U.S. BUREAU OF THE CENSUS

Record Linkage Software

USER DOCUMENTATION

Submitted by

System Resources Corporation
400 Virginia Avenue, SW, Suite 350
Washington, D.C. 20024

and

Synectics for Management Decisions, Inc.
3030 Clarendon Boulevard, Suite 305
Arlington, Virginia 22201

TABLE OF CONTENTS

LIST OF FIGURES

1.0 BACKGROUND

This section provides a background and context for the software by presenting the purposes of record linkage and the statistical theory and methodology behind it.

1.1 Purposes of Record Linkage

Record linkage is an effort to identify records on two different electronic files that contain information about the same "entity." These records often refer to a person, organization, or institution. However, the entity could also be a dwelling unit, a real estate property, a criminal case, a license to carry out an activity such as the sale of pesticides or drugs, or anything else that could be the subject of an electronic record.

There are two basic reasons to try to link records: data collation and list construction.

- *Data Collation* — A project might require data that are not all available from the same source. Such a project might involve checking the consistency between earnings reported on income tax returns and earnings reported by employers to the Social Security Administration. A research study might require comparing data from the Census of Agriculture with data from the Economic Census.

- *List Construction* — Some projects require a list of all the members of a population to serve as a sampling frame, the contact list for a census, or for collation with data from other sources. It often happens that there is no single list of the population, but that a combination of several lists can be expected to include all or most of it. Usually, there is some overlap between these lists. To avoid biasing the sample or census, we must delete duplications so that each member of the population is included only once. This "unduplication" requires that we identify people, or other entities, that are included in more than one list.

1.2 Theory and Methodology of Record Linkage

The concepts of record linkage were introduced by Howard Newcombe in the late 1950's. The programs documented here are based on statistical theories first developed by two Canadian statisticians, Fellegi and Sunter.

There are two kinds of record linkage: one-to-one matching and many-to-one matching.

- *One-to-one* — One-to-one matching is used if each entity is represented once on a particular file. The record linkage seeks to establish unique pairs of records, both referring to the same entity.

- *Many-to-one* Many-to-one matching links several records in one file to a single record in another file. For example, imagine wanting to link a file of export licenses and a file of businesses. Each company or establishment in the business file might hold several export licenses. In this case, the purpose of the record linkage would be to match the appropriate business to *each* of the export licenses.

Many-to-one matching: An example

One common example of many-to-one matching is "geocoding," so called because one of its uses is to match addresses to larger geographic units such as ZIP codes or Census blocks or tracts.

2.0 RECORD LINKAGE COMPUTER PROGRAMS

Computer programs have been developed to perform record linkage tasks. The programs developed by the Bureau of the Census and documented here are designed to perform one-to-one matching.

> AutoMatch is available within the Bureau on a few selected computers; the only one accessible to all Bureau personnel is in the Statistical Research Division (SRD).

Outside programs such as the commercially-available AutoMatch must be used to perform many-to-one matching. AutoMatch has a companion standardization program, AutoStan. Canlink, by Statistics Canada, runs on Unix workstations only, with Oracle, and includes no standardization fucntions.

2.1 Census Bureau Programs

Census has developed four different programs to perform different parts of the record linkage process.

- *SRD Name Standardizer* — The Statistical Research Division developed a program to standardize the names of businesses and individuals.
- *GD Address Standardizer* — The Geography Division developed a program to put business and personal addresses into a common format.
- *GDRIVER Program* — The Statistical Research Division GDRIVER program integrates the name and address standardizers and allows both names and addresses to be converted to a common format in a single computer run. This program can also be used as a driver for the name standardizer alone or the address standardizer alone. These programs could be used alone if the files do not include both names and addresses or if another standardization program is used for one of the elements.
- *SRD Record Linkage Program* — The Statistical Research Division also developed a record linkage program. Since standardization of information is a key element of the linkage process, this program depends on use of the standardizer programs or other programs that can standardize names and addresses.
- *DMD Name Standardizer* — The Decennial Management Division also developed a *Name Standardizer* that puts the names of individuals into a common format. This program is not documented here. The available documentation indicates that it is particularly appropriate for standardizing files that include military personnel. For more information about this program, call Anne McGaughey, now (March, 1999) in the Decennial Statistical Studies Division, at 301-457-4110.

2.2 Overview of the System

This section includes a general overview of the process of record linkage as it is performed by the Bureau of the Census programs.

2.2.1 General Overview

Figure 1 is a general flowchart of the record standardization and matching process. That process begins with two electronic files, designated **File A** and **File B**. (These file designations are used in the statistical theory of record matching and will be used throughout this documentation.)

(1) Both files must first be standardized using the programs described in Section 4.

(2) The standardized files must then be sorted according to the blocking variables, producing a pair of sorted files.

"Blocking" provides a method to limit the number of record pairs being examined. It is described in Section 3.3.

(3) It is these sorted files that the matching program can collate.

(4) After each pass through the matching program, record pairs are assigned either to a file of matched records, or to a "residual" file of unmatched records.

These unmatched records must be re-sorted by a new set of blocking variables. The re-sorted files can then be run through the matching program just like the original two files.

The process of examining unmatched records, re-sorting, and rematching can be repeated until no or almost no new matches are produced.

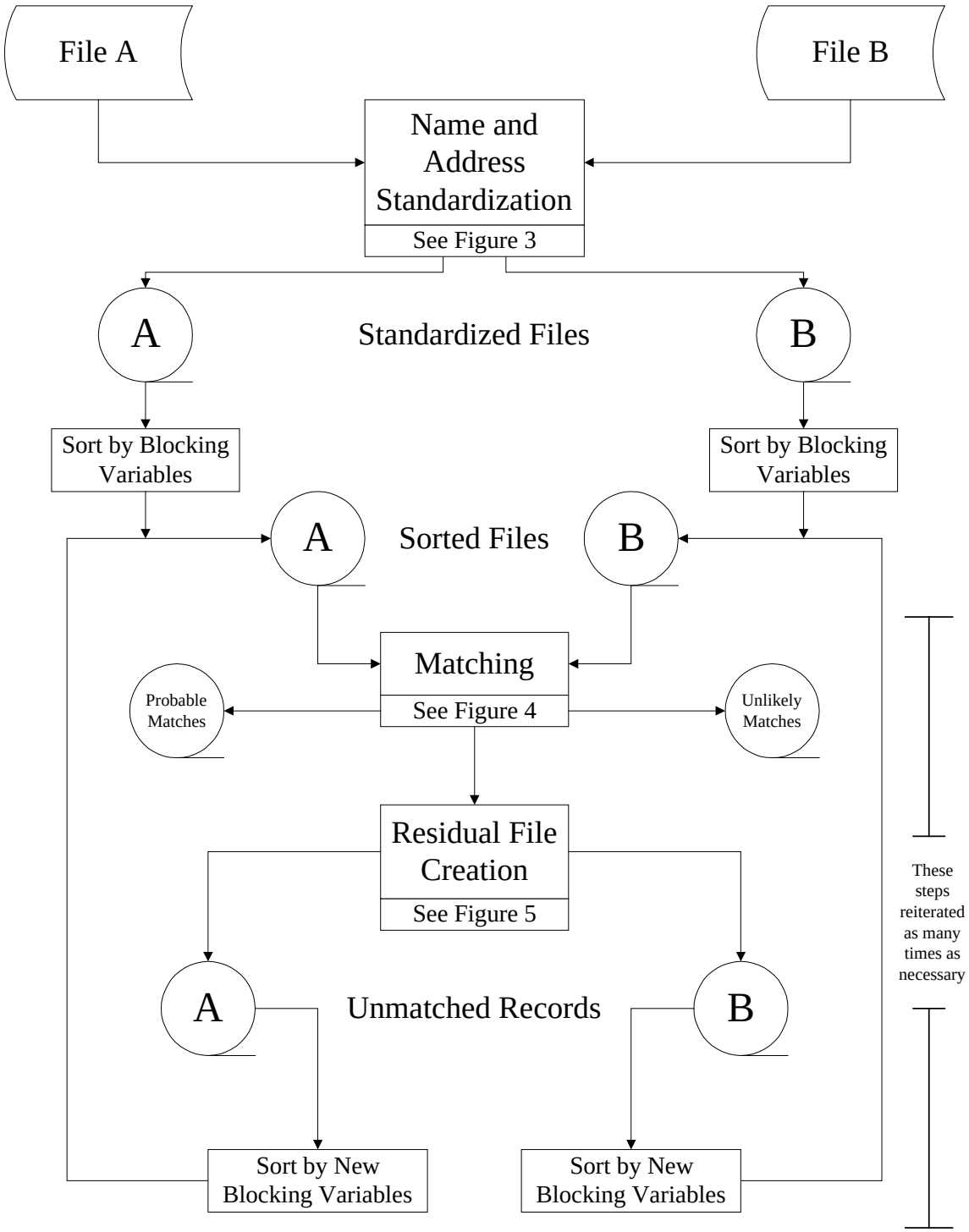
At that point, the user has a set of files of matched records to use in his project and the functions of the matching software have been completed.

2.2.2 Data Standardization

Before any attempt is made to link records, the data in them must be standardized.

The Bureau of the Census record matching system includes modules to standardize names, both business and personal, and street addresses. A single driver program, GDRIVER, allows the user to standardize both names and addresses within a single file. The names and addresses in the second file must be standardized in a separate step.

Figure 1: General Overview



2.2.3 Sorting and Blocking

The record matching program will work only on files that have been sorted and standardized. These standardized files must be sorted in the order of the blocking variables.

Sorting in the order of the blocking variables: An example

If the blocking variables are state and the first initial of the last name, all records must be sorted so that everyone from Alabama whose last name begins with "A" is at the beginning of the file, followed by all Alabamians whose last names begin with "B." Records of people from Wyoming whose last names begin with "Z" must be at the end of the file.

The Census system makes use of information provided by the user for blocking, but does not include a utility for sorting the records into the required order.

- Section 5.0 of this documentation includes instructions for using a freeware sorting program.

- If large files are being matched, the user might have to use a different sorting program with a larger capacity, such as the sort utility of SAS.

2.2.4 Record Matching and Clerical Review

Once the data have been standardized and sorted, the matching module can be executed.

The matching program requires instructions through a parameter file, *parmmf.dat*, that is described in Section 6.3.

The output of the matching program includes a sorted list of pairs of records with their matching scores.

The user must examine this list by hand ("clerical review") to determine the minimum score above which all matches found by the program are, in fact, matched pairs, and the score below which no pairs are true matches. Usually, there will be a range between these two numbers that includes some pairs that really are matches and some that the user can determine are not or are probably not true matches. Use of the output files of matches is not further supported by the matching program.

2.2.5 Iteration

The matching program also identifies "residuals," records that are not involved in matched pairs.

It is usually advisable to run the residual records through the matching program again after adjusting the matching score parameters and sorting and blocking on a somewhat different set of variables. A second matching run should produce a different set of matched pairs of records. It is up to the user to merge these records with the matches produced on the previous matching run.

This iterative process may produce additional matches on a second pass.

3.0 DATA PREPARATION FOR RECORD LINKAGE

Record linkage programs must either assume that the data to be matched are in a common format, leaving the user to ensure that this is the case, or include a standardization module or subroutine.

3.1 Data Preparation Requirements

Record linkage often involves electronic files compiled at different times or by different organizations, using different conventions for formatting and abbreviating the data. This causes differences in the format of identical information.

(1) Even when data are collected by the same organization, different individuals might respond to different information requests and thus use different name and address formats.

(2) The same individual might respond in different ways at different times, such as by omitting a middle name or initial on one form but including it on another, or abbreviating part of his address on one but not on another.

For example, compare the following two business records:

Northwest Regional Hospital	OBGYN DEPT
Department of Obstetrics & Gynecology	NW REG HOSP
223 Franklin D. Roosevelt Drive	223 FDR DR

Similarly, compare the following two personal records:

Dr. William S.T. Freylinghuysen III	WM S. FRELINGHUYSEN
501 East 108th Street, Apartment 8-D	501 E 108 ST #8D
New York, New York 10011	NY, NY 10011-2345

Although each pair of records pertains to the same hospital or individual, within each pair the two records differ in the information included:

(1) One record includes Dr. Frelinghuysen's title, a generational suffix, and two middle initials. The other record omits the title, suffix, and one of the middle initials.

(2) One record is in upper and lower case letters. The other is all in capital letters.

(3) One record makes the maximum use of abbreviations. The other uses no abbreviations.

A computerized linkage program is not likely to connect these pairs of records unless the information is formatted much more consistently.

3.2 Data Standardization Programs

The data standardization programs documented here work by "parsing" name or address elements, that is, breaking them down into a set of elements that are then rearranged in a common order and abbreviated in common ways.

External programs can standardize other kinds of data.

For instance, the SRD name standardizer recognizes eight elements (coded 1 to 4 and 6 to 9) that might be found in a personal name:

Code	Element	Examples
1	Conjunction	and, &, or
2	Generational suffix	Senior, Sr., Junior, Jr., III, 3rd
3	Title	Professor, Reverend, Dr., Ms.
4	Occupational suffix	M.D., R.N., Ph.D., Esq., C.P.A.
6	Initial	A., B.
7	Last name	Smith, Martinez
8	Last name prefix	Saint, da, los, van, Mac
9	First name	James, Mary, Rafael

The program compares these elements to lists in files that it accesses.

- The list of conjunctions is quite short, the list of first names is quite long.
- A user dealing with a special population can improve the standardization of the data by adding terms to one of these lists.
- The Census name standardizer recognizes the various elements in whatever order they occur, converts variations like nicknames to a standard form, and rearranges elements in a common order before consolidating them into three fields: first name, middle name, and last name.

Special populations: An example

If matching involves a list of military personnel, the title file might be augmented with the various terms for military ranks, such as "sergeant" and "captain."

A similar process can be performed with address elements. The Bureau of the Census address standardizer recognizes 10 elements that might be found in a street address:

Note: The Bureau of the Census address standardizer operates on street addresses only, not on city names.

