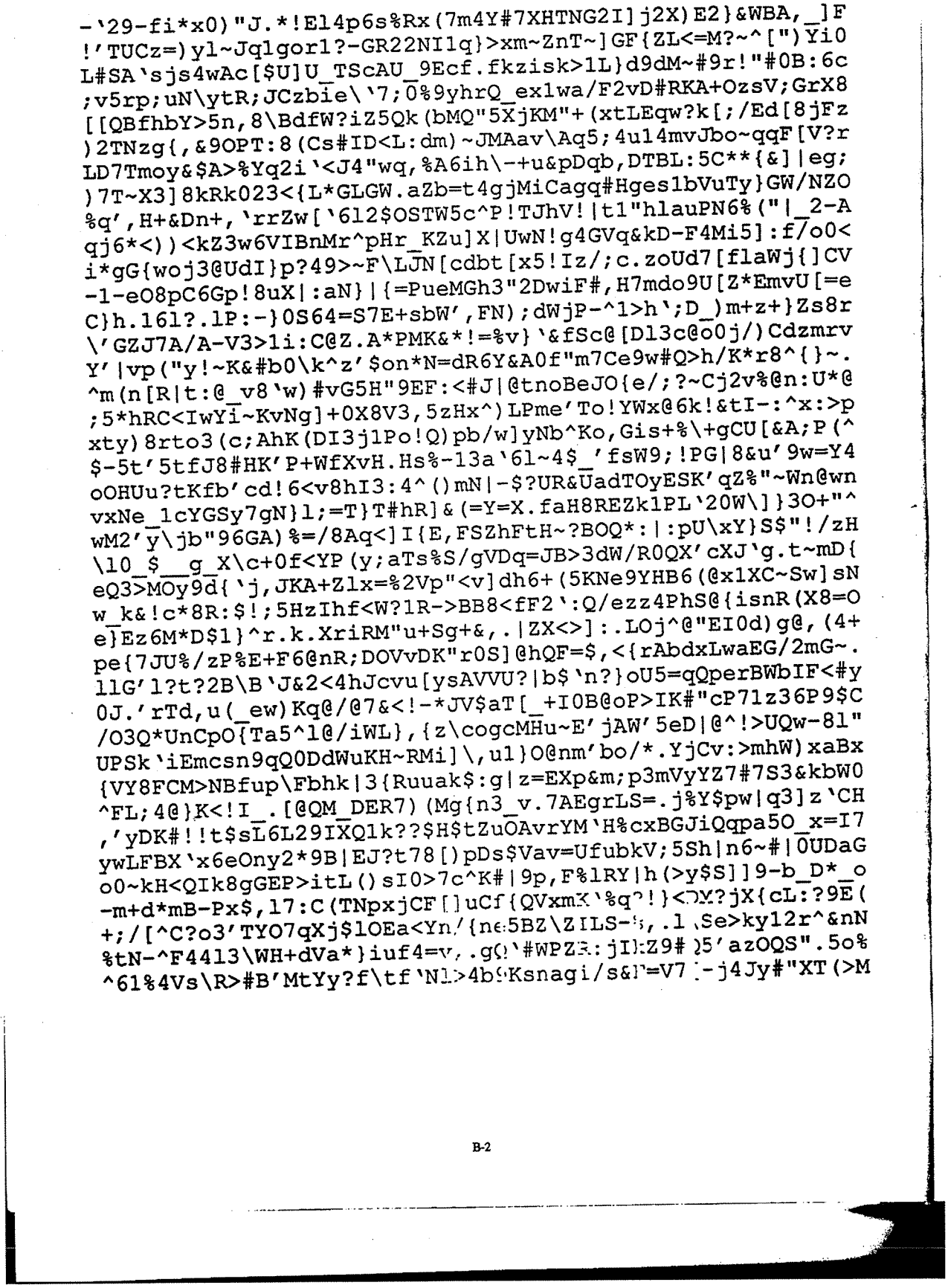
[3] WG10 "JPEG", committee draft ISO/IEC CD 10198-1, "Digital Compression and Coding of Continuous-Tone Still Images," March 3, 1991.

