NIST Special Database 19
Handprinted Forms and Characters Database

Patrick J Grother
Kayee K Hanaoka
Image Group
Information Access Division
National Institute of Standards and Technology

patrick.grother@nist.gov
kayee.hanaoka@nist.gov

September 13, 2016

Contents

1 Introduction

The Special Database 19 2nd Edition(2016) (SD19 2nd(2016)) contains 4.95 millions Portable Network Graphics (PNG) images.

The Special Database 19 (SD19 1st) contains the full page binary images of 3699 Handwriting Sample Forms (HSFs) and 814255 segmented handprinted digit and alphabetic characters from those forms. This document details the origin, publication history, organization, and use of the HSF derived images. The final database has been mastered and replicated onto ISO-9660 formatted CD-ROM discs for permanent archiving and distribution in 1995. The database can be download at https://www.nist.gov/srd/nist-special-database-19. Those segmented characters each occupy a 128x128 pixel raster and are labelled by one of 62 classes corresponding to "0"-"9", "A"-"Z" and "a"-"z". The classes of each segmentation have been manually checked such that the residual character classification error is about 0.1%.

Section 2 shows that although some of the material has been published previously (about 60% of the forms and 50% of the characters) the Special Database 19 CD represents NIST's most comprehensive and probably final release of class referenced images for handprinted optical character recognition. It contains all the data of Special Databases 3 and 7 which it supersedes. Further, SD19 almost replaces Special Database 1 which contains images of just the isolated box fields from the HSF pages. Section 3 quantifies the breadth of this and previous CD releases. The segmented character images are included in multiple organizations suited to different recognition applications. The characters are given by writer, by class, by caseless class, and by field origin. These data hierarchies are described in section 4, suggestions and tips for their use are given in section 5 and the various file formats are defined in section 6.

2 Source Materials

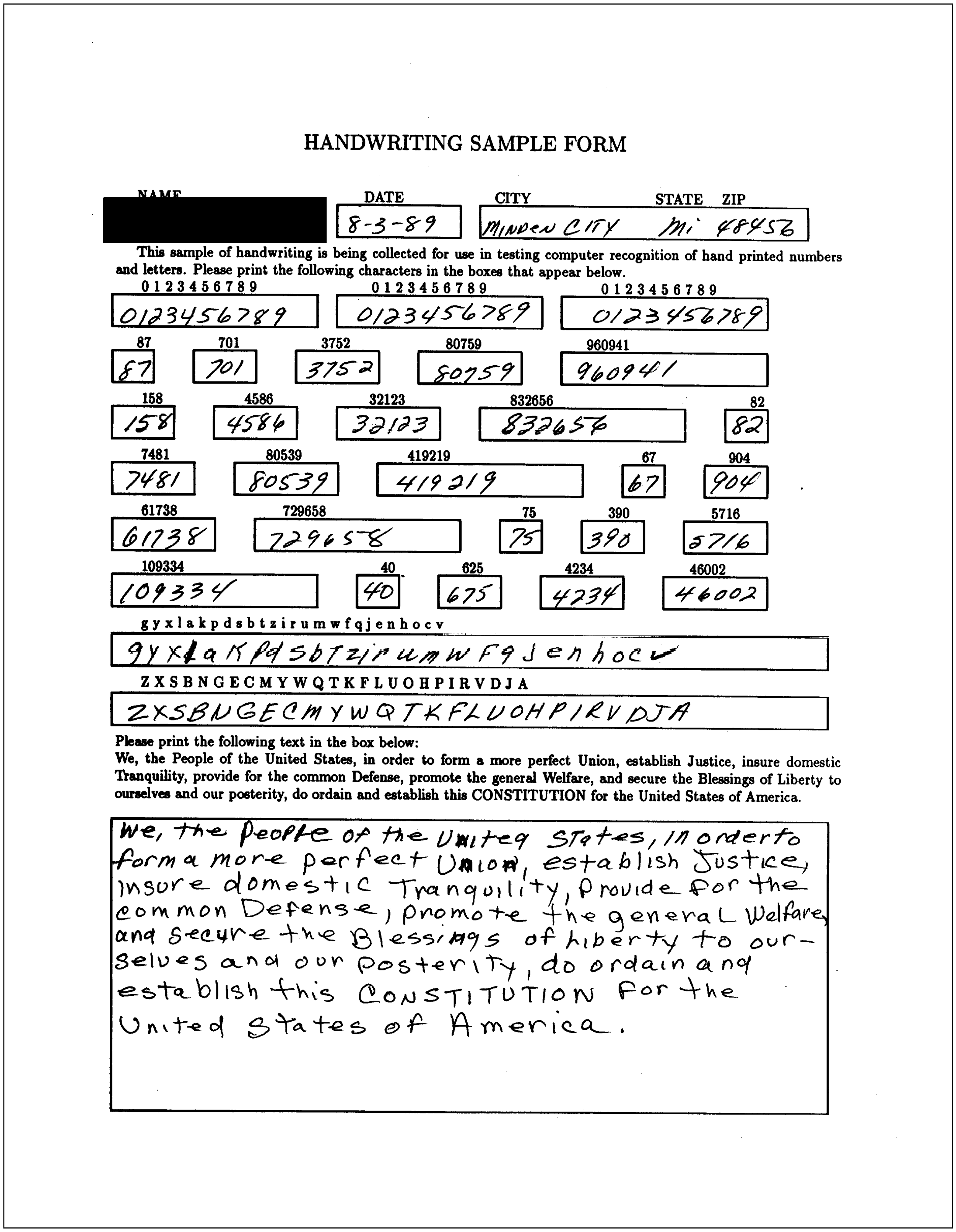
This database contains eight series of HSF images, denoted by *hsf_{0,1,2,3,4,6,7,8}*. The HSFs were scanned at 11.8 dots per millimeter (300 dots per inch) and contain 34 fields: name and date entries, a city/state field, 28 digit fields, one upper-case field, one lower-case field, and an unconstrained Constitution text paragraph. An example of a completed HSF page is given in figure 1. Characters segmented from all field types are included in this database. All images are binary and are stored in NIST's IHEAD format.

Name	Date	Release	Contents	Notes
Special Database 1	May 1990	HWDB Release 1-1.1	Forms and fields	
Special Database 3	February 1992	HWSC Release 4-1.1	Characters	
Special Database 7	April 1992	TST1 Release 6-1.1	Characters	aka Test Data 1.

Table 1: Previous NIST Handprint Databases.