Element Name	Examples
House number	123
Street name prefix	N, S, E, W
Main street name	Oak, Main, Martin Luther King
Secondary street name (street type)	Avenue, Blvd., Place, Court
Within-structure description	Bldg., Apt., #
Within-structure identifier	5-W, 405
Rural route description	RR
Rural route identifier	Usually a number
Box description	Box, PO, PO Box
Box identifier	A letter or number

The address standardization program refers to a file of possible patterns to recognize these elements by their content and position, and rearranges them into a standard order. At the same time, the program converts elements like street types to standard abbreviations.

Both the name and address standardization programs refer to files of name and address elements that are based on experience gained in previous matching projects. These files are described in Sections 4.5 and 4.6.

- A user may want to modify these files to reflect the particular populations or type of records being examined.

- You must be familiar with the data in the files you are going to be matching. It may be necessary to do one or more matching runs and review a sample of the matches to determine what kinds of terms need to be added.

Modifying files for particular populations: An example

A file called *skpl.dat* contains elements of business names that are too common to be useful for matching, such as “Administrator” and “Industries.” It is based on Agricultural Census requirements and many of the terms are peculiar to agriculture-related businesses, such as “Creamery” and “Ranch.” This list would not be suitable for a project involving export-import firms, and would have to be augmented with terms appropriate to that population.

If you think the matching program is not recognizing correct matches, you may want to consult with the Statistical Research Division about adapting the program to your data.

3.3 Record Blocking

The main objective of a record linkage program is to classify every possible pair of records as either a matched pair or an unmatched pair.

It would be impractical to compare all record pairs between two files of any reasonable size, since the total number of pairs is the product of the number of records on each file.

Therefore, the records on each of the files being compared must be *blocked.*

Blocking is a division of an entire file into mutually exclusive subsets.

When records on both files are blocked, the records on File A are compared only to records in the corresponding block on File B.

If there were still too many comparisons to execute within a reasonable time after a first selection of blocking variables, both files would have to be divided more finely.

Since blocking precludes most of the possible matches from being checked, selection of the blocking variables is a very important step in the record linkage process. For instance, if two files both contain addresses, they can be blocked by state. This would divide each file into 50 blocks and greatly reduce the number of potential matches to be attempted.

Comparing files for duplication without blocking: An example

Determining whether there are any duplications in two small files of 1,000 cases each would require 1,000 × 1,000 = 1,000,000 comparisons. In this example, at most 1,000 matches could be found, and the remaining 999,000 would be "unmatched" pairs. Determining whether there are any duplications in two modest files of 10,000 cases each would require 10,000 × 10,000 = 100,000,000 comparisons. Determining whether any of the 10,000 cases in a modest-sized file are already included in a large file of 20 million records would require 10,000 × 20,000,000 = 200,000,000,000 or 200 billion comparisons. This is too many for the Bureau of the Census computers to handle within a reasonable run time.

Comparing files for duplication with blocking: An example

If two 10,000-record files were each divided into 20 equal blocks of 500, instead of making one set of 100 million comparisons, the linkage program could make 20 sets of 500 × 500 = 250,000 comparisons for a total of only 250,000 × 20 = 5,000,000 comparisons. Note, however, that dividing a 20 million-record file into 20 blocks would still leave 1 million records per block, and even against blocks of only 500 in the smaller file, within each pair of blocks there would still be 500 million comparisons. Performing 500 million comparisons within 20 blocks would require a total of 10 billion comparisons instead of 200 billion.

Several factors must be considered before deciding to block on state, however.

First, although blocking on state would create 50 blocks (51 if the District of Columbia were included—even more if the files included Puerto Rico and the territories), the blocks would not be of equal size. Although the *average* state has only 2 percent of the nation's population, California has about 12 percent and Wyoming has only 0.2 percent.

Possible size of files when blocking by state: An example

If our hypothetical files were distributed in proportion to the population, the California blocks would include 1,200 records in the file of 10,000 and 2,400,000 records in the file of 20 million, thus requiring 1,200 × 2,400,000 = 2,880,000,000 comparisons within the California block: still too many.

It is necessary, therefore, to ensure that the blocking variables selected do not have any unusually large categories by determining the sizes of the potential blocks in each file and computing their products.

A second problem in blocking involves the accuracy of the data recorded in the blocking variable.

For instance, if through keying errors some Maine addresses have been entered as "MD" (Maryland) instead of "ME," or if some Louisiana addresses have been transposed from "LA" to "AL" (Alabama), then the miscoded addresses will never be compared and the correct matches will not be found.

For example, neither of the following pairs of records, pertaining to the same individual, would be matched:

FIRSTNAME	MI	LASTNAME	STNO	STREET	CITY	STATE	ZIP	SEX	DOB	SSAN
ALFRED	H	SMITH	415	WALKER CT	AUGUSTA	ME	04338	M	060747	057401833
ALFRED	H	SMITH	415	WALKER CT	AUGUSTA	MD	04338	M	060747	057401833
FILBERT	Z	ZWACH	1499	IVANHOE PKWY	BATON ROUGE	LA	70829	M	022964	998776655
FILBERT	Z	ZWACH	1499	IVANHOE PKWY	BATON ROUGE	AL	70829	M	022964	998776655

Each of these two pairs differs only by a single letter in the STATE variable, and any probabilistic matching algorithm would identify each pair of records as pertaining to the same individual. However, if the state code were being used as a blocking variable the pairs of records would never be compared.

It is thus important to block records on variables that are known to be highly accurately coded. In these two cases, the errors in the state codes could be detected by comparing them to the ZIP codes, since Maryland ZIP codes begin with "2," not "0," and Alabama ZIP codes begin with "3," not "7."

Because of the problems associated with errors in blocking variables, it is important to reblock the unmatched records on different variables and make another attempt to match the records. As noted earlier, the Census matching program outputs the matching records to a separate residual file so that the remaining records can be run through the program again.

4.0 NAME AND ADDRESS STANDARDIZATION

Standardization is a separate process that must be carried out *before* records can be matched.

Section 4.1 describes the standardization options. Sections 4.2-4.5 specify how to execute each of the options supported by Bureau of the Census software.

4.1 Standardization Options

Data standardization options are shown in **Figure 2**.

The GDRIVER program is set up to standardize both names and addresses (see Section 4.2), but changing the specifications in one line of a required file—*parmdr.txt*—will let you standardize names only (see Section 4.3) or addresses only (see Section 4.4).

You can standardize names alone when the files to be matched do not include addresses or the addresses have already been standardized. Likewise, you can standardize addresses alone when the files to be matched do not include names or the names have already been standardized. You may also use any other standardization program to which you have access and which you know how to use.

SRD GDRIVER: GDRIVER uses the SRD name standardization program, which can deal with either personal or corporate names, and the GD address standardizer.

Decennial Management Division (DMD) name standardizer: DMD has also developed a name standardizer; it is similar to the SRD name standardizer but with different name element lists. In particular, DMD accesses a title file that includes military rank terms, making it particularly appropriate for use with files of armed forces personnel.

4.2 Use of GDRIVER for Name and Address Standardization

The following is a step-by-step guide to standardizing both the name and address fields for two files using the GDRIVER standardization program. Perform the following steps on File A and then repeat the steps for File B.

The use of GDRIVER is illustrated in **Figure 3**. The program calls several reference files, for both name and address standardization. The reference files for name standardization must be in the directory where the standardization will take place. If the reference files for address standardization are not in this directory, the file *initfiles.dat* needs to be created. This file should contain the path of where these files are located. The user must also tell GDRIVER the location of the file of records to be standardized, through the *parmn.txt* file, and the layout of that file, through *parmdr.txt.* GDRIVER creates two output files, one of standardized records *(stanout.txt)* and one of records it could not standardize *(otherout.txt).* It also produces a summary report.

This guide assumes that the two files are fixed-length ASCII files with one record per entity. Words in *italics* are actual filenames.

Figure 2: Standardizer Options

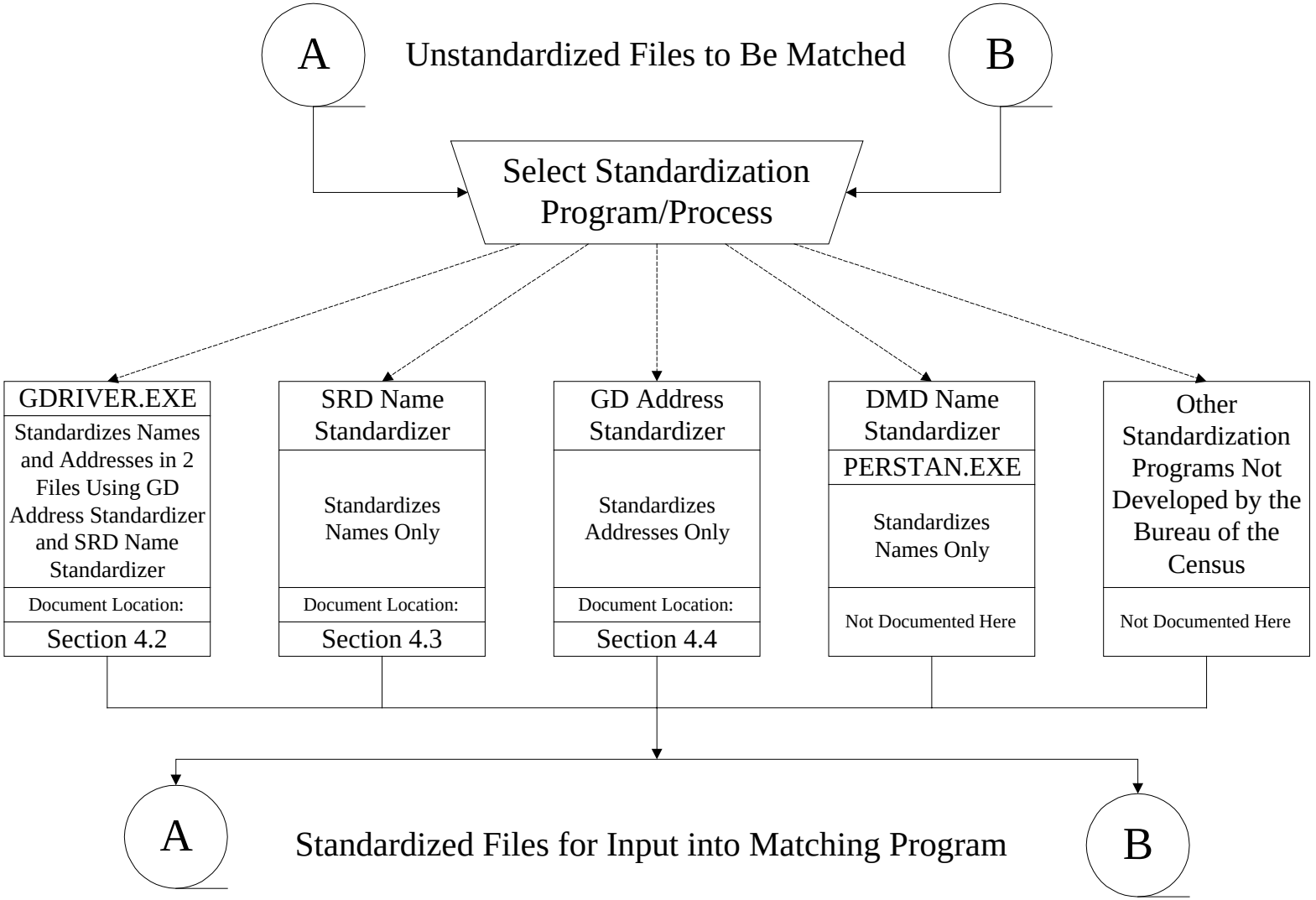
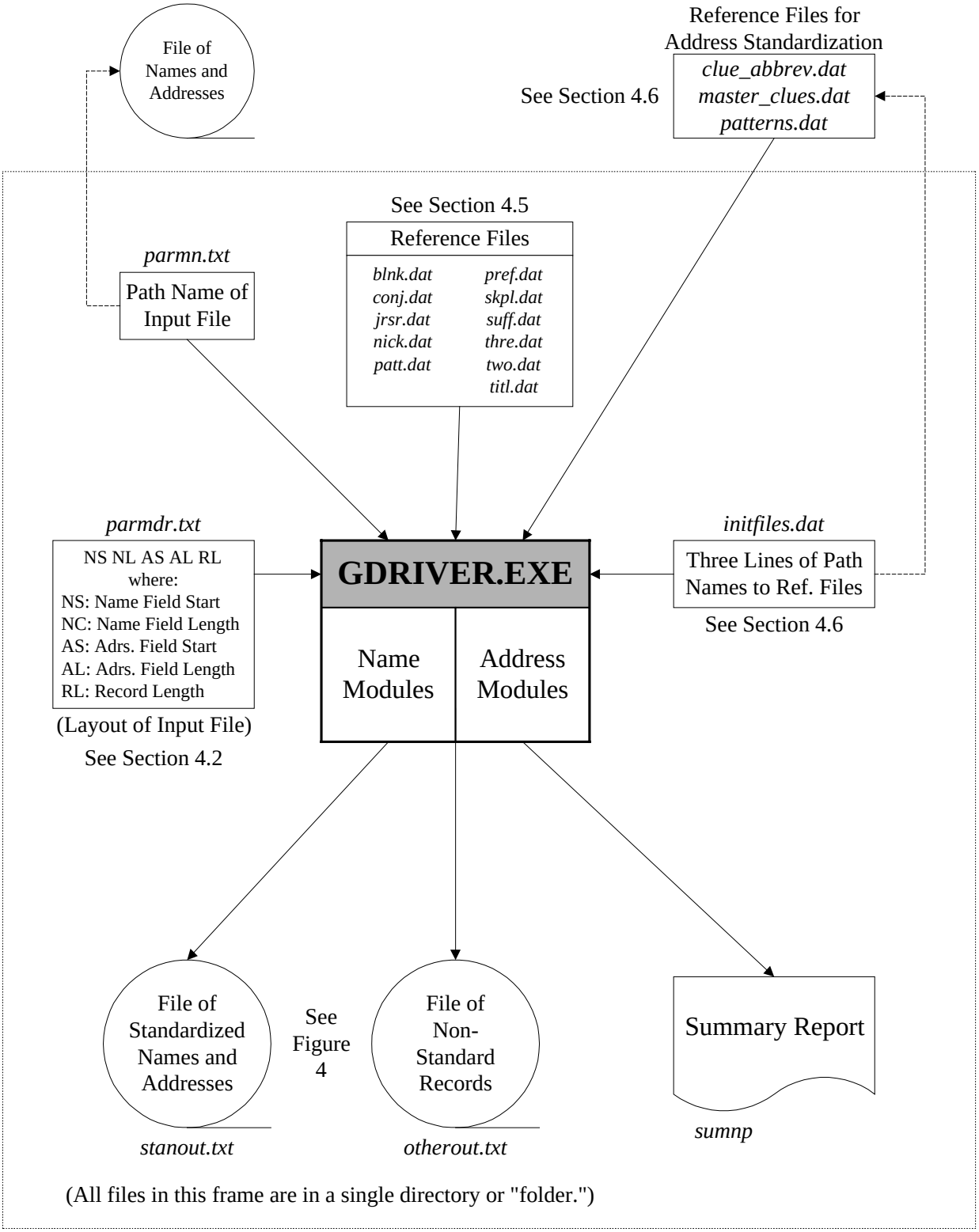


Figure 3: GDRIVER Standardizer



The first step is to prepare the two files that will be standardized.

