NISTIR 4950

NIST Scoring Package User's Guide Release 1.0

Michael D. Garris
Stanley A. Janet

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Advanced Systems Division
Image Recognition Group
Gaithersburg, MD 20899

October 1992

U.S. DEPARTMENT OF COMMERCE
Barbara Hackman Franklin, Secretary

TECHNOLOGY ADMINISTRATION
Robert M. White, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director

NISTIR 4950

NIST Scoring Package User's Guide Release 1.0

Michael D. Garris
Stanley A. Janet

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Advanced Systems Division
Image Recognition Group
Gaithersburg, MD 20899

NIST Scoring Package User's Guide

Release 1.0

Michael D. Garris and Stanley A. Janet

National Institute of Standards and Technology
Gaithersburg, MD 20899

Table of Contents

1. Introduction

The increased performance and connectivity of computers has touched off an imaging revolution. One area of application which is benefitting immensely from advancements in imaging technology is automated document processing and automated data entry through the use of optical character recognition. As this technology continues to advance, the number of commercially available products is increasing. Multiple products are emerging, all of which are designed for optical character recognition problems. Improved recognition algorithms have enabled the accuracy of these products to steadily increase, but each product is based on a different, often proprietary, set of algorithms. This presents potential users of optical character recognition technology with many different choices and options and leads to a series of significant questions: How does one determine when the technology has matured enough to make it economically advantageous to deploy? How does a potential user determine which product is best for his or her specific needs? How can a system developer, who has the ability to choose from a large variety of diverse algorithmic approaches, intelligently choose and then track progress when developing optical character recognition systems? The answer to these questions lies in objective system performance measurement. This is the motivation behind the development of the NIST Scoring Package.

Application requirements germane to a specific automated character recognition problem are embodied in a representative set of referenced images. Associated with each reference image is the ASCII textual information that is to be recognized in the image. NIST has produced three referenced image databases of digitized forms which are available to the public and distributed through NIST's Standard Reference Data Division on CD-ROM. *NIST Special Database 1* (SD1)[1] contains 2,100 digitized pages of a hand-print collection form completed by 2,100 different writers geographically distributed across the United States. Each full-page image in the database is a form comprised of 33 entry fields. Each entry field is demarcated by a separate box on the form. These fields include 28 numeric fields totalling 130 hand-printed digits, 1 alphabetic field containing the 26 lower-case letters, 1 alphabetic field containing the 26 upper-case letters, and a text paragraph field containing the first sentence from the Preamble to the Constitution of the United States. *NIST Special Database 2* (SD2)[2] contains 5,590 digitized tax forms from the IRS 1040 Package X for the year 1988 completed with machine-print. These include Forms 1040, 2106, 2441, 4562, and 6251 together with Schedules A, B, C, D, E, F, and SE. *NIST Special Database 6* (SD6)[3] contains 5,595 digitized tax forms from the same list completed with hand-print. The information provided on these images of tax forms has been generated by a computer and does not represent real people or real tax data.

Two other referenced databases are available to the public from NIST. They contain images of isolated characters that are useful for testing in isolation the character classification components of full-scale recognition systems. *NIST Special Database 3* (SD3)[4] contains 313,389 images of segmented characters from the 2,100 writers in SD1. SD3 is comprised of 223,125 digits, 44,951 upper-case letters, and 45,313 lower-case letters. These images have been verified to contain correctly segmented characters and do not include images of split and merge characters. Associated with every character image in this database is a reference value specifying the class of the character in the image. A second character image database, *NIST Special Database 7* (SD7)[5], was intended to be used primarily for testing hand-print character classifiers. SD7 contains hand-print from 500 writers and has approximately 83,000 isolated character images including 59,000 digits and 24,000 upper-case and lower-case letters. Because SD7 was a testing database, the reference classifications for each character image are distributed on floppy disk separately from the character images which are distributed on CD-ROM.

The reference information in these databases serve as ground truth for measuring recognition performance. The images are presented to a recognition system, and the system's results are returned. This includes hypothesized text of what the system located and recognized. The Scoring Package reconciles the hypothesized text with the reference text, accumulating statistics used to compute performance measures. Figure 1 illustrates the use of referenced images and the Scoring Package to assess the performance of a recognition system.

The model in Figure 1 has several advantages. First, knowledge of the internal details of a system being tested is not required. This is critical when testing systems comprised of proprietary functional components. Second, the performance measures are computed in an automated way without any human inspection. This is extremely important when assessing the performance of optical character recognition (OCR) technology, especially large-scale character recognition systems. An example is the NIST massively parallel model recognition system whose character recognition component is capable of classifying character images up to 1000 characters per second[6]. This system is capable of processing 2,100 pages of forms containing 130 hand-printed digits per form for a total of 273,000 digits in approximately 4 hours. The visual inspection of the system output from a single 4 hour processing session took a technician 6 months. In order to conduct tests in a reasonable amount of time, the compiling and computing of performance measures must be automated.

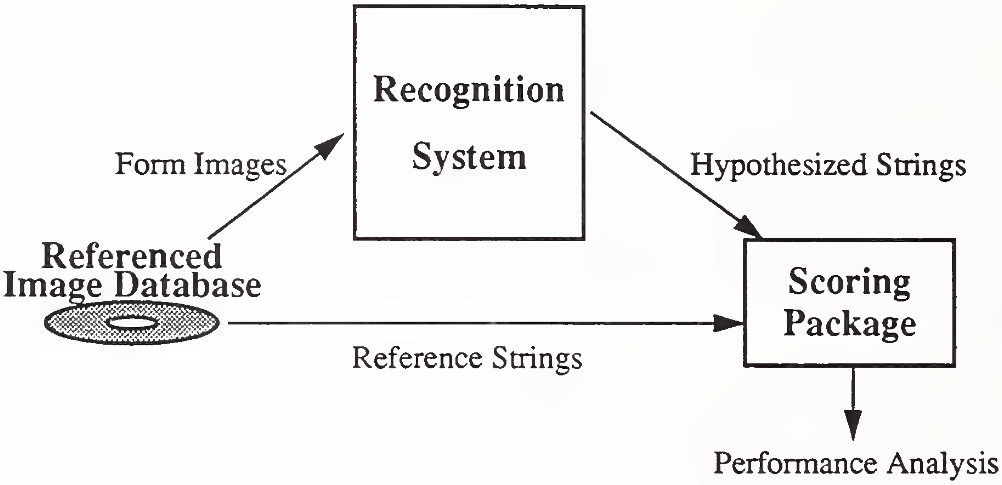


Figure 1: Testing paradigm for recognition systems using referenced images and the Scoring Package.

Using the system testing paradigm in Figure 1, potential users of character recognition technology can design a collection of referenced images representative of their specific needs. The set of images can then be presented to different candidate systems, and using the Scoring Package, performance measures can be computed from the output of each system for the purpose of system comparison. Likewise, a system developer can take a set of referenced images and present them to several variations of a single system. For example, one system configuration may use algorithmic approach A for character segmentation, whereas another system configuration may use algorithmic approach B. By presenting the same set of referenced images to both system configurations, performance measures can be computed and used to compare the two algorithmic approaches within the context of a fully operational system.

The NIST Scoring Package is a reference implementation of the draft, “Standard Method for Evaluating the Performance of Systems Intended to Recognize Hand-printed Characters from Image Data Scanned from Forms”, which has been submitted to ANSI X.3A. As the draft standard is modified and ultimately adopted by ANSI, the Scoring Package will be periodically updated to remain consistent with the standard. The software has been developed on a UNIX workstation and is implemented with a combination of utilities written in the ‘C’ programming language and the UNIX shell facility. Section 2 presents the concepts of scoring forms processing systems and character classifiers. Section 3 discusses the concepts and algorithm used for dynamic string alignment. Section 4 defines the files and the formats required as input to the Scoring Package. Section 5 documents how the Scoring Package software is installed and invoked.

2. Concepts of Scoring

The Scoring Package has the ability to analyze recognition results from tests conducted with two different types of images (images of forms and images of isolated characters). Form-based scoring is designed to analyze the results of processing form images and to measure system performance at the form, field, and character levels. Form-based scoring is useful when comparing form processing systems and is discussed in Section 2.1. Character-based scoring is designed to analyze the results of recognizing isolated character images and to measure classification error rates. Character-based scoring is useful when comparing character classifiers and is discussed in Section 2.2.

2.1 Form-Based Scoring

The Scoring Package has been developed to measure the performance of character recognition systems, and more specifically, automated forms processing systems such as those used to process the images in SD2 and SD6. Figure 2 illustrates four different forms processing tasks addressed by the draft standard. These tasks include form identification, field identification, field recognition, and character recognition. In general, the first step to processing a form requires proper identification of the form type. Based on the identified type, fields can be located through the use of a spatial template. If fields cannot be unambiguously identified by position alone, then other contexts may be required such as reading the label printed on the form next to each field. This is referred to as field identification. Once a field has been located and identified, it then can be recognized. Typically the recognition is done character by character, and if all the characters in a field have been correctly classified, the field is considered to be correctly recognized. This definition of field recognition makes it dependent on the results of character recognition, which is emphasized in the figure by character recognition being nested within field recognition. Currently, the Scoring Package is able to measure the system performance of the form identification, field recognition, and character recognition tasks. The ability to measure the task of field identification has yet to be implemented.

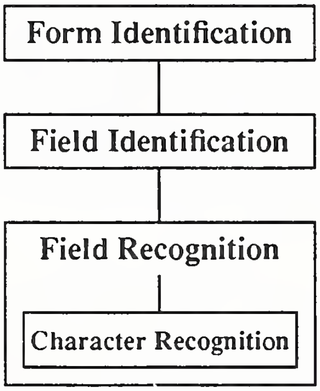


Figure 2: Four tasks of a generic forms processing system.

By establishing form identification as the first task, the Scoring Package does not address system issues such as pages missing from a multiple-page document, and other page handling issues. The Scoring Package has been designed to use forms for which the reference information is complete, accurate, and stored in a specified machine-readable file format. The forms are typically imaged and, together with the reference information, stored on CD-ROM. Only those forms organized in this fashion can be used by the Scoring Package.

The diagram in Figure 2 should be not be mistaken as a model for implementing forms processing systems. It should be viewed as a flexible framework by which forms processing systems can be analyzed and compared. If a specific system does not perform one of the tasks, for example a system may not conduct field identification, then the output resulting from that task is not used in measuring system performance. Note that these system variations are primarily dependent on the types of forms being processed, so that as long as the same set of form images are presented to each system, a consistent set of performance measurements will be computed resulting in a valid comparison. These four tasks embody the primary functions which distinguish forms processing from other applications such as free-formatted correspondence reading. Also notice that these tasks in no way limit the implementation of a forms processing system by dictating a presumed set of algorithmic procedures. For example, traditional character recognition systems conduct character segmentation prior to character classification.[6][7] Methods of combining segmentation and classification into a single concurrent process have recently been developed.[8][9][10] Regardless of the algorithmic techniques used, both types of systems produce character classifications that can be analyzed and compared, and both systems can be analyzed according to the tasks listed in Figure 2.

A more detailed diagram of the forms processing tasks is shown in Figure 3. This figure illustrates the possible outcomes resulting from each of the four tasks. Forms identification can either result in a correctly identified form or an incorrectly identified form. Likewise, field identification can either result in a correctly identified field or and incorrectly identified field. Character recognition can result in a character being correctly recognized, incorrectly recognized, or missed. Characters are frequently missed due to errors during segmentation. If all the characters in a field have been correctly recognized, then the field is considered to be correctly recognized. Otherwise, the field is considered to have been incorrectly recognized. Performance measurements can be computed by compiling statistics at each of these possible outcomes.

For each form image used to test a forms processing system, the Scoring Package is given the form's type, a list of the form's field identities, and a list of text strings corresponding to what was entered on the form, field by field. The files and formats used as input to the Scoring Package are discussed in detail in Section 4. Using this reference information, the Scoring Package can determine the level of error the system achieves when performing each of the four tasks. If the type of a form is correctly identified, then the form is tallied as correctly identified and scoring continues at the field identification task. If form identification is incorrect, then no faith can be placed on the outcomes from any subsequent tasks and scoring is discontinued. The form is tallied as incorrectly identified and the fields and characters on the form are tallied as missing. The same is true at the field identification task. If the field is correctly identified, then the field is tallied as correctly identified and scoring continues at the field and character recognition tasks. If the field identification is incorrect, no faith can be placed on the outcomes from any subsequent tasks and scoring is discontinued. The field is tallied as incorrectly identified and characters in the field are tallied as missing.

Field recognition is dependent on the outcomes from character recognition so that character recognition analysis is conducted first. For each field which is correctly identified from a correctly identified form, the hypothesized characters generated by the recognition system when reading the field are reconciled with the reference string of what was entered in the field. This is done through the use of a dynamic string alignment algorithm which is discussed in detail in Section 3. The alignments produced are used to tally the number of correct, incorrect, and missing characters. If all the characters in the reference string are recognized by the system correctly and no additional characters are falsely inserted, then the field is tallied as being correctly recognized. Otherwise, the field is tallied as incorrectly recognized. This is true when character level rejections do not exist or are ignored. The next section discusses how system rejections impact scoring.

2.1.1 Effects of Rejection

Up to this point, the effects of system rejections on scoring have not been addressed. Systems have the potential to reject the outcomes from each of the four forms processing tasks. For example, a system may choose to reject the hypothesized form type assigned to a specific form image, or a system may choose to reject the hypothesized classification assigned to a segmented character image. Rejecting outcomes gives a system the ability to flag low confidence decisions as unknown, so that they may be verified by human inspection.

Provisions have been made in the Scoring Package to account for several types of system rejections. If the hypothesized identification of a form is rejected, the Scoring Package considers all the fields and characters on the form to be rejected. Only those fields belonging to forms whose identification is accepted continue to be analyzed at the field identification task. In a similar way, if a field identification is rejected, the Scoring Package considers all the characters in the field to be rejected. Only those characters belonging to fields whose identification is accepted continue to be analyzed at the field recognition and character recognition tasks. In the character recognition task, any classification resulting from the recognition of a segmented image may be rejected. It is desirable for a system to reject classifications associated with incorrectly segmented images such as split or merged characters and images of noise. These segmentation errors result in characters being missed (deletion errors) and in erroneous additional classifications being made (insertion errors). It is also desirable to reject incorrect classifications associated with correctly segmented character images. These represent the substitution errors in the system. Unfortunately, rejection mechanisms are not perfect, so that occasionally, correctly classified character images are also rejected. Having described the various instances of character level rejections, a field is considered correctly recognized only if every character in the field's reference string has been correctly classified with no characters missed and there are no additional (inserted) classifications remaining after rejection.

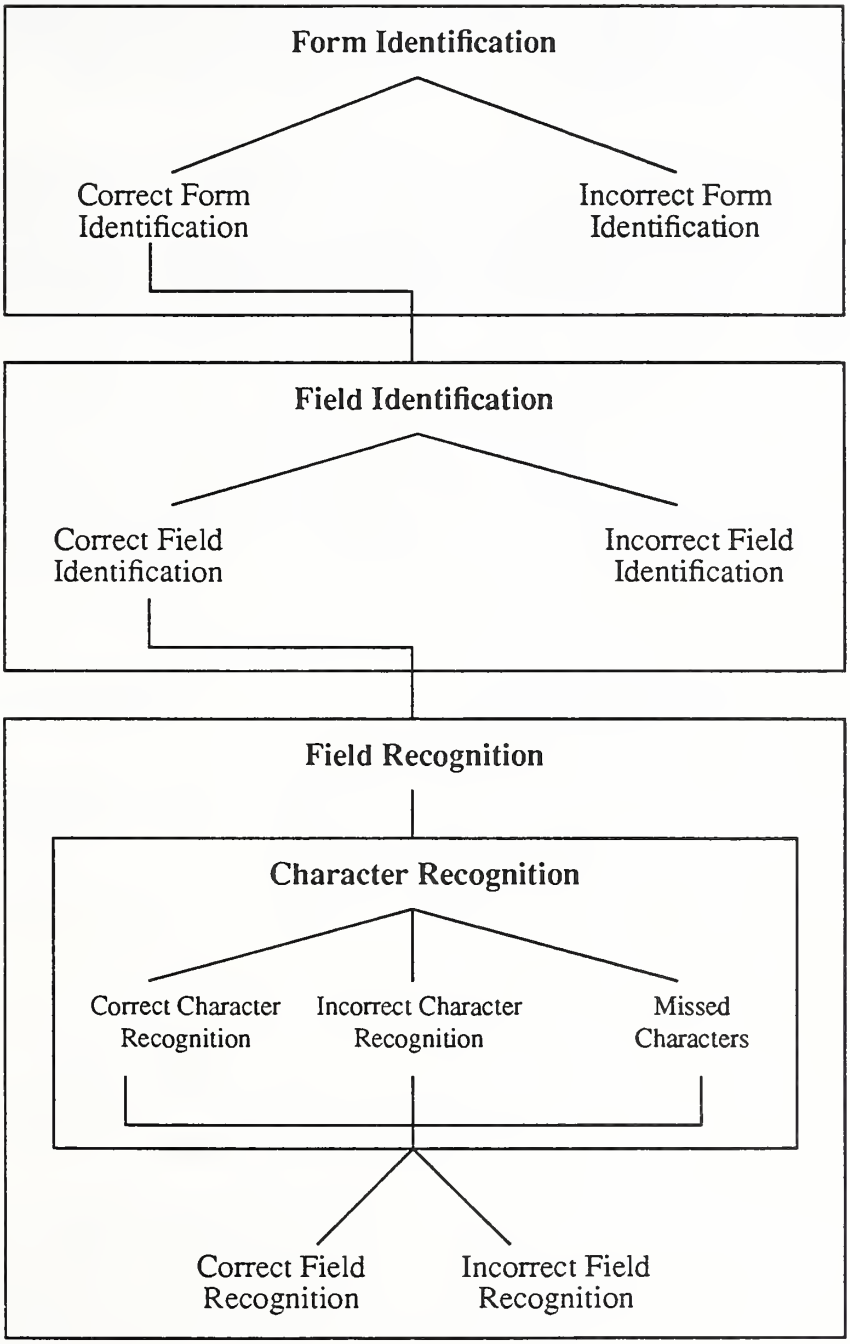

Form Identification
Correct Form Identification
Incorrect Form Identification
Field Identification
Correct Field Identification
Incorrect Field Identification
Field Recognition
Character Recognition
Correct Character Recognition
Incorrect Character Recognition
Missed Characters
Correct Field Recognition
Incorrect Field Recognition

Figure 3: The possible outcomes resulting from each of the four forms processing tasks.

2.2 Character-based Scoring

The Scoring Package can also be used to measure the classification errors of character classifiers. Here, instead of analyzing the processing of form images, the Scoring Package analyzes the recognition of isolated character images such as those distributed with SD3 and SD7. The class assigned to each of these images is typically taken from one of the digits '0' through '9', one of the upper-case alphabetic letters 'A' through 'Z', one of the lower-case alphabetic letters 'a' through 'z', one of the punctuation characters, or one of a handful of special characters.

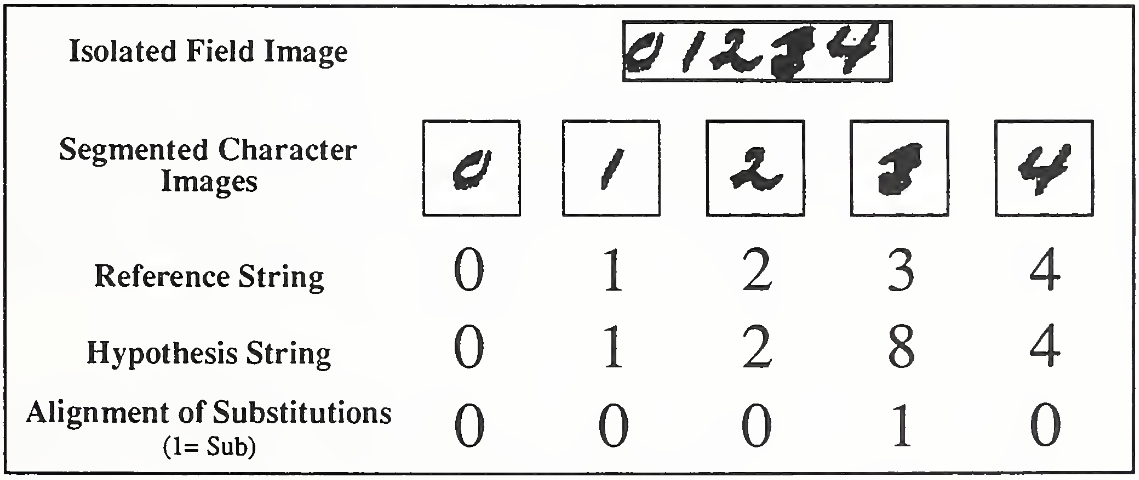
Character-based scoring was used in the first Census Optical Character Recognition Systems Conference sponsored by the Bureau of the Census and hosted by NIST. The report from this conference can be used as a case study of character-based scoring.[11] The files and formats used for character-based scoring differ from those used for form-based scoring. The differences are discussed in detail in Section 4. Though the files and formats differ, the concepts for character-based scoring are quite similar to the character recognition task analyzed in form-based scoring. Character-based scoring only addresses the classification of correctly segmented character images, so that images are either correctly recognized or incorrectly recognized, and their associated classifications may be rejected.