Special Database 1 [1] contains the isolated field images of the 2100 HSF pages of the *hsf_{0,1,2,3}* partitions, but it omits the segmented characters which first appeared on Special Database 3 [2]. That database is composed of the classed and checked segmentations of the digit, upper and lower case fields, but not the Constitution text. Special Database 3 was released as the official training materials for the First Census OCR Systems Conference (see section 5). The writers of the SD3 partitions were Census Bureau field personnel stationed throughout the United States. Those segmented characters were followed by the unclassed characters of the *hsf_4* partition which were released as the testing materials for the conference as Special Database 7 [3] CD. Those images were obtained from a further 500 HSF forms completed by high school students in Bethesda, Maryland. The handchecked classes of those testing images were subsequently released on floppy disc.

This CD republishes the 2100 HSF images already released on Special Database 1 and the character images of Special Databases 3 and 7. It releases, for the first time, the 500 parent HSF pages of the Special Database 7 characters, the HSF forms and the segmentations of a further 1069 writers, the Constitution box characters of



HANDWRITING SAMPLE FORM

NAME	DATE	CITY	STATE	ZIP
[redacted]	8-3-89	MINDEN CITY	Mi	48456

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9		
0123456789	0123456789	0123456789		
87	701	3752	80759	960941
87	701	3752	80759	960941
158	4586	32123	832656	82
158	4586	32123	832656	82
7481	80539	419219	67	904
7481	80539	419219	67	904
61738	729658	75	390	5716
61738	729658	75	390	5716
109334	40	625	4234	46002
109334	40	675	4234	46002

g y x l a k p d s b t z i r u m w f q j e n h o c v

gyxlaKpdsbTzirumwFqJenhocv

Z X S B N G E C M Y W Q T K F L U O H P I R V D J A

ZXSBNGECMYWQTKFLUOHPIRVDJA

Please print the following text in the box below:
We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

We, the People of the United States, In orderto
form a more perfect Union, establish Justice,
insure domestic Tranquility, Provide for the
common Defense, promote the general Welfare,
and Secure the Blessings of Liberty to our-
selves and our posterity, do ordain and
establish this CONSTITUTION for the
United States of America.

Figure 1: Example HSF Image. This is the file *hsf_page/hsf_0/f0002_01.pct*. Notice that the first field on this form, the name field, has been intentionally occluded, on some others it remains blank. All fields except those on the first line have been segmented and recognized by NIST.

partition	writers	forms	digits	uppers	lowers	const	writer origin
hsf_0	0000-0499	SD1	SD3	SD3	SD3	SD19	Census Field
hsf_1	0500-0999	SD1	SD3	SD3	SD3	SD19	Census Field
hsf_2	1000-1499	SD1	SD3	SD3	SD3	SD19	Census Field
hsf_3	1500-2099	SD1	SD3	SD3	SD3	†	Census Field
hsf_4	2100-2599	SD19	SD7	SD7	SD7	ø	High School
hsf_5	2600-3099	retained by NIST for future tests				ø	High School
hsf_6	3100-3599	SD19	SD19	SD19	SD19	ø	Census MD
hsf_7	3600-4099	SD19	SD19	SD19	SD19	ø	Census MD
hsf_8	4100-4169	SD19	†	†	†	†	Census MD

Table 2: First publications of the various partitions. SD3, SD7 are Special Databases 3 and 7 released as as the training and testing materials for the First Census OCR Systems Conference. A † indicates that those fields were completed but not processed at the time of the CD release. A ø indicates that the field was never filled out.

the first 1500 writers of the original Special Database 1, and the writer identities of of the Special Database 3 and 7 characters.

The publication statuses of the various writer partitions and field types are given in the table 2. The first 429 writers of *hsf_5* were completed by the Bethesda high school students while the final 71 forms, and all those of partitions *hsf_*{*6,7,8*} were obtained from Census Bureau employees in Suitland, Maryland.

3 Special Database 19 Contents

The universal descripton of a writer on this database is of the form fyyyy_zz. For example the directory f3100_45 uniquely identifies writer 3100 and the second index, 45, identifies which of the 100 possible form templates was used in the form image. The number, size, and location of the fields is the same across the variations, the only change being a different random ordering of characters intended for the digit, upper and lower case fields. The two digit index can be used to access the form's corresponding reference file in the directory *hsf_pages/truerefs*. This file gives the text that the writer was instructed to print in each field. These files are useful in aligning recognized characters with their putatively true classes. Such a process is used by the NIST Scoring Package [4]. The reference file for the image in figure **??** is ref_01 and is shown in figure 3. Similarly the directory *hsf_pages/template* contains TeX and Postscript files of each unfilled HSF variation, again labelled by the two digit index, that could be used for further data collection.

Field 0 was occluded or left incomplete on all HSF images. Fields 1 and 2 have never been processed by NIST, fields 3 through 30 hold only digits, field 31 is the sole lower case character box, field 32 is likewise the only upper case box, and field 33 contains the free format text of the U.S. Constitution preamble.

The fields of each form were segmented into isolated characters. Each resulting image was automatically assigned a classification. All the referenced character images have been manually checked, with an image's class being changed upon detection of an error. This checking [5] process was performed twice, each time by a different person. In order to retain examples close to class boundaries the philosophy employed in the checking process was to not discard or correct anything that could plausibly have been the intended class even if, in hindsight, it looked more like an example of another class. The resulting segmented images were then organized by writer then by one of four field types: digits, upper and lower case alphabetics, and the restricted alphabetic set from the Constitution text. Directories and files in the SD19 data hierarchies referring to those fields are denoted variously with the names *digit, upper, lower, const* and the prefixes *"d"*, *"u"*, *"l"* and *"c"*. The HSF writer identification scheme is carried over to these segmentations. Thus the resulting images are identified by one of seven *hsf_?* partitions, the writer index and, for good measure, the template id.

```
fld_0
fld_1
fld_2
fld_3 0123456789
fld_4 0123456789
fld_5 0123456789
fld_6 87
fld_7 701
fld_8 3752
fld_9 80759
fld_10 960941
fld_11 158
fld_12 4586
fld_13 32123
fld_14 832656
fld_15 82
fld_16 7481
fld_17 80539
fld_18 419219
fld_19 67
fld_20 904
fld_21 61738
fld_22 729658
fld_23 75
fld_24 390
fld_25 5716
fld_26 109334
fld_27 40
fld_28 625
fld_29 4234
fld_30 46002
fld_31 gyxlakpdsbtzirumwfqjenhocv
fld_32 ZXSBNGECMYWQTKFLUOHPIRVDJA
fld_33
```

Figure 2: Reference file *hsf_page/truerefs/ref_01.txt*.

