Nist Special Database 6

Structured Forms Database 2 Users' Guide

D. L. Dimmick and M. D. Garris

National Institute of Standards and Technology

Advanced Systems Division

Image Recognition Group

September 16, 1992

Users' Guide

Contents

1.0 Introduction

2.0 Image Synthesis

    2.1 Format File

    2.2 Image File Format

3.0 Hand-Print Fonts

4.0 Databse Content and Organization

    4.1 Hiearchy

5.0 Source Code For DatabaseAccess

    5.1 Compilaton

    5.2 Dumpihdr

    5.3 Ihdr2sun

    5.4 Sunalign

6.0 Entry Field Documentation Tables

References

Appendices

1.0 Introduction

This report describes the secondNIST Structured Forms Reference Set database, *NIST Special Database 2* (SD6), 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 digitized from structured forms containing hand-printed data. The structured forms used in SD6 are twelve 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. *Nist Special Database 2* (SD2). [1]

SD6 contains 5,595 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 SD6 appear to be real forms prepared by individuals but the images have been automatically derived and synthesized using a computer and contain no "real" tax data.

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 hand-printed data explained in Section 3.0.. 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 variations in hand-print. 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 12 pixels per millimeter binary. Figure 1 displays a synthesized tax form.

Figure 1 is a 1040 Form and Figure 2 is the format file for Figure 1.

2.1 Answer File Formats

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 referred to as a format file. This format 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. SD6 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. The form faces used in SD6 are contained in Appendix A and are the same as those used in SD2.

Each successive line in the answer file is an entry field identification followed by an entry field value. 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 and are identical to those used in SD2 except for corrections to labeling errors found in the documentation for SD2. The entry field value may be empty or it may contain a computer derived value. Typically, any value listed for an entry field references the precise character information entered into the form image, and empty entry field values model sparsely filled forms. Exceptions exist for ICON entry fields and Continuation Alpha fields.

Entry fields of type ICON contain non-character information such as box check marks and signatures. The presence of this kind of non-character information in an ICON field is represented with an entry field value of 1. If no ICON information is present, then the entry field value is left empty.

Continuation Alpha fields (CA) are used in conjunction with alphanumeric fields (A) to represent a single alphanumeric response that spans multiple entry fields on a form. One example of a CA field is contained in the textual response to question 9b on the Schedule A form. Figure 3 displays a subimage rom a Schedule A form containing question 9b.

Three entry fields exist for question 9b, one numeric field and two text fields. The two text fields located to the left of the numeric field on the form contain a single response, a person's name and address. The first text entry field in labeled SchA_9b_V1 and has an entry field type of A. The second entry field is labeled SchA_9b_H1_V2 and has an entry field type of CA. All the entry field types used in this database are listed in Figure 15.

Figure 4 lists the entry field values stored in the form's format file for the textual response to question 9b. The textual response, *"Paine X. Teton, 57 Kearny Avenue"*, is divided across the two text entry fields. The entry field SchA_9b_V1 contains "*Paine X*," and the entry field SchA_9b_H1_V2 contains the remainder of the response "*Teton, 57 Kearny Avenue*". As can be seen from the entry field values in Figure 4, SchA_9b_V1 contains the entire textual response and SchA_9b_H1_V2 is empty. Also notice that the entry field value for SchA_9b_V1 is separated across two lines. The portion of the entry field value on the first line represents the characters entered in SchA_9b_V1, while the second line represents the characters entered in SchA_9b_H1_V2.

Figures 3 and 4

The first entry field of a multiple line response is considered the primary entry field containing the entire response in its value. The value of the primary entry field is separated across multiple lines consistent with the way the response spans multiple entry fields on the form. Each subsequent line in the primary entry field's value corresponds to the characters in each subsequent CA field. The lines in the primary entry field's value are separated by the unique sequence of a new-line character followed by a tab character, ``\n\t''. Entry field documentation tables, provided as text files in SD6 and included in Appendix C, contain entry field types so that CA entry fields can be identified.

The nearest preceding entry field of type A is a CA field's primary entry field. Using this convention, CA field values are obtained through referencing the primary entry field's value, and the values to the left of CA field identifiers remain empty in the format file. It is possible for a multiple line response to span less than the total number of entry fields available to contain that response. CA fields obtain their values sequentially from the primary entry field's value until all the lines of the response have been assigned, at which point, subsequent CA field values are left blank.

The exceptional use of CA fields in the format file are the result of historical design decisions. It is anticipated that their use in future databases will be modified to be consistent with the format of the other non-ICON entry field types.