3. Dynamic String Alignment

The importance of automating the performance assessment of large scale character recognition systems was emphasized in the introduction. The automation of the Scoring Package is largely due to the use of a dynamic string alignment algorithm. This algorithm is responsible for determining how errors occurring in the character recognition task in form-based scoring should be assessed. The alignment algorithm reconciles the reference string (what was entered in a field) with the hypothesized string generated by the recognition system. String alignment concepts are discussed in Section 3.1 and the actual algorithm used is described in Section 3.2.

3.1 String Alignment Concepts

In this section, several different examples are presented in order to demonstrate how string alignments can be used to automatically assess the performance of character recognition systems. A familiar system error is a substitution error in which the recognition system assigns an incorrect classification to a segmented character image. Figure 4 displays an alignment produced by the Scoring Package of a substitution error caused by an ambiguous character, a '3' classified as an '8'. The hand-printed '3' is malformed so that it really does look ambiguously like an '8' when read by a human. The top image in the figure is an isolated field containing the five hand-printed digits '0', '1', '2', '3', and '4'. The second line of images are the result of segmenting the isolated field into separate images, one character per image. The third line in the figure lists the reference string of what truly was printed in the field. The fourth line lists the hypothesis string corresponding to the assigned classifications generated by the recognition system. The last line in the figure marks the substitution errors identified by the Scoring Package with a '1' representing a substitution error made by the recognition system. As shown in the figure, the segmented character image containing the malformed '3' is classified by the recognition system as an '8' and is identified as a substitution error by the Scoring Package by reconciling the hypothesis string with the reference string.

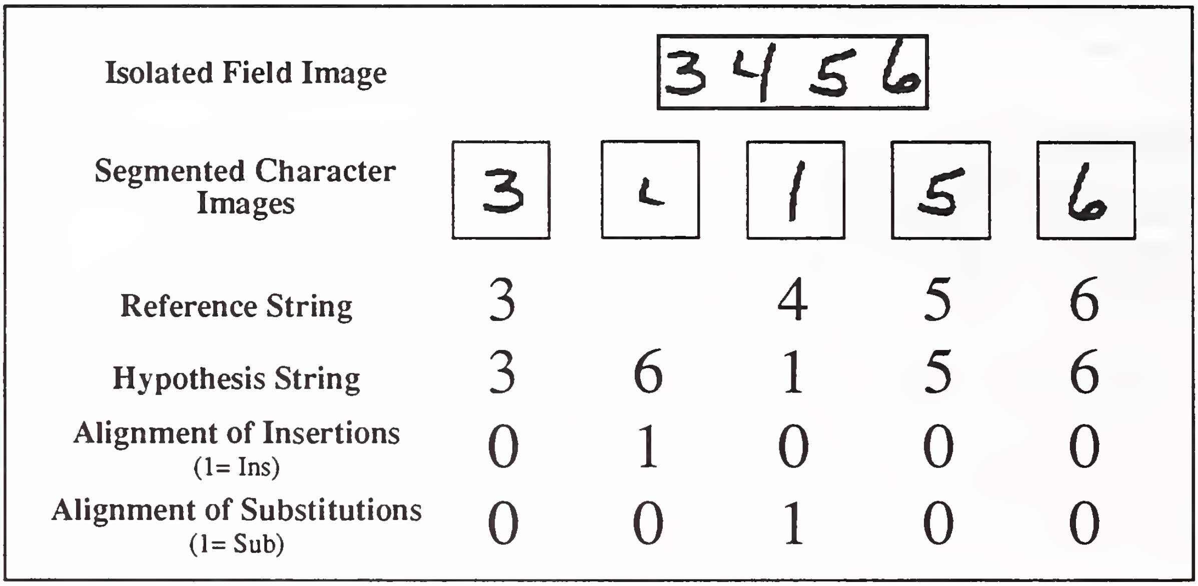


Reference String	0	1	2	3	4
Hypothesis String	0	1	2	8	4
Alignment of Substitutions (1= Sub)	0	0	0	1	0

Figure 4: Scoring Package alignment of a substitution error caused by a malformed character.

Another source of character recognition errors comes from incorrectly segmented character images. Most character classifiers are designed to recognize characters one character image at a time. With unconstrained hand-print, characters frequently touch or overlap making the clean separation of characters difficult. Unfortunately, characters are not always segmented correctly. This results in isolated images containing partial characters, multiple characters, and noise. These segmented images are in turn passed to the system's character classifier. Typical segmentation failures result in the insertion of character-like images into, and the deletion of legitimate character images from, the recognition system. This is demonstrated in the examples shown in Figure 5 and Figure 6.

Figure 5 shows an example alignment produced by the Scoring Package of an insertion error caused by a segmentation failure. The top image is an isolated field containing the four hand-printed digits '3', '4', '5', and '6'. The second line of images is the result of segmenting the isolated field into separate images, which are assumed to be one character per image. Notice the '4' has been incorrectly separated into two pieces resulting in two isolated images with two strokes forming a right angle in the left image and a vertical stroke in the right image. This is an example of a segmentation failure, the splitting of a character into multiple images. The third line in the figure lists what was printed in the field. The fourth line lists the hypothesis string corresponding to the assigned classifications generated by the recognition system.



Reference String	3		4	5	6
Hypothesis String	3	6	1	5	6
Alignment of Insertions (1= Ins)	0	1	0	0	0
Alignment of Substitutions (1= Sub)	0	0	1	0	0

Figure 5: Scoring Package alignment of an insertion error caused by a segmentation failure and resulting in a substitution error.

Due to the segmentation failure, the character classifier in the recognition system is presented the two pieces of the '4' rather than one complete character. The result can be seen in the hypothesis string where the first piece of the '4' is classified as a '6' and the second piece of the '4' is classified as a '1'. The fifth line in the figure marks the insertion error identified by the Scoring Package with a '1' representing the inserted classification of a '6'. Often, a single segmentation failure introduces multiple errors into the system. This can be seen by the last line in the figure which marks a substitution error at the position of the second piece of the '4', the vertical stroke. If segmentation failures go undetected, then the character classifier assumes the resulting isolated images are correct and the classifier will assign a classification to each isolated image it is permitted to see. By reconciling the reference string to the hypothesis string, the Scoring Package labeled the second piece of the '4' as a substitution error, knowing that a '4' was truly printed in the field.

Figure 6 shows an example alignment produced by the Scoring Package of a deletion error caused by a segmentation failure. The top image is an isolated field containing the five digits '4', '5', '6', '7', and '8'. The second line of images is the result of segmenting the isolated field into separate images, which are assumed to be one character per image. Notice the '5' and '6' have been merged into a single isolated image. This is another example of a segmentation failure, the merging of multiple characters into a single segmented image. The third line in the figure lists the reference string of what truly was printed in the field. The fourth line lists the hypothesis string corresponding to the assigned classifications generated by the recognition system.

Due to the segmentation failure, the character classifier in the recognition system is presented a single image containing two characters rather than two separate images each containing one character. This is another example of how, if a segmentation failure goes undetected, the character classifier will assign a classification to each isolated image it is permitted to see.The result can be seen in the hypothesis string where the number of assigned classifications is one less than the length of the reference string, and the merged image containing the '5' and '6' is classified as a '7'. The fifth line in the figure marks the deletion error identified by the Scoring Package with a '1' representing the position of the deleted character. The last line in the figure marks the substitution error resulting from the merged character image being incorrectly classified. By reconciling the reference string to the hypothesis string, the Scoring Package located the position of the deleted character and labeled the classification assigned to the merged character image as a substitution error.

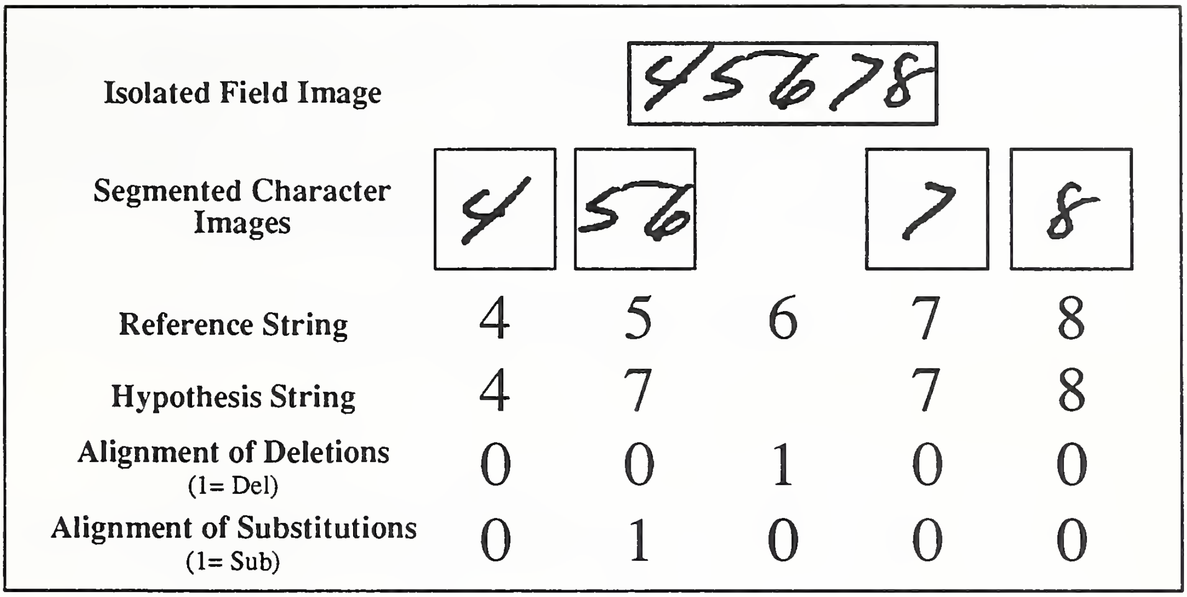

Isolated Field Image
45678
Segmented Character Images
4
56
7
8
Reference String 4 5 6 7 8
Hypothesis String 4 7 7 8
Alignment of Deletions (1= Del) 0 0 1 0 0
Alignment of Substitutions (1= Sub) 0 1 0 0 0

Figure 6: Scoring Package alignment of a deletion error caused by a segmentation failure and resulting in a substitution error.

The examples in Figure 5 and Figure 6 demonstrate how system errors have a cascading effect, resulting in multiple errors being introduced into a single hypothesis string. The alignment examples shown are, by design, easy to understand and are easily derived. In practice, multiple errors frequently occur in a single hypothesis string resulting in many different possible alignments. The Scoring Package analyzes each candidate alignment and chooses the one that assesses the least amount of penalty. The Scoring Package does this in a consistent and logical way so that, when given the same hypothesis string and reference string, the Scoring Package will always generate the same alignment. As multiple errors are introduced into the hypothesis string, it becomes increasingly more difficult for the Scoring Package to unambiguously determine insertion errors from substitutions errors. This distinction often requires human inspection which would compromise the degree to which the Scoring Package is automated. Therefore, the Scoring Package does not distinguish substitution errors from insertion errors and lumps them together into a single category called false positives.

The examples shown in this section have been for form-based scoring. Note that the same dynamic string alignment algorithm used for form-based scoring can be used for character-based scoring. This is realized by treating each isolated character image in character-based scoring as a field containing only one character in form-based scoring. Using this conversion scheme, the alignment algorithm simply determines if a character classifier's hypothesized classification matches the reference class associated with the isolated character image.

3.2 Dynamic String Alignment Algorithm

The dynamic string alignment algorithm used in the Scoring Package has been adapted from the Levenstein Distance algorithm.[12] This algorithm uses dynamic programming to find the minimum distance between two strings given penalties for character substitutions, deletions and insertions. The algorithm was modified to return the information needed to construct aligned reference and hypothesis strings.

First, two 2-dimensional arrays are filled. One array contains the minimum cumulative penalties using the Levenstein Distance that represent the mutation of the reference string into the hypothesis string along all possible paths. The other array holds the decisions (substitution, insertion, deletion, or no change) that yielded the minimum additional penalty to arrive at each point. An array of the series of decisions is then gathered, and the aligned reference and hypothesis strings are generated from that.

The algorithm was extended in four other ways: 1.) It can accept input strings of any length given sufficient memory; 2.) The penalties are not fixed, but are parameters which can be modified from the command line; 3.) Another parameter controls the way the algorithm breaks ties between equal-cost paths; and 4.) The penalties are context-sensitive, that is they are not scalar values, but are functions of the characters.

Context-sensitive alignment can result in more logical alignments. For example, given a reference string "h" and hypothesis strings "k-" and "-k", if the penalty for a 'k' being substituted for a 'h' is less than the penalty for a '-' being substituted for a 'h', then the alignment algorithm will produce output which, more often than not, reflects what was actually confused by a recognition system as shown in Figure 7. The 'k' is aligned with the 'h' and the '-' is scored as an insertion. If the penalties were constants, substitutions between dissimilar characters would occur as often as those between similar characters. Currently, the only command-line interface to the context mappings specifies that case-insensitive alignment be employed.

Context-Sensitive Alignments

h	h
k -	- k

Figure 7: Alignments resulting from less penalty for a 'k' being substituted for a 'h' than a '-' being substituted for a 'h'.

4. File Formats

Strict adherence to file formats is essential for successful Scoring Package operation. In light of their critical importance, this section is devoted to file format specifications. Section 4.1 deals with the files and formats required for form-based scoring, while Section 4.2 deals with the files and formats required for character-based scoring.

4.1 Form-Based Scoring Files

There are five unique file types utilized by the NIST Scoring Package for form-based scoring: Table_A files, reference files, hypothesis files, confidence files, and rejection files. Three of these file types are required (Table_A files, reference files, and hypothesis); two are optional (confidence files and rejection files). Figure 8 lists these files identifying their source and frequency of occurrence. The *tester* designs and administers a scoring test; the *subject* takes the test. Of the two file types generated by the tester (Table_A files and reference files), the Table_A files must be distributed to the subject in addition to test images, while the reference files are for the tester's use only and are never released. Each of these five files are discussed in detail in the following sections.

FILE	SOURCE	OCCURANCE	STATUS
Table_A	Tester	Per Form Template	Required
Reference	Tester	Per Form Sample	Required
Hypothesis	Subject	Per Form Sample	Required
Confidence	Subject	Per Form Sample	Optional
Reject	Subject	Per Form Sample	Optional

Figure 8: Five different file types required by the NIST Scoring Package for form-based scoring.

4.1.1 Form and Format Terminology

In this document, a form template refers to each unique variation or version of a blank form. Each form face such as the 1040 page 1, the 1040 page 2, the Schedule A, etc. is identified as a unique form template. A separate form template must also be identified for variations of the same form face when a different number or order of entry fields exist due to changes from year to year or due to printing variations among tax preparers and tax packages. In this document, a form sample refers to an instance of a form template with its entry field values filled in. The form samples are the images in a test database.

To simplify file format descriptions, several terms must be defined. A Single-Value ASCII String Representation (SVASR) is a buffer of variable length containing any number of printable ASCII characters in the hexadecimal range 21 to 7E. A SVASR is void of any space characters, hexadecimal 20. A Multiple-Value ASCII String Representation (MVASR) is a buffer of variable length containing any number of printable ASCII characters in the hexadecimal range 20 to 7E including any number of space characters. An ASCII Delimiter Character (ADC) is a single space character, hexadecimal 20. The ADC is used to separate a line of contiguous SVASR's or to separate a SVASR followed by a MVASR. An ASCII Line Representation (ALR) is a buffer of variable length containing any number and combination of SVASRs, MVASRs, and ADCs terminated by the ASCII LF character, hexadecimal 0A. This means that the ASCII CR character, hexadecimal 0D, cannot occur anywhere in an ALR, or in place of, or in combination with the ASCII LF character 0A at the end of the ALR. Also note that all files described in this document do not contain any end-of-file marker or end-of-file character.

4.1.2 Table_A Files

Table_A files are created by the tester. A separate Table_A file is required for each unique form template or layout existing in the test database. This table is comprised of three columns of information including entry field identifications, entry field types, and an optional column of entry field context labels. Each line of the table data represents a single entry field found on the form template. The entry field identification strings listed in the first column dictate the identity of each entry field on the form. These references must match identically to the field answers generated by the tester in the reference files and the subject's responses in the hypothesis files. Reference files and hypothesis files will be discussed in detail later. Appendix A contains a blank template of the first page of a 1988 1040 form labeled with the entry field identifications used in the corresponding Table_A file which is also included in the appendix. The contents of the Table_A file shown in Figure 35 has been listed in two adjacent text columns.

The type of an entry field can be, for example, one of four possible choices (A, F, I, and ICON). Figure 9 lists the field types used in SD2 and in SD6 to describe entry fields. The type "A" should be used for any alphanumeric field, "F" should be used for any floating point field, "I" should be used for any integer field, and "ICON" should be used for any field which is not of the previous three types. The "ICON" type should be used to represent annotations in the margin of a form, box check-marks, signatures, and other types of fields which will not be recognized character by character.

TYPE	DEFINITION
A	Alphanumeric Fields
F	Floating Point Fields
I	Integer Fields
ICON	Non-Character Fields (box markings, signatures)

Figure 9: Four different entry field types used in Table_A files for SD2 and SD6.

The third column in a Table_A file contains entry field context labels. If provided, the context labels can be used to identify and score subsets of entry fields uniquely. The context labels used in SD2 and SD6 include Data, Name, and SSN and are listed in Figure 10. For example, using this context convention, the social security numbers on forms can be isolated and scored apart from the other entry fields on the form. The context assigned to an entry field is optional. This means the Table_A file for a given form template must contain the identification label and the type of each entry field on the form, but context labels may or may not be specified. If a context label is desired for a single entry field on form template, then context labels for all entry fields on the form template should be provided. If no context labels are desired for a given form template, then none need be provided.

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

Figure 10: Three different entry field contexts used in SD2 and SD6.

As stated earlier, a Table_A file is comprised of two required entry field columns (identifications and types) and one optional column (context labels). The tester is responsible for generating and assigning each of the entry field identifications, types, and context labels. The only entry field type automatically distinguished by the Scoring Package is ICON. This enables the scoring of fields containing character data to be automatically separated from the scoring of fields containing non-character data. By inventing and assigning entry field types and context labels, the tester can design a flexible test in which both global scores and scores computed on specific subsets of fields may be specified. For example, by designing a Table_A file such that all fields containing the names of people are identified with the context label "NAME", the tester can invoke the Scoring Package specifying that only the fields containing the names of people be included in the computation of scores.

A Table_A file is comprised of a variable number of ALRs, one ALR per entry field on the form template being represented. If context labels are being used for the given form template, then each ALR is comprised of three SVASRs, representing in order, an entry field's identification string, type, and context label. One ADC is used to separate the identification string from the type, and one ADC is used to separate the type from the context label. If context labels are not being used for the given form template, then each ALR is comprised of two SVASRs, representing in order, an entry field's identification string and type with one ADC used to separate the two.

Figure 11 lists the first ten lines of a Table_A file for the first page of a 1988 1040 form for which context labels are included. Notice that there are no graphical lines or heading in this file, only character data organized in columns. Figure 12 lists byte for byte the hexadecimal representation of the ten lines listed in Figure 11. Notice the hexadecimal 0A character terminating each line and the hexadecimal 20 character separating each SVASR. Figure 13 and Figure 14 list the first ten lines of a Table_A file for the first page of a 1988 1040 form for which no context labels are provided. Notice in Figure 14 there is only one hexadecimal 20 character separating the entry field identification string from the type with each line terminated by a hexadecimal 0A character. In order to readily identify which form template a Table_A file represents, the form type should be embedded in the table's file name. For example "1988_1040_1_a.tab" could be used to store the table information listed in either Figure 11 or Figure 13.

```
1040_1_L_H1_V1 A DATA
1040_1_L_H2_V1 A DATA
1040_1_L_H3_V1 A DATA
1040_1_L_H1_V2 A NAME
1040_1_L_H2_V2 A SSN
1040_1_L_H1_V3 A DATA
1040_1_L_H2_V3 A SSN
1040_1_L_H1_V4 A DATA
1040_1_L_H1_V5 ICON DATA
1040_1_L_H2_V5 ICON DATA
```

Figure 11: Top of an example Table_A file containing context labels.

```
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 31 20 41 20 44 41 54 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 31 20 41 20 44 41 54 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 33 5F 56 31 20 41 20 44 41 54 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 32 20 41 20 4E 41 4d 45 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 32 20 41 20 53 53 4E 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 33 20 41 20 44 41 54 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 33 20 41 20 53 53 4E 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 34 20 41 20 44 41 54 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 35 20 49 43 4F 4E 20 44 41 54 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 35 20 49 43 4F 4E 20 44 41 54 41 0A
```

Figure 12: Hexadecimal listing of the Table_A portion listed in Figure 11.

```
1040_1_L_H1_V1 A
1040_1_L_H2_V1 A
1040_1_L_H3_V1 A
1040_1_L_H1_V2 A
1040_1_L_H2_V2 A
1040_1_L_H1_V3 A
1040_1_L_H2_V3 A
1040_1_L_H1_V4 A
1040_1_L_H1_V5 ICON
1040_1_L_H2_V5 ICON
```

Figure 13: Top of an example Table_A file excluding context labels.

```
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 31 20 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 31 20 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 33 5F 56 31 20 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 32 20 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 32 20 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 33 20 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 33 20 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 34 20 41 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 35 20 49 43 4F 4E 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 35 20 49 43 4F 4E 0A
```

Figure 14: Hexadecimal listing of the Table_A portion listed in Figure 13.