digit	lower	upper	constitution	
			upper	lower
0	a	A	A	a
1	b	B	B	b
2	c	C	C	
3	d	D	D	d
4	e	E	E	e
5	f	F	F	
6	g	G		g
7	h	H	H	h
8	i	I	I	
9	j	J	J	
	k	K		
	l	L	L	l
	m	M	M	
	n	N	N	n
	o	O	O	
	p	P	P	
	q	Q		q
	r	R	R	r
	s	S	S	
	t	T	T	t
	u	U	U	
	v	V	V	
	w	W	W	
	x	X		
	y	Y	Y	
	z	Z		

digit	lower	upper	constitution	
			upper	lower
40363	3210	3033	4436	8467
44704	3136	3026	1500	2876
40072	3286	3344	8489	
41112	3111	3020	2321	8749
39154	3084	2892	2893	25639
36606	2961	3053	7569	
39937	2966	2964		1310
41893	3253	2925	748	6964
39579	3152	4490	9504	
39533	2213	2958	1430	
	2957	2850		
	4454	3375	2511	13399
	3109	3077	7410	
	3150	3128	6460	10166
	3215	3176	25963	
	2816	3127	6617	
	2648	3018		851
	3113	3061	2821	13312
	3136	3129	21143	
	3058	2981	8415	18169
	3312	3143	11461	
	3378	3237	2196	
	3164	3122	2379	
	3292	3203		
	2746	2966	2575	
	3176	3165		

Table 3: Classes present and their abundancies by field. Blank Constitution field entries indicate that the class is not present in the segmentations of that text, either because it does not occur in the text, because no examples survived segmentation and checking, or because its upper or lower case counterpart is equivalent. Note large differences in the occurrence frequencies.

The intended numbers of each class in the digit, upper, and lower case fields are approximately equal, although the final segmented and checked characters are less evenly distributed. Table 3 shows that the text of the Constitution box does not contain any examples of certain classes, and indeed that the occurrence frequencies of the characters that do appear are widely variant, though similar to the frequency distribution of English text.

Partition	SD1 1990	SD3 (*SD7) 1992				SD19 1995				
		Number writers with at least one character successfully segmented and checked					Number writers with at least one character successfully segmented and checked			
	Forms	Digit	Upper	Lower	Const	Forms	Digit	Upper	Lower	Const
hsf_0	500	500	488	490	†	500	500	488	490	500
hsf_1	500	498	486	483	†	500	498	486	483	499
hsf_2	500	497	493	495	†	500	497	493	495	500
hsf_3	600	586	577	572	†	600	586	577	572	†
hsf_4		500*	500*	500*	ø	500	500	500	500	ø
hsf_5						500	500	500	500	ø
hsf_6						499	499	499	498	ø
hsf_7						500	500	500	499	ø
hsf_8						70	†	†	†	ø

Table 4: Writer abundancies. Note that hsf_5 is not included, and has been held in reserve for future use. A † indicates that those fields were completed but not processed at the time of the CD release. A ø indicates that the field was never filled out.

	digits	uppers	lowers	const	total
hsf_0	53449	10790	10968	99208	174415
hsf_1	53312	10662	10784	87965	162723
hsf_2	52467	10863	10883	61570	135783
hsf_3	63896	12636	12678	†	89210
hsf_4	58646	11941	12000	ø	82587
hsf_6	61094	12479	12205	ø	85778
hsf_7	60089	12092	11578	ø	83759
total	402953	81463	81096	248743	814255
hsf_5	59071	12399	12139	ø	83609
hsf_8	†	†	†	ø	†

Table 5: Group abundancies. Note that hsf_5 is not included, and has been held in reserve for future use. A † indicates that those fields were completed but not processed at the time of the CD release. A ø indicates that the field was never filled out.

0	**1**	2	3	4	**5**	6	7	8	9
40363	44704	40072	41112	39154	36606	39937	41893	39579	39533
A	B	C	D	E	F	G	H	I	J
7469	4526	11833	5341	5785	10622	2964	3673	13994	4388
K	L	M	N	**O**	P	Q	R	S	T
2850	5886	10487	9588	29139	9744	3018	5882	24272	11396
U	V	W	X	Y	Z				
14604	5433	5501	3203	5541	3165				
a	b	c	d	**e**	f	g	h	i	**j**
11677	6012	3286	11860	28723	2961	4276	10217	3152	2213
k	l	m	n	o	p	q	r	s	t
2957	17853	3109	13316	3215	2816	3499	16425	3136	21227
u	v	w	x	y	z				
3312	3378	3164	3292	2746	3176				

Table 6: Class abundancies totalled over all partitions and all fields. Bold type indicates the classes with maximum and minimum representation.

4 Data Hierarchies

There are five directories in the *data* subtree. The first *hsf_page* contains the full page HSF images. The other four directories, *by_**, each have alternative organizations of the segmented character images suited to different recognition applications. The characters are given by writer, by class, by field origin, and finally by caseless class.

The file formats are discussed in detail in a section 6. For the purposes of this section the user need only know that the following file extensions correspond to particular files, thus:

.mis – a file containing multiple isolated character images.

.pct – a file containing a full page HSF image.

.cls – a file containing the checked classes of the images held in the accompanying *.mis* file.

.mit – where appropriate, a file containing a pointer to the source misfile. It contains the writer identity and exact location (pathname and index offset) to the original segmentations of that writer for the characters held in the accompanying *.mis* file.

4.1 *hsf_page* Full HSF Page Images

This tree contains the unprocessed images of the HSF forms in the *hsf_{0,1,2,3,4,6,7,8}* directories. The *truerefs* directory holds the text reference files for the 100 form types. These latter files are necessary for the automated scoring of any package which recognizes HSF images (see section 7). A sample HSF form is given in figure **??** and its reference file is on page 5. All such images are in the *hsf_** subtrees have IHEAD format, are named with the writer and template indices, and carry the *.pct* extension. The *template* directory contains postscript (*.ps*) and LaTeXfiles (*.tex*) for the unfilled HSF forms, which may be reproduced for further data collection. The file hierarchy is given below.

```
hsf_page
|
|
hsf_0 ... hsf_8                  truerefs                 template
|                                |                        |
|                                |                        |
f0000_14.pct ... f0499_10.pct    ref_00.txt ... ref_99.txt ref_00.ps  ... ref_99.ps
                                                          ref_00.tex ... ref_99.tex
```

4.2 *by_write* – By Author

This tree contains the segmented characters organized by *hsf_?* partition then by writer. This organization is generally not particularly useful for OCR studies since the image files contain multiple classes. The files are, however, the primary output of the segmentation and checking process, and the other hierarchies that follow were derived from it.