Back to Contents

2.2 Image File format

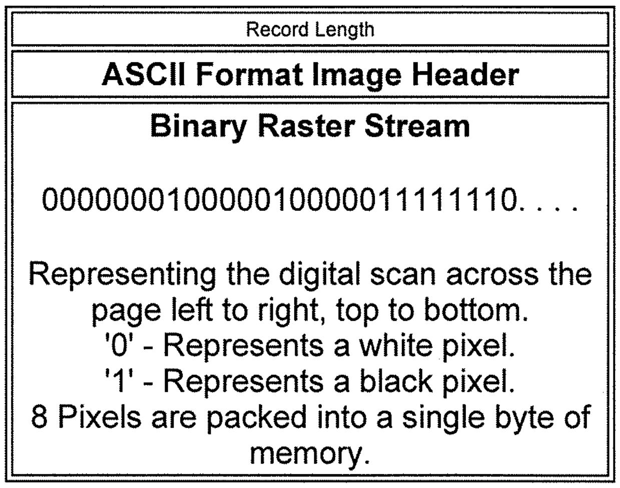
Image file formats and effective data compression and decompression are critical to the usefulness of image archives. Each of a completed form face was synthesized at 12 dots per millimeter binary, 2-dimensionally compressed using CCITT Group 4 [2][3], and temporarily archived onto computer magnetic 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 of 0 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 5 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 SD6. 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 5. The IHead structure definition as written in the 'C' programming language is listed in Figure 6.



Representing the digital scan across the page left to right, top to bottom.
'0' - Represents a white pixel.
'1' - Represents a black pixel.
8 Pixels are packed into a single byte of memory.