4.1.3 Reference Files

For every form sample in a test database an associated reference file is required. Reference files are created by the tester for use in system testing. These files contain the identification of the form template contained in the form sample followed by the actual data entered in each field on the form. The Scoring Package treats the form identification and entry field values recorded in the reference file as ground truth. The integrity of any test is completely dependent on the accuracy of these files. Appendix B contains an image of a completed first page of a 1988 1040 tax form and the reference file associated with the form image is listed in two adjacent text columns in Figure 36. Note that the information contained in the form was derived from a computer and does not contain real tax information.

A reference file is comprised of a variable number of ALRs with the first ALR identifying the sample's form template followed by one ALR per entry field on the sample. The form identification is the first ALR in the reference file and is represented as a SVASR. Each ALR following the form identification ALR corresponds to a specific entry field on the form. These entry field ALRs contain a required entry field identification string and a conditional entry field value. The identification string is represented as a SVASR and the entry field value is represented as an MVASR. If an entry field ALR contains the conditional entry field value, then the ALR is comprised of a SVASR and MVASR separated by an ADC. If an entry field ALR does not contain an entry field value, then the ALR is comprised of a SVASR representing the identification string only. If an entry field contains data, then its value should contain exactly what was entered in the field. If an entry field is blank, then its value should be omitted from the ALR including the omission of the ADC.

Figure 15 lists the first ten lines of a reference file for the first page of a 1988 1040 form. The first line identifies the form template contained in the form sample. The remaining lines correspond to the first 9 entry fields contained on the first page of the 1040 form. Notice that the first three entry fields (1040_1_L_H1_V1, 1040_1_L_H2_V1, 1040_1_L_H3_V1) have no entry field value entered in the reference file because their corresponding fields on the form were left empty. Figure 16 lists byte for byte the hexadecimal representation of the ten lines listed in Figure 15. Notice the hexadecimal 0A character terminating each line. Also notice that the first three entry field ALRs contain only a single SVASR representing identification strings without associated values. This represents three entry fields which were left blank on the form sample. The remaining six entry field ALRs contain both identification strings and values. These are entry fields that were filled in on the form sample. Notice that the identification strings are represented as SVASRs, the values are represented as MVASRs, and that there is a single ADC, hexadecimal 20, separating the two.

```
1988_1040_1
1040_1_L_H1_V1
1040_1_L_H2_V1
1040_1_L_H3_V1
1040_1_L_H1_V2 Berry K. & Loras A. Boyle
1040_1_L_H2_V2 A15 86 7384
1040_1_L_H1_V3 7861 Fairfield Street
1040_1_L_H2_V3 A73 28 5386
1040_1_L_H1_V4 Boyle, MT 30073
1040_1_L_H1_V5 1
```

Figure 15: Top of an example reference file.

```
31 39 38 38 5F 31 30 34 30 5F 31 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 31 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 31 0A
31 30 34 30 5F 31 5F 4C 5F 48 33 5F 56 31 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 32 20 42 65 72 72 79 20 4B 2E 20 26 20 4C 6F 72 61 73 20 41 2E 20 42 6F 79 6C 65 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 32 20 41 31 35 20 38 36 20 37 33 38 34 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 33 20 37 38 36 31 20 46 61 69 72 66 69 65 6C 64 20 53 74 72 65 65 74 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 33 20 41 37 33 20 32 38 20 35 33 38 36 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 34 20 42 6F 79 6C 65 2C 20 4D 54 20 33 30 30 37 33 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 35 20 31 0A
```

Figure 16: Hexadecimal listing of the reference file portion listed in Figure 15.

Entry field 1040_1_L_H1_V5 is an example of an ICON entry field. Notice that this field's value is a '1' which signifies that the field contains a box check mark. If the ICON entry field was empty on the form, then a value of '0' would be used in the reference file. This convention reflects a format change in the way ICON entry fields are represented in the reference file. In the past, such as in SD2 and SD6, the entry field value was left blank when no information was present, and in SD2 a value of "_ICON_" was used in place of the '1' when information was present.

The entry field identification strings listed in the reference file must match exactly in name and in order to the identification strings recorded in the Table_A file associated with the sample's form template. The SVASR used for the form identification should be embedded in the associated Table_A file name. In our previous file name example, Table_A was named "1988_1040_1_a.tab". Notice that the form identification in the reference file example is "1988_1040_1". For historical reasons, the reference files used for form-based scoring have also been called format files. All reference files should have a consistent extension such as "fmt".

4.1.4 Hypothesis Files

For every form sample in a test database, the subject must return an associated hypothesis file. Each hypothesis file contains the form template identified by the subject's system followed by the results of what his system captured and recognized from each entry field on the form sample. The Scoring Package aligns the subjects's results with the true entry field values contained in the form sample's associated reference file in order to compute error rates. Appendix B contains an example of a hypothesis file corresponding to the completed form displayed in the appendix. The hypothesis file shown in Figure 37 is listed in two adjacent text columns.

Hypothesis files are identical in format to that of reference files. A hypothesis file is comprised of a variable number of ALRs with the first ALR containing the form template identified by the subject's system followed by one ALR per entry field on the form. The form identification is the first ALR in the hypothesis file and is represented as a SVASR. Each ALR following the form identification ALR corresponds to a specific entry field on the form. These entry field ALRs contain a required entry field identification string and a conditional entry field value. The identification string is represented as a SVASR and the value is represented as an MVASR. If a subject's system detected and captured data within an entry field, then the recognized value is included and the entry field ALR is comprised of a SVASR and MVASR separated by one ADC. If a subject's system detected a blank field, then the value is omitted including the ADC, and the entry field ALR is comprised only of a SVASR representing the identification string.

It is common for alphanumeric entry fields to be made up of more than one word. Therefore, the recognition of spacing must be addressed. When capturing and recognizing fixed-spaced machine generated text, spaces between words are clearly detectable. When capturing and recognizing proportionally-spaced machine generated text, the detection of spaces becomes slightly obscure. When capturing and recognizing hand-printed data, the detection of spaces without the use of dictionaries and grammars becomes practically impossible. In light of this, the subject has the choice of reporting recognition results with or without the recognition of spaces. If the subject chooses to report the recognition of spaces, then the value of an entry field detected to contain multiple words will contain a space character wherever the subject's system detected one. Remember that the value of an entry field ALR in the hypothesis file is a MVASR which includes the existence of space characters, hexadecimal 20. If the subject chooses not to report the recognition of spaces, then the value of all entry field ALRs, even if the entry field really is comprised of multiple words, will contain no space characters. The Scoring Package can handle either hypothesis format for entry field values.

Figure 17 lists the first ten lines of an example hypothesis file where the subject chose not to report the recognition of spaces. This example represents perfect recognition of the form corresponding to the reference file in Figure 15. Figure 18 lists byte for byte the hexadecimal representation of the ten lines listed in Figure 17. Notice that the multiple word values do not have any space characters, hexadecimal 20. Also note that the fields having recognized information retain the use of the ADC to separate the entry field identification string from the entry field value in the hypothesis file.

```
1988_1040_1
1040_1_L_H1_V1
1040_1_L_H2_V1
1040_1_L_H3_V1
1040_1_L_H1_V2 BerryK.&LorasA.Boyle
1040_1_L_H2_V2 A15867384
1040_1_L_H1_V3 7861FairfieldStreet
1040_1_L_H2_V3 A73285386
1040_1_L_H1_V4 Boyle,MT30073
1040_1_L_H1_V5 1
```

Figure 17: Top of an example hypothesis file where the subject chose not to report the recognition of spaces.

```
31 39 38 38 5F 31 30 34 30 5F 31 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 31 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 31 0A
31 30 34 30 5F 31 5F 4C 5F 48 33 5F 56 31 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 32 20 42 65 72 72 79 4B 2E 26 4C 6F 72 61 73 41 2E 42 6F 79 6C 65 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 32 20 41 31 35 38 36 37 33 38 34 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 33 20 37 38 36 31 46 61 69 72 66 69 65 6C 64 53 74 72 65 65 74 0A
31 30 34 30 5F 31 5F 4C 5F 48 32 5F 56 33 20 41 37 33 32 38 35 33 38 36 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 34 20 42 6F 79 6C 65 2C 4D 54 33 30 30 37 33 0A
31 30 34 30 5F 31 5F 4C 5F 48 31 5F 56 35 20 31 0A
```

Figure 18: Hexadecimal listing of the hypothesis file portion listed in Figure 17.

The entry field identification strings listed in the hypothesis file must match exactly in name and in order to the identification strings in the Table_A file associated with the sample's form template and the corresponding reference file. The tester must distribute Table_A files to the subject so that the subject can return hypothesized answers referenced by correct entry field identification strings. The SVASR returned from the subject for the form identification must also be one from a list of form templates provided to the subject from the tester. All hypothesis files should have a consistent extension such as "HYP".

4.1.5 Confidence Files

Character classifiers typically produce a floating point value on the range 0.0 to 1.0, representing how confident the classifier is of its recognition decision. By setting thresholds on these values, a subject can tune his system to desired levels of performance trading off throughput for accuracy. The Scoring Package is capable of conducting basic analyses with only the subject's hypothesis file aligned with the form sample's reference file. However, through the optional use of confidence files, the Scoring Package can do additional analyses if the subject provides confidence values for each character classified. Appendix B contains an example of a confidence file corresponding to the completed form displayed in the appendix. Note that the line breaks within single entry field specifications in Figure 38 are due to the wrap-around properties of the listing. Line breaks within an entry field specification do not indicate the presence of new-line characters in the file.

Confidence files are comprised of a variable number of ALRs with the first ALR containing the confidence of the recognition system's identification of the form template followed by one ALR per entry field on the form sample. The confidence of the form identification is the first ALR in the confidence file and consists of the form identification (also included in the hypothesis file) and the actual confidence value. Both the form identification and the confidence value are represented as SVASRs separated by an ADC. Each ALR following the form identification ALR corresponds to a specific entry field on the form. These entry field ALRs contain a required entry field identification string and a conditional list of confidence values. The identification string and confidence values are represented as SVASRs separated by ADCs. If a subject's system detected and captured data within an entry field, then the confidence values are included and the entry field ALR contains the identification string and one confidence value for each individual character captured and classified. If a subject's system detected a blank field, then the confidence values are omitted, and the entry field ALR contains a SVASR representing the entry field identification string.

A confidence value must be a number ranging from 0.0 through 1.0. The number of digits to the right of the decimal point must be less than 17. Whether or not a subject chooses to report his recognition results of spaces, the number of bytes comprising an entry field's value MVASR in the hypothesis file must equal the number of individual confidence values reported in the confidence file for the entry field. Failure of the number of bytes in the hypothesis file's MVSAR to equal the number of confidence values in the confidence file will result in the entry field being removed from the analysis conducted by the Scoring Package.

Figure 19 lists the first five lines of an example hypothesis file where the subject chose to report his recognition results of spaces. Figure 21 lists the corresponding lines from an example confidence file. Notice that the last line in Figure 19 contains a space character in the MVASR "B. Boyle" which is listed as hexadecimal in Figure 20 as "42 2e 20 42 6f 79 6c 65". Also notice that the eight bytes in the MVASR are assigned exactly eight confidence values in the last line of Figure 21. Had the subject chosen not to report his recognition results of spaces, then the MVASR of the last line in Figure 19 would be "B.Boyle" without the space character. The hexadecimal listing for the MVASR would be "42 2e 42 6f 79 6c 65", omitting the hexadecimal 20. In turn, the list of confidence values in Figure 21 would be reduced from eight values to seven with the confidence value "0.258367" omitted.

The entry field identification strings listed in the confidence file must match exactly in name and in order to the identification strings dictated in the Table_A file associated with the sample's form template and the corresponding reference and hypothesis files. All confidence files should have a consistent extension such as "CON".

```
1988_1040_1
1040_1_L_H1_V1
1040_1_L_H2_V1
1040_1_L_H3_V1 87
1040_1_L_H1_V2 B. Boyle
```

Figure 19: Top of a hypothesis file where the subject chose to report the recognition of spaces.

```
31 39 38 38 5f 31 30 34 30 5f 31 0a
31 30 34 30 5f 31 5f 4c 5f 48 31 5f 56 31 0a
31 30 34 30 5f 31 5f 4c 5f 48 32 5f 56 31 0a
31 30 34 30 5f 31 5f 4c 5f 48 33 5f 56 31 20 38 37 0a
31 30 34 30 5f 31 5f 4c 5f 48 31 5f 56 32 20 42 2e 20 42 6f 79 6c 65 0a
```

Figure 20: Hexadecimal listing of the hypothesis file portion listed in Figure 19.

```
1988_1040_1 0.989425
1040_1_L_H1_V1
1040_1_L_H2_V1
1040_1_L_H3_V1 0.786324 0.998934
1040_1_L_H1_V2 0.675347 0.994671 0.258367 0.683123 0.876284 0.391576 0.4987481 0.719952
```

Figure 21: Top of an example confidence file corresponding to the hypothesis file in Figure 19.

```
31 39 38 38 5f 31 30 34 30 5f 31 20 30 2e 39 38 39 34 32 35 0a
31 30 34 30 5f 31 5f 4c 5f 48 31 5f 56 31 0a
31 30 34 30 5f 31 5f 4c 5f 48 32 5f 56 31 0a
31 30 34 30 5f 31 5f 4c 5f 48 33 5f 56 31 20 30 2e 37 38 36 33 32 34 20 30 2e 39 39 38 39 33 34 0a
31 30 34 30 5f 31 5f 4c 5f 48 31 5f 56 32 20 30 2e 36 37 35 33 34 37 20 30 2e 39 39 34 36 37 31 \
         20 30 2e 32 35 38 33 36 37 20 30 2e 36 38 33 31 32 33 20 30 2e 38 37 36 32 38 34 \
         20 30 2e 33.39 31 35 37 36 20 30 2e 34 39 38 37 34 38 31 20 30 2e 37 31 39 39 35 32 0a
```

Figure 22: Hexadecimal listing of the confidence file portion listed in Figure 21.

4.1.6 Rejection Files

A second type of optional file which a subject can return is rejection files. Given the confidence values from a recognition system, a subject can use very sophisticated methods for determining whether a recognition decision should be accepted or rejected. Therefore, rather than return raw confidence values in a confidence file, the subject may choose instead to specify explicitly which classifications should be rejected and which should be accepted in a rejection file. Once again, the Scoring Package is capable of conducting basic analyses with only the subject's hypothesis file. However, through the optional use of rejection files the Scoring Package can do additional analyses if the subject provides reject values for each character classified. Appendix B contains an example of a rejection file. Once again, note that the line breaks within single entry field specifications in Figure 39 are due to the wrap-around properties of the listing and do not indicate the presence of new-line characters in the file.

Rejection files are comprised of a variable number of ALRs with the first ALR containing information as to whether the recognition system accepted or rejected the identification of the form template followed by one ALR per entry field on the form sample. The rejection line corresponding to the form identification is the first ALR in the rejection file and consists of the form identification (also included in the hypothesis file) and a binary reject value. Both the form identification and the rejection value are represented as SVASRs separated by an ADC. Each ALR following the form identification ALR corresponds to a specific entry field on the form. These entry field ALRs contain a required entry field identification string and a conditional list of reject values. The identification string and reject values are represented as SVASRs separated by ADCs. If a subject's system detected and captured data within an entry field, then the reject values are included and the entry field ALR contains the identification string and one reject value for each individual character captured and classified. If a subject's system detected a blank field, then the reject values are omitted, and the entry field ALR contains a SVASR representing the entry field identification string only.

Reject values must be a binary number equal to '0' or '1'. A '1' indicates that the classification should be scored as unknown rather than as a correct or incorrect classification. A '0' indicates that the classification should be scored correct if the hypothesized character is identical to the reference character and scored incorrect otherwise. Regardless if a subject chooses to report his recognition results of spaces or not, the number of bytes comprising an entry field's value MVASR in the hypothesis file must equal the number of individual reject values reported in the rejection file for the entry field. Failure of the number of bytes in the hypothesis file's MVSAR to equal the number of reject values in the rejection file will result in the entry field being removed from the analysis conducted by the Scoring Package.

Figure 23 lists the first five lines of a rejection file corresponding to the example Hypothesis file listed in Figure 19. Notice that in the last line of Figure 23 there is a reject value ('0' or '1') for each and every byte of the MVASR listed in the last line of Figure 19 including a reject value for the space character. If the subject had chosen not to report the recognition results of space characters, then the reject value for the space character would be omitted.

```
1988_1040_1 0
1040_1_L_H1_V1
1040_1_L_H2_V1
1040_1_L_H3_V1 0 0
1040_1_L_H1_V2 1 0 0 0 0 0 1 0
```

Figure 23: Top of a rejection file corresponding to the hypothesis file shown in Figure 19.

```
31 39 38 38 5f 31 30 34 30 5f 31 20 30 0a
31 30 34 30 5f 31 5f 4c 5f 48 31 5f 56 31 0a
31 30 34 30 5f 31 5f 4c 5f 48 32 5f 56 31 0a
31 30 34 30 5f 31 5f 4c 5f 48 33 5f 56 31 20 30 20 30 0a
31 30 34 30 5f 31 5f 4c 5f 48 31 5f 56 32 20 31 20 30 20 30 20 30 20 30 20 30 20 31 20 30 0a
```

Figure 24: Hexadecimal listing of the rejection file portion listed in Figure 23.

The entry field identification strings listed in the rejection file must match exactly in name and in order to the identification strings dictated in the Table_A file associated with the sample's form template and the corresponding reference and hypothesis files. All rejection files should have a consistent extension such as "REJ".

4.2 Form-Based Scoring Output

Two form-based scoring output files are listed in Appendix C. The first file is a summary report listing the performance measures computed consistent with the draft standard. The second file is a fact sheet that gives a detailed accounting of all fundamental events accumulated by the Scoring Package. An explanation of how these files were generated is given in Section 5.5.1. Line by line descriptions of the output files are included in the appendix immediately following the two listings.

4.3 Character-Based Scoring Files

There are four unique file types utilized by the NIST Scoring Package for performing character-based scoring: classification files, hypothesis files, rejection files, and confidence files. Two of these file types are required (classification files and hypothesis files); two are optional (rejection files and confidence files). Figure 25 lists these files identifying their source. Once again, the *tester* designs and administers a scoring test; the *subject* takes the test. The classification files are for the tester's use only and are never released. Each of these four files are discussed in detail in the following sections. Additional information on character-based scoring files and how they are used for testing character classifiers can be found in the conference report from the first Census Optical Character Recognition System Conference.[11]

FILE	SOURCE	STATUS
Classification	Tester	Required
Hypothesis	Subject	Required
Rejection	Subject	Optional
Confidence	Subject	Optional

Figure 25: Four different file types utilized by the NIST Scoring Package for character-based scoring.

4.3.1 File and Format Terminology

The main distinction between form-based scoring and character-based scoring is made with respect to the types of images being recognized by a subject's system. In form-based scoring, the subject is given a collection of images of completed forms to test his system. Using these form images, as described in Section 4.1, allows performance measurements to be calculated at the form, field, and character levels. In character-based scoring, the subject is given a collection of isolated characters, one character per image, to test just his system's character classifier. This results in performance measurements being calculated at just the character level.

Several large collections of isolated character images have been gathered by NIST. These images are typically stored in Multiple Image Set files, or MIS files. One database using the MIS file format is SD3. Briefly, an MIS file contains one or more isolated character images. The last raster row of pixels comprising a previous character image is concatenated with the first raster row of pixels comprising the next character image. In this way, the MIS file is stored as one tall raster image file when really the file contains a column of concatenated character images. A collection of MIS files can be used to represent an entire test set of character images. Each MIS file contains an ASCII header developed by NIST called IHead which is prefixed to the raster image data. This header contains all the information necessary to effectively interpret the image data stored in the file, including attributes such as the pixel width and height of the image. The IHead file format is described in detail in Appendix F. A complete description of the MIS file format is included in Appendix G.

In order to accurately specify file formats for character-based scoring, two terms must be defined. An ASCII String Representation (ASR) is a buffer of variable length containing any number of printable ASCII characters, where the printable ASCII characters include all characters in the hexadecimal range 20 to 7E. An ASCII Line Representation (ALR) is an ASR terminated by the ASCII LF character, hexadecimal 0A. This means that the ASCII CR character 0D cannot occur anywhere in an ALR, or in place of, or in combination with the ASCII LF character 0A at the end of the ALR.