Each writer directory contains files for each field type; digit, upper, lower and, where present (see table 3), constitution alphas. All images are in the *.mis* format and are accompanied by a *.cls* class reference file. No *.mit* files are included as they would redundantly reference themselves.

```
by_write
```

```
|
|
hsf_0    ...    hsf_7
|
|
f0000_14  f0001_41 ... f0499_10
|
|
d0000_14.mis u0000_14.mis l0000_14.mis c0000_14.mis
d0000_14.cls u0000_14.cls l0000_14.cls c0000_14.cls
```

4.3 *by_field* – By Field Type

This hierarchy contains characters organized by *hsf_?* partition then by field, and finally by class. Writer information is discarded though the files' entries are included by concatenation of the writer characters from the *by_write* tree. The digit files contained here are identical to those contained in the *by_class* tree since digits are not found in other fields. This is not true for upper and lower case alphabetics, since they may generally be found in the Constitution text as well.

```
by_field
|
|
hsf_0    ...    hsf_7
|
|
digit  upper  lower  const
|
|
30.mis ... 39.mis
30.cls ... 39.cls
30.mit ... 39.mit
```

4.4 *by_class* – By Hexadecimal Class

The tree contains images organized by class, then by database. Both writer and field information are discarded: there is no distinction between an "e" from the constitution box of writer 0000 and one from the lower case field of writer 4044. This hierarchy contains relatively few and large files, and is of primary interest to class separability and class recognition studies. Note that in the directory structures that follow the second layer directories have labels which are the hexadecimal ASCII representations of the textual class labels. This nomenclature is described in section 6.3.

```
by_class
|
|
30 ... 39 41 ... 5a 61 ... 7a
|
|
hsf_0.mis ... hsf_7.mis train_30.mis
hsf_0.cls ... hsf_7.cls train_30.cls
hsf_0.mit ... hsf_7.mit train_30.mit
```

The file *train_30.mis* contains the "0"s of all writers of partitions *hsf_{0,1,2,3,6,7}*. For reasons given in section 5 the *train_??* files comprise the suggested training set for OCR studies. The *hsf_4* is likewise earmarked as a standard testing results reporting set. Note that the class files are redundant in this tree, since they contain only one unique hexadecimal class string, and the class has already been indicated in the parent directory name. The files are included for completeness and for user programs that may need a *.cls* file for each *.mis* argument.

4.5 *by_merge* – By Merged Classes

The class abundancies, given in table 3, show up to an order of magnitude disparity between classes. This situation may be ameliorated for certain applications by folding the upper and lower case letters of some classes into one another; for instance an upper case "W" is largely equivalent for recognition purposes to its lower case analogue "w". Indeed it could be argued that a classifier could equally be trained to recognize classes of different appearance, "A" and "a" for example, on the basis that, although examples of the two classes may form separate clusters in a representative feature space, some classifiers will still perform well. For this hierarchy the upper and lower case examples of the following thirteen classes have been merged:

C I J K L M O P S U V W X Y Z

The resulting tree hierarchy contains exact replicas of the files of the unmerged classes from the *by_class* tree, and the merged classes labelled by the hexadecimal codes of the upper and lower case labels delimited by a period. The final number of classes is 37. Note that, unlike in the *by_class* hierarchy, the *train_??* files have been omitted. They may be generated using the *concmis* utility described in section 7.1.

```
by_merge
|
|
30 ... 39
   41 42 44 45 46 47 48 4e 51 52 54 61 62 64 65 66 67 68 6e 71 72 74
      43_63 49_69 4a_6a 4b_6b 4c_6c 4d_6d 4f_6f
      50_70 53_73 55_75 56_76 57_77 58_78 59_79 5a_7a
|
|
hsf_0.mis ... hsf_7.mis
hsf_0.cls ... hsf_7.cls
hsf_0.mit ... hsf_7.mit
```

5 Using Special Database 19

5.1 Standard Training and Testing Sets

The First Census OCR Systems Conference discussed the performance of 45 OCR systems submitted by 26 academic and industrial organizations. NIST supplied the isolated images of the *hsf_4* partition to the participants and, after a two week processing period, scored the submitted hypotheses against the known classes. The results were published as [6].

The participants were free to use whatever training images they had available. NIST supplied a large set of training materials, the digit, upper and lower images of partitions *hsf_{0,1,2,3}*, and typically entrants to the conference used these as a supplement to in-house character databases. The general conclusion of the conference was that the testing images of *hsf_4* are more difficult, in a recognition sense, than the images of *hsf_{0,1,2,3}*. This was subsequently demonstrated in a cross validation study [7].

For that reason it is **strongly suggested that** *hsf_4* **be used only as standard OCR reporting set**, so that recognition results will then be meaningfully comparable to the performances made public in the Census Conference report. The user's attention is drawn to the NIST Special Software Scoring Package [8, 9, 10]. Given that restriction, NIST suggests the *by_class/??/train_??.mis* files for training, one for each of the 62 classes. They are the concatentation of characters from all partitions except *hsf_4*. For example the 5420 class "E" images in *by_class/45/hsf_{0,1,2,3,6,7}.mis* are all joined to make the file *by_class/45/train_45.mis*, where "45" is the hexadecimal ASCII representation of class "E".

5.2 Caveats

1. The *by_** partitions contain different orderings of the same 814255 segmented images, and generally simultaneous use of two hierarchies will give redundant replication.
2. The *by_{class,merge,field}* trees contain files made by concatentation of like entries from the primary *by_write* tree. If the leading M entries of such an N member file are used, then, when $M \ll N$, the writer population is greatly restricted. This typically reduces diversity, and the probable classification difficulty of the subset. The solutions to this problem in decreasing order of preference, are:
   - To use the entire file, making reported results comparable.
   - To shuffle the contents of the file, possibly randomly, to impart more writer homogeneity to the resulting leading M entries. This can be achieved using the *shuflmis* utility described in section 7.1. Use of subsets of randomly sorted datasets makes the comparison of reported results problematic.
3. Many operating systems and shells possess upper limits on the number of files that may be handled. Users of the *by_write* and *hsf_page* trees may experience difficulties in the following regards:
   - Wildcard expansions can be time-consuming. Many operating systems and shells have upper limits on the number of arguments that may be supplied on the command line to installed programs.
   - UNIX filesystems require an "inode" for each file. If the largest tree of this database is copied to a filesystem then that disk must be configured to have enough inodes. Similar issues affect DOS-based systems. See UNIX's *df -i*.
   - Special Database 19 holds in excess of 600Mb of data. The destination drive of any copying operations must be large enough.
4. If a user possesses previous NIST OCR Special Databases and uses the supplied CCITT Group 4 compression code then it should be updated with the SD 19 release. This modified code allows large *.mis* files to be handled. Use of the old compression version on these files **will result in the loss of all the data**.