A. Create a directory for one of the files.

B. Copy the GDRIVER.EXE file to the directory you just created.

C. Copy the reference files for name standardization to the directory.

> See Section 4.5 for a full list of and discussion of the reference files for name standardization.

D. Copy the file *parmn.txt* to the directory.

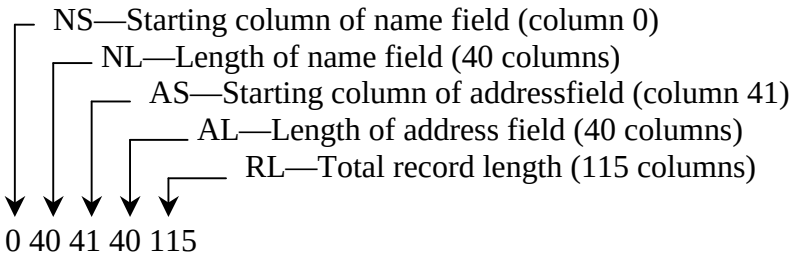
The file *parmn.txt* should contain only the name and path of the file to be standardized.

E. Copy the file *parmdr.txt* to the directory.

The file *parmdr.txt* should contain a single line specifying the record layout of the file to be standardized in the following format:

NS NL AS AL RL

For example, in *parmdr.txt*



F. Copy the file *initfiles.dat* to the directory.

G. The file *initfiles.dat* should contain the names and paths of the following three reference files for address standardization (each on a separate line):

- *clue_abbrev.dat*
- *master_clues.dat*
- *patterns.dat*

> See Section 4.6 for a full discussion of the reference files for address standardization listed in the file *initfiles.dat*.

If these files have been copied to the same directory as the other files discussed here, *initfiles.dat* will not be necessary.

The next step is to use the GDRIVER program to standardize the names and addresses in a file. To do this, simply execute GDRIVER.EXE in the directory. Once the GDRIVER program has completed its standardization, review the output.

The GDRIVER program produces four files; each is described here.

- *stanout.txt* (File of standardized records)—each record of this file contains all of the data from the original records that the program could standardize on both name and address, plus the standardized data. The original record layout remains unchanged. Appended to the end of each record are the standardized address and name fields in the following format:

Field Name	Length
House number	6
Street name prefix direction	5
Main street name	20
Secondary street name	10
Within-structure description	4
Within-structure identifier	5
Rural route description	3
Rural route identifier	5
Box description	4
Box identifier	5
Title	6
First name	13
Middle name	13
Last name	14

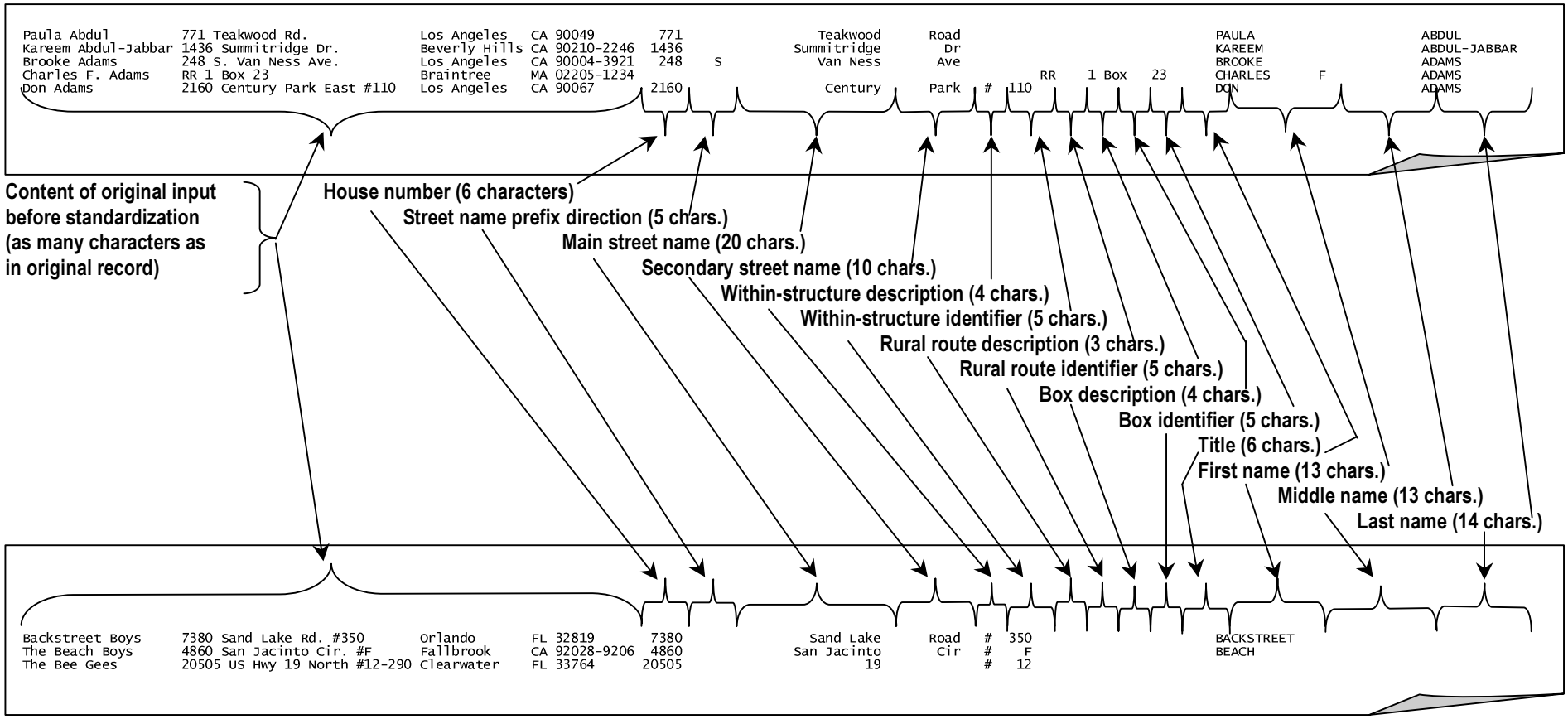
The following eight lines are a sample of records from an unstandardized file:

```
Paula Abdul          771 Teakwood Rd.               Los Angeles   CA 90049
Kareem Abdul-Jabbar 1436 Summitridge Dr.           Beverly Hills CA 90210-2246
Brooke Adams         248 S. Van Ness Ave.          Los Angeles   CA 90004-3921
Charles F. Adams     RR 1 Box 23                   Braintree     MA 02205-1234
Don Adams            2160 Century Park East #110   Los Angeles   CA 90067
Backstreet Boys      7380 Sand Lake Rd. #350       Orlando       FL 32819
The Beach Boys       4860 San Jacinto Cir. #F      Fallbrook     CA 92028-9206
The Bee Gees         20505 US Hwy 19 North #12-290 Clearwater    FL 33764
```

Figure 4 displays five lines extracted from the *stanout.txt* file. They show what each record looks like after standardization.

Figure 4: Sample Output from *stanout.txt* and *otherout.txt*

stanout.txt



otherout.txt

- *otherout.txt* (File of non-standard records)—this file contains the original records that the program could not standardize on either name or address. The original record layout remains unchanged. Appended to the end of each record are the fields that the program could standardize in the following format:

Field Name	Length
House number	6
Street name prefix direction	5
Main street name	20
Secondary street name	10
Within-structure description	4
Within-structure identifier	5
Rural route description	3
Rural route identifier	5
Box description	4
Box identifier	5
Title	6
First name	13
Middle name	13
Last name	14

Three lines in **Figure 4** are extracted from the *otherout.txt* file. They show what each record looks like after standardization. Note that these records could only be standardized on some of the fields.

- *Sumnp* (Summary report)—this report shows the following: 1) the number of entries in each of the reference files; 2) the number of records read; and 3) the number of names standardized. Below is an example of this report.

```
Conjunction table size is          6
 Jr/sr table size is              13
 Nickname table size is          491
 Pattern table size is           851
 Prefix table size is             23
 Skip list table size is        1146
 Suffix table size is             19
 Title table size is               9
 Two suffix table size is          5
 Three suffix table size is        3
 Blank table size is               5

 Number of records read is                1135
 Number of names standardardized is       1135
```

Example of report from *sumnp*

- *output2* Ignore this file. *Output2* is opened during the execution of the GDRIVER.EXE, but nothing is written to it, so when GDRIVER finishes executing, it is a null file.

At this point you are ready to create a separate directory for File B and repeat the entire process above using File B. After you have completed standardizing File B, use the two *stanout.txt* files as the input for the next stage of the record matching process.

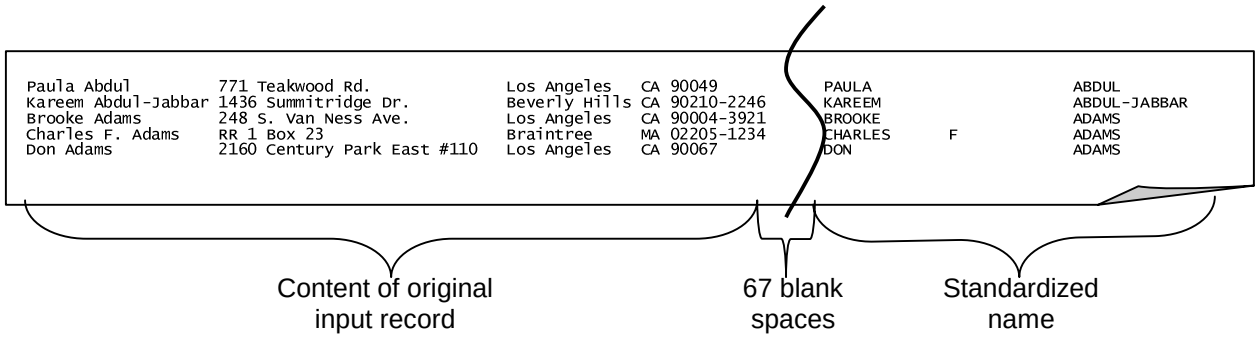
4.3 Use of GDRIVER for Name Standardization Only

To use the GDRIVER to standardize names alone, follow the same basic procedures described in Section 4.2. The only difference comes in step E. After copying the file *parmdr.txt* to the directory, edit this file: enter "0" as the length of the address field. For example:

```
0  30  30  0  100
```

Here the fourth number (0) specifies that the length of the address field is zero, so the GDRIVER program ignores the field. Note that the starting position of the address field (the second 30) isn't "zeroed" out. The GDRIVER program ignores the starting column when the length is zero.

This change means the *stanout.txt* file would look different than what is shown in **Figure 4**: the fields for the standardized address are blank. In other words, the output file will show each original input record, 67 blank spaces, and then the standardized name fields in the layout specified in Section 4.2.



***stanout.txt* for Standardization of Names Only**

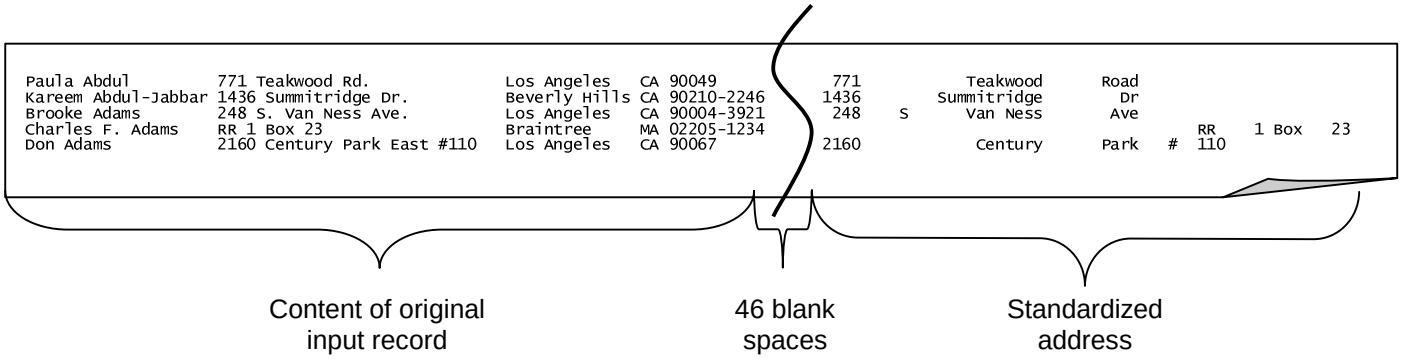
4.4 Use of GDRIVER for Address Standardization Only

To use the GDRIVER to standardize addresses alone, follow the same basic procedures described in Section 4.2. The only difference comes in step E. After copying the *parmdr.txt* file to the directory, edit this file: enter "0" as the length of the name field. For example:

```
0  0  30  40  100
```

Here the second number (0) specifies that the length of the name field is zero, so the GDRIVER program ignores the field. Note that the starting position of the name field (the first 0) remains. The GDRIVER program ignores the starting column when the length is zero.