A Multiple Feature Set (MFS) file is a file of ALRs. Each MFS file is associated with a unique MIS file. The first line of the MFS file contains the ASR of a decimal number, which is the number of lines in the file minus one, and also the number of images in the associated MIS file. No ASCII space characters, hexadecimal 20, are allowed in the ASR for the first line. Each line following the first line of an MFS file is an ALR containing information about the corresponding image in the associated MIS file. Each of the four files utilized by the Scoring Package for character-based scoring are MFS formatted files.

4.3.2 Classification (CLS) Files

In order to analyze character recognition results, the Scoring Package requires reference classification information that can be compared against a subject's recognition results. The reference classifications associated with each character in an MIS file are stored in a classification (CLS) file. A CLS file is a file in the MFS file format. Each line following the first line contains an ASR of the correct class assigned to the corresponding image in the associated MIS file. The ASR in each line consists of two ASCII characters. These are the ASCII characters that represent the hexadecimal number that represents the ASCII character of the class. No space characters are allowed on any line of this type of file. One CLS file should be created by the tester for each MIS file included in the test set. The name of a CLS file should correspond to the same name used for the associated MIS file in the test set and have a consistent extension such as "CLS". An example of a classification file corresponding to a collection of isolated hand-print characters is shown in Appendix D.

Figure 26 shows an example of a CLS file that contains reference classifications for the five characters ‘G’, ‘r’, ‘L’, ‘S’, and ‘w’. The tester would have created this CLS file in order to reference a MIS file containing the images of the five characters. An ASCII listing (that recognized the convention that 0A is the end of line marker) of the file is shown in the figure. Similarly, a hexadecimal listing of the same CLS file is shown in Figure 27. In this example, the upper-case ‘C’ in “4C” could just as well be a lower-case ‘c’, resulting in a hexadecimal 43 instead of a hexadecimal 63.

```
5
47
72
4C
53
77
```

Figure 26: ASCII listing of a CLS file containing the five characters ‘G’, ‘r’, ‘L’, ‘S’, and ‘w’.

```
35 0A 34 37 0A 37 32 0A 34 63 0A 35 33 0A 37 37 0A
```

Figure 27: Hexadecimal listing of a CLS file containing the five characters ‘G’, ‘r’, ‘L’, ‘S’, and ‘w’.

4.3.3 Hypothesis (HYP) Files

A hypothesis (HYP) file is a file in the MFS file format containing a subject’s hypothesized character classifications generated by his system’s character classifier. Each line following the first line contains an ASR of the hypothesized class assigned to the corresponding image in the associated MIS file. The ASR in each line consists of two ASCII characters. These are the ASCII characters that represent the hexadecimal number that represents the ASCII character of the hypothesized class. No space characters are allowed on any line of this type of file. One HYP file should be returned by the subject for each MIS file processed by his system. The name of a HYP file should correspond to the same name used for the associated MIS file in the test set and have a consistent extension such as “HYP”. An example of a hypothesis file is included in Appendix D.

For example, consider a HYP file generated by a subject’s system when given the MIS file associated with the CLS file in Figure 26. An ASCII listing of the HYP file is shown in Figure 28. Notice that the last character classification in the HYP file is incorrect. The lower-case ‘w’ in the MIS file was mistakenly identified by the subject as a lower case ‘m’, hexadecimal 6D. Similarly, a hexadecimal listing of the same HYP file is shown in Figure 29.

```
5
47
72
4C
53
6D
```

Figure 28: ASCII listing of a HYP file containing the five characters ‘G’, ‘r’, ‘L’, ‘S’, and ‘m’.

```
35 0A 34 37 0A 37 32 0A 34 63 0A 35 33 0A 36 44 0A
```

Figure 29: Hexadecimal listing of a HYP file containing the five characters ‘G’, ‘r’, ‘L’, ‘S’, and ‘m’.

4.3.4 Rejection (RJX) Files

A rejection (RJX) file is a file in the MFS file format in which the ASR on each line following the first line is an ASCII ‘0’ or an ASCII ‘1’. A ‘1’ indicates that the classification should be scored as unknown rather than as a correct or an incorrect classification. A ‘0’ indicates that the classification should be scored correct if identical with the true classification and scored incorrect otherwise. The name of an RJX file should correspond to same name used for the associated MIS file and have a consistent extension such as “RJ0”. An example of a rejection file is included in Appendix D.

A subject may use the RJX files to return information on the reliability of the hypothesized classifications obtained from his OCR system. This format is useful if the system does not provide confidence levels or activations. The use of RJX files is also preferred if the subject's system has an accept/reject criterion that is more complex than setting a threshold on the highest confidence level or activation. An example of a RJX file associated with the HYP file in Figure 28 is shown in Figure 30. Similarly, a hexadecimal listing of the same RJX file is shown in Figure 31.

```
5
0
1
1
0
1
```

Figure 30: ASCII listing of a RJX file associated with the HYP file in Figure 28.

```
35 0A 30 0A 31 0A 31 0A 30 0A 31 0A
```

Figure 31: Hexadecimal listing of the RJX file shown in Figure 30.

4.3.5 Confidence (CON) Files

A confidence (CON) file is a file in the MFS file format in which the ASR on each line after the first line gives the decimal representation of the confidence level (or activation) assigned to the classification on the corresponding line of the HYP file that is associated with the same MIS file. The confidence level must be a number ranging from 0.0 through 1.0. The number of digits to the right of the decimal point must be less than 17. The name of a CON file must be the same as the name of the associated MIS file and have a consistent extension such as "CON". An example of a confidence file is included in Appendix D.

A subject may use CON files to return the confidence levels assigned by his OCR system to the hypothesized classifications obtained from his system, provided that such information is available, and provided that his system makes its accept/reject decisions by comparing the contents of these files with a subject-specified threshold. For example, consider the same HYP file used for the last example. An ASCII listing of an associated CON file is shown in Figure 32. Similarly, a hexadecimal listing of the same CON file is shown in Figure 33. Leading zeros are optional as shown in the example.

```
5
0.375
.9
.7
.4
.8
```

Figure 32: ASCII listing of a CON file associated with the HYP file in Figure 28.

```
35 0A 30 2E 33 37 35 0A 2E 39 0A 2E 37 0A 2E 34 0A 2E 38 0A
```

Figure 33: Hexadecimal listing of the CON file shown in Figure 32.

4.4 Character-Based Scoring Output

Two character-based scoring output files are listed in Appendix E. The first file is a summary report listing the performance measures computed consistent with the draft standard. The second file is a fact sheet that gives a detailed accounting of all fundamental events accumulated by the Scoring Package. An explanation of how these files were generated is given in Section 5.5.2. These two output files use the same formats that are used for reporting the form-based scoring results described in Appendix C.

5. Software Documentation

5.1 Release Notes

Release 1.0 of the Scoring Package contains several changes in the format of files used for form-level scoring. These new file formats represent changes with respect to the files contained in the databases SD2 and SD6. SD2 identifies an ICON field that contains information with the string "_ICON_" in the reference file, and identifies void ICON fields by leaving the entry field value in the reference file empty. SD6 identifies ICON fields containing information with the value of '1' in the reference file and represents void ICON fields by leaving the entry field value in the reference file empty. Release 1.0 of the Scoring Package requires an entry field value of '1' to signify the presence of ICON information and a '0' to signify the absence of ICON information. This change allows proper recording of ICON field decisions, so that ICON fields can have associated confidence values and be rejected for both the presence of information and the absence of information in the field.

A second file format change is associated with the use of Continuation Alpha (CA) fields. CA fields were introduced for the purpose of form synthesis when modeling a single textual response potentially spanning more than one entry field on a form. Both SD2 and SD6 use the CA field type in their Table_A files, and CA field values were recorded in reference files in an unnecessarily complex way. Release 1.0 of the Scoring Package expects the numeric, textual, or ICON data contained in an entry field to be recorded on the corresponding entry field's line in the reference file with no exceptions. Using this convention, CA fields are scored identically to Alphanumeric (A) entry fields, so that the use of CA field types for scoring purposes is made obsolete.

Command utilities and a special option flag have been provided with this release of the Scoring Package to provide backward compatibility with old file formats. This is especially useful when scoring files from SD2 and SD6. Any scoring files created between now and the next release of the Scoring Package, should adhere strictly to the format guidelines outlined in this document.

5.2 Installation Procedures

The Scoring Package software is distributed as a combination of 'C' language source code and UNIX shell scripts. Installation instructions have also been provided on the CD-ROM in the file **install.txt** found in the directory **doc**. In order to run the Scoring Package, the source code must be compiled. The CD-ROM is a read-only media, requiring the source code to be copied to a read-writable file system prior to compilation. This section outlines the procedures necessary to install the software from CD-ROM. These instructions assume installation of the package in the directory /usr/local/score on a UNIX system running SunOS[1] 4.1.1. To install the software in another directory, change all occurrences of /usr/local/score to the absolute path name of the destination directory. These instructions assume the use of the UNIX C-Shell command interpreter "csh". Before every execution of "make", you should actually run "make -n" to see what commands will be executed and to verify that they make sense and won't harm any existing data.

The first installation step is to mount the CD-ROM containing the Scoring Package and copy the software to a read-writable file system:

```
# mount -v -t hsfs -o ro /dev/sr0 /mnt
# make -f /mnt/makefile.txt cdcopy CDROOT=/mnt IROOT=/usr/local/score
# umount -v /mnt
```

where CDROOT is the absolute path name of the CD-ROM file system, and IROOT is the absolute path name under which the Scoring Package should be copied and compiled. The default for CDROOT is /mnt. The default for IROOT is /usr/local/score. This step copies the entire contents from the CD-ROM into IROOT which will require approximately 5 megabytes after compilation. By installing in /usr/local/score, all executable commands for the Scoring Package will reside in /usr/local/score/bin.

The second installation step adds the Scoring Package's directory of executable commands to the execution path in the current shell:

```
# set path=( /usr/local/score/bin $path )
# rehash
```

[1] Certain commercial systems may be identified in order to adequately support the subject matter of this work. 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.

To add the Scoring Package's directory of executable commands to the execution path in future shells, edit ~/.cshrc:

```
# vi ~/.cshrc
```

and add /usr/local/score/bin to the list of directories assigned to the shell variable "path".

The third installation step compiles the provided source code and installs the resulting executable commands:

```
# cd /usr/local/score
# make clean
# make compile
# make install BINDIR=/usr/local/score/bin
# rehash
```

The Scoring Package software has been developed and is supported under SunOS 4.1.1. The software requires a multi-tasking operating system which has the ability to spawn sub-processes. The options to **merge** that spawn sub-processes are **reffilter**, **hypfilter**, **cnffilter**, **rejfilter**, **table_a_filter**, and **mrgfilter**. The options to **score** that spawn sub-processes are **zcat**, **tar**, and **ztar**. All of these options are described in the following section. Compilation on other UNIX systems may result in undefined functions. One possible solution is to check the GNU C library for source code corresponding to any undefined functions. GNU source code can be acquired via "ftp" from prep.ai.mit.edu [18.71.0.38] in the file /pub/gnu/glibc-1.04.tar.Z (or the latest version). If access to the network is not available, the source code may also be acquired by writing:

Free Software Foundation
675 Mass Ave
Cambridge, MA 02139
phone: 617/876-3296.

All users of the Scoring Package should register their copy of the package with NIST by sending their name, company, US Mail address, e-mail address, phone number, and the number of their package's release. To register via electronic mail, please send the requested information to scoring@magi.ncsl.nist.gov [129.6.48.150]. If access to the network is not available, please register via US Mail or FAX to:

Stanley A. Janet
NIST
225/A216
Gaithersburg, MD 20899
phone: 301/975-2916
FAX: 301/590-0932

Given this information, NIST can announce any software changes or new releases. Please send all problems, questions, and bugs via electronic mail to scoring-bugs@magi.ncsl.nist.gov. In addition to a complete description of the problem, identify the machine model and operating system under which the problem occurs. This software has been thoroughly tested by NIST on UNIX systems running SunOS 4.1.1. Any attempt to run this software on any other computer operating system is strictly the responsibility of the user.

5.3 Command Executions

This section describes the various utilities included with this release of the Scoring Package and specifies how they may be invoked. Three commands, **merge**, **score**, and **ocrmerge.sh** control the actual scoring process. Two other utilities, **convref** and **convhyp**, have been provided to handle outdated file formats discussed in the release notes above. Documentation on the use of these utilities is also included on the CD-ROM in the directory **doc**. Finally, documentation for IHead and MIS file utilities provided with this release is provided in Appendix H.

5.3.1 Merge

Merge is required by the form-based scoring process to create files, in a format called the merge file format, from a reference file, a hypothesis file, an optional confidence file, and zero or more optional rejection files. Dividing the scoring process into two parts separates the logic of the data input and data scoring tasks. **Merge** assembles the various scoring input files into a single file of a canonical form that can be processed by **score** and provides input file format checking and conversion while facilitating visual checking and manual creation of merge files for testing. Frequently, a single set of merge files is processed multiple times by **score**. Using the merge utility requires format checking and conversion to be conducted once, rather than every time **score** is invoked. Merge may be invoked as follows:

# merge [options] file . . .

Options:

-h	prints usage message and exits
-V	prints version and exits
-v	turns on verbose output
-o *merge-options*	selects merge options described below

The Scoring Package permits various file tree structures for storing the input files required for scoring. Therefore, a highly flexible input utility such as **merge** is required. The merge-options are specified on the command line using the "-o" option. Each option selected may be specified with a separate "-o" flag or as a list of options separated by commas associated with a single "-o" flag. These merge-options can be divided into four groups: input options, output options, debugging options, and miscellaneous options.

1. Input Options

explicit specifies that all files are named explicitly on the command line. This is the default. The command line must consist of one or more scoring sets, each set containing three or more file names. The first two files of a scoring set are the reference and hypothesis files. The final file of each set is the merge file. If a confidence file and/or rejection files are specified, then they must follow the hypothesis file with the confidence file first. The following C-Shell command sequence executes merge on two sets of files with arguments that specify the presence a confidence file and two rejection files per set. The resulting merge files "x.mrg" and "y.mrg" are then used by **score**.

```
# set xfiles = ( x.{fmt,hyp,con,rj1,rj2,mrg} )
# set yfiles = ( y.{fmt,hyp,con,rj1,rj2,mrg} )
# merge -o explicit,conf=c,nrej=2,formtypes $xfiles $yfiles
# score -s output=Ad x.mrg y.mrg
```

implicit instructs **merge** to generate file names from each argument on the command line, using each argument as a file's root name, appending a period and an extension, and optionally prefixing a directory path (unless the first character of the argument is a slash). **Merge** generates scoring files and merge files that reference the current working directory by default. To specify a different directory, the following assignments should be included in the merge-options on the command line:

refdir=*value*
hypdir=*value*
cnfdir=*value*
rejdir=*value*
mrgdir=*value*

In implicit mode, the extensions of generated files names may be specified and appended to the arguments on the command line. The default extensions are "fmt", "HYP", "CON", "REJ" and "mrg" respectively. To specify alternative extensions, the following assignments should be included in the merge-options on the command line:

refext=*value*
hypext=*value*
cnfext=*value*
rejext=*value*
mrgext=*value*

Using these flags, only one rejection file may be specified for each scoring set when the implicit mode is used. Therefore, the implicit mode merge command that corresponds to the explicit mode example above, except with only one rejection file specified per scoring set, would be:

```
merge -o implicit,conf=c,nrej=1,formtypes,hypext=hyp,cnfext=con,rejext=rj1 x y
```

nrej=*#* instructs **merge** to expect the specified number of rejection files on the command line for every scoring file set, where each scoring file set is comprised of at least a reference and hypothesis file pair. This option is shown in the explicit mode example above. The default is zero (0) rejection files per set.

conf=*value* instructs **merge** to expect a confidence file with every scoring file set when a value of 'c' is used. The default is a value of 'n' which instructs merge not to expect confidence files. This option is shown in the explicit mode example above.

table_a=*file* specifies the name of the Table_A file to be used in scoring.

no_table_a prohibits the use of any Table_A file.

table_a_dir=*value*

instructs **merge** to read the specified Table_A file from the given directory. The path specified for **table_a_dir** is appended to the file name specified for **table_a**, if the file name is a relative path. If the file name specified for **table_a** is a complete path beginning with the slash character, then the value specified for **table_a_dir** is ignored.

newca instructs **merge** that Continua Alpha (CA) fields will have any corresponding text after their field label. This is the default. Refer to the release notes for changes made relative to the use of CA fields.

oldca instructs **merge** to queue lines in reference files that begin with a tab and assign the first remaining line from the queue to any CA field subsequently encountered.

newicons instructs **merge** that the presence of ICON data is denoted in reference and hypothesis files by a '1', and that the absence of ICON data is denoted by a '0'. This is the default.

oldicons instructs **merge** that the presence of ICON data is denoted in reference and hypothesis files by the string "_ICON_" or a '1', and absence is denoted by either a '0' or no text following the field label.

newformats sets both "newca" and "newicons" flags. This is the default. Refer to the release notes for changes made relative to the file formats.

oldformats instructs **merge** to accept old formats for CA and ICON fields instead of the new formats. This option is required when scoring reference files from SD2 and SD6.

formtypes instructs **merge** to expect form types to be present on the first line of the scoring input files. Confidence files have a confidence value following the form type. Rejection files have a "0" or "1" following the form type denoting form type acceptance or rejection respectively. This option is shown in the explicit mode example above.

noformtypes instructs **merge** to assume there are no lines present in the scoring input files with form type hypotheses. This is the default. This option is useful when scoring results from systems that do not make form identifications. Using this option in conjunction with **formtypes** is not permitted.

formids is reserved for future use.

noformids is reserved for future use.

hyp_ignore instructs **merge** to ignore field data in the hypothesis, confidence and rejection files. For character fields, empty lines are written to the merge file. For ICON fields, zeroes are written instead.

reffilter=*cmd* specifies how a reference file should be filtered prior to being processed.

hypfilter=*cmd* specifies how a hypothesis file should be filtered prior to being processed.

cnffilter=*cmd* specifies how a confidence file should be filtered prior to being processed.

rejfilter=*cmd* specifies how a rejection file should be filtered prior to being processed.

table_a_filter=*cmd*

specifies how a Table_A file should be filtered prior to being processed. These filter options utilize the system call popen(3) so that multiple instances of the same filter type are appended to create a pipe. For example:

```
-o reffilter='sed s:foo:bar:',reffilter='tr A a'
```

is equivalent to:

```
-o reffilter='sed s:foo:bar: | tr A a'
```

The term *filter* represents any user-specified program (UNIX command, shell-script, or custom program) that reads from standard input, modifies the data stream in some specified way, and writes the modified data stream back out to standard output. These filter options are useful when on-line modifications and format changes to scoring input files are deemed useful and necessary. These options are not typically used for general scoring purposes when the input file formats and contents match the precise specifications of the Scoring Package.

skip=# instructs **merge** to skip the specified number of lines into the reference, hypothesis, confidence and rejection files. This option is useful when local headers on all input file types are used.

refskip=# instructs **merge** to skip the specified number of lines into reference files.

hypskip=# instructs **merge** to skip the specified number of lines into hypothesis files.

cnfskip=# instructs **merge** to skip the specified number of lines into confidence files.

rejskip=# instructs **merge** to skip the specified number of lines into rejection files.

2. Output Options

fillconf=*value* specifies that confidence values output should be derived from random values between 0.0 and 1.0 when the value is the string "random". If the value is a number, then the confidence values output should be the given number.

divider=*value* sets the divider in merge files to the given string. The default divider is the string "%%%%%".

mrgfilter=*cmd* specifies that the output going to the merge file is first piped through the given command. See the **reffilter** example above.

3. Debugging Options

nofree prevents memory obtained using malloc(3) and associated calls from being deallocated. Applies only to calls to score_free() in **merge**, not inside library calls.

quit stops processing of files on the first occurrence of an error. By default, **merge** proceeds to the next set of files on error.

mallocdb={0|1|2}

sets a debug level for calls to malloc(3). This option is compiled into **merge** if /usr/lib/debug/malloc.o exists (it does in SunOS 4.1.1). If the file does not exist, the option currently has no effect. See malloc_debug(3) for more information on the possible debug levels.

4. Miscellaneous Options

name=*value* overrides the program name used for example in error messages. This option is useful when **merge** is started up by other programs, such as in **ocrmerge.sh**, so that error messages will make better sense. The default value for the program name is the base-name of argv[0].

seed=# initializes the random number generator used by merge using the specified seed.

5. Notes and Caveats

Lines in any of the input files that begin with '#' are considered comments and are ignored. The skip operations are performed after any filtering, and do not include comment lines.

The implicit mode should not be used when more than one rejection file per reference file are being specified via "**nrej=#**".

The merge files created by a version of **merge** should only be scored by the corresponding version of **score**, as the internal merge file format may change.

