Nist Special Database 2

Structured Forms Database Users' Guide

D. L. Dimmick, M. D. Garris, and C. L. Wilson

National Institute of Standards and Technology

Advanced Systems Division

Image Recognition Group

December 1, 1991

1.0 Introduction

This report describes the NIST Structured Forms Reference Set database, *NIST Special Database 2*, containing binary images of synthesized documents. Databases of this magnitude are necessary to further the research and development of automated document processing systems. This database is being distributed as a reference data set to be used by developers of document recognition and data capture systems to test and report results on a common corpus of images derived from structured forms containing machine printed data. The structured forms used in this database are 12 different tax forms from the IRS 1040 Package X for the year 1988. These include Forms 1040, 2106, 2441, 4562, and 6251 together with Schedules A, B, C, D, E, F, and SE. Eight of these forms contain two pages or form faces making a total of 20 different form faces represented in the database.

The database contains 5,590 full page images of completed tax forms. Each image is stored in the bilevel black and white raster format defined in Section 2.2. The images in this database appear to be real forms prepared by individuals but the images have been automatically derived and synthesized using a computer.

2.0 Image Synthesis

The entry field values on these forms have been automatically generated by a computer in order to make the data available without the danger of distributing privileged tax information. The computer derived entry field values are synthesized as images from one or more fonts of machine printed data. An image of an entry field value is produced by combining images of each character in the value. An entry field image is then inserted in a selected location within the corresponding field within a form image. The image data entered in a field in this way has been translated and rotated by small amounts to simulate effects of imperfect printing and imperfect alignment of a form in the printing device. Multiple examples of the digital representation of each character are used so that the pattern of the binary pixels representing each character is not consistently replicated but varies as it would in a sample of real tax forms. Both the form templates and the character examples are digitized at 300 pixels per inch binary. Figure 1 displays a synthesized tax form.

The values entered on the forms have been derived by a computer. These entry field values are stored separately from the image in an ASCII text file. This text file, one per completed structured form image, serves as an answer file which can be used to score the values hypothesized by a recognition system. An example of one of the answer files in the database is listed in Figure 2. These text files are the ground truth against which recognition responses may be compared.

The information in Figure 2 has been listed in two adjacent text columns. The first line in this file contains the identification of the form face in the referenced image. *NIST Special Database* 2 contains multiple form faces and therefore can be used for testing the forms identification ability of a document recognition system. The form type identification can be used to compute a system's accuracy in correctly identifying the form face contained in an image.

Each successive line in the answer file is an entry field identification and entry field value pair. The field identification string uniquely identifies which entry field is being referenced on a structured form. The field identifications used in this database are labeled on the form faces contained in Appendix A. The entry field value may be empty or it may contain a computer derived value. Empty entry field values model sparsely filled forms. An entry field value containing the token string "_ICON_" represents the existence of non-character information. Examples of this type of information includes box check marks and signatures. Any other value listed for an entry field references the precise character information entered into the form image.

2.2 Image File format

Image file formats and effective data compression and decompression are critical to the usefulness of image archives. Each of a completed form face was synthesized at 300 dots per inch binary, 2-dimensionally compressed using CCITT Group 4, and temporarily archived onto computer magnetic mass storage. Once all forms were synthesized, the images 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 form. The 2-dimensional area of the form is divided into discrete locations according to the resolution of a specified grid. Each cell of this grid is represented by a single bit value or 1 called a pixel; 0 represents a cell predominately white, 1 represents a cell predominately black. This 2-dimensional sampling grid is then stored as a 1-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 concatenated together.

After digitization, certain attributes of an image are required to be known 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 is used to manipulate the raster data of an image, is able to first read the header and determine the proper interpretation of the data which follows it. Figure 3 illustrates this file format.

A header format named IHead has been developed for use as an image interchange format.

Numerous image formats exist; some are widely supported on small personal computers, others supported on larger workstations; most are proprietary formats, few are public domain. 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.

IHead has been successfully ported and tested on several systems including UNIX workstations and servers, DOS personal computers, and VMS mainframes. The attribute fields in IHead can be loaded into main memory in two distinct ways. Since the attributes are represented by the ASCII character set, the attribute fields may be parsed as null-terminated strings, an input/output format common in the 'C' programming language. IHead can also be read into main memory using

record-oriented input/output. The fixed length of the header is prefixed to the front of the header as shown in Figure 3. The IHead structure definition as written in the 'C' programming language is listed in Figure 4.