This change means the *stanout.txt* file would look different than what is shown in **Figure 4**: the fields for the standardized name are blank. In other words, the output file will show each original input record, 46 blank spaces, and then the standardized address fields in the layout specified in Section 4.2.



***stanout.txt* for Standardization of Address Only**

4.4 Name Standardization Reference Files Used in GDRIVER

There are 11 reference files that must be in the directory where GDRIVER.EXE resides. Each of these files modifies text strings that have been determined through experience to require special attention. These files can be used as is, or you can edit these files, customizing the standardization process to the particular needs of your project. Each of the reference files is listed below, and its corresponding special processing is briefly discussed. The entries in these files must appear in alphabetical order.

- *blnk.dat* A list of words which the standardizer replaces with blanks if they appear in the "name" field; examples include "Construction," "Excavating," and "Owner." (Note: these strings are only blanked when the standardizer associates them with a *last* name.) These are terms that would appear in the names of businesses but would not be useful for matching because of their common occurrence.

- *conj.dat* A list of conjunctions that may appear in the name field, indicating that two or more standardized records are to be produced; examples include "&," "+," and "C/O."

- *jrsr.dat* A list of generational suffixes; examples include "JR," "III," and "2ND." The software also converts words such as "THIRD" to "III," or "2ND" to "II."

- *nick.dat* A list of names that are used to replace variations on first or middle names; examples include "Al" replacing "Albert," "Alberto," "Alvin," and "Alva" and "Barbara" replacing "Barb" and "Barbie."

- *patt.dat* A list of name patterns that the different name elements might exhibit; the name standardizer uses these patterns as instructions about how to rearrange the elements from their order in the file being standardized to a common order.

- *pref.dat* A list of prefixes found in last names; examples include "de," "La," and "Vander." When the software encounters these prefixes it attaches them to the beginning of the last name. For example, "Jim St John" becomes "Jim Stjohn."

- *skpl.dat* A list of words indicating that the entity in question should be skipped by the name standardizer; examples include "Academy," "Church," "Theatre," "Inc," and "Corp." These terms also indicate business or institutional names.

- *suff.dat* A list of occupational suffixes that the standardizer identifies; examples include "DDS," "Esq," and "PHD."

- *thre.dat* A list of three-letter occupational suffixes with embedded spaces; the standardizer deletes the spaces and then looks for the suffix in *suff.dat*; examples include "C P A" and "D M D."

- *titl.dat* a list of prefix titles that the standardizer identifies; examples include "Mr," "Dr," and "Rev."

- *two.dat* a list of two-letter occupational suffixes with embedded spaces; the standardizer deletes the spaces and then looks for the suffix in *suff.dat*; examples include “P C” and “M D.”

4.5 Address Standardization Reference Files Used in GDRIVER

The file *initfiles.dat* refers to a collection of files that store key address words, their variant spellings, and the various address patterns to be recognized. A brief description of the files is provided below:

- *clue_abbrev.dat* (Clue Word Abbreviation Table)—This file contains all the variant spellings or alternate representations for the key words to be identified. The file also indicates what type or types of keyword this clue could be. For example, “Ave” is a street type; “RD” could be a street type or a rural route type.

- *master_clues.dat* (Master Clue Word Table)—This file has a single record for each keyword. For example, there is one record for “Avenue,” not separate records for “Av,” “Ave,” “Avenue,” etc. Each record has a space for several “official” variant forms of the word that might be desired on output, including the full name and an official USPS abbreviation.

- *patterns.dat* (Pattern Table)—The Pattern Table contains input patterns of keyword types, and indicates which output field each of the input key words should get mapped to, as part of the parsing process. Each pattern also has a priority code, which supports a scoring system that is used to resolve ambiguous clue words.

5.0 SORTING AND BLOCKING RECORDS FOR MATCHING

We have presented some of the theoretical considerations concerning blocking records in Section 2.2.3. This section specifies how to execute your blocking strategy. Once you have decided which variables you will block on, you need to sort both of the files that will be used in the match by these variables. The programs distributed by SRD do not include a sort utility. There are many different options available for sorting files, with one such sort utility being QSORT. This is a public domain sorting utility available at the Bureau of the Census, and it is the utility that we are documenting here.

The following is a guide to using QSORT:

(1) Create a directory for the sorting to take place.

(2) Place the QSORT.EXE program in this directory.

(3) Place the file(s) to be sorted in this directory.

(4) Rename any input file having a long file name to one with eight or fewer characters (e.g., *stanoutput.txt* to *stanouta.txt*).

Once this is done, you are ready to execute the sorting utility. The sort can be run in the Windows or DOS environment. An example of the command needed to execute the sort is as follows:

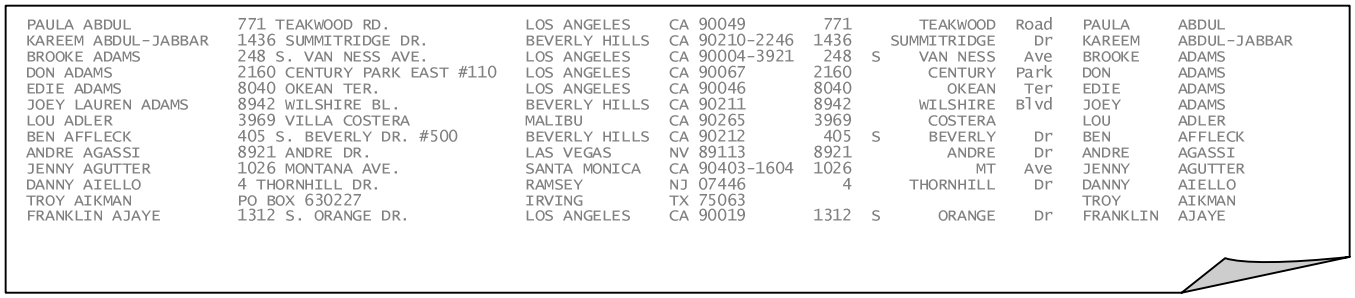
```
QSORT  STANOUTA.TXT SORTA.DAT /73:2 /96:5
```

This command is made up of four components:

`QSORT`	This command calls the program (QSORT.EXE).
`STANOUTA.TXT`	This is the input file (which will be the output from the standardization program or the unmatched records from one of the matching runs).
`SORTA.DAT`	This is the name of the file that will be created from the sorting utility.
`/73:2 /96:5`	This component tells the sort program which variable(s) to sort by. The command must be in this format for each variable; that is, a slash, followed by a starting position, followed by a colon, followed by the number of positions occupied by the variable.

In this example, the sort program will sort first on the data in positions 73–74 and then on the data in positions 96–100.

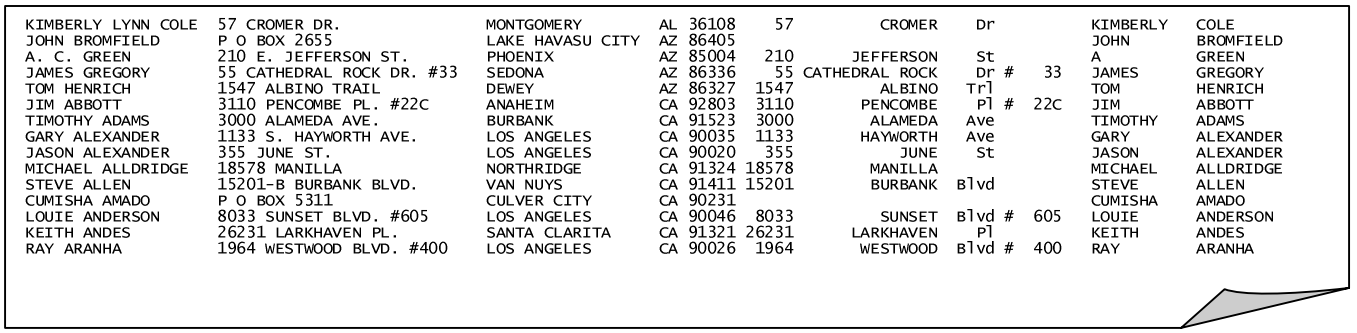
The output file created from the sort utility will have the same file layout as the input file.



PAULA ABDUL	771 TEAKWOOD RD.	LOS ANGELES	CA	90049	771		TEAKWOOD	Road	PAULA	ABDUL
KAREEM ABDUL-JABBAR	1436 SUMMITRIDGE DR.	BEVERLY HILLS	CA	90210-2246	1436		SUMMITRIDGE	Dr	KAREEM	ABDUL-JABBAR
BROOKE ADAMS	248 S. VAN NESS AVE.	LOS ANGELES	CA	90004-3921	248	S	VAN NESS	Ave	BROOKE	ADAMS
DON ADAMS	2160 CENTURY PARK EAST #110	LOS ANGELES	CA	90067	2160		CENTURY	Park	DON	ADAMS
EDIE ADAMS	8040 OKEAN TER.	LOS ANGELES	CA	90046	8040		OKEAN	Ter	EDIE	ADAMS
JOEY LAUREN ADAMS	8942 WILSHIRE BL.	BEVERLY HILLS	CA	90211	8942		WILSHIRE	Blvd	JOEY	ADAMS
LOU ADLER	3969 VILLA COSTERA	MALIBU	CA	90265	3969		COSTERA		LOU	ADLER
BEN AFFLECK	405 S. BEVERLY DR. #500	BEVERLY HILLS	CA	90212	405	S	BEVERLY	Dr	BEN	AFFLECK
ANDRE AGASSI	8921 ANDRE DR.	LAS VEGAS	NV	89113	8921		ANDRE	Dr	ANDRE	AGASSI
JENNY AGUTTER	1026 MONTANA AVE.	SANTA MONICA	CA	90403-1604	1026		MT	Ave	JENNY	AGUTTER
DANNY AIELLO	4 THORNHILL DR.	RAMSEY	NJ	07446	4		THORNHILL	Dr	DANNY	AIELLO
TROY AIKMAN	PO BOX 630227	IRVING	TX	75063					TROY	AIKMAN
FRANKLIN AJAYE	1312 S. ORANGE DR.	LOS ANGELES	CA	90019	1312	S	ORANGE	Dr	FRANKLIN	AJAYE

Example of *sorta.dat*

After standardizing File B, rerun the QSORT utility, renaming the input *stanouta.txt* to *stanoutb.txt*, and replacing *sorta.dat* with *sortb.dat* as the destination filename.



KIMBERLY LYNN COLE	57 CROMER DR.	MONTGOMERY	AL	36108	57		CROMER	Dr	KIMBERLY	COLE
JOHN BROMFIELD	P O BOX 2655	LAKE HAVASU CITY	AZ	86405					JOHN	BROMFIELD
A. C. GREEN	210 E. JEFFERSON ST.	PHOENIX	AZ	85004	210		JEFFERSON	St	A	GREEN
JAMES GREGORY	55 CATHEDRAL ROCK DR. #33	SEDONA	AZ	86336	55		CATHEDRAL ROCK	Dr # 33	JAMES	GREGORY
TOM HENRICH	1547 ALBINO TRAIL	DEWEY	AZ	86327	1547		ALBINO	Trl	TOM	HENRICH
JIM ABBOTT	3110 PENCOMBE PL. #22C	ANAHEIM	CA	92803	3110		PENCOMBE	Pl # 22C	JIM	ABBOTT
TIMOTHY ADAMS	3000 ALAMEDA AVE.	BURBANK	CA	91523	3000		ALAMEDA	Ave	TIMOTHY	ADAMS
GARY ALEXANDER	1133 S. HAYWORTH AVE.	LOS ANGELES	CA	90035	1133		HAYWORTH	Ave	GARY	ALEXANDER
JASON ALEXANDER	355 JUNE ST.	LOS ANGELES	CA	90020	355		JUNE	St	JASON	ALEXANDER
MICHAEL ALLDRIDGE	18578 MANILLA	NORTHRIDGE	CA	91324	18578		MANILLA		MICHAEL	ALLDRIDGE
STEVE ALLEN	15201-B BURBANK BLVD.	VAN NUYS	CA	91411	15201		BURBANK	Blvd	STEVE	ALLEN
CUMISHA AMADO	P O BOX 5311	CULVER CITY	CA	90231					CUMISHA	AMADO
LOUIE ANDERSON	8033 SUNSET BLVD. #605	LOS ANGELES	CA	90046	8033		SUNSET	Blvd # 605	LOUIE	ANDERSON
KEITH ANDES	26231 LARKHAVEN PL.	SANTA CLARITA	CA	91321	26231		LARKHAVEN	Pl	KEITH	ANDES
RAY ARANHA	1964 WESTWOOD BLVD. #400	LOS ANGELES	CA	90026	1964		WESTWOOD	Blvd # 400	RAY	ARANHA

Example of *sortb.dat*

6.0 RECORD MATCHING PROGRAM

After Files A and B have been standardized and sorted, they can be run through the SRD record matching program. The Appendix describes a sample matching task using two files containing names and addresses.

Figure 5 illustrates the record matching process. After the two standardized files, A and B, have been sorted by the blocking variables, the user must supply the locations of the sorted, standardized files through creation of the file *parmn.txt* (see Section 6.1). The user must also provide a set of parameters through creation of *parmmf.txt* (see Section 6.2). The matcher (MATCHER.EXE) refers to these files and produces three output files and a summary report. The three output files are:

- *prntd1.dat* Possible matches
- *prntd2.dat* Unlikely matches with scores less than 0
- *prntmf.dat* A pointer file used in the next step to process and sort the residuals or unmatched records

The possible matches in *prntd1.dat* must be sorted by descending score; then, run PRINTER.EXE to produce a list of matches from *prntd1.dat* ordered from highest to lowest score. The user must review this file to decide on cutoff scores to input into the next step. Together, *prntd1.dat* and *prntd2.dat* will include all matches with scores between the two cutoff values specified in Set 5 of the parameter file *parmmf.txt* (see Section 6.2).