FIG. 5. An illustration of the IHead raster file format.

```
/***************************************************************************
        File Name: IHead.h
        Package:  NIST Internal Image Header
        Author:    Michael D. Garris
        Date:      2/08/90
***************************************************************************/
/* Defines used by the ihead structure                */
#define                                   /* len of hdr record (always even
IHDR_SIZE        288                      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]                       /* idenfification/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 1r_r1;                            /* 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 6. IHead C language definition.

Figure 7 lists the header values from an IHead file corresponding to the structure members listed in Figure 6. This header information belongs to the isolated box image displayed in Figure 8. Referencing the structure members listed in Figure 6, 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.

IMAGE FILE HEADER

Identity	: box_03.pct "0123456789"
Header Size	: 288 (bytes)
Date Created	: Thu Jan 4 17:34:21 1990
Width	: 656 (pixels)
Height	:135 (pixels)
Bits per Pixel	: 1
Resolution	: 300 (ppi)
Compression	: 2 (code)
Compress Length	: 874 (bytes)
Scan Alignment	: 16 (bits)
Image Data Unit	: 16 (bits)
Byte Order	: High-Low
MSBit	: First
Column Offset	: 0 (pixels)
White Pixel	: 0
Data Units	: Unsigned
Scan Order	: Row Major,
	: Top to Bottom
	: Left to Right
Parent	: hsf_0/f0000_14/f0000_14.pct
X Origin	:: 192 (pixels
Y Origin	: 732 (pixels)

Figure 7. The IHead values for the isolated subimage displayed in Figure 8.

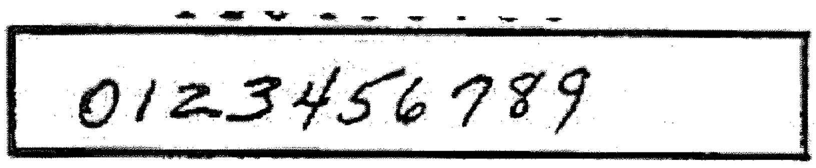


Figure 8. An IHead image of an isolated box.

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 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, 300 dots per inch. The next two fields deal with compression.

Back to Contents

In the IHead format, images may be compressed with virtually any algorithm. The IHead data 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 7 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 10.1 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 7 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 7, the **parent** field contains the full path name to the image from which the image displayed in Figure 8 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.

Back to Contents

3.0 Data Base Content and Organization

In order to provide a realistic sample of structured forms containing hand-print, it was desirable to use digitized hand-print from a large number of writers. For this database, the segmented character images from the 2100 writers found in *NIST Special Database 3* [4] were used. A goal in SD6 was to provide the appearance of a sole authorship on each form. One approach to this problem would be to use characters solely from a single writer when synthesizing a form. The characters in SD3 were extracted from the form images found in *NIST Special Database 1* (SD1) [5]. Each writer printed 13 unique instances of each digit `0' and only one unique instance of each upper and lower case alphabetic character `A' through `Z'. This results in too few examples of each character per writer to accurately reflect the variations naturally occurring in digitized hand-print.

An alternate approach was developed to more realistically model hand-print variations. SD3 was analyzed and divided into hand-print similarity across multiple writers. Four similarity measures were used to group the hand-printed images in SD3. These were pixel height, pixel width, slant and pixel density. Pixel height is the number of pixel rows between the highest black pixel and the lowest black pixel on a character in it image. Pixel width is the number of pixel columns between the left-most black pixel and right-most black pixel on a character in its image. Slant is a computed quantifier representing the amount of horizontal shearing required to make a character in its image vertical. Pixel density is computed by dividing the number of black pixels contained in the character image by the result of multiplying pixel height and pixel width.

These measures were used to divide the character images into sixteen independent groups. Four similarity values were computed for the images of each class of character, for example all the zeros in SD3, creating four distributions for each class. Median values were calculated from each of the resulting distributions. The four similarity values for each character image were compared to the median values of the four measurement distributions associated with the character's class. If the similarity value was greater than the corresponding median value, the character image was recorded as belonging to bin 1. If the similarity value was less than the corresponding median value, the character image was recorded as belonging to bin 0. This resulted in four binary identifiers being recorded for each character image, one identifier per similarity measure. Combining the four identifiers in sequential order, created sixteen unique binary sequences. For this database, all the character images having the same unique binary sequence became one of sixteen possible hand-print fonts. These hand-print fonts were used to synthesize the structured forms in SD6.

Figures 9 through 12 provide the reader with examples of the range of hand-print found on the structured forms in SD6. Figure 9 contains an address synthesized with the hand-print font containing the characters for which all four similarity values were below the median values of their respective measurement distributions.

Figures 9 through 12

Back to Contents

4.0 Database Content and Organization

NIST Special Database 6 contains 5,595 full page images of completed structured forms and correspondingly contains 5,595 ASCII text format files. SD6 is approximately 630 Megabytes in size and is distributed on an ISO-9669 formatted CD-ROM disc. The binary images in the database have been 2-dimensionally compressed. Uncompressed SD6 would require approximately 5.95 Gigabytes of storage.

4.1 Hierarchy

Figure 13 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 5.0. 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 14.

Figures 13 and 14.

Figure 14. The file organization of the form images and format files contained in *NIST Special Database 6*.

There are 5,595 full-page images of completed forms distributed across 8 subdirectories within **data**. The subdirectories **sfrs2_0**, **sfrs2_1**, through **sfrs2_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 SD6. Each submission is represented as a directory. An example of a submission directory **r0200** through **r0299**. The images associated with submission 200 are stored in the subdirectory **r0200**. There are on average 6.22 form images stored in a submission directory. In Figure 14, **r0200** contains 6 synthesized form faces stored as the files **r0200_00.pct**, **r0200_01.pct**, through **r0200_05.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 format file for the image **r0200_00.pct** is **r0200_00.fmt**, **r0200_01.pct** is **r0200_01.fmt**, and so on. In this way 5,595 form images are stored on the CD with their 5,595 corresponding format files accounting for 11,190 individual files in all.

Back to Contents

5.0 Source Code For Data Base Access

In addition to images and format files, SD6 contains documentation and software written in the 'C' programming language and developed on UNIX scientific workstations.. 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**.

5.1 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
```

5.2 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 7 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**.

5.3 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 IHead 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**.

5.4 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. 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 convened using **ihdr2sun**. The main routine for this program is found in the file **sunalign.c**.

Back to Contents

6.0 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, one for each of the twenty different form faces found in SD6. 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 15 and 16 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	Floating Point Fields
I	Integer Fields

ICON	Non-Character Fields (box markings, signatures)

FIG. 15. The set of possible entry field data types.

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

FIG. 16. The set of possible entry field contexts.

References

[1] D.L. Dimmick, M. D. Garris, and C. L. Wilson, Structured Forms Database, Technical Report Special Database 2, SFRS, National Instituttte of Standards and Technology, December 1991.

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

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

[4] M. D. Garris and R. A. Wilkinson, Handwritten segmented characters database. Technical Report Special Database 3, HWSC, National Institute of Standards and technology, February 1992.

[5] C. L. Wilson and M. D. Garris, Handprinted character database, National Institute of Standards and Technology, Special Database 1, HWDB, April 18, 1990.

Appendices

Back to Contents