6 NIST Data Formats

6.1 Image files

NIST has developed the IHEAD image header and has incorporated it into all images of all the Special Databases. Its design and functionality are described extensively in [11]. All members of the header are ASCII coded with some fields being in 'C' string format and other fields being single ASCII characters. The actual structure definition for an NIST header and a description of the use and purpose of each member is shown in Appendix A. The header is of a fixed allocated length, currently 296 bytes, regardless of its content. The first eight bytes of the header contain the length of the remainder, i.e. 288 bytes all of which can be read into structure memory with a single "fread" system call. Also note the eight character ASCII header length record stored in front of the actual header in an image file.

All the SD19 images have been compressed using an implementation of the CCITT Group 4 algorithm developed by the CALS Test Network and adapted by NIST for use with files which have compressed images and uncompressed headers. The IHEAD header within each image (described in Appendix A) remains uncompressed with the appropriate members set to reflect the fact that the following raster data has been compressed before being saved to a file. The file *src/lib/image/readrast.c* contains the C routine *ReadBinaryRaster* that when given an IHEAD image file will allocate, load, and return the header structure with the decompressed raster data. Although an IHEAD file may contain uncompressed data, the reading routine will invoke, if necessary, routines to decompress the image data. The files *src/lib/image/grp4{comp,deco}.c* contain modified source code for compressing and decompressing binary rasters. If previous distributions of this code are used to compress large *.mis* files **data will be lost** and the user will not automatically be notified of this.

The character files with the *.mis* extension are read using the *readmisfile* routine in *src/lib/mis/readmis.c* which calls the underlying *ReadBinaryRaster* routine.

6.2 Character image files

All character images contained on this CD are held in the NIST developed Multiple Image Set (MIS) format. An MIS file allows multiple homogeneous images to be stored together; one or more images are stacked vertically as a contiguous raster, the last scanline of one image is followed by the first scanline of the another. Individual characters are MIS entries and the contiguous raster collection is the MIS memory. The MIS memory contains entries of equal width, height, and depth (precision) and is stored using the IHEAD header format. The IHEAD structure's *width* attribute holds the width of the MIS memory, and likewise the structure's *height* entry gives the height of the MIS memory. In this way, the MIS collection can be compressed, stored and manipulated just like a single image. Given fixed dimensions for all entries, the IHEAD structure's *width* attribute also specifies the width of the entries, but the IHEAD definition lacks a facility to hold the height and number of entries. So we let the interpretation of two of the IHEAD attribute fields, *par_x* and *par_y*, change when the IHEAD header is being used to describe a MIS memory. The *par_x* and *par_y* fields are used to hold the width of and height of the MIS entries, whence the number of entries can be calculated by dividing the entry height into the total MIS memory height.

The MIS structure definition written in 'C' is given below. It contains an IHEAD structure, *head*, and the MIS memory, *data*. In addition, there are 6 other attribute fields which hide the details of the IHEAD interpretation from application programs that manipulate MIS memories.

```
typedef struct misstruct{
    IHEAD *head;
    unsigned char *data;
    int misw;
    int mish;
    int misd;
    int entw;
    int enth;
    int ent_num;
    int ent_alloc;
} MIS;
```

The attributes *misw* and *mish* give the width and height of the MIS memory. They are also contained in the IHEAD structure pointed to by *head*. The attributes *entw* and *enth* specify the width and height of the entries and are the same as the *par_x* and *par_y* attributes of the IHEAD structure. The *ent_alloc* attribute specifies how many MIS entries of dimension *entw* and *enth* have been allocated to the MIS memory, *data*, and *ent_num* gives the number currently used out of the total allocated.

6.3 Character Class Files

For each segmented character image in the database there is an associated character classification which has been assigned and manually checked. Although no punctuation characters are presented on Special Database 19, they have influenced the design of the class labelling. Because operating systems and shells treat punctuation as special characters, we have developed a method to avoid such ambiguities. The classification of each image is just the hexadecimal representation of the character's ASCII value. Thus the digits are labelled "30"-"39", the upper case alphas as "41"-"5a" and the lowers as "61"-"7a", such that a handprinted "2" has the class "32", a lower-case "z" is denoted by "7a".

These classifications are stored in *.cls* files and have one-to-one correspondence to the entries in the associated *.mis* file. The *.cls* file is an ASCII file with the first line containing the integer number of classes listed in the file, one value per successive line. Each ASCII text line is terminated with the linefeed character, 0x0A. All *.mis* files on Special Database 19 have an associated *.cls* file.

6.4 Character Origin Files

Each character image in the various *by_** hierarchies were obtained from one of the 4169 writers. The *by_write* tree contains the characters as they were originally segmented from their particular fields. The other trees contain the same characters in different organizations and the *.mit* files indicate precisely where in the *by_write .mis* files such characters came from. Each *.mit* file contains as many entries as its accompanying *.mis* file and, like the *.cls* counterpart, is an ASCII file with an integer head giving the number of lines that follow. Each entry is a pointer consisting of a pathname to the "parental" *.mis* file and a zero-oriented index into that file. Figure 3 shows nine *.mis* entries, and the associated *.cls* and *.mit* files.

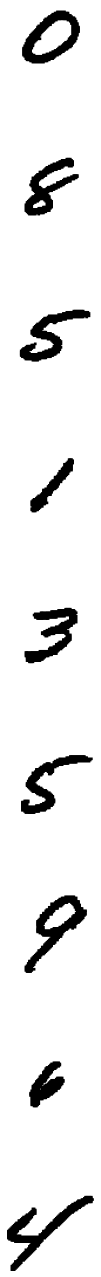
7 Software

The full page HSF images in *hsf_page* are the conventional input to NIST's public domain Standard Reference Optical Character Recognition System [12] released on CD ROM in 1994. Its functionality and availability are given below. Further, Special Database SD19 contains several software utilities for the handling of the isolated character files.

NIST FORM-BASED HANDPRINT RECOGNITION SYSTEM
Michael D. Garris (mdg@magi.ncsl.nist.gov)
James L. Blue, Gerald T. Candela, Darrin L. Dimmick, Jon Geist,
Patrick J. Grother, Stanley A. Janet, and Charles L. Wilson

National Institute of Standards and Technology,
Building 225, Room A216
Gaithersburg, Maryland 20899
Phone: 301 975 2928 FAX: 301 840 1357

The National Institute of Standards and Technology (NIST) has developed a standard reference form-based handprint recognition system for evaluating optical character recognition (OCR). NIST is making this recognition system freely available to the general public on an ISO-9660 format CD-ROM. The recognition system processes the Handwriting Sample Forms distributed with NIST Special Database 1 and NIST Special Database 3. The system reads handprinted fields containing digits, lower case letters, upper case letters, and reads a text paragraph containing the Preamble to the U.S. Constitution.



```
9
30
38
35
31
33
35
39
36
34
```

```
9
f3804_08/d3804_08.mis 0
f3804_08/d3804_08.mis 8
f3804_08/d3804_08.mis 15
f3804_08/d3804_08.mis 21
f3804_08/d3804_08.mis 23
f3804_08/d3804_08.mis 30
f3804_08/d3804_08.mis 33
f3804_08/d3804_08.mis 48
f3804_08/d3804_08.mis 63
```

Figure 3: An example MIS image file. At right are its CLS class and MIT reference files.