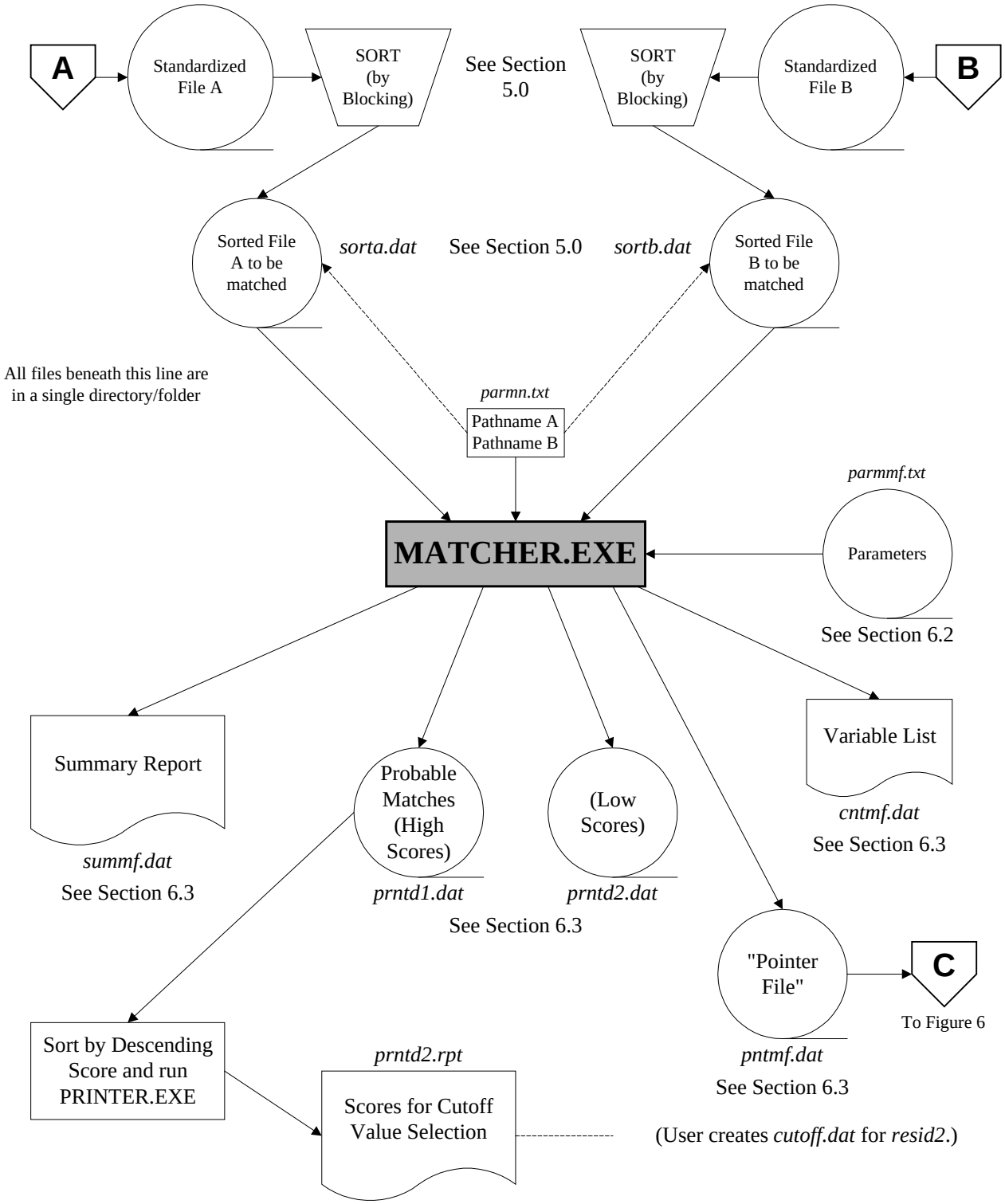
6.1 Specifying the Location of the Input Files

Create *parmn.txt*. This file contains two lines with the names and locations of the two input files—File A and File B. If File A and File B are located in the directory where the matching will occur, *parmn.txt* only needs to contain the file names.

```
c:\stanoutA.srt
c:\stanoutB.srt
```

Example of *parmn.txt*

Figure 5: Matching



6.2 Creating the Parameter File

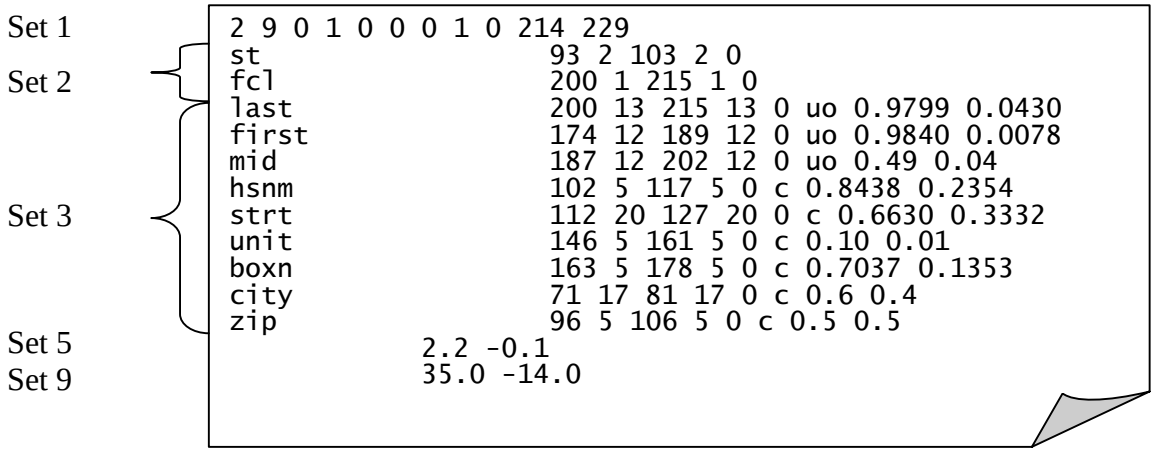
Create *parmmf.txt*. This file is the user's mechanism for controlling the functioning of the matching program. You may choose to modify many of the third set of variables based on the contents of *init.dat* (an output from Advance Automatic Adjustment Features).

The parameter file *init.dat* is optional and is used to replace matching probabilities in *parmmf.txt*. For the replacement to succeed, the first three characters of the variable name in *init.dat* must agree with the first three characters of the corresponding variable name in *parmmf.txt*.

An example of *parmmf.txt* is shown below. The labels "Set 1," "Set 2," etc. are not part of the file but are used for reference. Since some sets vary in number of lines, they are not referred to here by line number.

Please note:

- Each line in the parameter file must end with a blank space.
- Maximum input record size is 499.
- The maximum size of a single field (e.g., first) is 60.



Example of *parmmf.txt*

Our example does not include Sets 4, 6, 7, and 8 because they are not generally used. However, these sets are briefly described in the appropriate places in the discussion.

Set 1 consists of the following parameters. The parameters are separated by spaces.

- Number of blocking fields. (there are **2** blocking fields in the example below, so the program will interpret two lines as containing information about the blocking variables)

- Number of matching variables. (there are **9** comparison fields in the example, so the program will interpret nine lines as containing information about the matching variables)

- Number of locations of sequence numbers. Users not familiar with this feature should use 0; 1 is for one location. More than one location may not be specified. (**0** is used in the example)

- Cutoff switch. If this switch is set to 1, the two cutoffs in Set 9 are read in; if it is set to 0, they are not. (the cutoff switch is set to **1** in the example) See Set 9, described below.

- Number of tables. This must be a value between 0 and 2. Use 0 unless you know how to use the options for 1 and 2 (not documented here). (there are **0** tables in the example)

- Downweight switch. This switch should always be set to 0 (downweight switch = 0 means that no adjustments are made to the weights). If it is set to 1, 2, or 3 (not documented here), various errors are likely to occur. (the downweight switch is set to **0** in the example)

- Number of ID fields. This is used for testing in situations where the truth of matches is known (i.e., the user obtains truth data sets, that can be used to improve the matching. When the ID field contains a 1, the pointer file—and the print files—will have the indicator "t" or "f" placed in them. If the number of ID fields is 0, then "f" is placed in all appropriate locations in the pointer file and the print files. (there are **0** ID fields in the example)

- Print switch. If this switch is set to 1, a subset of the matches will be written to the two files, *prntd1.dat* and *prntd2.dat,* from which they may be printed. Together these will include all the matches with scores between the high and low cutoff values specified in Set 9 described below. If the switch is set to 0, no matches will be written to the *prntd1.dat* and *prntd2.dat.* (the print switch is set to **1** in the example)

- Exact switch. This switch activated features in earlier versions of the software that are no longer included; therefore, it should always be set to 0. (the exact switch is set to **0** in the example)

- Length, in characters, of records in File A. (**214** characters long in the example)

- Length, in characters, of records in File B. (**229** characters long in the example)

Set 1

```
2 9 0 1 0 0 0 1 0 214 229
st                      93 2 103 2 0
fcl                     200 1 215 1 0
last                    200 13 215 13 0 uo 0.9799 0.0430
first                   174 12 189 12 0 uo 0.9840 0.0078
mid                     187 12 202 12 0 uo  0.49 0.04
hsnm                    102 5 117 5 0 c 0.8438 0.2354
strt                    112 20 127 20 0 c 0.6630 0.3332
unit                    146 5 161 5 0 c 0.10 0.01
boxn                    163 5 178 5 0 c 0.7037 0.1353
city                    71 17 81 17 0 c 0.6 0.4
zip                     96 5 106 5 0 c 0.5 0.5
                2.2 -0.1
                35.0 -14.0
```

Set 1 of *parmmf.txt*

Set 2 consists of one line for each blocking field. Each line must include the following information in the order given. The parameters must be separated by spaces.

- Name of blocking field. This parameter must be exactly 20 character positions long. If the name of the blocking field is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the blocking field labels in the example below are **st** and **fcl**)

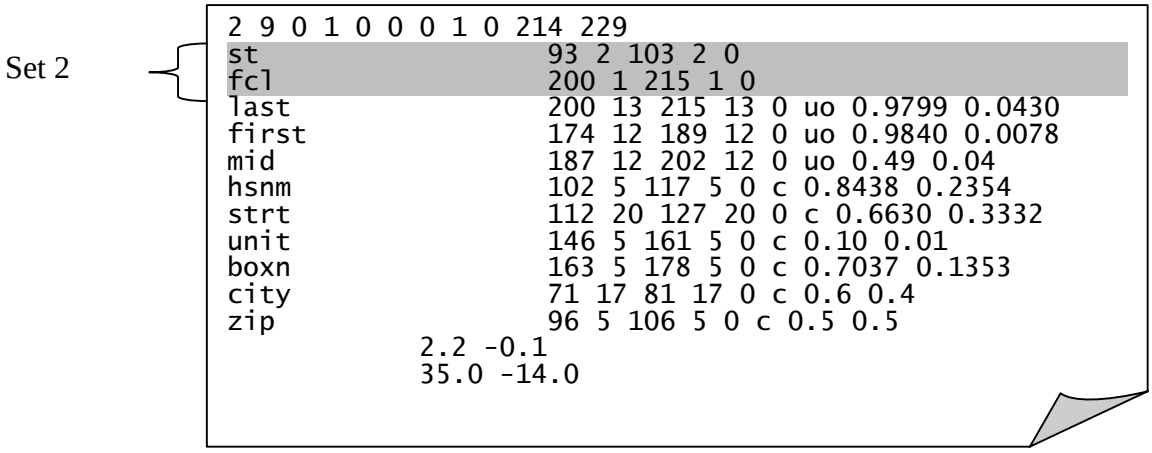
- Starting position of the blocking field in File A. ("st" starts in position **93** in File A in the first line of Set 2 in the example)

- Length, in characters, of the blocking field in File A. (the blocking field is **2** characters long in File A in the first line of Set 2 in the example)

- Starting position of the blocking field in File B. ("st" starts in position **103** in File B in the first line of Set 2 in the example)

- Length, in characters, of the blocking field in File B. (the blocking field is **2** characters long in File B in the first line of Set 2 in the example)

- Null-data switch. This parameter is currently nonoperational; use 0 as a place-holder. (the null-data switch is set to **0** in the first line of Set 2 in the example)



Set 2 of *parmmf.txt*

Set 3 consists of one line for each set of comparison or matching variables. Each line must include the following information in the order given. The parameters must be separated by spaces.