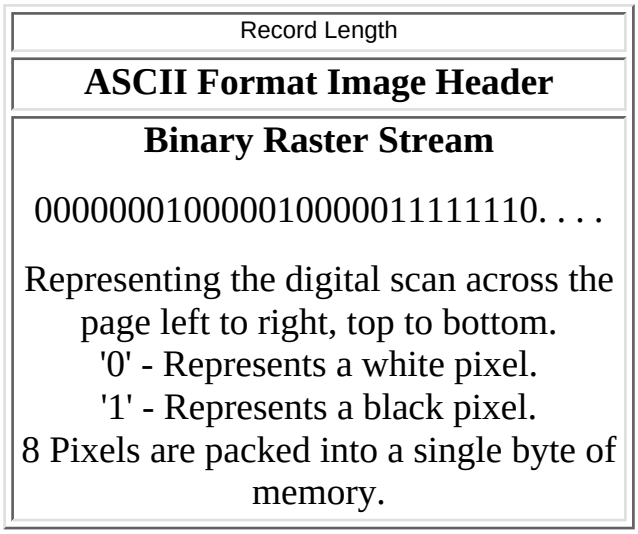

FIG. 3. An illustration of the IHead raster file format.

FIGURE 4. IHead C language definition.

Figure 5 lists the header values from an IHead file corresponding to the structure members listed

Figure 4. This header information belongs to the isolated box image displayed in Figure 6.

Referencing the structure members listed in Figure 4, 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 reference string the writer was instructed to print in the box. The reference string is delimited by double quotes.

FIGURE 5. The lHead values for the isolated subimage displayed in Figure 6.

FIGURE 6. An lHead image of an isolated box.

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 e scan resolution of the image; in this case, 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 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 that 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. For example, the image described in Figure 5 has a compression code of 2. This signifies that CCITT Group 4 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 is used to load the compressed block of data into main memory. Once the compressed image data has been loaded into memory, CCITT Group 4 decompression can be used to produce an image that has the pixel dimensions consistent with those stored in its header. Using CCITT Group 4 compression and this compression scheme on the images in this database, a compression ratio of 10 to 1 was achieved.

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 image described in Figure 5 is 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. In Figure 5, the parent field contains the full path name to the image from which the image displayed in Figure 6 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. The IHead image format contains the minimal amount of ancillary information required to successfully manage binary and gray scale images.

3.0 Data Base Content and Organization

NIST Special Database 2 contains 5,590 full page images of completed structured forms and correspondingly contains 5,590 ASCII text answer files. The database is approximately 610 Megabytes in size and is distributed on an ISO-9660 formatted CD- RO M disc. The binary images in the database have been 2-dimensionally compressed. Uncompressed the database would require 5.9 Gigabytes of storage.

3.1 Hierarchy

Figure 7 illustrates the top-level directory tree in the database. The directories **doc**, **man**, and **src**, contain documentation and utilities necessary to manipulate the image data on the CD discussed in Section 4. The **data** directory contains files of images and entry field value answer

files described in Section 2.0. The organization of these files is illustrated in Figure 8.

FIGURE 7. The top-level directory tree in the database.

FIGURE 8. The file organization of the form images and answer files contained in *NIST Special Database* 2.

There are 5,590 full-page images of completed forms distributed across 8 subdirectories within **data**. The subdirectories **sfrs_0**, **sfrs_1**, through **sfrs_8** each contain 100 synthesized tax submissions comprised of a random collection of completed form faces generated by a computer. Therefore there are 900 total tax submissions in this database. Each submission is represented as a directory. An example of a submission directory **r0200** is illustrated in Figure 8. In this example,

the directory **sfrs_2** contains the 100 submission directories **r0200** through **r0299**. The images associated with submission 200 are stored in the subdirectory **r0200**. There are on average 6.2 form images stored in a submission directory. In Figure 8, **r0200** contains 9 synthesized form faces stored as the files **r0200_00.pct**, **r0200_01.pct**, through **r0200_08.pct** where the last two digits in the file name uniquely index the form images. For each form face image, there is a corresponding answer file. The answer file for the image **r0200_00.pct** is **r0200_00.fmt**, **r0200_01.pct is r0200_01.fmt**, and so on. In this way 5,590 form images are stored on the CD with their 5,590 corresponding answer files accounting for 11,180 individual files in all.