This is a source code distribution written primarily in C and is organized into 11 libraries. There are approximately 19,000 lines of code supporting more than 550 subroutines. Source code is provided for form registration, form removal, field isolation, field segmentation, character normalization, feature extraction, character classification, and dictionary-based post-processing.

The NIST standard reference recognition system is designed to run on UNIX workstations and has been successfully compiled and tested on [1] a Digital Equipment Corporation (DEC) Alpha, Hewlett Packard (HP) Model 712/80, IBM RS6000, Silicon Graphics Incorporated (SGI) Indigo 2, SGI Onyx, SGI Challenge, Sun Microsystems (Sun) IPC, Sun SPARCstation 2, Sun 4/470, and a Sun SPARCstation 10. Scripts for installation and compilation on these architectures are provided with this distribution.

A distribution of this standard reference system can be obtained free of charge at https://nist.gov/srd/nist-special-database-19. This system or any portion of this system may be used without restrictions. However, redistribution of this standard reference recognition system is strongly discouraged as any subsequent corrections or updates will be sent to registered recipients only. This software was produced by NIST, an agency of the U.S. Government, and by statute is not subject to copyright in the United States. Recipients of this software assume all responsibilities associated with its operation, modification, and maintenance.

The character images used in the preparation of the Karhunen Loève feature extractor and Probabilistic Neural Network classifier of the standard reference system were obtained from the *hsf_{4,6}* partitions. Four classifiers were used for their respective fields; the digit classifier used just the *.mis* files of the *by_write/hsf_6/f** writers. Similarly the upper and lower case recognizers respectively used the upper and lower case *.mis* files of both the *hsf_4* and *hsf_6* writers. The Constitution text classifier used the characters of these two writer sets amalgamated so that there was no differentiation between upper and lower case. The use of the *hsf_4* partition for training purposes is in contravention of the guidelines suggested in section 5.1.

7.1 Utilities

A number of C coded utilities have been included in this CD firstly to provide tools for handling the provided image data, and secondly to give examples of how to handle the NIST IHEAD and MIS data structures. In addition directories named *bin* and *lib* must be created. Given a copy of these trees in, for example, the directory */usr/local/nist.cd*, the executable binaries for various architectures can be produced in the *bin* directory by invoking the UNIX commands thus:

```
# mkdir /usr/local/nist.cd
# cd /usr/local/nist.cd
# cp -rp /cd/src /cd/include /cd/man /cd/makefile.mak .
# mkdir bin lib
# make -f makefile.mak instarch PROJDIR=/usr/local/nist.cd INSTARCH=sgi
# make -f makefile.mak depend PROJDIR=/usr/local/nist.cd
# make -f makefile.mak install PROJDIR=/usr/local/nist.cd
```

where the environment variable PROJDIR is the root path to the *src* directory, and INSTARCH, which indicates the desired machine architecture, is one of *sun*, *sgi*, *aix*, *hp*, *sol* or *osf*. A brief description of the software utilities is given below. The manual pages that follow offer a more complete description.

concmis This utility produces a single *.mis* file by concatenation of an arbitrary number of *.mis* files given on the command line.

conccls This utility produces a single *.cls* or *.mit* file by concatenation of an arbitrary number of *.cls* or *.mit* files given on the command line.

[1]Specific hardware and software products identified were used in order to adequately support the development of this technology. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment identified is necessarily the best available for the purpose.

dumpihdr This utility dumps the textual information of the header of a NIST ihead file (both *.mis* files and *.pct* files fall in this category) to standard output.

misfrmit This utility builds *.mis* files. The entries in an output file are named in an input *.mit* file, which specifies arbitrary *.mis* files that will contribute and identifies which elements in those files.

mistopng This utility converts and parses .mis files into png image files. The entries in each .mis file are parsed into individual graphic file format - PNG. The output images are typically sized 128 width and 128 height.

normmis This utility performs size and orientation normalization of the images in *.mis* files. The input images are typically held as 128x128 *.mis* entries. The output images are held in 32x32 *.mis* entries. Each image is processed independently. A full description of the algorithms in use is given in [12].

numcls Given a number of *.cls* and *.mit* files, this utility dumps the number of entries in each to standard output.

nummis Given a number of *.mis* files, this utility dumps the number of entries in each to standard output.

showmis Produces a text representation of the binary images held in a *.mis* file using the "#" and "." characters to represent dark ink and white space pixels respectively. The utility is best used on size normalized *.mis* files (typically 32x32 entries obtained using, for example, *normmis*) instead of the raw 128x128 entries which are textually too wide for many output devices.

shuflmis Shuffles the entries in *.mis* files. Optionally the utility will reorder the *.cls* counterparts.

References

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

[2] 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.

[3] R. A. Wilkinson. Handprinted Segmented Characters Database. Technical Report Test Database 1, **TST1**, National Institute of Standards and Technology, April 1992.

[4] Michael D. Garris and Stanley A. Janet. NIST Scoring Package User's Guide Release 1.0. Technical Report NISTIR 4950, National Institute of Standards and Technology, October 1992.

[5] R. Allen Wilkinson, Michael D. Garris, and Jon Geist. Machine-Assisted Human Classification of Segmented Characters for OCR Testing and Training. In D. P. D'Amato, editor, , volume 1906. SPIE, San Jose, 1993.

[6] R. A. Wilkinson, J. Geist, S. Janet, P. J. Grother, C. J. C. Burges, R. Creecy, B. Hammond, J. J. Hull, N. J. Larsen, T. P. Vogl, and C. L. Wilson. The First Optical Character Recognition Systems Conference. Technical Report NISTIR 4912, National Institute of Standards and Technology, August 1992.

[7] Patrick J. Grother. Cross Validation Comparison of NIST OCR Databases. In D. P. D'Amato, editor, , volume 1906. SPIE, San Jose, 1993.

[8] M. D. Garris. NIST Scoring Package Cross-Reference for use with NIST Internal Reports 4950 and 5129. Technical Report NISTIR 5249, National Institute of Standards and Technology, August 1993.

[9] M. D. Garris and S. A. Janet. NIST Scoring Package User's Guide Release 1.0. *NIST Special Software 1*, NIST Internal Report 4950 and CDROM, October 1992.