- Name of matching variable. This parameter must be exactly 20 character positions long. If the name of the matching variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the first comparison field label is **last** in the example below)

- Starting position of the matching variable in File A. (the field starts in position **200** in File A in the first line of Set 3 in the example)

- Length, in characters, of the matching variable in File A. (**13** characters long in the first line of Set 3 in the example)

- Starting position of the matching variable in File B. (the field starts in position **215** in File B in the first line of Set 3 in the example)

- Length, in characters, of the matching variable in File B. (**13** characters long in the first line of Set 3 in the example)

- Null-data switch. This parameter is currently nonoperational; use 0 as a place-holder. (the null-data switch is set to **0** in the first line of Set 2 in the example)

- Type of comparison (**uo** is used in the first line of Set 3 in the example)
  Use one of the following eight comparisons:

c	exact comparison: the variables being matched must agree character-by-character in order to receive the full agreement weight
p	prorated comparison by 15 percent for numeric: for a quantitative variable such as age, this method assigns the full agreement weight if the values are within 1 but adjusts it downward in a linear matter if the difference in the two variables is 15 percent or less of the lower value; if the difference is greater than 15 percent, the full disagreement weight is assigned
y	calls a feature developed for a specific project; should not be used
uo	ordinary string comparator/weighting
uf	special string comparator/weighting for first name
ul	special string comparator/weighting for last name
un	special string comparator/weighting for numeric address component
u	modified ordinary string comparator/weighting

  One of the following four codes may be used with the comparison types above (these are discussed further on the following page):

I	inversion: if the current fields are not alike, compares the field with the next field on the list to see if the fields have been inverted and, if they match, assigns the same weight as if they had been in the same order

i	inversion: if the current fields are not alike, compares the field with the next field on the list to see if the fields have been inverted and, if they match, assigns half the weight as if they had been in the same order
x	cross: if two or more important fields match, their matching weight is doubled
k	critical variable in a cross match involving three or more variables; see below

Beginning users should not use the following two comparisons:

f1	frequency-based comparison using *table1.dat*
f2	frequency-based comparison using *table2.dat*

These two parameters (f1 and f2) support frequency-based matching. They require advanced knowledge of record matching and should not be employed by beginning users. Their use requires additional programs and documentation that are not included here.

- Probability #1 (**0.9799** in the first line of Set 3 in the example). This is the probability, m_i, that two records agree on the variable if they are actually a match. $m_i = P$ (i^{th} field agrees | *(a,b)* 0 *M*)

- Probability #2 (**0.0430** in the first line of Set 3 in the example). This is the probability, n_i, that two records agree on the variable if they are *not* a match. $n_i = P$ (i^{th} field agrees | *(a,b)* 0 *U*)

If Probabilities #1 and #2 are far apart, the matching variable has a large distinguishing power; if they are both close to 0.5, the matching variable has little distinguishing power. Based on experience or intuition, you may specify values for these probabilities for each matching variable in Set 3 of *parmmf.txt.* You may adjust these values based on the contents of the files to be matched by using the counter program and expectation-maximization algorithm documented in Section 7.0.

Set 3

```
2 9 0 1 0 0 0 1 0 214 229
st                  93 2 103 2 0
fcl                 200 1 215 1 0
last                200 13 215 13 0 uo 0.9799 0.0430
first               174 12 189 12 0 uo 0.9840 0.0078
mid                 187 12 202 12 0 uo 0.49 0.04
hsnm                102 5 117 5 0 c  0.8438 0.2354
strt                112 20 127 20 0 c 0.6630 0.3332
unit                146 5 161 5 0 c 0.10 0.01
boxn                163 5 178 5 0 c 0.7037 0.1353
city                71 17 81 17 0 c 0.6 0.4
zip                 96 5 106 5 0 c 0.5 0.5
          2.2 -0.1
          35.0 -14.0
```

Set 3 of *parmmf.txt*

Comparison of multiple fields: The "inversion," "cross," and "critical" comparisons have been designed for some special situations.

Inversion comparison. The "i" or "I" options allows users to compare a field with the next field on the list to see if the fields have been inverted. Imagine that the last names in each file are compared and a corresponding weight is assigned.

```
last                1 12 1 12 0 uoi 0.99223 0.32432
first               14 12 14 12 0 uo 0.99231 0.21421
```

However, there exists the possibility that the data in the two fields were inverted. For example, last name "Edward" and first name "Porter" would appear in the other files as last name "Porter" and first name "Edward."

By adding the "i" option following the "uo," the program compares the "last" field against the "first" field. This inversion comparison uses the same matching methodology as the first field match; that is, a "uoi" uses the string comparator. A "cI" parameter specifies an exact match. If the inversion code is a lower-case letter (i), a match between the first field on one record and the second field on the other record is assigned a weight of one-half of the weight specified for the first matching variable. If the inversion code is a capital letter (I), a match between the first field on one record and the second field on the other record is assigned the full weight for the first variable.

Cross comparison. In this type of comparison, if all of the variables marked as cross variables match, then their matching weight is doubled. If only some of the cross variables match, then only the associated weights for those variables in agreement are assigned. Suppose that a user discovers that if two or more particular fields match it increases the probability of a match; for example "first name," "month of birth," "day of birth," and "year of birth." The following is an example of the parameters in Set 3 for these four matching variables:

```
first               14 12 14 12 0 uox 0.99231 0.21421
month               28 2 28 2 0 cx 0.93932 0.2132
day                 30 2 30 2 0 cx 0.9232 0.343
year                32 4 32 4 0 cx 0.9 0.2
```

If the four matching variables were specified as "uo" and "c" instead of "uox" and "cx," and all four matched, the matching weight would only be increased by the variable weights associated with those variables. However, these variables are designated by the "x" extension as "cross comparison" variables. Therefore, if all four match, the total matching weight will be augmented by twice as much: once for matching like any other matching variables, and once more because these variables are designated as "cross comparison" variables.

Critical comparison. If one of the variables in a cross match is designated as "critical" by replacing the "x" code with "k," then only three of four or more cross match fields have to agree for the weights of the agreeing fields to be doubled as long as one of the agreeing fields is the critical field. For example, the following parameters might be included in Set 3:

```
first                14 12 14 12 0 uok 0.99231 0.21421
month                28 2 28 2 0 cx 0.93932 0.2132
day                  30 2 30 2 0 cx 0.9232 0.343
year                 32 4 32 4 0 cx 0.9 0.2
```

If first name, month of birth, and year of birth agreed, the matching weight would be increased by the sum of the weights for those three variables twice, once for ordinary matching and once again because the variables are cross-match variables and include the critical variable.

Set 4 is present only if the third variable of Set 1 contains a 1. The sequence field allows an individual to check matching results which are brought together as pairs against the records in the original files being matched. It consists of the following specifications, separated by spaces.

Set 4 is rarely used, so it is not shown in the example.

- Name of sequence variable. This parameter must be exactly 20 character positions long. If the name of the sequence variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters.

- Starting position of sequence variable on File A.

- Length, in characters, of sequence variable on File A.

- Starting position of sequence variable on File B.

- Length, in characters, of sequence variable on File B.

Set 5 contains the high and low cutoffs for the match, if this option is being used. If the fourth parameter in Set 1 is equal to 1 then this option is being used. It consists of two specifications.

- High cutoff value, the score below which you are certain very few correct linkages exist. (**2.2** in the example below)

- Low cutoff value, the score above which you are certain few incorrect linkages exist. (**-0.1** in the example)

Pairs of records with scores between these two values must be reviewed individually to determine whether they are matches or residuals.

If the probabilities are assigned by logistic regression, the probability that a linkage with a score of *X* is correct equals *X/(X+1).* For example, the probability that a pair with a score of 15.5 is a correct match is 15.5/16.5 = 0.939. In other words, you are 93.9 percent confident that a match with a score of 15.5 is correct.

Set 5

```
2 9 0 1 0 0 0 1 0 214 229
st                  93 2 103 2 0
fcl                 200 1 215 1 0
last                200 13 215 13 0 uo 0.9799 0.0430
first               174 12 189 12 0 uo 0.9840 0.0078
mid                 187 12 202 12 0 uo 0.49 0.04
hsnm                102 5 117 5 0 c 0.8438 0.2354
strt                112 20 127 20 0 c 0.6630 0.3332
unit                146 5 161 5 0 c 0.10 0.01
boxn                163 5 178 5 0 c 0.7037 0.1353
city                71 17 81 17 0 c 0.6 0.4
zip                 96 5 106 5 0 c 0.5 0.5
            2.2 -0.1
            35.0 -14.0
```

Set 5 of *parmmf.txt*

Sets 6–7 support advanced functions and features included in the software for specific applications. They should not be altered by beginning users. Set 6 is associated with the f1 and f2 parameters in Set 5. The Set 7 parameter was developed for a special project and should always be set to 0.

Sets 6-8 are rarely used, so they are not shown in the example.

Set 8 is a toggle switch for the high and low cutoffs specified in Set 5. If the value of the parameter is 1, then the program will use the high and low cutoffs defined in Set 5. If it is 0, Set 5 is bypassed.

Set 9 consists of one line that contains the high cutoff and low cutoff values for printing. If the eighth parameter in Set 1 is equal to 1, then this option is being used. It consists of two specifications.

- High cutoff value, the score above which you do not wish to view any records from the match. (**35.0** in the example below)

- Low cutoff value, the score below which you do not wish to view any records from the match. (**–14.0** in the example)

These values may be equal to the match cutoff values, but do not have to be. The pairs with a positive match weight will be output to *prntd1.dat* and the pairs with a negative weight will be output to *prntd2.dat*.

Set 9

```
2 9 0 1 0 0 0 1 0 214 229
st                  93 2 103 2 0
fcl                 200 1 215 1 0
last                200 13 215 13 0 uo 0.9799 0.0430
first               174 12 189 12 0 uo 0.9840 0.0078
mid                 187 12 202 12 0 uo 0.49 0.04
hsnm                102 5 117 5 0 c 0.8438 0.2354
strt                112 20 127 20 0 c 0.6630 0.3332
unit                146 5 161 5 0 c 0.10 0.01
boxn                163 5 178 5 0 c 0.7037 0.1353
city                71 17 81 17 0 c 0.6 0.4
zip                 96 5 106 5 0 c 0.5 0.5
            2.2 -0.1
            35.0 -14.0
```

Set 9 of *parmmf.txt*

If the program fails to run properly, error messages will be in *summf.dat.* Error messages typically relate to increasing the size of the data structures that hold data.

6.3 Running the Program

Once the required parameter files have been created, you are ready to run the program. Before running MATCHER.EXE, make sure the following input files are in the directory:

- *parmn.txt* This file contains the name and location of the files to be matched.
- *parmmf.txt* This file contains all of the parameters necessary to perform the match.
- *sorta.dat* The sorted and standardized records from File A.
- *sortb.dat* The sorted and standardized records from File B.
- MATCHER.EXE The matching program has to be present in this directory.

Creation of *sorta.dat* and *sortb.dat* is discussed in Section 5.0.

Run MATCHER.EXE. It will produce the following files.

- *summf.dat*

  This is the summary statistics file. It contains information on:

  (1) The number of records subjected to match from both files.
  (2) The total number of pairs examined.
  (3) The number of records from each file that could not be paired to a record in the other file.
  (4) The number of linked pairs for which the matching score exceeds the high cutoff parameter of Set 5 in *parmmf.txt* (i.e., matched pairs).
  (5) The number of linked pairs for which the matching score falls between the high cutoff and the low cutoff parameter of Set 5 in *parmmt.txt* (i.e., clerical review pairs).
  (6) The sizes of the largest three blocks in each match.

To access *summf.dat, prntd1.dat, prntd2.dat,* or *pntmf.dat* for viewing and printing, you can use any text editor or word processing program.

- *prntd1.dat* and *prntd2.dat*

  These files identify the best linkage from File B to File A. Each record in the file contains the following information in the following order:

  (1) The score;
  (2) The block identifiers;
  (3) The value of each matching variable from File A (listed in the order that the matching variables are listed in Set 3 of the parameter file, *parmmf.txt*);
  (4) The value of each matching variable from File B (listed in the order that the matching variables are listed in Set 3 of the parameter file, *parmmt.txt*).

Prntd1.dat contains the content for a list of pairs with high scores (positive scores); *prntd2.dat* contains the content for a list of pairs with low scores (negative scores).

- *pntmf.dat* This file contains instructions, based on the results of the match, specific to reading the input data files that are being compared. This is a necessary input file for the residuals program.

- *cntmf.dat* This file contains a list of the matching variables, and their associated m- and u-probabilities.

6.4 Interpreting the Output

To interpret the output, you need to sort the output files from the matching run, and then print the files in a readable format.

The files need to be sorted in descending order (based on the weights from the matching run). QSORT, which had been used previously, is unsuitable for this purpose (i.e., QSORT does not have the capability to sort in descending order). For this sort, we are recommending the DOS sort utility. An example of the command needed to execute the sort utility is as follows:

```
SORT /R < PRNTD1.DAT > PRNTD1.SRT
```

This command is made up of six components:

SORT	This command calls the DOS sort utility.
/R	This component instructs the sort utility to sort in descending order.
<	This instructs DOS that the name of the input file is to follow.
PRNTD1.DAT	This is the name of the input file (which is the output file from the MATCHER program that contains matches with a positive weight).
>	This instructs DOS that the name of the output file is to follow.
PRNTD1.SRT	This is the name of the output file that will be created from the sort utility.

After the files have been sorted in descending order, you are ready to run PRINTER.EXE.

Before running PRINTER.EXE, make sure the following input file is in the directory:

- *prntd1.srt*

An example of the command needed to execute PRINTER.EXE is as follows:

```
PRINTER < PRNTD1.SRT > PRNTD1.RPT
```

This command is made up of five components:

PRINTER	This command calls the PRINTER.EXE program.
<	This instructs DOS that the name of the input file is to follow.
PRNTD1.SRT	This is the name of the input file (which is the output file from the sort utility.
>	This instructs DOS that the name of the output file is to follow.
PRNTD1.RPT	This the name of the output file that will be created from the PRINTER program.

The other output file from the matching program, *prntd2.dat*, should also be sorted, and then run through PRINTER.EXE.

6.5 Using the Residuals

The next step in the matching process is to use the residuals process, illustrated in **Figure 6**. Like MATCHER.EXE, the residuals program, RESID2.EXE, requires a user-provided file, *parmn.txt*, to tell it where to find the sorted data files, A and B. The user must also supply the file *cutoff.dat*, containing the selected cutoff scores based on inspection of the initial matcher output. The residuals program module creates two files of non-matches, for Files A and B respectively, named *r_sorta.dat* and *r_sortb.dat.* These files are substituted for the original sorted data files in a second pass through the matcher, MATCHER.EXE.