5.3.2 Score

Score performs the entire scoring task and accepts both files and directories as legal arguments. If files are provided, they must be of the merge file format and created by the corresponding version of **merge**. Merge files are scored directly. If directories are provided, then by default, **score** searches the directories recursively for merge files. **Score** may be invoked as follows:

```
# score [options] { file | dir } . . .
```

Options:

-h[h[h]]	prints a short (h), medium (hh), or long (hhh) usage message and exits
-V	prints version and exits
-v	turns on verbose output
-A	selects options that modify default string alignment parameters described below
-o	selects options that modify global parameters described below
-s	selects options that create a scoring profile described below

There are three classes of options used by **score**: global options, scoring profile options, and alignment options. Global options control the basic actions of **score**. Scoring profile options create subsets of input data for a single execution of **score**. The alignment options control the way reference and hypothesis character strings are aligned. The algorithm employed by **score** chooses the minimum penalty alignment as found from a modified Levenstein distance algorithm[12]. Given these classes of options, global options can be specified separately (-o <option1> -o <option2>) or together in a comma-separated list (-o <option1>,<option2>). Likewise, alignment options can be specified separately (-A<option1> -A<option2>) or together in a comma-separated list (-A<option1>,<option2>). Scoring profile options differs in that each instance of a comma-separated list of options specified with a unique "-s" flag represents a separate scoring profile.

1. Global Options

name=*value* overrides the program name used for example in error messages. This option is useful when **score** is started up by other programs, such as in **ocrmerge.sh**, so that error messages will make better sense. The default value for the program name is the base-name of argv[0].

zcat instruct **score** to assume files with the extension ".Z" are compressed files. Files having this extension are decompressed prior to any subsequent processing. This option is useful for processing large numbers of merge files that need to be compressed in order to save disk space.

tar instructs **score** to assume files with the extension ".tar" are tar archives. Tar archives have all their contents extracted in a temporary directory. That directory is processed as if it appeared on the command line, then removed. This option is useful for processing large numbers of merge files that need to be bundled in order to save file system nodes.

ztar instructs **score** to assume files with the extension ".tar.Z" are compressed tar archives. Files with this extension are first decompressed and then handled the same way tar archives using the **tar** option are processed. This option is useful in saving both disk space and file system nodes.

recurse turns on recursive directory processing, which forces **score** to process all files contained in a directory that is either a specified command line argument or a temporary directory that tar archives are extracted into. By default, directories are searched recursively.

norecurse turns off recursive directory processing. Only the files in a directory will be processed, not those in any subdirectories.

nocase instructs score to consider case mismatches to be correct.

case instructs score to consider case mismatches to be substitutions. This is the default.

nowhite instructs score to ignore the reported recognition of spaces by removing blanks and tabs from reference strings and from hypothesis strings. Any confidence and/or reject values corresponding to a reported space are also removed.

yesicon=*string* overrides the string denoting the presence of ICON data in reference and hypothesis files. The default string signifying the presence of ICON data is "1".

noicon=*string* overrides the string denoting the absence of ICON data in reference and hypothesis files. The default string signifying the absence of ICON data is "0".

nofree prevents memory obtained using malloc(3) and associated calls from being deallocated. This option applies only to calls to score_free() in **score**, not inside library calls.

quit stops the scoring of files on the first occurrence of an error. By default, **score** proceeds to the next merge file on error.

mallocdb={0|1|2}

sets a debug level for calls to malloc(3). This option is compiled into **score** if /usr/lib/debug/malloc.o exists (it does in SunOS 4.1.1). If the file does not exist, the option currently has no effect. See malloc_debug(3) for more information on the possible debug levels.

linebuf buffers lines being printed to standard output. This option is useful for watching **score's** progress or for debugging if **score** prematurely exits with buffers not being flushed.

prdatasize instructs **score** to produce output for monitoring the size of its data segment. The output is printed before and after allocating space for all scoring profiles, and after every file is processed. The value printed at each point is the pointer to the start of the program's data space. This option is useful for watching how much memory has been allocated and not freed. See sbrk(2).

maxfiles=# instructs **score** to stop processing after the specified number of merge files have been scored. This option is useful for getting sample numbers from a fraction of the merge files in a large set that is conveniently accessed via wild-carding. For example:

```
# score ... -o maxfiles=50 data/*.mrg
```

2. Scoring Profile Options

Scoring profile options create subsets of input data to be processed during a single execution of **score**. For example, if two rejection files are provided for a set of input scoring files during the merge phase, the alignments can be scored using both rejection files by creating two scoring profiles, one profile specifying the first rejection file should be applied and the second profile specifying the second rejection file should be applied. An example of this follows:

```
# set opts = output=FCd
# score -s ${opts},rejline=1,of=r1.out -s ${opts},rejline=2,of=r2.out x.mrg y.mrg
```

Notice the two "-s" option sequences which represent two different scoring profiles. At least one scoring profile must be present on **score's** command line. Adding more profiles will require more calculations and will increase the amount of memory utilized by **score**. The maximum number of scoring profiles that can be created is limited only by the amount of available memory.

In the options listed below, a leading exclamation point inverts the option, a plus sign leading a number changes an equality test to a greater than test, and a minus sign leading a number changes an equality test to a less than test. The options that use the term "character field" have no effect on ICON fields, which are currently always scored (unless ignored based on field number).

sel=[!]*rangelist* selects fields based on their number within their files. A range list is one or more slash-separated range specifications, where a range specification is a single number representing a field number, or two dash-separated numbers representing a range of field numbers. Field number are 1-oriented. The following are all legal range lists:

1 (first field)
3-5 (third, fourth and fifth fields)
1/3-5/10 (first, third, fourth, fifth and tenth fields)

formtype=[!]*name*

scores only the fields on forms of the specified type. Fields from forms whose type was not specified during the invocation of **merge** are not scored.

fieldtype=[!]*name*

scores only the fields of the specified entry field type as defined by a corresponding Table_A file. Fields from forms where no Table_A file was specified during the invocation of **merge** are not scored.

fieldcontext=[!]*name*

scores only the fields having the specified context label as defined by a corresponding Table_A file. Fields from forms where no Table_A file was specified during the invocation of merge are not scored.

lencmp=*test* scores only those character fields where the reference string length is equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal to the hypothesis string length. These conditional tests may be specified from the corresponding list:

eq, neq, lt, le, gt, ge

astr=[!]*regex* scores only character fields with alignment strings matching the provided regular expression. The alignment string consists of the following characters representing, respectively, character matches, substitutions, insertions and deletions in the alignment:

'-', 'S', 'I', 'D'

Therefore, **astr=**'^S' would select fields that begin with a substitution error. See regex(3) for information on the syntax of regular expressions.

refstr=[!]*regex* scores only character fields with reference strings matching the provided regular expression.

hypstr=[!]*regex* scores only character fields with hypothesis strings matching the provided regular expression.

alen=[!|+|-]# scores only character fields with alignment lengths equal to the provided number.

rlen=[!|+|-]# scores only character fields with reference lengths equal to the provided number.

hlen=[!|+|-]# scores only character fields with hypothesis lengths equal to the provided number.

charfields scores all character fields in the scoring profile.

nocharfields scores no character fields in the scoring profile.

ldist=[+|-]# scores all character fields that, based on string alignments with rejections ignored, have a Levenstein distance equal to the number provided.

right scores all character fields that, based on string alignments with rejections ignored, have no substitutions, insertions, or deletions.

wrong scores all character fields that, based on string alignments with rejections ignored, have at least one substitution, insertion, or deletion.

nok=[+|-]# scores all character fields that, based on string alignments with rejections ignored, have a total number of correct characters equal to the provided number.

nerr=[+|-]# scores all character fields that, based on string alignments with rejections ignored, have a total number of substitutions, insertions, and deletions equal to the provided number.

nsub=[+|-]# scores all character fields that, based on string alignments with rejections ignored, have a total number of substitutions equal to the provided number.

nins=[+|-]# scores all character fields that, based on string alignments with rejections ignored, have a total number of insertions equal to the provided number.

ndel=[+|-]# scores all character fields that, based on string alignments with rejections ignored, have a total number of deletions equal to the provided number.

rejthr=# instructs **score** to ignore provided rejection data and generate its own by rejecting hypothesized characters whose corresponding confidence values are greater than or equal to the specified threshold value.

rejline=# instructs **score** to use a specific set of rejection data. The default is to use the data from the first rejection file included in each set of scoring files during the merge phase. The example shown at the beginning of the scoring profile section above demonstrates the use of this option.

tmin=# specifies the minimum time between outputs (in seconds) as outlined in the draft standard.

peak_tput=# specifies the peak throughput (in outputs per second) as outlined in the draft standard.

tresp=# specifies the average time for the first response (in seconds) as outlined in the draft standard.

tav_base=# specifies the base time for which the system was available (in hours) as outlined in the draft standard.

tunavail=# specifies the time the system was unavailable (in hours) as outlined in the draft standard.

output=*flags* specifies what information is printed in the scoring summary. The available output flags are:

all	(selects all output flags)
none	(selects no output flags)
F	(selects character fields)
I	(selects ICON fields)
C	(selects characters)
t	(selects form types)
A	(selects alignments)
d	(selects miscellaneous draft standard measures)

The second flag, "none", selects no other output flags, which is the default, and is useful for checking the formats of merge files and checking for errors internal to **score**. Multiple occurrences of the flag 'A' causes only alignments with errors, ignoring all rejections, to be printed instead of printing every alignment in the scoring profile. If both 'F' and 'I' are specified, then the scoring results from character fields and ICON fields are combined when computing performance measures.

of=*file* sends scoring summary output including the fundamental accumulators to the specified file. The default is to send the scoring summary to standard output. It is recommended that output be sent to files when more than one scoring profile is specified.

af=*file* sends selected alignments to the specified file. The default is to send the alignments to the same place that the scoring summary output is sent.

cf=*file* sends basic scoring counts including fundamental accumulators to the specified file. By default, this data is not printed.

linebuf buffers lines output to any files specified by the above options.

unbuf does not buffer lines output to any files specified by the above options.

3. Alignment Options

case treats case mismatches as substitutions during the alignment. This is the default.

nocase treats case mismatches as correct identifications during the alignment.

dir=*right* aligns hypotheses to the right in case of ties. Ties can occur when more than one possible alignment have the same minimum penalty. This option favors insertions during backtracking and is the default.

dir=*left* aligns hypotheses to the left in case of ties. This option favors deletions during backtracking.

4. Notes and Caveats

Lines in any merge files beginning with "#" are considered comments and are ignored.

If no rejection values are provided or created from confidence values from a specified threshold, output that is a function of rejections should be assumed to have been generated from no rejections.

If no confidence values are provided, output that is a function of confidence should be assumed to have been generated from full confidence.

5.3.3 Ocrmerge.sh

Ocrmerge.sh is a front-end to **merge** implemented as Bourne Shell script and is required for character-based scoring. This utility takes character-based scoring files (CLS, HYP, CON, and RJX) and creates merge files which then can be processed by score. Ocrmerge.sh may be invoked as follows:

```
# ocrmerge.sh [merge-options] file . . .
```

This script executes **merge** with options that instruct it to filter the character-based scoring files into the form-based scoring file formats prior to creating merge files. The script executes the following command sequence:

```
XEQ=merge
exec ${XEQ} \
    -o name=ocrmerge.sh \
    -o no_table_a \
    -o noformtypes \
    -o noformids \
    -o reffilter=ocrreff.sh \
    -o hypfilter=ocrhypf.sh \
    -o cnffilter=ocrcnff.sh \
    -o rejfilter=ocrrejf.sh \
    “$@”
```

Ocrmerge.sh passes along all arguments on its command line to **merge**. Therefore, for example, either explicit or implicit merge modes can still be used.

5.3.4 Convref and Convhyp

Convref converts obsolete form-based reference file formats to the reference file format current with this release of the Scoring Package. **Convhyp** converts obsolete form-based hypothesis file formats to the hypothesis file format current with this release of the Scoring Package. Prior to conversion, the files provided on the command line are copied to back-up files using the <file>~ naming convention. The old format files are then overwritten by the new format files.

One old format, used in SD2, called for the presence of ICON data to be specified by the string “_ICON_” following the entry field label. The current format calls for a ‘1’ to be used to specify the presence of ICON data in a field. All old formats, used in SD2 and SD6, called for the absence of ICON data to be specified by an empty entry field value. The current format calls for a ‘0’ to be used to specify the absence of ICON data in a field. All old formats, also used in SD2 and SD6, called for lines beginning with a tab character to be queued and substituted for entry field values whenever Continuation Alpha (CA) fields were incurred. The current format does not recognize tabbed lines and requires all entry field values to be included on the same line as their associated entry field identifications regardless of the type of field. **Convref** and **Convhyp** may be invoked as follows:

```
# convref [options] file . . .
# convhyp [options] file . . .
```

```
Options:
    -h          prints a usage message and exits
    -V          prints version and exits
    -v          turns on verbose output
    -t table_A       specifies an associated Table_A file
```

5.4 Fundamental Accumulators

The summary output generated by the Scoring Package contains a section which reports the values compiled for a set of fundamental accumulators tallied at the character level across each of the four forms processing tasks listed in Figure 2. These accumulators are listed in Figure 34. TP represents the total number of correct character classifications assigned by the system prior to any rejections. FP represents the total number of combined substitution and insertion errors made by the system prior to any rejections. M represents the total number of deletion errors made by the system within the character recognition task plus the total number of characters missed due to identification errors at either the form or field identification tasks. Notice that M is not included in the fundamental accumulator FP. RT is the subset of correct classifications in TP which are rejected by the system. RF is the subset of incorrect classification in FP which are rejected by the system. RM represents the total number of characters missed due to form or fields being rejected.

TP	Correct Character Classifications
FP	Substituted and Inserted Characters
M	Deleted and Missed Characters
RT	Correct Character Classifications Rejected
RF	Substituted and Inserted Characters Rejected
RM	Missed Characters Rejected

Figure 34: Fundamental accumulators reported by the Scoring Package.

Using these fundamental accumulators, the number of correct character classifications remaining after rejection is equal to (TP - RT), and the number of incorrect character classifications and classifications of non-character images after rejection is equal to (FP - RF). The total number of rejected characters is equal to (RT + RF + RM). These accumulators are used to compute the performance measures defined in the draft standard on evaluating character recognition systems.

5.5 Provided Data and Examples

Three groups of example scoring files have been provided with this release of the Scoring Package. All three collections are stored in the directory **data** on the CD-ROM. The data directory **form** contains images and scoring files for a collection of IRS tax forms, the data directory **char** contains a collection of referenced images of isolated characters and associated scoring files, and the data directory **test** contains scoring files for a collection of simulated test forms.

5.5.1 Form-Based Scoring

The data directory **form** contains referenced images and scoring files for 11 IRS tax forms, each of which are stored in their own subdirectory, **r0000** to **r0010**. The form images are in the IHead format which is defined in Appendix F and stored in files having an extension of "pct". The data entered on these forms has been derived by a computer so that the forms do not contain real tax data. The first five forms, **r0000** to **r0004**, are the first page of a 1988 1040 form completed with machine-print and are similar to the forms images distributed in SD2. The second five forms, **r0005** to **r0009**, are the same form face completed with hand-print and are similar to the forms image distributed in SD6. The final form, **r0010**, is a 1988 Schedule A form completed with hand-print. A set of reference, hypothesis, rejection, and confidence files are stored with each form image and have the extensions "fmt", "hyp", "con", and "rej" respectively. The hypothesis, rejection, and confidence files represent plausible system results from a fictitious forms processing system. This collection of scoring files has been provided so that a user of the Scoring Package may experiment with the package's form-based scoring capabilities.

These form-based scoring input files were used by the Scoring Package to produce the example output files **form.sum** and **form.fct** in the directory **form**. These two output files are included in Appendix C and represent results obtained from scoring all 11 tax forms collectively. The intermediate merge files used by **score** are included in each of the corresponding form subdirectories and have the extension "mrg". For example, the following **merge** options were used to create the merge file in **r0000**:

```
# merge -o formtypes,conf=c,nrej=1 -o table_a=tabels/1040_1.tab r0000/r0000_00.{fmt,hyp,con,rej,mrg}
```

Upon creating the merge files, the following **score** options were used to create the two scoring output files:

```
# score -o nowhite -s output=FCItdA, of=form.sum,cf=form.fct r????/r????_??.mrg
```

5.5.2 Character-Based Scoring

The data directory **char** contains referenced images and scoring files for a collection of isolated characters similar to those distributed in SD3. The directory **d0000** contains an MIS file of 20 isolated images of digits, the directory **u0000** contains an MIS file of 20 isolated images of upper-case letters, and the directory **l0000** contains an MIS file of 20 isolated images of lower-case letters. The MIS file format is defined in Appendix G, and the MIS files are stored in files having an extension of "mis". A set of classification, hypothesis, rejection, and confidence files are stored with each MIS file and have the extensions "cls", "hyp", "con", and "rj0" respectively. This collection of scoring files has been provided so that a user of the Scoring Package may experiment with the package's character-based scoring capabilities.

These character-based scoring files were used by the Scoring Package to produce example output files which have the extension "sum" and "fct" stored in each of the three subdirectories in **char**. For example, the two output files **l0000.sum** and **l0000.fct** in **l0000**, which are included in Appendix E, represent scores obtained from the scoring input files shown in Appendix D. The intermediate merge file use by **score** is included in the file **l0000.mrg** and was created using the following to **ocrmerge.sh**:

```
# ocrmerge.sh -o conf=c,nrej=1 -o implicit,refext=cls,hypext=hyp,cnfext=con,rejext=rj0,mrgext=mrg l0000/l0000
```

Upon creating the merge files, the following **score** options were used to create the two scoring output files:

```
# score -s output=FCItdAA, of=l0000/l0000.sum,cf=l0000/l0000.fct l0000/l0000.mrg
```

5.5.3 Installation Testing

Once the Scoring Package has been installed, the provided form-based and character-based scoring files can be used to produce locally derived scoring outputs. The new output files can be compared with those output files distributed on the CD-ROM in order to determine if the installation of the Scoring Package has been successful. A third set of scoring files has been included with this release of the Scoring Package and is found in the data directory **test**.

Each of the directories in **test** contains a simulated form reference file and simulated system results (hypothesis, confidence, and rejection files) that have been designed to test specific logic and format aspects of the Scoring Package. The scoring outputs generated by the Scoring Package for each of these test forms have been stored in files which have the extension "sum" and "fct". The scoring output files included in **test's** subdirectories were created similarly to the form-based scoring example above except that each test form was scored independently rather than collectively, and the test form in **of1** requires the use of **oldformats** when invoking **merge**. These tests forms can be used to determine if the installation of the Scoring Package was successful, but more importantly, they can be used to validate the porting of the Scoring Package to other architectures in which attention to implementation details is critical.

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

[3] D. L. Dimmick and M. D. Garris. Structured Forms Database 2, Technical Report Special Database 2, **SFRS2**, National Institute of Standards and Technology, September 1992.

[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] R. A. Wilkinson. Handprinted segmented characters database. Technical Report Test Database 1, **TST1**, National Institute of Standards and Technology, April 1992.

[6] M. D. Garris, et al. Massively parallel implementation of character recognition systems. In *Conference on Character Recognition and Digitizer Technologies*, volume 1661, pages 269-280, San Jose California, February 1992. SPIE.

[7] H. P. Graf, C. Nohl, and J. Ben. Image segmentation with networks of variable scale. J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume IV, pages 480-487. Morgan Kaufmann, Denver, December 1991.

[8] M. D. Garris and C. L. Wilson. A neural approach to concurrent character segmentation and recognition. In *Southcon 92 Conference Record*, pages 154-159, Orlando, March 1992. IEEE.

[9] G. L. Martin. Centered-object integrated segmentation and recognition for visual character recognition. J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume IV, pages 504-511. Morgan Kaufmann, Denver, December 1991.

[10] J. D. Keeler and D. E. Rumelhart. Self-organizing segmentation and recognition neural network. J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume IV, pages 496-503. Morgan Kaufmann, Denver, December 1991.

[11] R. A. Wilkinson, et al. The first Census optical character recognition system conference. Technical Report NISTIR 4912, National Institute of Standards and Technology, July 1992.

[12] H. G. Zwakenberg. Inexact Alphanumeric Comparison. *The C Users Journal*, pages 127-131. May 1991.

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

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

Appendix A: Form Template Files

Form **1040** Department of the Treasury—Internal Revenue Service

U.S. Individual Income Tax Return 1988 (0)

1040_1_L_H3_V1

For the year Jan.-Dec. 31, 1988, or other tax year beginning 1040_1_L_H1_V1, 1988, ending 1040_1_L_H2_V1, 19 | OMB No. 1545-0074

Label

Use IRS label. Otherwise, please print or type.

LABEL HERE

Your first name and initial (if joint return, also give spouse's name and initial) Last name
1040_1_L_H1_V2

Your social security number
1040_1_L_H2_V2

Present home address (number, street, and apt. no. or rural route). (If a P.O. Box, see page 6 of Instructions.)
1040_1_L_H1_V3

Spouse's social security number
1040_1_L_H2_V3

City, town or post office, state, and ZIP code
1040_1_L_H1_V4 1040_1_L_H2_V5

For Privacy Act and Paperwork Reduction Act Notice, see Instructions.