[10] M. D. Garris. Methods for Evaluating the Performance of Systems Intended to Recognize Characters from Image Data Scanned from Forms. Technical Report NISTIR 5129, National Institute of Standards and Technology, February 1993.

[11] Michael D. Garris. Design, Collection, and Analysis of Handwriting Sample Image Databases. *The Encyclopedia of Computer Science and Technology*, 1994. to be published in 1994.

[12] Michael D. Garris, James L. Blue, Gerald T. Candela, Darrin L. Dimmick, Jon Geist, Patrick J. Grother, Stanley A. Janet, and Charles L. Wilson. NIST Form-Based Handprint Recognition System. Technical Report NISTIR 5469, National Institute of Standards and Technology, July 1994.

NAME

conccls - produces a cls file which is the concatenation of cls files or a mit file which is the concatenation of mit files.

SYNOPSIS

conccls output.cls input1.cls [input2.cls ...] **conccls** output.mit input1.mit [input2.mit ...]

DESCRIPTION

conccls is a utility that combines cls or mit files. It produces one output file which is the concatenation of the entries of the files in the order they were supplied on the command line.

OPTIONS

None.

EXAMPLES

Given the SD19 data hierarchy, the class files associated with the digits of the hsf_0 writers may be gathered into one file using:

```
conccls d_hsf_0.cls by_write/hsf_0/f*/d*.cls
```

FILES

ocr.cd/src/include/mfs.h

SEE ALSO

concmis

DIAGNOSTICS

If the first line of any of the input files is not an integer then **conccls** fails with an appropriate message and a -1 exit status.

BUGS

Many environments provide a maximum number of arguments that may be supplied or pattern matched when a command is invoked.

NAME

concmis - produces a mis file which is the concatenation of mis files.

SYNOPSIS

concmis output.mis input1.mis [input2.mis ...]

DESCRIPTION

concmis is a utility that combines multiple character mis files. It produces one output mis file which is the concatenation of the characters of the files in the order they were supplied on the command line.

OPTIONS

None.

EXAMPLES

Given the SD19 data hierarchy, the digits of the hsf_0 writers may be gathered into one file using:

```
concmis d_hsf_0.mis by_write/hsf_0/f*/d*.mis
```

FILES

ocr.cd/src/include/ihead.h
ocr.cd/src/include/mis.h

SEE ALSO

conccls

DIAGNOSTICS

The input mis files must all be congruent: that is entry precision (bits per pixel), width, and height must be the same for each file.

BUGS

Many environments provide a maximum number of arguments that may be supplied or pattern matched when a command is invoked.

NAME
dumpihdr - prints an image file's header information.

SYNOPSIS
dumpihdr [input1.mis | input2.pct] ...

DESCRIPTION
dumpihdr is a utility that dumps image header information to standard output for multiple NIST IHEAD format image files.

OPTIONS

EXAMPLES
Given the SD19 data hierarchy, the IHEAD structures of the first writer's completed HSF and digit segmentations are written to standard output using:

```
dumpihdr hsf_page/hsf_0/f0000_14.pct by_write/hsf_0/f0000_14/d0000_14.mis
```

FILES
ocr.cd/src/include/ihead.h
ocr.cd/src/include/mis.h

SEE ALSO
showmis

BUGS
Many environments provide a maximum number of arguments that may be supplied or pattern matched when a command is invoked.

NAME

misfrmit - prepares a misfile by extracting mis entries from other misfiles.

SYNOPSIS

misfrmit [-p pathname] mitfiles...

DESCRIPTION

misfrmit this utility makes a misfile by gathering mis entries from arbitrary misfiles. The specific entries are named in a "mitfile", the entries of which point to a source misfile. A mit entry is just the name of the misfile to be scavenged and a zero-oriented index into that file. Each of the input mitfiles, with the standard ".mit" extension, yields one output misfile with the ".mis" extension.

OPTIONS

-p pathto. If the entries in the mitfile do not point directly to the required misfile, and require a pathname to be prepended then this option is followed by the required path. The same effect can be obtained by moving to and then running misfrmit in the appropriate parent directory.

EXAMPLES

Given the SD19 data hierarchy, the first and last digits of the last writer (0499) of the hsf_0 partition may be placed in a single mis file thus: cd sd19.cd/data

```
echo 2 > toy.mit
echo by_write/hsf_0/f0499_10/d0499_10.mis 0 >> toy.mit
echo by_write/hsf_0/f0499_10/d0499_10.mis 109 >> toy.mit
misfrmit toy.mit
showmis toy.mis
```

Alternatively, to be more parsimonious with the use of long pathnames, we can equivalently use the -p option thus:

```
cd sd19.cd/data
echo 2 > toy.mit
echo d0499_10.mis 0 >> toy.mit
echo d0499_10.mis 109 >> toy.mit
misfrmit -p by_write/hsf_0/f0499_10 toy.mit
showmis toy.mis
```

FILES

ocr.cd/src/include/ihead.h
ocr.cd/src/include/mis.h
ocr.cd/src/include/mit.h
ocr.cd/src/include/mfs.h

DIAGNOSTICS

The mit named mis entries must all be congruent: that is entry precision (bits per pixel), width, and height must be the same for each mis entry collected.

BUGS

The utility does not attempt to gather the class entries corresponding to the mis entries which may be useful or necessary.

NAME

mistopng - generate individual png file held in mis file

SYNOPSIS

mistopng input.mis

DESCRIPTION

mistopng is a utility that converts pixels of binary mis entries to graphics file format - PNG.

OPTIONS

EXAMPLES

Given the SD19 data hierarchy, the uppers of the first hsf_0 writer may be displayed using:

```
mistopng   by_write/hsf_0/f0000_14/u0000_14.mis
```

FILES

ocr.cd/src/include/ihead.h
ocr.cd/src/include/mis.h

SEE ALSO

showmis may be used to produce a text representation of the images in mis file.
(nummis) gives the total number of entries to be generate from a ".mis" file.

BUGS

Many environments provide a maximum number of arguments that may be supplied or pattern matched when a command is invoked.

NAME
normmis - normalizes 128x128 mis file entries to 32x32.

SYNOPSIS
normmis [-v] input.lis

DESCRIPTION
normmis this utility applies the size and orientation normalization algorithms of the NIST Public Domain Recognition System to isolated characters. The program takes a text file containing an integer followed by that number of mis filenames. For each ".mis" file supplied in this file a corresponding ".nrm" file is produced.

OPTIONS
-v The verbose flag enables notification after completion of each mis entry.