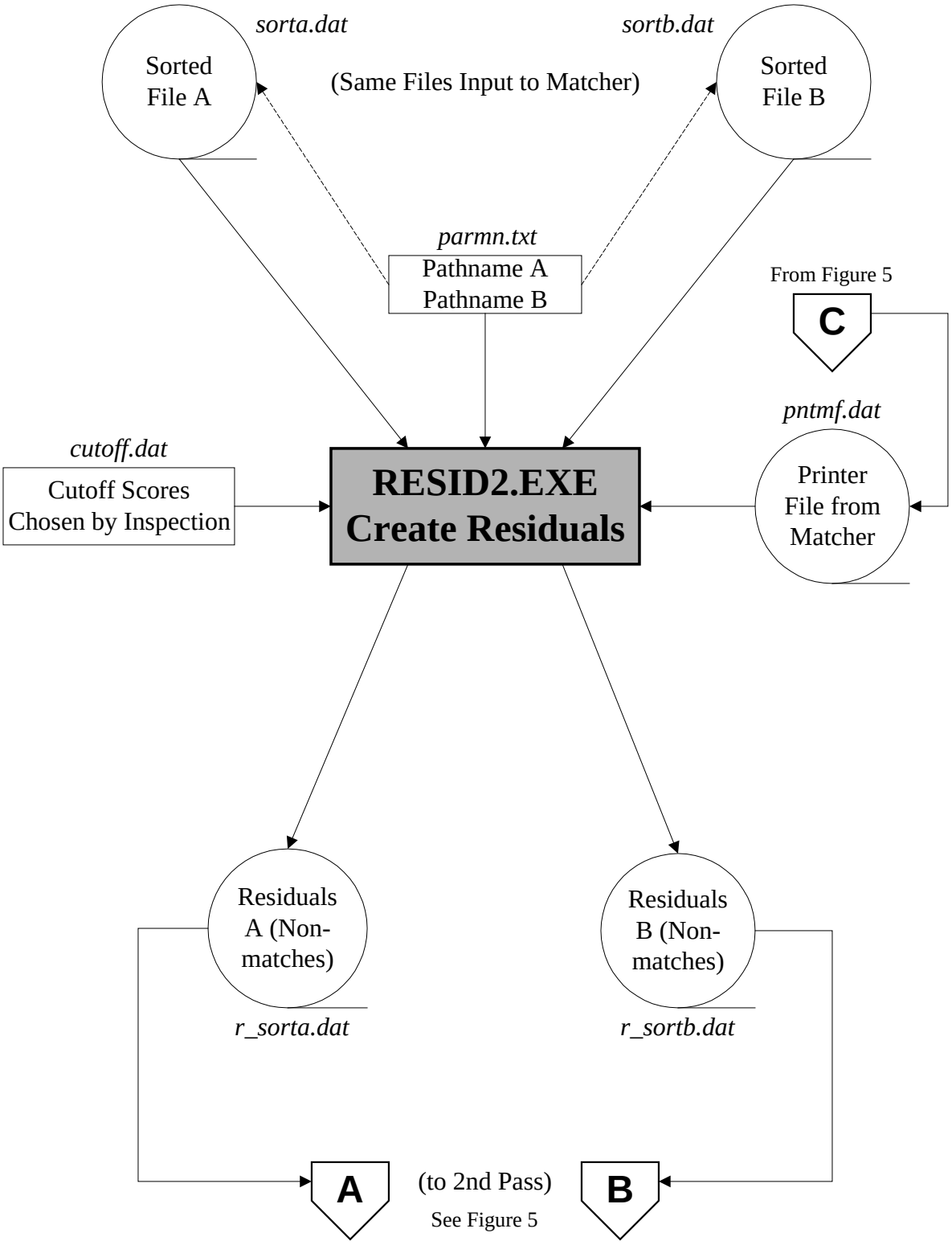
The file *parmn.txt* has been described in section 6.1.

Begin by creating the file *cutoff.dat*: Review the matches with positive scores listed in *prntd1.rpt*. Find the score of the first occurrence that appears to be a non-match. This will be the upper cutoff. Then find the score below which all records appear to be non-matches. This will be the lower cutoff. In the example, the beginning score for non-matches is the upper cutoff of 3.0 and the lowest score for a match is the lower cutoff value of 1.0.

```
3.0 1.0
```

Example of *cutoff.dat*

Figure 6: Residuals



Before running RESID2.EXE, make sure the following input files are in the directory:

- *sorta.dat* Consists of the cases in the original File A after standardization and sorting. This file was created when the standardized file was sorted by the blocking variables; see Section 5.0.
- *sortb.dat* Consists of the cases in the original File B after standardization and sorting. This file was created when the standardized file was sorted by the blocking variables; see Section 5.0.
- *parmn.txt* Specifies the names and locations of *sorta.dat* and *sortb.dat.*
- *parmmf.txt* Described in Section 6.2.
- *pntmf.dat* Created by MATCHER.EXE.
- *cutoff.dat* As explained above.

Run RESID2.EXE.

The following files contain the non-matches, which will be sorted on new blocking variables and re-matched on a subsequent run of the matching program.

- *r_sorta.dat*
- *r_sortb.dat*

After sorting, they are ready for a second matching attempt. See Section 5.0 for more information about matching. ***The second match should be run in a different directory from the first. Otherwise, the old versions of several files will be lost.***

6.6 Executing the Second and Subsequent Matching Runs

For a second or subsequent matching run, four input files are required:

- *r_sorta.srt* This is the sorted residual File A (i.e., those records from File A that did not match on the first match).
- *r_sortb.srt* This is the sorted residual File B (i.e., those records from File B that did not match on the first match).
- *parmn.txt* Specifies the names and locations of the input files.
- *parmmf.txt* This file contains the necessary parameters to run the match.

Rerun MATCHER.EXE by repeating the steps beginning in Section 6.1.

7.0 ADVANCED AUTOMATIC ADJUSTMENT FEATURES

The advanced automatic adjustment features allow additional control over the matching process by users who are well versed in the mathematical theory of record matching.

7.1 Creating Sorted Data Files

Rename the first sorted data file as *sorta.dat.*

```
ab11akbar       ali             25000001
ab11akbar       susan           24000002
ab11akbar       john            3000003
ab11smith       robert          35000004
ab11smith       barbara         35000005
ab11smith       stephen         12000006
ab12jonesc      ali             25000007
ab12jonesc      susan           24000008
ab12smithc      john            3000009
ab12smithc      robert          35000010
ab21barker      john            44000011
ab21barker      robert          42000012
ab21marx        carla           22000013
```

Example of *sorta.dat*

Rename the second sorted data file *sortb.dat.*

```
ab11akvar     ali             23000001
ab11akbar     susan           24000002
ab11akbar     john            3000003
ab11smith     robert          35000004
ab11smith     barbara         35000005
ab11smith     stephen         12000006
ab13jonesp    ali             25000007
ab13jonesp    susan           24000008
ab13smithp    john            3000009
ab13smithp    robert          35000010
ab21barker    rodert          38000011
```

Example of *sortb.dat*

The input files must be sorted by the logical blocking criteria given in the second set of parameter file lines (see Section 7.2 below, under “Set 2”). If they are not, the program will stop with an error message indicating which record in which file is out of sort order.

7.2 Creating the Parameter File *parmc.txt*

Create the file *parmc.txt*, which contains the set of input parameters. This file allows the specification of additional parameters, beyond those included in *parmmf.txt*, which was described above in 6.3. Change your parameters as required.

An example of *parmc.txt* is shown below. The labels "Set 1," "Set 2," and "Set 3" are not part of the file but are used for reference. As for *parmmf.txt*, the maximum record size is 499 and the maximum size of a single field is 60 characters.

Set 1
Set 2
Set 3

```
1 3 120 120
blk                 1 4 1 4 0
last                5 20 5 20 0 u 0.95 0.25
first               25 10 25 10 0 u 0.80 0.20
age                 40 3 40 3 1 p 0.70 0.30
```

Example of *parmc.txt*

Set 1 consists of a single line containing four parameters:

- Number of blocking fields. (there is **1** blocking field in the example below, so the program will interpret only one line as containing information about the blocking variable)
- Number of matching variables. (there are **3** comparison fields in the example, so the program will interpret three lines as containing information about the matching variables)
- Length, in characters, of records in File A. (**120** characters long in File A in the example)
- Length, in characters, of records in File B. (**120** characters long in File B in the example)

Set 1

```
1 3 120 120
blk                 1 4 1 4 0
last                5 20 5 20 0 u  0.95 0.25
first               25 10 25 10 0 u  0.80 0.20
age                 40 3 40 3 1 p  0.70 0.30
```

Set 1 of *parmc.txt*

Set 2 consists of a single line for each blocking variable. The number of lines must correspondent to the number in the first parameter in Set 1. Each line contains the following parameters:

- Name of the blocking variable. This parameter must be exactly 20 characters long. If the name of the blocking variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the first comparison field label is **blk** in the example below)

- Starting position of the blocking variable in File A. ("blk" starts in position **1** of File A in the example)

- Length, in characters, of the blocking variable in File A. (**4** characters long in the example)

- Starting position of the blocking variable in File B. ("blk" starts in position **1** of File B in the example)

- Length, in characters, of the blocking variable in File B. (**4** characters long in the example)

- Null-data switch. This parameter is currently nonoperational; use 0 as a place-holder. (the null-data switch is set to **0** in the first line of Set 2 in the example)

Set 2

```
1 3 120 120
blk                  1 4 1 4 0
last                 5 20 5 20 0 u  0.95 0.25
first                25 10 25 10 0 u  0.80 0.20
age                  40 3 40 3 1 p  0.70 0.30
```

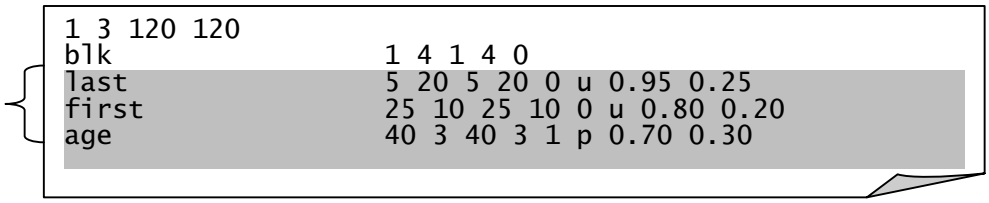
Set 2 of *parmc.txt*

Note that the blocking field may be part of the field occupied by a variable. For example, if the last name field began in position 1, the above parameters would specify blocking on the first four characters of the last name. The last names SMITh, SMIThers, SMIThson, and SMITs would all be in the same block; the last names ALBEniz, ALBErghetti, ALBErt, and ALBErtson would also be in the same block.

Set 3 consists of one line for each matching variable. The number of lines must correspond to the number in the second parameter in Set 1. Each line must include the following parameters:

- Name of the matching variable. This parameter must be exactly 20 characters long. If the name of the blocking variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the first comparison field label is **last** in the example below)
- Starting position of the matching variable in File A. (the field starts in position **5** in File A in the example)
- Length, in characters, of the matching variable in File A. (**20** characters long in File A in the example)
- Starting position of the matching variable in File B. (the field starts in position **5** in File B in the example)
- Length, in characters, of the matching variable in File B. (**20** characters long in File B in the example)
- Null-data switch. This parameter is currently nonoperational; use 0 as a place-holder. (the null-data switch is set to **0** in the first line of Set 2 in the example)
- Type of comparison (**u** is used in the first line of Set 3 in the example):
  - c exact
  - p prorated comparison by 15 percent for numeric
  - y calls a feature developed for a specific project; should not be used
  - uo ordinary string comparator/weighting
  - uf special string comparator/weighting for first name
  - ul special string comparator/weighting for last name
  - un special string comparator/weighting for numeric address component, modified ordinary string comparator/weighting
- An estimated probability #1. (0.95 in the first line of Set 3 in the example)
- An estimated probability #2. (0.25 in the first line of Set 3 in the example)

Set 3



```
1 3 120 120
blk                  1 4 1 4 0
last                 5 20 5 20 0 u 0.95 0.25
first                25 10 25 10 0 u 0.80 0.20
age                  40 3 40 3 1 p 0.70 0.30
```

Set 3 of *parmc.txt*

If the program fails to run properly, error messages will be in *chkc.dat.*

7.3 Running The Counter Program

Run C2: This is a counter program which requires parameter file *parmc.txt* and sorted data files. This will produce following files.

- *cntc.dat* This is a table of counts.
- *sumc.dat* This file contains summary statistics and/or error messages.
- *cntci.dat* This file can be used as input to a special three-class "em" parameter estimation program in subdirectory EM.

7.4 Creating Parameter File *parmu.txt*

Create *parmu.txt*, a parameter file that provides labels, locations, and length for each blocking and comparison variable the user intends to use.

An example of *parmu.txt* is shown below. The labels "Set 1," "Set 2," and "Set 3" are not part of the file but are used for reference. Each line of the parameter file must end with a blank.

Set 1
Set 2
Set 3

```
1 3 120 120
blk                 1 4 1 4
last                5 20 5 20 0
first               25 10 25 10 0
age                 40 3 40 3 1
```

Example of *parmu.txt*

Set 1 consists of a single line containing four parameters.