Presidential Election Campaign

Do you want $1 to go to this fund? . . . 1040_1_L_H1_V5 □ Yes □ No

If joint return, does your spouse want $1 to go to this fund? . . □ Yes □ No

1040_1_L_H1_V6 1040_1_L_H2_V6

Note: Checking "Yes" will not change your tax or reduce your refund.

Filing Status

Check only one box.

1040_1_1 1 □ Single 1040_1_2

2 □ Married filing joint return (even if only one had income)

1040_1_3_H1 3 □ Married filing separate return. Enter spouse's social security no. above and full name here. 1040_1_3_H2

1040_1_4_H1 4 □ Head of household (with qualifying person). (See page 7 of Instructions.) If the qualifying person is your child but not your dependent, enter child's name here. 1040_1_4_H2

1040_1_5_H1 5 □ Qualifying widow(er) with dependent child (year spouse died ▶ 19). (See page 7 of Instructions.) 1040_1_5_H2

Exemptions

(See Instructions on page 8.)

1040_1_6a 6a □ Yourself If someone (such as your parent) can claim you as a dependent, do not check box 6a. But be sure to check the box on line 33b on page 2. . . . .

1040_1_6b_H1 b □ Spouse . . . .

No. of boxes checked on 6a and 6b 1040_1_6b_H2

c Dependents:

(1) Name (first, initial, and last name)	(2) Check if under age 5	(3) If age 5 or older, dependent's social security number	(4) Relationship	(5) No. of months lived in your home in 1988
1040_1_6c_H1_V1	H2	: H3 :	H4	H5
1040_1_6c_H1_V2	H2	: H3 :	H4	H5
1040_1_6c_H1_V3	H2	: H3 :	H4	H5
1040_1_6c_H1_V4	H2	: H3 :	H4	H5
1040_1_6c_H1_V5	H2	: H3 :	H4	H5
1040_1_6c_H1_V6	H2	: H3 :	H4	H5

If more than 6 dependents, see Instructions on page 8.

No. of your children on 6c who:
- lived with you 1040_1_6c_H6_V1
- didn't live with you due to divorce or separation 1040_1_6c_H6_V2

No. of other dependents listed on 6c 1040_1_6c_H6_V3

d If your child didn't live with you but is claimed as your dependent under a pre-1985 agreement, check here . ▶ □

e Total number of exemptions claimed . . . . 1040_1_6d

Add numbers entered on lines above ▶ 1040_1_6e

Income

Please attach Copy B of your Forms W-2, W-2G, and W-2P here.

If you do not have a W-2, see page 6 of Instructions.

Please attach check or money order here.

7	Wages, salaries, tips, etc. *(attach Form(s) W-2)*	7	1040_1_7
8a	Taxable interest income *(also attach Schedule B if over $400)*	8a	1040_1_8a
b	Tax-exempt interest income (see page 11). DON'T include on line 8a 8b 1040_1_8b		
9	Dividend income *(also attach Schedule B if over $400)*	9	1040_1_9
10	Taxable refunds of state and local income taxes, if any, from worksheet on page 11 of Instructions .	10	1040_1_10
11	Alimony received	11	1040_1_11
12	Business income or (loss) *(attach Schedule C)*.	12	1040_1_12
13	Capital gain or (loss) *(attach Schedule D)*	13	1040_1_13
14	Capital gain distributions not reported on line 13 (see page 11)	14	1040_1_14
15	Other gains or (losses) *(attach Form 4797)*	15	1040_1_15
16a	Total IRA distributions . . 16a 1040_1_16a 16b Taxable amount (see page 11)	16b	1040_1_16b
17a	Total pensions and annuities 17a 1040_1_17a 17b Taxable amount (see page 12)	17b	1040_1_17b
18	Rents, royalties, partnerships, estates, trusts, etc. *(attach Schedule E)*	18	1040_1_18
19	Farm income or (loss) *(attach Schedule F)*	19	1040_1_19
20	Unemployment compensation (insurance) (see page 13)	20	1040_1_20
21a	Social security benefits (see page 13) . . . . . . 21a 1040_1_21a		
b	Taxable amount, if any, from the worksheet on page 13	21b	1040_1_21b
22	Other income (list type and amount—see page 13) 1040_1_22_H1	22	1040_1_22_H2
23	Add the amounts shown in the far right column for lines 7 through 22. This is your total income . ▶	23	1040_1_23

Adjustments to Income

(See Instructions on page 13.)

24	Reimbursed employee business expenses from Form 2106, line 13.	24	1040_1_24		
25a	Your IRA deduction, from applicable worksheet on page 14 or 15	25a	1040_1_25a		
b	Spouse's IRA deduction, from applicable worksheet on page 14 or 15	25b	1040_1_25b		
26	Self-employed health insurance deduction, from worksheet on page 15 .	26	1040_1_26		
27	Keogh retirement plan and self-employed SEP deduction. . .	27	1040_1_27		
28	Penalty on early withdrawal of savings	28	1040_1_28		
29	Alimony paid (recipient's last name 1040_1_29_V1 and social security no. :1040_1:29_H1_V2) .	29	1040_1_29_H2_V2		
30	Add lines 24 through 29. These are your total adjustments . . . . . . ▶			30	1040_1_30

Adjusted Gross Income

31	Subtract line 30 from line 23. This is your adjusted gross income. *If this line is less than $18,576 and a child lived with you, see "Earned Income Credit" (line 56) on page 19 of the Instructions. If you want IRS to figure your tax, see page 16 of the Instructions* . . . ▶	31	1040_1_31

```
1040_1_L_H1_V1 A DATA
1040_1_L_H2_V1 A DATA
1040_1_L_H3_V1 A DATA
1040_1_L_H1_V2 A NAME
1040_1_L_H2_V2 A SSN
1040_1_L_H1_V3 A DATA
1040_1_L_H2_V3 A SSN
1040_1_L_H1_V4 A DATA
1040_1_L_H1_V5 ICON DATA
1040_1_L_H2_V5 ICON DATA
1040_1_L_H1_V6 ICON DATA
1040_1_L_H2_V6 ICON DATA
1040_1_1 ICON DATA
1040_1_2 ICON DATA
1040_1_3_H1 ICON DATA
1040_1_3_H2 A NAME
1040_1_4_H1 ICON DATA
1040_1_4_H2 A NAME
1040_1_5_H1 ICON DATA
1040_1_5_H2 A DATA
1040_1_6a ICON DATA
1040_1_6b_H1 ICON DATA
1040_1_6b_H2 I DATA
1040_1_6c_H1_V1 A NAME
1040_1_6c_H2_V1 ICON DATA
1040_1_6c_H3_V1 A SSN
1040_1_6c_H4_V1 A DATA
1040_1_6c_H5_V1 I DATA
1040_1_6c_H6_V1 I DATA
1040_1_6c_H1_V2 A NAME
1040_1_6c_H2_V2 ICON DATA
1040_1_6c_H3_V2 A SSN
1040_1_6c_H4_V2 A DATA
1040_1_6c_H5_V2 I DATA
1040_1_6c_H6_V2 I DATA
1040_1_6c_H1_V3 A NAME
1040_1_6c_H2_V3 ICON DATA
1040_1_6c_H3_V3 A SSN
1040_1_6c_H4_V3 A DATA
1040_1_6c_H5_V3 I DATA
1040_1_6c_H6_V3 I DATA
1040_1_6c_H1_V4 A NAME
1040_1_6c_H2_V4 ICON DATA
1040_1_6c_H3_V4 A SSN
1040_1_6c_H4_V4 A DATA
1040_1_6c_H5_V4 I DATA
```

```
1040_1_6c_H1_V5 A NAME
1040_1_6c_H2_V5 ICON DATA
1040_1_6c_H3_V5 A SSN
1040_1_6c_H4_V5 A DATA
1040_1_6c_H5_V5 I DATA
1040_1_6c_H1_V6 A NAME
1040_1_6c_H2_V6 ICON DATA
1040_1_6c_H3_V6 A SSN
1040_1_6c_H4_V6 A DATA
1040_1_6c_H5_V6 I DATA
1040_1_6d ICON DATA
1040_1_6e I DATA
1040_1_7 F DATA
1040_1_8a F DATA
1040_1_8b F DATA
1040_1_9 F DATA
1040_1_10 F DATA
1040_1_11 F DATA
1040_1_12 F DATA
1040_1_13 F DATA
1040_1_14 F DATA
1040_1_15 F DATA
1040_1_16a F DATA
1040_1_16b F DATA
1040_1_17a F DATA
1040_1_17b F DATA
1040_1_18 F DATA
1040_1_19 F DATA
1040_1_20 F DATA
1040_1_21a F DATA
1040_1_21b F DATA
1040_1_22_H1 A DATA
1040_1_22_H2 F DATA
1040_1_23 F DATA
1040_1_24 F DATA
1040_1_25a F DATA
1040_1_25b F DATA
1040_1_26 F DATA
1040_1_27 F DATA
1040_1_28 F DATA
1040_1_29_V1 A NAME
1040_1_29_H1_V2 A SSN
1040_1_29_H2_V2 F DATA
1040_1_30 F DATA
1040_1_31 F DATA
```

Figure 35: Listing of a Table_A file corresponding to the form template on the previous page.

Appendix B: Form-Based Files

Form **1040** Department of the Treasury—Internal Revenue Service
U.S. Individual Income Tax Return **1988** 00

For the year Jan.-Dec. 31, 1988, or other tax year beginning July , 1988, ending July , 1988 OMB No. 1545-0074

Label

Use IRS label. Otherwise, please print or type.

LABEL HERE

Your first name and initial (if joint return, also give spouse's name and initial) Last name
Brainerd A. & Erskine W. Mitchell

Your social security number A1 : 88 : 1304

Present home address (number, street, and apt. no. or rural route). (If a P.O. Box, see page 6 of Instructions.)
99225 Lee Street

Spouse's social security number A59 : 02 : 1948

City, town or post office, state, and ZIP code
Russell, NJ 61920

For Privacy Act and Paperwork Reduction Act Notice, see Instructions.

Presidential Election Campaign ▶ Do you want $1 to go to this fund? ☒ Yes ☐ No
If joint return, does your spouse want $1 to go to this fund? ☒ Yes ☐ No
Note: Checking "Yes" will not change your tax or reduce your refund.

Filing Status

Check only one box.

1 ☒ Single
2 ☐ Married filing joint return (even if only one had income)
3 ☐ Married filing separate return. Enter spouse's social security no. above and full name here. ______
4 ☐ Head of household (with qualifying person). (See page 7 of Instructions.) If the qualifying person is your child but not your dependent, enter child's name here. ______
5 ☐ Qualifying widow(er) with dependent child (year spouse died ▶ 19). (See page 7 of Instructions.)

Exemptions

(See Instructions on page 8.)

6a ☒ Yourself If someone (such as your parent) can claim you as a dependent, do not check box 6a. But be sure to check the box on line 33b on page 2.
b ☐ Spouse

No. of boxes checked on 6a and 6b 1

c Dependents:

(1) Name (First, initial, and last name)	(2) Check if under age 5	(3) If age 5 or older, dependent's social security number	(4) Relationship	(5) No. of months lived in your home in 1988
Rider Harlan		A97 : 20 : 3766	Aunt	6
Tulane Banks		A97 : 08 : 19045	S-Law	12
Hunter Bell		A39 : 26 : 756	Da-Law	2

If more than 6 dependents, see Instructions on page 8.

No. of your children on 6c who:
- lived with you 8
- didn't live with you due to divorce or separation ______

No. of other dependents listed on 6c ______

d If your child didn't live with you but is claimed as your dependent under a pre-1985 agreement, check here ▶ ☐
e Total number of exemptions claimed

Add numbers entered on lines above ▶ 7

Income

Please attach Copy B of your Forms W-2, W-2G, and W-2P here.

If you do not have a W-2, see page 6 of Instructions.

Please attach check or money order here.

7	Wages, salaries, tips, etc. *(attach Form(s) W-2)*	7	3878
8a	Taxable interest income *(also attach Schedule B if over $400)*	8a	
b	Tax-exempt interest income (see page 11). DON'T include on line 8a 8b		
9	Dividend income *(also attach Schedule B if over $400)*	9	
10	Taxable refunds of state and local income taxes, if any, from worksheet on page 11 of Instructions	10	
11	Alimony received	11	
12	Business income or (loss) *(attach Schedule C)*	12	0
13	Capital gain or (loss) *(attach Schedule D)*	13	
14	Capital gain distributions not reported on line 13 (see page 11)	14	
15	Other gains or (losses) *(attach Form 4797)*	15	
16a	Total IRA distributions 16a ______ 16b Taxable amount (see page 11)	16b	
17a	Total pensions and annuities 17a ______ 17b Taxable amount (see page 12)	17b	
18	Rents, royalties, partnerships, estates, trusts, etc. *(attach Schedule E)*	18	
19	Farm income or (loss) *(attach Schedule F)*	19	
20	Unemployment compensation (insurance) (see page 13)	20	
21a	Social security benefits (see page 13) 21a		
b	Taxable amount, if any, from the worksheet on page 13	21b	
22	Other income (list type and amount—see page 13) Travel allowance	22	0
23	Add the amounts shown in the far right column for lines 7 through 22. This is your **total income** ▶	23	3878

Adjustments to Income

(See Instructions on page 13.)

24	Reimbursed employee business expenses from Form 2106, line 13	24	
25a	Your IRA deduction, from applicable worksheet on page 14 or 15	25a	2574
b	Spouse's IRA deduction, from applicable worksheet on page 14 or 15	25b	
26	Self-employed health insurance deduction, from worksheet on page 15	26	
27	Keogh retirement plan and self-employed SEP deduction	27	
28	Penalty on early withdrawal of savings	28	
29	Alimony paid (recipient's last name ______ and social security no. ___ : ___ : ___)	29	
30	Add lines 24 through 29. These are your **total adjustments** ▶	30	2574

Adjusted Gross Income

31 Subtract line 30 from line 23. This is your **adjusted gross income**. *If this line is less than $18,576 and a child lived with you, see "Earned Income Credit" (line 56) on page 19 of the Instructions. If you want IRS to figure your tax, see page 16 of the Instructions* ▶ 31 1303

```
1040_1
1040_1_L_H1_V1 July
1040_1_L_H2_V1 July
1040_1_L_H3_V1 88
1040_1_L_H1_V2 Brainerd A. & Erskine W. Mitchell
1040_1_L_H2_V2 A11 88 1304
1040_1_L_H1_V3 99225 Lee Street
1040_1_L_H2_V3 A59 02 1948
1040_1_L_H1_V4 Russell, NJ 61920
1040_1_L_H1_V5 1
1040_1_L_H2_V5 0
1040_1_L_H1_V6 1
1040_1_L_H2_V6 0
1040_1_1 1
1040_1_2 0
1040_1_3_H1 0
1040_1_3_H2
1040_1_4_H1 0
1040_1_4_H2
1040_1_5_H1 0
1040_1_5_H2
1040_1_6a 1
1040_1_6b_H1 0
1040_1_6b_H2 1
1040_1_6c_H1_V1 Rider Harlan
1040_1_6c_H2_V1 0
1040_1_6c_H3_V1 A97 20 3760
1040_1_6c_H4_V1 Aunt
1040_1_6c_H5_V1 6
1040_1_6c_H6_V1 8
1040_1_6c_H1_V2 Tulane Banks
1040_1_6c_H2_V2 0
1040_1_6c_H3_V2 A97 08 1904
1040_1_6c_H4_V2 Si-Law
1040_1_6c_H5_V2 12
1040_1_6c_H6_V2
1040_1_6c_H1_V3 Hunter Bell
1040_1_6c_H2_V3 0
1040_1_6c_H3_V3 A39 26 756
1040_1_6c_H4_V3 Da-Law
1040_1_6c_H5_V3 2
1040_1_6c_H6_V3
1040_1_6c_H1_V4
1040_1_6c_H2_V4 0
1040_1_6c_H3_V4
1040_1_6c_H4_V4
```

```
1040_1_6c_H5_V4
1040_1_6c_H1_V5
1040_1_6c_H2_V5 0
1040_1_6c_H3_V5
1040_1_6c_H4_V5
1040_1_6c_H5_V5
1040_1_6c_H1_V6
1040_1_6c_H2_V6 0
1040_1_6c_H3_V6
1040_1_6c_H4_V6
1040_1_6c_H5_V6
1040_1_6d 0
1040_1_6e 9
1040_1_7 3878
1040_1_8a
1040_1_8b
1040_1_9
1040_1_10
1040_1_11
1040_1_12 0
1040_1_13
1040_1_14
1040_1_15
1040_1_16a
1040_1_16b
1040_1_17a
1040_1_17b
1040_1_18
1040_1_19
1040_1_20
1040_1_21a
1040_1_21b
1040_1_22_H1 Travel allowance
1040_1_22_H2 0
1040_1_23 3878
1040_1_24
1040_1_25a 2574
1040_1_25b
1040_1_26
1040_1_27
1040_1_28
1040_1_29_V1
1040_1_29_H1_V2
1040_1_29_H2_V2
1040_1_30 2574
1040_1_31 1303
```

Figure 36: Listing of a reference file corresponding to the form displayed on the previous page.

```
1040_1
1040_1_L_H1_V1 July
1040_1_L_H2_V1 July
1040_1_L_H3_V1 88
1040_1_L_H1_V2 BnairerndA.&ErskinW.Mitchell
1040_1_L_H2_V2 A11881384
1040_1_L_H1_V3 99225LeeStret
1040_1_L_H2_V3 A59021948
1040_1_L_H1_V4 Russell,NJ61920
1040_1_L_H1_V5 1
1040_1_L_H2_V5 0
1040_1_L_H1_V6 1
1040_1_L_H2_V6 0
1040_1_1 1
1040_1_2 0
1040_1_3_H1 0
1040_1_3_H2
1040_1_4_H1 0
1040_1_4_H2
1040_1_5_H1 0
1040_1_5_H2
1040_1_6a 1
1040_1_6b_H1 0
1040_1_6b_H2 1
1040_1_6c_H1_V1 RiderHarlan
1040_1_6c_H2_V1 0
1040_1_6c_H3_V1 A97203760
1040_1_6c_H4_V1 Aunt
1040_1_6c_H5_V1 6
1040_1_6c_H6_V1 8
1040_1_6c_H1_V2 TulaneBanks
1040_1_6c_H2_V2 0
1040_1_6c_H3_V2 A97081904
1040_1_6c_H4_V2 Si-Law
1040_1_6c_H5_V2 12
1040_1_6c_H6_V2
1040_1_6c_H1_V3 HunterBell
1040_1_6c_H2_V3 0
1040_1_6c_H3_V3 A9326
1040_1_6c_H4_V3 Da-Law
1040_1_6c_H5_V3 2
1040_1_6c_H6_V3
1040_1_6c_H1_V4
1040_1_6c_H2_V4 0
1040_1_6c_H3_V4
1040_1_6c_H4_V4
```

```
1040_1_6c_H5_V4
1040_1_6c_H1_V5
1040_1_6c_H2_V5 0
1040_1_6c_H3_V5
1040_1_6c_H4_V5
1040_1_6c_H5_V5
1040_1_6c_H1_V6
1040_1_6c_H2_V6 0
1040_1_6c_H3_V6
1040_1_6c_H4_V6
1040_1_6c_H5_V6
1040_1_6d 0
1040_1_6e 9
1040_1_7 3873
1040_1_8a
1040_1_8b
1040_1_9
1040_1_10
1040_1_11
1040_1_12
1040_1_13
1040_1_14
1040_1_15
1040_1_16a
1040_1_16b
1040_1_17a
1040_1_17b
1040_1_18
1040_1_19
1040_1_20
1040_1_21a
1040_1_21b
1040_1_22_H1 Travelalbewance
1040_1_22_H2 0
1040_1_23 3878
1040_1_24
1040_1_25a 25174
1040_1_25b
1040_1_26
1040_1_27
1040_1_28
1040_1_29_V1
1040_1_29_H1_V2
1040_1_29_H2_V2
1040_1_30 2574
1040_1_31 03
```