EXAMPLES
Given the SD19 data hierarchy, the digit misfiles of the first 10 writers of the hsf_0 partition may be normalized thus:

```
echo 10  >  mislist
ls -1 by_write/hsf_0/f000?_??/d*.mis  >>  mislist
normmis mislist
showmis by_write/hsf_0/f0000_14/d*.nrm
```

FILES
ocr.cd/src/include/ihead.h
ocr.cd/src/include/mis.h

SEE ALSO
showmis. normmis is useful in providing a compact representation of raw mis images displayable with the **showmis** utility.

DIAGNOSTICS
The input mis files must all be congruent: that is entry precision (bits per pixel), width, and height must be the same for each file.

BUGS

NAME

numcls - gives the number of entries in a cls or mit file

SYNOPSIS

numcls input1.cls [input2.cls ...] **numcls** input1.mit [input2.mit ...]

DESCRIPTION

numcls is a simple utility that dumps the integer head of multiple cls and mit files to standard output.

OPTIONS

EXAMPLES

Given the SD19 data hierarchy, the number of digits corresponding to the first 10 writers of the hsf_0 partitions may be displayed thus:

```
numcls by_write/hsf_0/f000?_??/d*.cls
```

FILES

ocr.cd/src/include/mfs.h

SEE ALSO

nummis. The output of the above example is identical to that of:

```
nummis by_write/hsf_0/f000?_??/d*.mis
```

BUGS

Many environments provide a maximum number of arguments that may be supplied or pattern matched when a command is invoked.

NAME
nummis - gives the number of entries in a ".mis" file

SYNOPSIS
nummis input1.mis [input2.mis ...]

DESCRIPTION
nummis is a utility that dumps the integer number of entries in mis files to standard output. For multiple files the order in which results are printed depends on the order in which the shell expands wildcard pattern matches.

OPTIONS

EXAMPLES
Given the SD19 data hierarchy, the number of digits corresponding to the first 10 writers of the hsf_0 partitions may be displayed thus:

```
nummis  by_write/hsf_0/f000?_??/d*.mis
```

FILES
ocr.cd/src/include/ihead.h
ocr.cd/src/include/mis.h

SEE ALSO
numcls. The output of the above example is identical to that of:

```
numcls  by_write/hsf_0/f000?_??/d*.cls
```

DIAGNOSTICS

BUGS
Many environments provide a maximum number of arguments that may be supplied or pattern matched when a command is invoked.

NAME
showmis - produces a text representation of the images in mis file.

SYNOPSIS
showmis input.mis

DESCRIPTION
showmis is a utility that uses the text characters "." and "#" to represent the white false background and true dark ink pixels of binary mis entries. It provides a quick method of visualization of mis entries.

OPTIONS

EXAMPLES
Given the SD19 data hierarchy, the uppers of the first hsf_0 writer may be displayed using:

```
showmis by_write/hsf_0/f0000_14/u0000_14.mis
```

FILES
ocr.cd/src/include/ihead.h
ocr.cd/src/include/mis.h

SEE ALSO
normmis may be used to give a more useful display. The mis files of SD19 contain 128x128 square entries which are typically too wide for useful display on most terminals.

DIAGNOSTICS

BUGS
Many environments provide a maximum number of arguments that may be supplied or pattern matched when a command is invoked.

NAME

shuflmis - reorder the entries in .mis files, and optionally their .cls counterparts.

SYNOPSIS

shuflmis [-cmu -s seed] input.mis...

DESCRIPTION

shuflmis randomly reorders the images in .mis files. This has utility in producing homogenous writer distributions in writer ordered .mis files produced by concatenation. The output overwrites the input .mis file. Optionally **shuflmis** will reorder the .cls partners of the .mis arguments, will produce the .mit file that effects the shuffle (from .mis to .shf), will take a seed other than the default, and will produce a .bak file that when used with **misfrmit** will undo the .mis shuffle (from .shf to .mis)

OPTIONS

-c shuffle the corresponding class file, which must the same name as the .mis argument save for a .cls extension.
-m produce the .mit file that effects the shuffle. This is useful if the integer shuffling key is desired.
-u produce the mit file, with extension .bak, that will undo the .mis shuffle, when supplied as an argument to **misfrmit**.
-s seed Use a different integer seed to random number generator than the default which is 1234.

EXAMPLES

Given a copy of SD19 data hierarchy, the uppers of the first hsf_0 writer may be shuffled using:

```
shuflmis -c -u by_write/hsf_0/f0000_14/u0000_14.mis
```

Note that the .cls file is also reordered.

FILES

ocr.cd/src/include/ihead.h
ocr.cd/src/include/mis.h ocr.cd/src/include/mfs.h ocr.cd/src/include/mit.h

SEE ALSO

misfrmit can be used to recover the original .mis file, but only if the .bak mitfile, created with the -u flag to shuflmis , exists.

DIAGNOSTICS

BUGS

The random number generator is re-initialized for each .mis argument. Files with equal numbers of entries will be reordered in an identical manner.

If the -c option was applied to reorder the .cls file then the -u flag provides a mechanism for unshuffling the .mis file, !!! but not the .cls files. !!!

Many environments provide a maximum number of arguments that may be supplied or pattern matched when a command is invoked.

8 Appendix A

```
/* 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 date 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;
```

```
   1. id         - General identification field used to contain the
                       image file name and any other information useful for
                       image distinction.
   2. created    - Date when the image was created.
   3. width      - Pixel width of the image.
   4. height     - Number of scanlines in the image.
   5. depth      - Number of bits representing a single pixel.
   6. density    - Pixels per inch resolution of the image.
   7. compress   - ASCII code used to represent the compression
                       algorithm used on the image.
   8. complen    - Length of compressed image in bytes.
   9. align      - Even multiple in bits to which each scanline is padded.
  10. unitsize   - Size in bits of fundamental data units in the image.
  11. sigbit     - Order of most significant to least significant bits
                       within fundamental data units in the image.
  12. byte_order - Order of high and low bytes used if fundamental
                       data units are 2 bytes long in the image.
  13. pix_offset - Pixel offset into the data where the
                       image of interest actually begins.
```

14.	whitepix	- Intensity value of white pixels in the image.
15.	issigned	- Flag which signifies whether fundamental data units include a sign bit or are unsigned.
16.	rm_cm	- Flag which signifies whether the raster data is stored row-major or column-major.
17.	tb_bt	- Flag which signifies whether the raster data is stored top-to-bottom or bottom-to-top.
18.	lr_rl	- Flag which signifies whether the raster data is stored left-to-right or right-to-left.
19.	parent	- File name of the parent image if the image is a subimage.
20.	par_x	- X coordinate pixel value in the parent image where the subimage was cut.
21.	par_y	- Y coordinate pixel value in the parent image where the subimage was cut.