Back to Contents

# Appendix A: Non-random Page Image Example

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVW
XYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0
123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefg
hijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
xyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()
*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789
:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnop
qrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI
JKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY
Z[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./012
3456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghi
jklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@AB
CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
z{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQR
STUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+
,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`ab
cdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;
<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
stuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJK
LMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$
%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./01234
56789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijk
lmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCD
EFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{
|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRST
UVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcd
efghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=
>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrst
uvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&
'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456
789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklm
nopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCDEF
```

# Appendix B: Random Page Image Example




B-2

# Appendix C. Manual Pages for Supplied Software

**DCPLLJPG(1) USER COMMANDS DCPLLJPG (1)**

**NAME**

dcplljpg - jpeg lossless decompression for Ihead 8 bit gray scale images

**SYNOPSIS**

**dcpUjpg** *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 bit gray scale ihead raster image (previously compressed using jpegcomp4).

**EXAMPLES**

**dcplljpg foo.pct**

**FILES**

| | |
|---|---|
| **ihead.h** | NIST's header include file |
| **jpeg.h** | for jpeg algorithm |

**SEE ALSO**

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

**DIAGNOSTICS**

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

**BUGS**

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

Sun Release 4.1 Last change: 14 November 1991

Back to Contents

---

**DECOMP(1) USER COMMANDS DECOMP(1)**

**NAME**

**SYNOPSIS**

**DESCRIPTION**

**Decomp** takes a compressed ihead image file, examines the compression field of the header, and applies the correct decompression algorithm storing the results as a decompressed ihead image in the out file.

**OPTIONS**

*IHead file in*

Any NIST IHead raster image *IHead file out* Any NIST IHead raster image

**EXAMPLES**

**decomp foo.pct fool.pct**

**FILES**

**/usr/share/include/ihead.h**

NIST's raster header include file

**SEE ALSO**

**dumpihdr(l)**

**DIAGNOSTICS**

**Decomp** exits with a status of –1 if opening **ihdrfile** fails.

**BUGS**

Sun Release 4.1 Last change: 10 February 1992

---

**DUMPIHDR (1)** **USER COMMANDS** **DUMPIHDR (1)**

**NAME**

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

**SYNOPSIS**

**dumpihdr** *ihdrfitle*

**DESCRIPTION**

**Dumpihdr** opens an NIST IHead rasterfile and formats and prints its header contents to stdout

**OPTIONS**

*ihdrfile* any NIST IHead image file name

**EXAMPLES**

**dumpihdr foo.pct**

**FILES**

**ihead.h** NIST's raster header include file

**SEE ALSO**

**ihdr2sun(l), ReadIheadRaster(3), writeihdrfile(3), writeihdr(3), readihdr(3), printihdr(3)**

**DIAGNOSTICS**

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

**BUGS**

Sun Release 4.1 Last change: 15 March 1990

Back to Contents

---

**IHDR2SUN(1) USER COMMANDS IHDR2SUN(1)**

**NAME**

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

**SYNOPSIS**

**ibdr2sun** [ *-o outfile* ] *ihdrfile* [ *mapfile* ]

**DESCRIPTION**

**Ibdr2sun** converts an 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**

**ibdr2sun foo.pct gray.map**

**F1LES**

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

**SEE ALSO**

**dumpidhr(l), sunalign(l), 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.

Sun Release 4.1 Last change: 08 March 1990

**SUNALIGN(1) USER COMMANDS SUNALIGN(1)**

**NAME**

sunalign - takes a sun rasterfile and word aligns its scanlines

**SYNOPSIS**

**sunalign** *sunrasterfile*

**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**

*sunrasterfile* 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**

Sun Release 4.1 Last change: 08 March 1990

Back to Contents

**THRESHI(1) USER COMMANDS THRESH1(1)**

**NAME**

**thresh 1** - simplistic grey scale to binary thresholder

**SNOPSIS**

**thresh1** *<8-bit grey image> <1-bit resulting image><percentage threshold>*

**DESCRIPTION**

**Thresh1** is a simplistic grey scale to binary thresholder that sets all values below a threshold to 0 and all others to 1.

**OPTIONS**

*8-bit grey image*

Any NIST IHead grey scale raster image

*1-bit resulting image*

Any NIST IHead binary raster image

percentage threshold

The percentage times 256 decides the value to use in thresholding.
Everything less than this value becomes 0, otherwise it becomes 1.

**EXAMPLES**

**thresh1 foo_grey.pct foo_bin.pct .50**

**FILES**

/usr/share/include/ihead.h

NIST's raster header include file

**SEE ALSO**
**DIAGNOSTICS**
**BUGS**

Sun Release 4.1 Last change: 24 June 1992

---

**THRESH2(1) USER COMMANDS THRESH2 (1)**

**NAME**

thresh2 - grey scale to binary thresholder using weighted Gaussian sums of a region

**SYNOPSIS**

**thresh2** *<8-bit grey image> <1-bit resulting image> <outer window> <inner window> <percentage threshold>*

**DESCRIPTION**

**Thresh2** is a grey scale to binary thresholder that uses a weights average of a region to be thresholded. The outer window is the area that the weighted sum will be applied to. The inner window is used to determine the weight that will be applied to the value at a position in the outer window. The weights are generated using the Gaussian eA( -(i^2+j^2)/(inner window^2)). Then the weighted sum is divided by the summed weights to get the weighted average of the outer window. This weighted average is then thresholded. If the weighted average is less than 256 x the percentage then the value is set to 0, otherwise 1.

**OPTIONS**

*8-bit grey image*

Any NIST IHead grey scale raster image

*i-bit resulting image*

Any NIST IHead binary raster image

*outer window size*

This value determines the size of the area to be summed.
A value of 3 produces a windows size of 7x7 around the pixel in question.

*inner window size*

This value is used to determine the weighting factor.

*percentage threshold*

This percentage times 256 determines the value to be used as the threshold.

**EXAMPLES**

**thresh2 foo_grey.pct foo_bin.pct 3 1 .50**

**FILES**

**/usr/share/include/ihead.h**

NIST's raster header include file

**SEE ALSO**
**DIAGNOSTICS**
**BUGS**

Sun Release 4.1 Last change: 24 June 1992

Back to Contents

**READIHEADRASTER(3) C LIBRARY FUNCTIONS READIHEADRASTER(3)**

## 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 name **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), **writeibdr**(3), **dumpibdr**(l)

## DIAGNOSTICS

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

## BUGS

Sun Release 4.1 Last change: 05 March 1990

Back to Contents

**GRP4DECOMP(3 C LIBRARY FUNCTI0NS GRP4DECOMP(3)**

## NAME

grp4decomp - takes an CCITT Group 4 Compressed input buffer and returns an output buffer of uncompressed data

## SYNOPSIS

**grp4decomp(indata, inbytes, width, height, outdata, outbytes)**
**unsigned char *indata, *outdata;**
**int inbytes, width, height, *outbytes;**

## DESCRIPTION

**grp4decomp()** takes the input buffer **indata** of length **inbytes** and decompresses it returning the uncompressed data in the output buffer **outdata** with length **outbytes**. This procedure was developed by the CALS Test Network and adapted for use by NIST. The source is found under **src** in the file **g4decomp.c.**

**indata** - the compressed data input buffer

**inbytes** - the length of the input data in bytes

**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

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

**outbytes** - a pointer to the length in bytes of the uncompressed output data

## SEE ALSO

**readihdrfile**(3)

## BUGS

**NOTE: Grp4decomp** will only work with binary image data.

Sun Release 4.1 Last change: 15 March 1990

Back to Contents

**JPGLLDCP(3) C LIBRARY FUNCTI0NS JPGLLDCP(3)**

## NAME

jpglldcp - takes an jpeg lossless compressed input data buffer (with modified data header) and writes the uncompressed data to the output buffer

## SYNOPSIS

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

## DESCRIPTION

**jpgUdcp**() 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(l), ReadlheadRaster(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).

Sun Release 4.1 Last change: 14 January 1992

Back to Contents

## NAME

**readihdrfile** - loads into memory an IHead structure and corresponding binary image data from a file

## SYNOPSIS