Figure 37: Listing of a hypothesis file corresponding to the completed form.

```
1040_1
1040_1_L_H1_V1 0.85 0.86 0.90 0.81
1040_1_L_H2_V1 0.84 0.89 0.94 0.90
1040_1_L_H3_V1 0.99 0.83
1040_1_L_H1_V2 0.83 0.85 0.85 0.84 0.91 0.94 0.90 0.90 0.98
0.92 0.93 0.89 0.87 0.88 0.82 0.80 0.90 0.81 0.99 0.97 0.94 0.83
0.83 0.93 0.96 0.95 0.98 0.81
1040_1_L_H2_V2 0.92 0.92 0.90 0.94 0.87 0.87 0.82 0.81 0.89
1040_1_L_H1_V3 0.99 0.80 0.95 0.80 0.84 0.95 0.83 0.88 0.91
0.97 0.92 0.95 0.83
1040_1_L_H2_V3 0.93 0.90 0.91 0.90 0.98 0.82 0.84 0.82 0.92
1040_1_L_H1_V4 0.99 0.98 0.99 0.97 0.87 0.97 0.93 0.94 0.99
0.98 0.89 0.90 0.99 0.90 0.94
1040_1_L_H1_V5 0.99
1040_1_L_H2_V5 0.91
1040_1_L_H1_V6 0.88
1040_1_L_H2_V6 0.92
1040_1_1 0.82
1040_1_2 0.83
1040_1_3_H1 0.98
1040_1_3_H2
1040_1_4_H1 0.89
1040_1_4_H2
1040_1_5_H1 0.85
1040_1_5_H2
1040_1_6a 0.80
1040_1_6b_H1 0.91
1040_1_6b_H2 0.99
1040_1_6c_H1_V1 0.87 0.89 0.88 0.90 0.83 0.99 0.94 0.92 0.95
0.98 0.90
1040_1_6c_H2_V1 0.80
1040_1_6c_H3_V1 0.82 0.81 0.98 0.87 0.85 0.85 0.84 0.91 0.95
1040_1_6c_H4_V1 0.94 0.89 0.82 0.99
1040_1_6c_H5_V1 0.98
1040_1_6c_H6_V1 0.89
1040_1_6c_H1_V2 0.90 0.92 0.91 0.98 0.94 0.84 0.98 0.87 0.87
0.80 0.84
1040_1_6c_H2_V2 0.84
1040_1_6c_H3_V2 0.98 0.99 0.99 0.99 0.93 0.94 0.98 0.99 0.93
1040_1_6c_H4_V2 0.83 0.80 0.78 0.93 0.90 0.92
1040_1_6c_H5_V2 0.90 0.91
1040_1_6c_H6_V2
1040_1_6c_H1_V3 0.89 0.83 0.90 0.94 0.94 0.93 0.99 0.91 0.98
0.80
1040_1_6c_H2_V3 0.88
1040_1_6c_H3_V3 0.92 0.94 0.95 0.92 0.91
1040_1_6c_H4_V3 0.99 0.99 0.03 0.93 0.95 0.99
1040_1_6c_H5_V3 0.92
1040_1_6c_H6_V3
1040_1_6c_H1_V4
```

```
1040_1_6c_H2_V4 0.93
1040_1_6c_H3_V4
1040_1_6c_H4_V4
1040_1_6c_H5_V4
1040_1_6c_H1_V5
1040_1_6c_H2_V5 0.93
1040_1_6c_H3_V5
1040_1_6c_H4_V5
1040_1_6c_H5_V5
1040_1_6c_H1_V6
1040_1_6c_H2_V6 0.99
1040_1_6c_H3_V6
1040_1_6c_H4_V6
1040_1_6c_H5_V6
1040_1_6d 0.99
1040_1_6e 0.91
1040_1_7 0.89 0.89 0.82 0.88
1040_1_8a
1040_1_8b
1040_1_9
1040_1_10
1040_1_11
1040_1_12
1040_1_13
1040_1_14
1040_1_15
1040_1_16a
1040_1_16b
1040_1_17a
1040_1_17b
1040_1_18
1040_1_19
1040_1_20
1040_1_21a
1040_1_21b
1040_1_22_H1 0.92 0.84 0.85 0.84 0.80 0.98 0.92 0.92 0.90 0.95
0.96 0.96 0.98 0.87 0.87
1040_1_22_H2 0.80
1040_1_23 0.90 0.81 0.84 0.85
1040_1_24
1040_1_25a 0.99 0.93 0.98 0.98 0.82
1040_1_25b
1040_1_26
1040_1_27
1040_1_28
1040_1_29_V1
1040_1_29_H1_V2
1040_1_29_H2_V2
1040_1_30 0.92 0.84 0.84 0.89
1040_1_31 0.87 0.86
```

Figure 38: Listing of a confidence file corresponding to the completed form.

```
1040_1
1040_1_L_H1_V1 0 0 0 0
1040_1_L_H2_V1 0 0 0 0
1040_1_L_H3_V1 0 0
1040_1_L_H1_V2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0
1040_1_L_H2_V2 0 0 0 0 0 0 0 0 0
1040_1_L_H1_V3 0 0 0 0 0 0 0 0 0 0 0 0 0
1040_1_L_H2_V3 0 0 0 0 0 0 0 0 0
1040_1_L_H1_V4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1040_1_L_H1_V5 0
1040_1_L_H2_V5 0
1040_1_L_H1_V6 0
1040_1_L_H2_V6 0
1040_1_1 0
1040_1_2 0
1040_1_3_H1 0
1040_1_3_H2
1040_1_4_H1 0
1040_1_4_H2
1040_1_5_H1 0
1040_1_5_H2
1040_1_6a 0
1040_1_6b_H1 0
1040_1_6b_H2 0
1040_1_6c_H1_V1 0 0 0 0 0 0 0 0 0 0 0
1040_1_6c_H2_V1 0
1040_1_6c_H3_V1 0 0 0 0 0 0 0 0 0
1040_1_6c_H4_V1 0 0 0 0
1040_1_6c_H5_V1 0
1040_1_6c_H6_V1 0
1040_1_6c_H1_V2 0 0 0 0 0 0 0 0 0 0 0
1040_1_6c_H2_V2 0
1040_1_6c_H3_V2 0 0 0 0 0 0 0 0 0
1040_1_6c_H4_V2 0 0 1 0 0 0
1040_1_6c_H5_V2 0 0
1040_1_6c_H6_V2
1040_1_6c_H1_V3 0 0 0 0 0 0 0 0 0 0
1040_1_6c_H2_V3 0
1040_1_6c_H3_V3 0 0 0 0 0
1040_1_6c_H4_V3 0 0 1 0 0 0
1040_1_6c_H5_V3 0
1040_1_6c_H6_V3
1040_1_6c_H1_V4
1040_1_6c_H2_V4 0
1040_1_6c_H3_V4
1040_1_6c_H4_V4
```

```
1040_1_6c_H5_V4
1040_1_6c_H1_V5
1040_1_6c_H2_V5 0
1040_1_6c_H3_V5
1040_1_6c_H4_V5
1040_1_6c_H5_V5
1040_1_6c_H1_V6
1040_1_6c_H2_V6 0
1040_1_6c_H3_V6
1040_1_6c_H4_V6
1040_1_6c_H5_V6
1040_1_6d 0
1040_1_6e 0
1040_1_7 0 0 0 0
1040_1_8a
1040_1_8b
1040_1_9
1040_1_10
1040_1_11
1040_1_12
1040_1_13
1040_1_14
1040_1_15
1040_1_16a
1040_1_16b
1040_1_17a
1040_1_17b
1040_1_18
1040_1_19
1040_1_20
1040_1_21a
1040_1_21b
1040_1_22_H1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1040_1_22_H2 0
1040_1_23 0 0 0 0
1040_1_24
1040_1_25a 0 0 0 0 0
1040_1_25b
1040_1_26
1040_1_27
1040_1_28
1040_1_29_V1
1040_1_29_H1_V2
1040_1_29_H2_V2
1040_1_30 0 0 0 0
1040_1_31 0 0
```

Figure 39: Listing of a rejection file corresponding to the completed form.

Appendix C: Form-Based Scoring Output

C.1 Example Scoring Summary (.sum)

```
File: r0000/r0000_00.mrg #1
 vlen=0
 distance=0
 REF: ““
 HYP: ““
 RES: ““
 REJ: ““


File: r0000/r0000_00.mrg #2
 vlen=0
 distance=0
 REF: ““
 HYP: ““
 RES: ““
 REJ: ““


File: r0000/r0000_00.mrg #3
 vlen=0
 distance=0
 REF: ““
 HYP: ““
 RES: ““
 REJ: ““


File: r0000/r0000_00.mrg #4
 vlen=20
 distance=3
 REF: “BerryK.&LorasA.Boyle”
 HYP: “BerryK.&LonasA.Boyle”
 RES: “----------S---------”
 REJ: “00000000001000000000”
 CNF:  0.9700
       0.8900
        0.9500
         0.9300
          0.8900
           0.9900
            0.9400
             0.9800
              0.9100
               0.8700
                0.3000
                 0.8500
                  0.9300
                   0.9900
                    0.9500
                     0.9500
                      0.9200
                       0.9800
                        0.9100
                         0.8300

   confS:r->n
```

```
File: r0000/r0000_00.mrg #5
 vlen=9
 distance=0
 REF: “A57003582”
 HYP: “A57003582”
 RES: “---------”
 REJ: “000000000”
 CNF:  0.9800
       0.9400
        0.9800
         0.9900
          0.9200
           0.9300
            0.8200
             0.8700
              0.9400


                .
                .
                .

<Text Removed for Documentation Purposes>
                .
                .
                .


File: r0010/r0010_00.mrg #43
 vlen=22
 distance=0
 REF: “veredpensioninvestment”
 HYP: “veredpensioninvestment”
 RES: “----------------------”
 REJ: “0000000000000000000000”
 CNF:  0.9900
       0.9000
        0.9800
         0.8900
          0.9000
           0.9200
            0.9900
             0.9500
              0.9500
               0.9600
                0.9400
                 0.9200
                  0.9100
                   0.8100
                    0.8200
                     0.8400
                      0.8500
                       0.8200
                        0.9000
                         0.9400
                          0.9500
                           0.9200
```

```
File: r0010/r0010_00.mrg #44
 vlen=0
 distance=0
 REF: ""
 HYP: ""
 RES: ""
 REJ: ""


File: r0010/r0010_00.mrg #45
 vlen=3
 distance=0
 REF: "247"
 HYP: "247"
 RES: "---"
 REJ: "000"
 CNF:  0.9800
       0.9300
       0.9900


File: r0010/r0010_00.mrg #46
 vlen=3
 distance=0
 REF: "978"
 HYP: "978"
 RES: "---"
 REJ: "000"
 CNF:  0.9000
       0.9300
       0.9200




Summary:
  TOTALS  ( output=FCltdA,of=form.sum,cf=form.fct )

Draft standard measures:
Accumulators: TP=1648 FP=43 M=36 RT=45 RF=18 RM=164
 Character recognition decision:
  :                      accuracy:  88.8410%   ( 1648 / 1855 )
  :            accuracy (form right):  97.4571%   ( 1648 / 1691 )
 Character output:
  :                      accuracy:  98.4644%   ( 1603 / 1628 )
 Field accuracy:
  :       accuracy (including icons):  81.2762%   ( 777 / 956 )

Character rejection rates:
  :                         all:   2.7306%   ( 63 / 1882 )
  :               all hypotheses:   3.7256%   ( 63 / 1691 )
  :                     matches:   2.7306%   ( 45 / 1648 )
  :               substitutions:  44.1176%   ( 15 / 34 )
  :                  insertions:  33.3333%   ( 3 / 9 )
  :          all (due to form type):   8.7141%   ( 164 / 1882 )

Fields (excluding icons):
  :                      accuracy:  81.7010%   ( 634 / 776 )
  :       accuracy (with form right):  90.1849%   ( 634 / 703 )
  :      rejected (due to form type):   9.4072%   ( 73 / 776 )
  :      deleted (due to form wrong):   0.0000%   ( 0 / 776 )
```

```
Fields (including icons):
  :                      accuracy:  81.2762%   ( 777 / 956 )
  :       accuracy (with form right):  89.8266%   ( 777 / 865 )
  :      rejected (due to form type):   9.5188%   ( 91 / 956 )
  :      deleted (due to form wrong):   0.0000%   ( 0 / 956 )

Characters:
  :                      accuracy:  85.1753%   ( 1603 / 1882 )
  :       accuracy (with form right):  94.7960%   ( 1603 / 1691 )
  :      rejected (due to form type):   8.7141%   ( 164 / 1882 )
  :      deleted (due to form wrong):   0.0000%   ( 0 / 1882 )

Icons:
  :                      accuracy:  79.4444%   ( 143 / 180 )
  :       accuracy (with form right):  88.2716%   ( 143 / 162 )
  :      rejected (due to form type):  10.0000%   ( 18 / 180 )
  :      deleted (due to form wrong):   0.0000%   ( 0 / 180 )

Form type identification:
  :                      accuracy:  90.9091%   ( 10 / 11 )
  :                  failure rate:   9.0909%   ( 1 / 11 )
  :    accuracy (excluding rejected): 100.0000%   ( 10 / 10 )
  : failure rate (excluding rejected):   0.0000%   ( 0 / 10 )
  :                      rejected:   9.0909%   ( 1 / 11 )
```

C.2 Example Fact Sheet (.fct)

```
form type:
count: 11
 rejected: 1
 not rejected, right: 10
 not rejected, wrong: 0

icon fields:
count: 180
 form type rejected: 18
 form type wrong and not rejected: 0
 form type right and not rejected: 162
  right: 143
  wrong: 19
  rejected: 15
  not rejected: 147
  matches: 157
   rejected: 14
   not rejected: 143
  mismatches: 5
   rejected: 1
   not rejected: 4
  not present / not found: 115
  not present / found: 3
  present / not found: 2
  present / found: 42

character fields:
count: 776
 form type rejected: 73
 form type wrong and not rejected: 0
 form type right and not rejected: 703
  right: 634
  wrong: 69

characters:
 in alignments: 1891
 hypothesis: 1691
 reference: 1882
  form type rejected: 164
  form type wrong and not rejected: 0
  form type right and not rejected: 1691
   rejected: 63
   not rejected: 1628
   correct: 1648
    rejected: 45
    not rejected: 1603
   substitutions: 34
    rejected: 15
    not rejected: 19
   insertions: 9
    rejected: 3
    not rejected: 6
   deletions: 36

Accumulators: TP=1648 FP=43 M=36 RT=45 RF=18 RM=164
```

C.3 Scoring Summary Description

```
File: r0000/r0000_00.mrg #1                    < alignment of the 1st form's 1st field >
 vlen=0                                        < length of the alignment >
 distance=0                                    < Levenstein Distance>
 REF: ““‘                                      < empty reference string >
 HYP: ““‘                                      < empty hypothesis string >
 RES: ““‘
 REJ: ““‘
                .
                .
                .
<text removed for documentation purposes>
                .
                .
                .
File: r0000/r0000_00.mrg #4                    < alignment of the 1st form's 4th field >
 vlen=20                                       < length of alignment >
 distance=3                                    < Levenstein Distance >
 REF: “BerryK.&LorasA.Boyle”                   < reference string >
 HYP: “BerryK.&LonasA.Boyle”                   < hypothesis string >
 RES: “----------S---------”                   < alignment results >
 REJ: “00000000001000000000”                   < rejected character >
 CNF:  0.9700                                  < confidence values for each hypothesized character >
        0.8900
         0.9500
          0.9300
           0.8900
            0.9900
             0.9400
              0.9800
               0.9100
                0.8700
                 0.3000
                  0.8500
                   0.9300
                    0.9900
                     0.9500
                      0.9500
                       0.9200
                        0.9800
                         0.9100
                          0.8300

   confS:r->n                                  <summary of errors >
                .
                .
                .
<text removed for documentation purposes>
                .
                .
                .
```

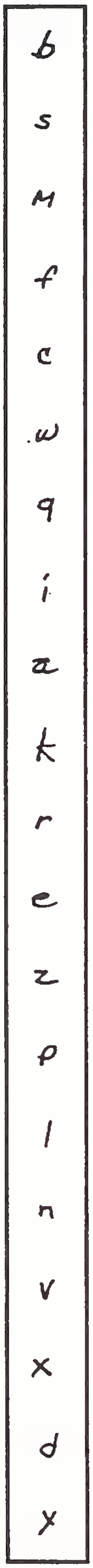
```
File: r0010/r0010_00.mrg #46                                  < alignment of the 11th form's 46th field >
 vlen=3                                                       < length of alignment >
 distance=0                                                   < Levenstein Distance >
 REF: "978"                                                   < reference string >
 HYP: "978"                                                   < hypothesis string >
 RES: "---"                                                   < alignment results >
 REJ: "000"                                                   < rejected characters >
 CNF:  0.9000                                                 < confidence values for each hypothesized character >
      0.9300
      0.9200




Summary:                                                      < beginning of summary report >
  TOTALS  ( output=FCItdA,of=form.sum,cf=form.fct )           < scoring profile options selected >

Draft standard measures:
Accumulators: TP=1648 FP=43 M=36 RT=45 RF=18 RM=164           < fundamental accumulators >
 Character recognition decision:
< eq. 6 in the draft standard >
  :                    accuracy:  88.8410%   ( 1648 / 1855 )
< eq. 6 ignoring rejected characters due to rejected form identifications >
  :            accuracy (form right):  97.4571%   ( 1648 / 1691 )
 Character output:
< eq. 7 in the draft standard >
  :                    accuracy:  98.4644%   ( 1603 / 1628 )
 Field accuracy:
< eq. 4 in the draft standard: # of fields correctly recognized / # of fields on all forms >
  :        accuracy (including icons):  81.2762%   ( 777 / 956 )

Character rejection rates:
< # of rejected characters on correctly identified and not rejected forms / # of reference characters on all forms >
  :                        all:   2.7306%   ( 63 / 1882 )
< # of rejected characters on correctly identified and not rejected forms / # of hypothesis characters on correctly identified and not
              rejected forms >
  :                all hypotheses:   3.7256%   ( 63 / 1691 )
< percentage of correctly recognized characters rejected >
  :                    matches:   2.7306%   ( 45 / 1648 )
< percentage of substituted characters rejected >
  :                substitutions:  44.1176%   ( 15 / 34 )
< percentage of inserted characters rejected >
  :                  insertions:  33.3333%   ( 3 / 9 )
< # of rejected characters due to rejected form identifications / # of reference characters on all forms >
  :          all (due to form type):   8.7141%   ( 164 / 1882 )

Fields (excluding icons):
< # of correctly recognized character fields / # of character fields on all forms >
  :                    accuracy:  81.7010%   ( 634 / 776 )
< # of correctly recognized character fields / # of character fields on correctly identified and not rejected forms >
  :        accuracy (with form right):  90.1849%   ( 634 / 703 )
< # of rejected character fields due to rejected form identifications / # of character fields on all forms >
  :       rejected (due to form type):   9.4072%   ( 73 / 776 )
< # of missed character fields due to incorrect and not rejected form identifications / # of character fields on all forms >
  :       deleted (due to form wrong):   0.0000%   ( 0 / 776 )
```

```
Fields (including icons):
< # of correctly recognized character and icon fields / # of character and icon fields on all forms >
   :                    accuracy:  81.2762%   ( 777 / 956 )
< # of correctly recognized character and icon fields / # of character and icon fields on correctly identified and not rejected forms >
   :        accuracy (with form right):  89.8266%   ( 777 / 865 )
< # of rejected character and icon fields due to rejected form identifications / # of character and icon fields on all forms >
   :       rejected (due to form type):   9.5188%   ( 91 / 956 )
<# of missed character and icon fields due to incorrect and not rejected form identifications / # of character and icon fields on all
            forms>
   :       deleted (due to form wrong):   0.0000%   ( 0 / 956 )

Characters:
< # of correctly recognized characters / # of reference characters on all forms >
   :                    accuracy:  85.1753%   ( 1603 / 1882 )
< # of correctly recognized characters / # of hypothesis characters on correctly identified and not rejected forms >
   :        accuracy (with form right):  94.7960%   ( 1603 / 1691 )
< # of rejected characters due to rejected form identifications / # of reference characters on all forms >
   :       rejected (due to form type):   8.7141%   ( 164 / 1882 )
< # of missed characters due to incorrect and not rejected form identifications / # of reference characters on all forms >
   :       deleted (due to form wrong):   0.0000%   ( 0 / 1882 )

Icons:
< # of correctly recognized fields / # of icon fields on all forms >
   :                    accuracy:  79.4444%   ( 143 / 180 )
< # of correctly recognized icon fields / # of icon fields on correctly identified and not rejected forms >
   :        accuracy (with form right):  88.2716%   ( 143 / 162 )
< # of rejected icon fields due to rejected form identifications / # of icon fields on all forms >
   :       rejected (due to form type):  10.0000%   ( 18 / 180 )
< # of missed icon fields due to incorrect and not rejected form identifications / # of icon fields on all forms >
   :       deleted (due to form wrong):   0.0000%   ( 0 / 180 )

Form type identification:
< # of correctly identified and not rejected forms / # of all forms >
   :                    accuracy:  90.9091%   ( 10 / 11 )
< 1.0 - previous value >
   :                  failure rate:   9.0909%   ( 1 / 11 )
< # of correctly identified and not rejected forms / # of all forms not rejected >
   :     accuracy (excluding rejected): 100.0000%   ( 10 / 10 )
< 1.0 - previous value >
   : failure rate (excluding rejected):   0.0000%   ( 0 / 10 )
< # of form identifications rejected / # of all forms >
   :                    rejected:   9.0909%   ( 1 / 11 )
```