4.0 Source Code For Data Base Access

In addition to images and answer files, the database contains documentation and software written in the 'C' programming language. Source code for 3 different programs: **dumpihdr**, **ihdr2sun**, and **sunalign** is included within the top-level database directory **src**. These programs, their 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 B 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 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 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 5 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 4 and is found in the file **ihead.h**.

4.3 Ihdr2sun <IHead file>

Ihdr2sun converts an image from NIST !Head 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 **readihdrfile()** which is found in the file **loadihdr.c**.

Readihdrfile() is a 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 CCITT Group 4, therefore **readihdrfile()** invokes the external procedure **grp4decomp()** which decompresses the raster data. Upon completion, **readihdrfile()** returns an initialized lHead structure, the uncompressed raster data, and the image's width and height in pixels. **Grp4decomp()** was developed by the CALS Test Network and adapted by NIST for use with this database and is found in the file **g4decomp.c** [1,2].

4.4 Sunalign <Sun rasterfile>

Sunalign is a program which ensures the Sun rasterfile passed has scan lines 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. lHead 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 convened using **ihdr2sun**. The main routine for this program is found in the file **sunalign.c**.

5.0 Entry Field Documentation Tables.

The final set of information provided with this database is a collection of tables. These tables contain general knowledge about each entry field found on a structured form. This knowledge can be applied by system developers to guide the recognition process of their document processing system. These tables specify the data type and context associated with each entry field found on the form faces labeled in Appendix A. Formatted copies of these tables are included in Appendix C and are found in the directory **tables** within the top-level database directory **doc**.

Appendix C contains 20 different tables, 1 for each of the 20 different form faces found in this database. Each line in these tables references a unique entry field from the corresponding form face. Entry fields are described by three columns of information. The first column in these tables contains entry field identifiers, the second column contains entry field data types, and the third column contains each entry field's associated context. Figures 9 and 10 list the possible entry field data types and contexts contained on the structured form faces used in this database.

TAG	DEFINITION
A, CA	Alphanumeric Field
F, FF, FP, FPER,	Floating Point Fields
FU	Integer Fields
I	Non-Character Fields (box markings,
ICON	signatures)

FIG. 9. The set of possible entry field data types.

TAG	DEFINITION
DATA	Generic Data
NAME	Names of People
SSN	Social Security Numbers

FIG. 10. The set of possible entry field contexts.

References

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

[2] CCITT, ``Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus, Fascicle VII.3 - Rec. T.6,'' 1984.

Appendix A: Labeled Form Faces 13

Appendix B: Manual Pages for Supplied Software

DUMPIHDR(1) USER COMMANDS

NAME

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

SYNOPSIS

dumpihdr *ihdrftle*

DESCRIPTION

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

OPTIONS

ihdrftle any NIST IHead image file name

EXAMPLES

dumpihdr foo.pct

FILES

ihead.h NIST's raster header include file

SEE ALSO

ihdr2sun(l)

DIAGNOSTICS

Dumpihdr exits with a status of -1 if opening i**hdrfile** fails.

BUGS

Sun Release 4.1 Last change: 15 March 1990

IHDR2SUN (1) USER COMMANDS

NAME

ihdr2sun - takes an NIST IHead binary raster file and converts it to a Sun rasterfile

SYNOPSIS

ibdr2sun *ihdrfile*

DESCRIPTION

Ibdr2sun converts an NIST IHead binary raster file to a Sun rasterfile.

The Sun image file created will have the root name of **ihdrfile** with the extension **.ras** appended.

OPTIONS

ihdrfile any IHead binary image

EXAMPLES

ibdr2sun r0000_00.pct

FILES

rasterfile.h Sun's raster header include file

ibead.h NIST's raster header include file

SEE ALSO

rasterfile(5)

DIAGNOSTICS

Ibdr2sun exits with a status of -1 if opening **ihdrfile** fails.

BUGS

Sun Release 4.1 **Last change: 15 March 1990**

I

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

FlLES

/ usrl 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

READIHDRFILE (3) C LIBRARY FUNCTIONS READIHDRFILE (3)

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

readihdrfile0 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 win uncompress the data before returning the data buffer. This routine also returns several integers convened 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

Release 4.1 Last change: 15 March 1990

GRP4DECOMP(3) **C LIBRARY FUNCTIONS** **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

in bytes - 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.