```
#include <ihead.h>

readihdrfile(file, head, data, width, height)
char *file;
IHead **head;
unsigned char **data;
int *width, *height;
```

## DESCRIPTION

**readihdrfile**() opens a file named **file** and allocates and loads into memory an IHead structure and its corresponding binary image data. If the image data is compressed, readihdrfile 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 under **src** in the file **loadhsf.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 dIe image's pixel height upon return

## SEE ALSO

**g4decomp**(3)

## DIAGNOSTICS

**readihdrfile**() exits with -1 when opening **file** fails or the image contains multiple bit planes.

## BUGS

Sun Release 4.1 Last change: 15 March 1990

Back to Contents

**PRINTIHDR (3) C LIBRARY FUNCTIONS PRINTIHDR (3 )**

## NAME

printihdr - prints an ihead structure to the passed file pointer

## SYNOPSIS

```
#include <ihead.h>

readihdrfile(file, head, data, width, height)
char *file;
IHead **head;
unsigned char **data;
int *width, *height;
```

## DESCRIPTION

**readihdrfile()** opens a file name **file** and allocates and loads into memory an IHead structure and its corresponding binary image data. If the image data is compressed, readihdrfile 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 under **src** in the file **loadhsf.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

## SEE ALSO

**g4decomp**(3)

## DIAGNOSTICS

**readihdrfile**() exits with –1 when opening **file** fails or the image contains multiple bit planes.

## BUGS

Sun Release 4.1 Last change: 15 March 1990

Back to Contents

**PRINTIHDR(3) C LIBRARY FUNCTIONS PRINTIHDR(3)**

**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(l)**

**BUGS**

Sun Release 4.1 Last change: 15 January 1992

Back to Contents

---

**READIHDR(3) C LIBRARY FUNCTIONS READIHDR(3)**

**NAME**

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

**SYNOPSIS**

```
#include

IHEAD *readihdr(fp)
FILE *fp;
```

**DESCRIPTION**

**Readihdr**() takes a file pointer to an **ihead** structured file. Then allocates and reads the header infor- mation 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(l)**

**BUGS**

Sun Release 4.1 Last change: 16 January 1992

---

**WRITEIHDRFILE(3) C LIBRARY FUNCTIONS WRITEIHDRFILE(3)**

**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 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(l)**

**DIAGNOSTICS**

**Writeihdrfile()** exits with –1 when opening **file** fails.

**BUGS**

Sun Release 4.1 Last change: 02 March 1990

Back to Contents

---

**WRITEIHDR(3) C LIBRARY FUNCTIONS WRITEIHDR(3)**

**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(l)**

**BUGS**

Sun Release 4.1 Last change: 02 March 1990

Back to Contents | Database Description | SRD Data Home