C.4 Fact Sheet Description

```
form type:                           < form-level accumulators >
count: 11                            < # of forms scored >
 rejected: 1                         < # of forms rejected >
 not rejected, right: 10             < # of forms not rejected and correctly identified >
 not rejected, wrong: 0              < # of forms not rejected and incorrectly identified >

icon fields:                         < icon field accumulators >
count: 180                           < # of icon fields scored >
 < indented counts are subsets of all icon fields scored >
 form type rejected: 18              < # of icon fields rejected due to form identification rejected >
 form type wrong and not rejected: 0 < # of icon fields on forms incorrectly identified and not rejected >
 form type right and not rejected: 162 < # of icon fields on forms correctly identified and not rejected >
  < indented counts are subsets of all forms correctly identified and not rejected >
  right: 143                         < # of correct icon fields after rejection >
  wrong: 19                          < # of incorrect icon fields after rejection >
  rejected: 15                       < # of icon fields rejected >
  not rejected: 147                  < # of icon fields accepted >
  matches: 157                       < # of correct icon fields ignoring rejection >
   < indented counts are subsets of all correct icon fields ignoring rejection >
   rejected: 14                      < # of correct icon fields rejected >
   not rejected: 143                 < # of correct icon fields accepted >
  mismatches: 5                      < # of incorrect icon fields ignoring rejection >
   < indented counts are subsets of all incorrect icon fields ignoring rejection >
   rejected: 1                       < # of incorrect icon fields rejected >
   not rejected: 4                   < # of incorrect icon fields accepted >
  not present / not found: 115       < # of empty icon fields detected correctly >
  not present / found: 3             < # of empty icon fields detected incorrectly >
  present / not found: 2             < # of non-empty icon fields detected incorrectly >
  present / found: 42                < # of non-empty icon fields detected correctly >

character fields:                    < character field accumulators >
count: 776                           < # of character fields scored >
 < indented counts are subsets of all character fields scored >
 form type rejected: 73              < # of character fields rejected due to form identification rejected >
 form type wrong and not rejected: 0 < # of character fields on forms incorrectly identified and not rejected >
 form type right and not rejected: 703 < # of character field on forms correctly identified and not rejected >
  < indented counts are subsets of all character fields on forms correctly identified and not rejected >
  right: 634                         < # of correct character fields after rejection >
  wrong: 69                          < # of incorrect character fields after rejection >

characters:                          < character-level accumulators >
 < indented counts are subsets of all character scored >
 in alignments: 1891                 < # of character alignment positions >
 hypothesis: 1691                    < # of hypothesized characters >
 reference: 1882                     < # of reference characters >
  < indented counts are subsets of all reference characters scored >
  form type rejected: 164            < # of characters rejected due to form identification rejected >
  form type wrong and not rejected: 0 < # of characters on forms incorrectly identified and not rejected >
  form type right and not rejected: 1691 < # of characters on forms correctly identified and not rejected >
   < indented counts are subsets of all reference characters on forms correctly identified and not rejected >
   rejected: 63                      < # of character rejected >
   not rejected: 1628                < # of characters accepted >
   correct: 1648                    < # of correct characters ignoring rejection >
    < indented counts are subsets of all correct characters ignoring rejection >
    rejected: 45                    < # of correct characters rejected >
    not rejected: 1603              < # of correct characters accepted >
   substitutions: 34                < # of substituted characters ignoring rejection >
    < indented counts are subsets of all substituted characters ignoring rejection >
    rejected: 15                    < # of substituted characters rejected >
    not rejected: 19                < # of substituted characters accepted >
   insertions: 9                    < # of inserted characters ignoring rejection >
    < indented counts are subsets of all inserted characters ignoring rejection >
    rejected: 3                     < # of inserted characters rejected >
    not rejected: 6                 < # of inserted characters accepted >
   deletions: 36                    < # of deleted characters >

Accumulators: TP=1648 FP=43 M=36 RT=45 RF=18 RM=164 < fundamental accumulators >
```

Appendix D: Character-Based Files



Classification File	Hypothesis File	Confidence File	Rejection File
20	20	20	20
62	62	0.83	0
73	73	0.90	0
6d	6d	0.85	0
66	66	0.89	0
63	65	0.78	0
77	77	0.82	0
71	71	0.85	0
69	69	0.85	0
61	61	0.92	0
6b	6b	0.87	0
72	6e	0.38	1
65	65	0.90	0
7a	73	0.08	1
70	70	0.83	0
6c	69	0.11	1
6e	6e	0.93	0
76	76	0.89	0
78	78	0.80	0
64	64	0.83	0
79	78	0.58	0

Figure 40: Isolated hand-print images and listings of corresponding character-based scoring files.

Appendix E: Character-Based Scoring Output

E.1 Example Scoring Summary (.sum)

```
File: 10000/10000.mrg #5
 vlen=1
 distance=3
 REF: “c”
 HYP: “e”
 RES: “S”
 REJ: “0”
 CNF:  0.7800

   confS:c->e

File: 10000/10000.mrg #11
 vlen=1
 distance=3
 REF: “r”
 HYP: “n”
 RES: “S”
 REJ: “1”
 CNF:  0.3800

   confS:r->n

File: 10000/10000.mrg #13
 vlen=1
 distance=3
 REF: “z”
 HYP: “s”
 RES: “S”
 REJ: “1”
 CNF:  0.0800

   confS:z->s

File: 10000/10000.mrg #15
 vlen=1
 distance=3
 REF: “l”
 HYP: “i”
 RES: “S”
 REJ: “1”
 CNF:  0.1100

   confS:l->i

File: 10000/10000.mrg #20
 vlen=1
 distance=3
 REF: “y”
 HYP: “x”
 RES: “S”
 REJ: “0”
 CNF:  0.5800

   confS:y->x
```

```
Summary:
  TOTALS  ( output=FCItdAA,of=10000/10000.sum,cf=10000/10000.fct )

Draft standard measures:
Accumulators: TP=15 FP=5 M=0 RT=0 RF=3 RM=0
 Character recognition decision:
  :                       accuracy:  75.0000%   ( 15 / 20 )
  :            accuracy (form right):  75.0000%   ( 15 / 20 )
 Character output:
  :                       accuracy:  88.2353%   ( 15 / 17 )
 Field accuracy:
  :        accuracy (including icons):  75.0000%   ( 15 / 20 )

Character rejection rates:
  :                            all:  0.0000%   ( 3 / 20 )
  :                 all hypotheses:  15.0000%   ( 3 / 20 )
  :                        matches:  0.0000%   ( 0 / 15 )
  :                  substitutions:  60.0000%   ( 3 / 5 )
  :                     insertions:  0.0000%   ( 0 / 0 )
  :          all (due to form type):  0.0000%   ( 0 / 20 )

Fields (excluding icons):
  :                       accuracy:  75.0000%   ( 15 / 20 )
  :        accuracy (with form right):  75.0000%   ( 15 / 20 )
  :       rejected (due to form type):  0.0000%   ( 0 / 20 )
  :       deleted (due to form wrong):  0.0000%   ( 0 / 20 )

Fields (including icons):
  :                       accuracy:  75.0000%   ( 15 / 20 )
  :        accuracy (with form right):  75.0000%   ( 15 / 20 )
  :       rejected (due to form type):  0.0000%   ( 0 / 20 )
  :       deleted (due to form wrong):  0.0000%   ( 0 / 20 )

Characters:
  :                       accuracy:  75.0000%   ( 15 / 20 )
  :        accuracy (with form right):  75.0000%   ( 15 / 20 )
  :       rejected (due to form type):  0.0000%   ( 0 / 20 )
  :       deleted (due to form wrong):  0.0000%   ( 0 / 20 )

Icons:
  :                       accuracy:  0.0000%   ( 0 / 0 )
  :        accuracy (with form right):  0.0000%   ( 0 / 0 )
  :       rejected (due to form type):  0.0000%   ( 0 / 0 )
  :       deleted (due to form wrong):  0.0000%   ( 0 / 0 )

Form type identification:
  :                       accuracy: 100.0000%   ( 1 / 1 )
  :                   failure rate:  0.0000%   ( 0 / 1 )
  :    accuracy (excluding rejected): 100.0000%   ( 1 / 1 )
  : failure rate (excluding rejected):  0.0000%   ( 0 / 1 )
  :                       rejected:  0.0000%   ( 0 / 1 )
```

E.2 Example Fact Sheet (.fct)

```
form type:
count: 1
 rejected: 0
 not rejected, right: 1
 not rejected, wrong: 0

icon fields:
count: 0
 form type rejected: 0
 form type wrong and not rejected: 0
 form type right and not rejected: 0
  right: 0
  wrong: 0
  rejected: 0
  not rejected: 0
  matches: 0
   rejected: 0
   not rejected: 0
  mismatches: 0
   rejected: 0
   not rejected: 0
  not present / not found: 0
  not present / found: 0
  present / not found: 0
  present / found: 0

character fields:
count: 20
 form type rejected: 0
 form type wrong and not rejected: 0
 form type right and not rejected: 20
  right: 15
  wrong: 5

characters:
 in alignments: 20
 hypothesis: 20
 reference: 20
  form type rejected: 0
  form type wrong and not rejected: 0
  form type right and not rejected: 20
   rejected: 3
   not rejected: 17
   correct: 15
    rejected: 0
    not rejected: 15
   substitutions: 5
    rejected: 3
    not rejected: 2
   insertions: 0
    rejected: 0
    not rejected: 0
   deletions: 0

Accumulators: TP=15 FP=5 M=0 RT=0 RF=3 RM=0
```

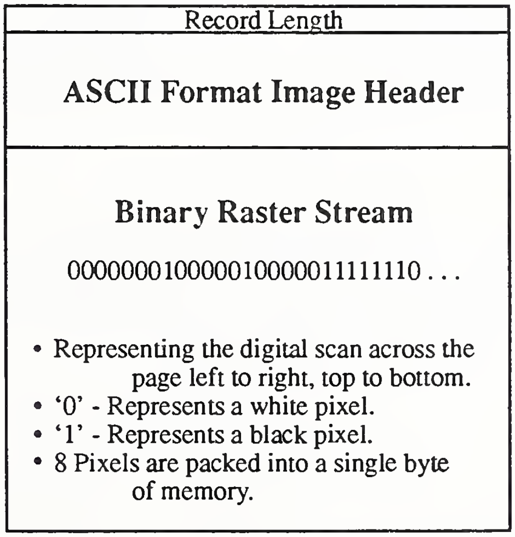
Appendix F: IHead File Format

Image file formats and effective data compression and decompression are critical to the usefulness of image archives. Each binary image distributed in the referenced databases produced by NIST have been digitized at 12 dots per millimeter and 2-dimensionally compressed using CCITT Group 4.[13][14] These raster images are digital encodings 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 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 41 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 SD2 and 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 41. The IHead structure definition as written in the 'C' programming language is listed in Figure 42.



Record Length

ASCII Format Image Header

Binary Raster Stream

000000010000010000011111110 . . .

- 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.

Figure 41: 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 IHDR_SIZE          288       /* len of hdr record (always even bytes) */
#define SHORT_CHARS        8         /* # of ASCII chars to represent a short */
#define BUFSIZE            80        /* default buffer size */
#define DATELEN            26        /* character length of data string */

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

Figure 42: IHead C language definition.

Figure 43 lists the header values from an IHead file corresponding to the structure members listed in Figure 42. This header information belongs to the isolated box image displayed in Figure 44. Referencing the structure members listed in Figure 42, 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 43: The IHead values for the isolated subimage displayed in Figure 44.

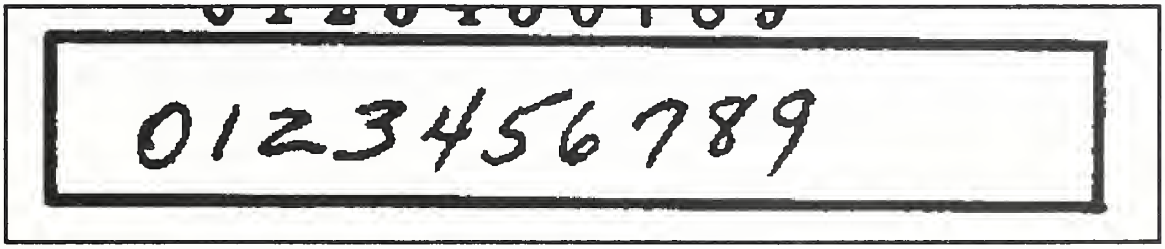


Figure 44: An IHead image of an isolated box.

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.

In the IHead format, images may be compressed with virtually any algorithm. The IHead header 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 which signifies which compression technique, if any, has been applied to the raster image data which follows the header. If the compression code is zero, then the image data is not compressed, and the data dimensions: width, height, and depth, are sufficient to load the image into main memory. However, if the compression code is nonzero, then the **complen** field must be used in addition to the image's pixel dimensions. For example, the image described in Figure 43 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 which has the pixel dimensions consistent with those stored in its header. Using CCITT Group 4 compression and this compression scheme on images of tax forms, a compression ratio of 10.1 to 1 has been 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 43 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 43, the **parent** field contains the full path name to the image from which the image displayed in Figure 44 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.

Appendix G: MIS File Format

Based on experience gained from creating and manipulating large on-line image databases, NIST has developed a number of diversified file formats. One such file format has been developed to manage large volumes of segmented character images. Storing character images in individual files has proven to be very inefficient especially when databases of several hundred thousand characters are being developed.

Devoting a separate file node for each character image creates enormous file system overhead. Unreasonably large directory tables must be allocated. Rarely are experiments conducted on only a single character image in isolation. Rather, most experiments require a large sample of characters. Experience has shown that the gathering of a large sample of characters from a file system where the images have been stored in individual files greatly burdens the disk controller. This results in slow experiment loading times as well as limiting the access of other applications to data stored on the same storage device.

In addition to creating large directory tables, storing segmented character images in individual files results in sparse usage of the storage device. This sparseness is even more exaggerated when the images are compressed. For example, segmented character images in SD3 and SD7 have been centered within a 128 by 128 binary pixel image. The resulting image size is 2,344 bytes, 296 bytes for the IHead header and 2,048 bytes of image data. These files when CCITT Group 4 compressed average 360 bytes in size, 296 bytes for the IHead header and only 64 bytes of compressed image data. Storing these compressed image files onto CDROM for example, which uses a 2,048 block size, would be extremely wasteful. Only 18% of each block containing image data would be used.

In light of these observations, NIST has developed a Multiple Image Set (MIS) file format. The MIS format allows multiple images of homogeneous dimensions and depth to be stored in one file. MIS is a simple extension or encapsulation of the IHead format described in Appendix F. It can be seen in Figure 45 that the IHead structure is included as a member in the MIS definition.

```
/***************************************************************
          Filename: Mis.h
          Author: Michael D. Garris
          Date: 7/18/90
***************************************************************/
typedef struct misstruct{
     IHEAD *head;
     unsigned char *data;
     int misw;
     int mish;
     int entw;
     int enth;
     int ent_num;
     int ent_alloc;
} MIS;
```

Figure 45: MIS C language definition

An MIS file contains one or more individual images stacked vertically within the same contiguous raster memory, the last scanline of the previous image concatenated to first scanline of the next image. The individual images which are concatenated together are referred to as MIS *entries*. The resulting contiguous raster memory is referred to as the MIS *memory*. The MIS memory containing one or more entries of uniform width, height, and depth is stored using the IHead header format. The IHead attribute fields are sufficient to describe the MIS memory. The IHead structure's width attribute specifies the width of the MIS memory, and likewise the IHead structure's height attribute specifies the height of the MIS memory. In this way, the MIS memory can be stored just like any normal raster image including possible compression.

Due to the uniform dimensions of MIS entries, the IHead structure's width attribute also specifies the width of the entries in the MIS memory. What is lacking from the original IHead definition is the uniform height of the MIS entries and the number of MIS entries in the MIS memory. Realize that given the uniform height of the MIS entries the number of entries in the MIS memory can be computed by dividing the entry height into the total MIS memory height. The interpretation of two of the IHead attribute fields, **par_x** and **par_y**, changes when the IHead header is being used to describe an MIS memory. The **par_x** field is used to hold the uniform width of the MIS entries, and the **par_y** field is used to hold the uniform height of the MIS entries. In other words, **width** and **height** represent MIS memory width and MIS memory height respectively, while **par_x** and **par_y** represent MIS entry width and MIS entry height respectively. Using this convention, an MIS file is treated like an IHead file.

Figure 45 lists the MIS structure definition written in the 'C' programming language. The structure contains an IHead structure, **head**, and an 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. The MIS attributes **misw** and **mish** specify the width and height of the MIS memory. These values are the same as the **width** and **height** attributes contained in the IHead structure pointed to by **head**. The MIS attributes **entw** and **enth** specify the uniform width and height of the MIS entries. These values are the same as the **par_x** and **par_y** attributes contained in the IHead structure pointed to by **head**. The MIS attribute, **ent_alloc**, specifies how many MIS entries of dimension **entw** and **enth** have been allocated to the MIS memory **data**. The MIS attribute, **ent_num**, specifies how many entries out of the possible number allocated are currently and contiguously contained in the MIS memory **data**.

Appendix H: Source Code Documentation for IHead and MIS files

Source code for 8 different programs: **decomp**, **dumpihdr**, **fragmis**, **htoc**, **ihdr2sun**, **sunalign**, **xtrctcls** and **xtrctmis** are included within the source code directory **image** in this release of the Scoring Package. These programs, their primary supporting subroutines, and associated file names are described below. These routines are provided as an example to software developers of how IHead and MIS images may be manipulated.

H.1 decomp <IHead file in> <IHead file out>

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

The procedure **readihdrfile()** is responsible for loading an IHead image from a file into main memory and is found in the file **loadihdr.c**. This routine reads the image's header data returning an initialized IHead structure by calling **readihdr()**. In addition, the image's raster data is returned to the caller uncompressed. The images in this release 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 [13][14] and adapted by NIST for use with this release and is found in the file **g4decomp.c**.

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

H.2 dumpihdr <IHead file>

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

H.3 fragmis <misfile> <rootname>

The program **fragmis** takes the concatenated MIS entries contained in a single MIS file and writes each entry to a separate IHead image file. The program is given the MIS file to be fragmented and the root name to be used in creating the resulting IHead image files. A sequential index and an extension **pct** will be added to the specified root name in order to create unique file names. The main routine for **fragmis** is found in **fragmis.c** and calls the external function **fragmis()**.

H.4 htoc <hex value>

The program **htoc** is a program which takes a hexadecimal value as input and returns to standard output the ASCII character that the value represents. This program is useful for determining the ASCII character a character classification value represents when examining CLS files. The main routine for **htoc** is found in **htoc.c**.

H.5 ihdr2sun <IHead file>

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

H.6 sunalign <Sun rasterfile>

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

H.7 xtrctcls -[c,h] <clsfile> <index>

The program **xtrctcls** is used to copy a specific class item from a CLS file. The copied item is returned via standard output in one of two formats, hexidecimal (the '-h' option) or character value (the '-c' option). The input to **xtrctcls** is the CLS file to be used and the index to the item to be copied. The index is zero-oriented. The main routine for **xtrctcls** is found in **xtrctcls.c.**

H.8 xtrctmis <misfile> <outfile> <index>

The program **xtrctmis** copies a specified MIS entry into a separate IHead image file. The inputs are the MIS file to be used, the file in which the MIS entry is to be stored, and the index of the MIS entry to be copied. Once again the index is zero oriented. The main routine for this program is in **xtrctmis.c** and calls the external routine **xtrctmis**().

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4950

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
OCTOBER 1992

4. TITLE AND SUBTITLE

NIST Scoring Package User's Guide - Release 1.0

5. AUTHOR(S)

Michael D. Garris and Stanley A. Janet

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

The increased performance and connectivity of computers has touched off an imaging revolution. Multiple products are emerging, all of which are designed for optical character recognition problems. This gives the potential user of optical character recognition technology many different choices and options which leads to a series of significant questions. How does one determine when the technology has matured enough to make it economically advantageous to deploy? How does a potential user determine which product is best for his or her specific needs? How can a system developer, who has the ability to choose from a large variety of diverse algorithmic approaches, intelligently choose and then track progress when developing optical character recognition systems? The answer to these questions lie in objective system performance measurement. This is the motivation behind the development of the NIST Scoring Package. Application requirements germane to a specific automated character recognition problem are embodied in a representative set of referenced images. Along with image data, a referenced image has associated with it the ASCII textual information that is to be recognized in the image. This reference information serves as ground truth for measuring recognition performance. The images are presented to a recognition system, and the system's results are returned. This includes hypothesized text of what the system located and recognized. The Scoring Package reconciles the hypothesized text with the referenced text, accumulated statistics used to compute performance measures. The NIST Scoring Package is a reference implementation of the draft, Standard Method for Evaluation the Performance of Systems Intended to Recognize Hand-printed Characters from Image Data Scanned from Forms, which has been submitted to ANSI X.3A. This document presents the concepts of scoring forms processing systems and character classifiers, discusses the concepts and algorithm used for dynamic string alignment, defines the files and their formats required as input to the Scoring Package, and documents how the Scoring Package software is installed and invoked.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

Character Classification; Dynamic String Alignment; Forms Identification; Forms Processing Systems; Hand-print Recognition; Image Databases; Isolated Character Images; Machine Print Recognition; Optical Character Recognition (OCR); Performance Assessment; Referenced Images; Recognition Performance Assessment Standard; Scoring; Structured Forms.

13. AVAILABILITY

☒ UNLIMITED

□ FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).

□ ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.

☒ ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
74

15. PRICE
A04

ELECTRONIC FORM