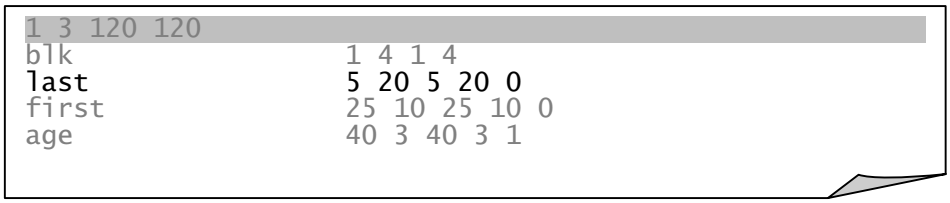
- Number of blocking fields. (there is**1** blocking field in the example below, so the program will interpret only one line as containing information about the blocking variable)

- Number of matching variables. (there are **3** comparison fields in the example, so the program will interpret three lines as containing information about the matching variables)

- Length, in characters, of records in File A. (**120** characters long in the example)

- Length, in characters, of records in File B (**120** characters long in the example)

Set 1



Set 1 of *parmu.txt*

Set 2 consists of a single line for each blocking variable. The number of lines must correspond to the number in the first parameter in Set 1. Each line contains the following parameters:

- Name of the blocking variable. This parameter must be exactly 20 characters long. If the name of the blocking variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the blocking field label is **blk** in the example below)

- Starting position of the blocking variable in File A. ("blk" starts in position **1** of File A in the example)

- Length, in characters, of the blocking variable in File A. (**4** characters long in the example)

- Starting position of the blocking variable in File B. ("blk" starts in position **1** of File B in the example)

- Length, in characters, of the blocking variable in File B. (**4** in the example)

Set 2

```
1 3 120 120
blk                  1 4 1 4
last                 5 20 5 20 0
first                25 10 25 10 0
age                  40 3 40 3 1
```

Set 2 of *parmu.txt*

Set 3 consists of one line for each matching variable. The number of lines must correspond to the number in the second parameter in Set 1. Each line must include the following parameters:

- Name of the matching variable. This parameter must be exactly 20 characters long. If the name of the blocking variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the first comparison field label is **last** in the example below)

- Starting position of the matching variable in File A. (the field starts in position **5** of File A in the example)

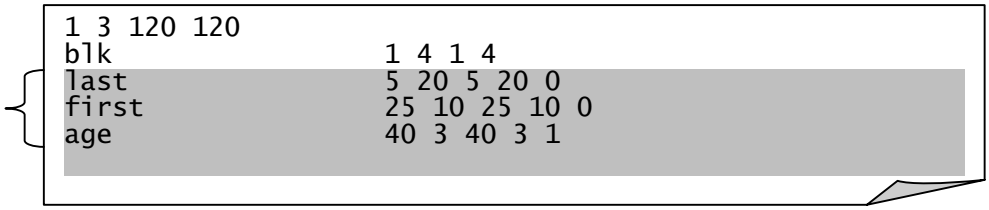
- Length, in characters, of the matching variable in File A. (**20** characters long in File A in the example)

- Starting position of the matching variable in File B. (the field starts at position **5** in File B in the example)

- Length, in characters, of the matching variable in File B. (**20** characters long in File B in the example)

- Null-data switch. This parameter is currently nonoperational; use 0 as a place-holder. (the null-data switch is set to **0** in the first line of Set 2 in the example)

Set 3



Set 3 of *parmu.txt*

If the program fails to run properly, error messages will be in *checku.dat*.

7.5 Running the U-Probability Parameter Generation Program

Run U1. This program provides an alternative method for obtaining u-probabilities that can be used in checking the u-probabilities produced by the EM program. The EM program produces both m- and u-probabilities. The u-weight program U1.C produces random-agreement probabilities that are often close to the u-probabilities from the EM program.

The u-probability is the probability that the content of a pair of fields agree, given that the records being compared do not match. Effectively, it is the probability that the content of the fields agree at random. This program will produce file *uprob.dat.*

7.6 Running the Expectation-Maximization (EM) Program

Create *parmem.txt*. Parameters include the number of fields (variables), an upper bound on the proportion of pairs estimated as matches, the epsilon used in fitting, and the maximum number of allowable iterations.

```
10  0.5000  0.00500000  0050
```

Example of *parmem.txt*

Run EM. This refers to two files generated by C2, *cntc.dat* and *parmem.txt.* This program generates more accurate probabilities. EM will produce two files:

- *emout1.dat*
- *em.trk*

APPENDIX: RUNNING A MATCH ON A SAMPLE FILE WITH TWO PASSES

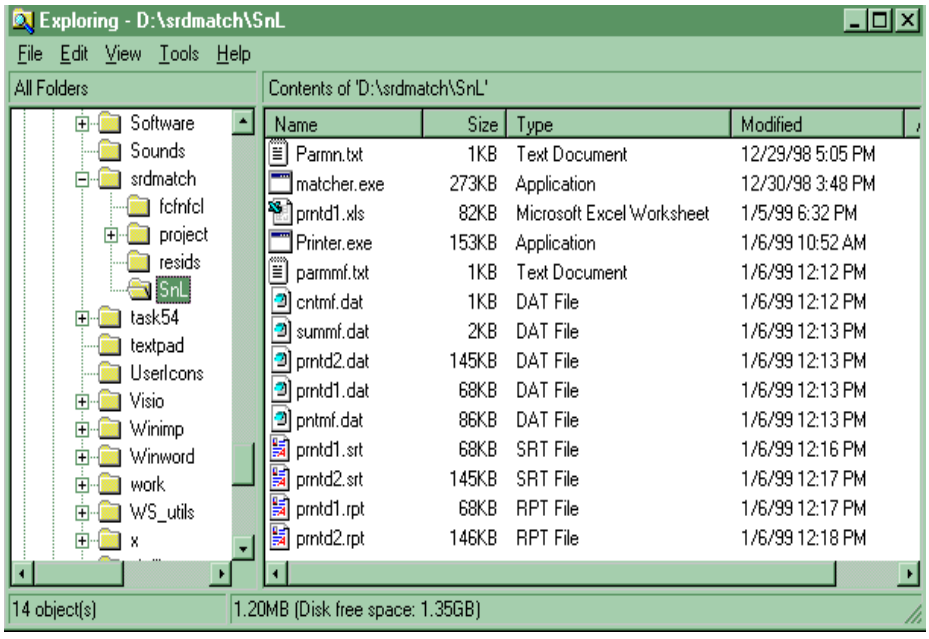
This appendix describes a sample matching job run against two sample files that SRD had compiled from publicly-available data about celebrities. Because the data were from public sources, there are no confidentiality issues in their use. The data consist of names and addresses only. One file contained 1,161 records; the other file contained 1,600 records. Names and addresses had already been standardized.

A.1 General Description of Sample Matching Job

The celebrity data were put through a two-pass match. In the first run of the matcher against the two files, 279 matched pairs and 17 clerical review pairs were found. Clerical review pairs are pairs with matching scores between the high and low cutoff values. When the residuals (919 records from the file of 1,161 and 1,357 records from the file of 1,600) were passed through a second match using new blocking and new cutoff values, it picked up 64 new matches and 44 clerical review pairs. This illustrates the need to block in different ways in order to pick up all the matches.

Three directories were used, two for the matching runs, and one for the residuals run. This made it easier to keep track of things, and avoided having to rename files from one run in order to get out of the way of the next run and preserve results, a problem if one tries to do all of this in one folder.

The three directories in this case were named SnL (for **S**tate a**n**d **L**ast initial), Resids, and fcfnfcl (for **f**irst **c**haracter **f**irst **n**ame and **f**irst **c**haracter **l**ast name), viz.:



A.2 Setting Up the Matching Run

The data files were in another directory used for standardization and had been sorted by state and first letter of last name. These are pointed to by *parmn.txt*, which contains:

```
D:\matchcel\stanoutB.srt
D:\matchcel\stanoutA.srt
```

If these files had been copied into the SnL directory instead, the "D:\matchcel\" part of the path could be dropped and the entire directory contents would be portable to any other compatible machine and directory without change.

For the first match run, the parameters in *parmmf.txt* were:

```
2 9 0 1 0 0 0 1 0 214 229
st                  93 2 103 2 0
fcl                 200 1 215 1 0
last                200 13 215 13 0 uo 0.9799 0.0430
first               174 12 189 12 0 uo 0.9840 0.0078
mid                 187 12 202 12 0 uo  0.49 0.04
hsnm                102 5 117 5 0 c  0.8438 0.2354
strt                112 20 127 20 0 c  0.6630 0.3332
unit                146 5 161 5 0 c  0.10 0.01
boxn                163 5 178 5 0 c  0.7037 0.1353
city                71 17 81 17 0 c  0.6 0.4
zip                 96 5 106 5 0 c 0.5 0.5
2.2 -0.1
35.0 -14.0
```

The cutoffs 2.2 and –0.1 were reasonable values, not optimized by experience with the data. If a file named *init.dat* had been present due to using the EM facility described in Section 7.0, the values in that file would have overridden the values from *parmmf.txt* seen here. In this sample run, no *init.dat* file was employed.

Note that the matching fields include "zip" with 0.5 specified for both the match and non-match probabilities. This gave "zip" no role whatsoever in the matching scoring, but forced it to appear in the output files, *prntd1.dat* and *prntd2.dat.*

A.3 Running the Matcher

Executing MATCHER.EXE produced the summary file, *summf.dat.* This was accomplished by the following two commands:

```
Matching files [D:\matchcel\stanoutB.srt] with [D:\matchcel\stanoutA.srt].
```

File *summf.dat* had the following content:

```
No init file
Blocking Field:         st
Blocking Field:        fcl

Matching Field:       last   agree:   3.1263   disagree:  -3.8631
Matching Field:      first   agree:   4.8375   disagree:  -4.1273
Matching Field:        mid   agree:   2.5055   disagree:  -0.6325
Matching Field:       hsnm   agree:   1.2766   disagree:  -1.5882
Matching Field:       strt   agree:   0.6880   disagree:  -0.6824
Matching Field:       unit   agree:   2.3026   disagree:  -0.0953
Matching Field:       boxn   agree:   1.6489   disagree:  -1.0710
Matching Field:       city   agree:   0.4055   disagree:  -0.4055
Matching Field:        zip   agree:   0.0000   disagree:   0.0000


MATCHING SUMMARY STATISTICS

        1161      records in the First file.
        1600      records in the Second file.
       63229      pairs of records compared.
         112      records skipped in the First File.
         127      records skipped in the Second File.
         279      matched pairs.
          17      clerical review pairs.
         753      unmatched records in the First file.
        1176      unmatched records in the Second file.
           0      blank records in the First file.
           0      blank records in the Second file.
        1161      records read in the First file.
        1600      records read in the Second file.

      LARGEST BLOCKS:

       First       Second
          87          171
          77          144
          65          136
```

A.4 Reviewing the Output and Choosing Cutoffs

Sorted the files *prntd1.dat* and produced *prntd1.srt*. File *printd1.srt* contained the score, the blocking value, and the specified matching field values from both celebrity files in each of its records. This rather long record is not appropriate for review by human eye. Therefore, we ran PRINTER.EXE to reformat the output to *prntd1.rpt* in a way suitable for printing and review. PRINTER.EXE replaces the tilde character (~) with a carriage return so that each record is displayed as three lines.

```
sort /r <prntd1.dat >prntd1.srt
printer <prntd1.srt >prntd1.rpt
```

We then printed *prntd1.rpt* and examined it. The top score, that is, the first two records on the file, was:

```
MOORE          MARY          TYLER          510                  86th    21A        NEW YORK            10028
MOORE          MARY          TYLER          510                  86th    21A        NEW YORK            10028
```

Our mission was to find the first occurrence that appeared to be a non-match, then find the score below which all records appeared to be non-matches. The first conceivable non-match was found in line 670 (record 224):

```
+0005.4433   CAM
   MARTINI          STEVEN                1964              WESTWOOD    400        LOS ANGELES         90025
   MARTIN           STEVE                                                    929 BEVERLY HILLS         90213
```

But many valid matches followed this case, so this did not appear credible as the upper cutoff score. Looking further, it appeared that non-matches began to appear frequently starting with this record pair at line 832:

```
+0002.9271   CAM
   MORGAN           HARRY                 13172             BOCA CANON          LOS ANGELES         90049
   MORGAN           HENRY                 13172             BOCA CANON          LOS ANGELES         90049
```

So our choice for the upper cutoff was 3.0.

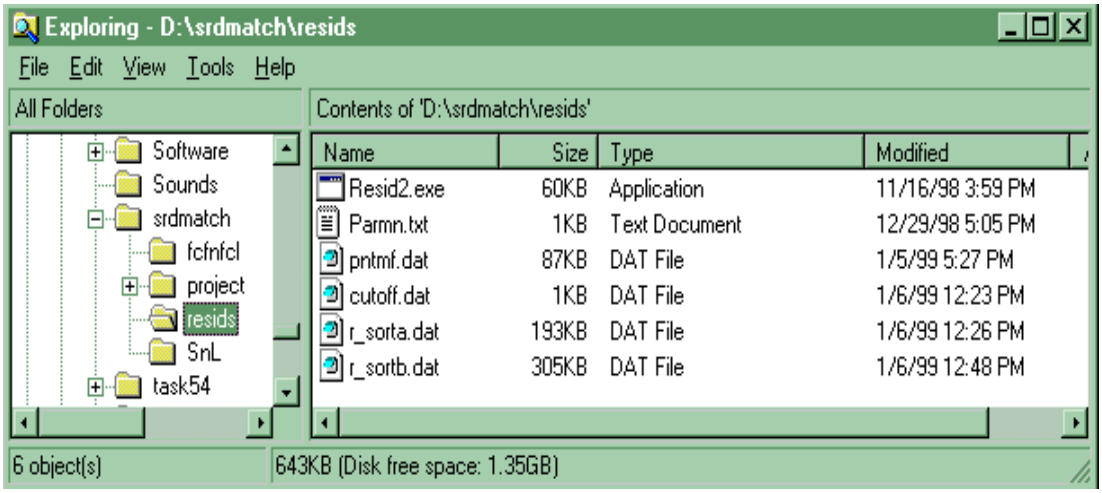
After the following records, the file contained nothing but what the user regarded as non-matches:

```
+0001.3690   CAB
   BRIDGES          DYLAN                  5525             JED SMITH           HIDDEN HILLS        91302
   BRIDGES          BEAU                   5525             JED SMITH           HIDDEN HILLS        91302
+0000.6747   CAS
   SADLER           WILLIAM                1114                 GRANT           VENICE              90291
   SCHALLERT        WILLIAM               14920                 RAMOS           PACIFIC PALISADES   90272
```

So our choice for the lower cutoff value was 1.0.

A.5 Running the Residuals

Now we were ready to run the residuals program in the Resids folder:



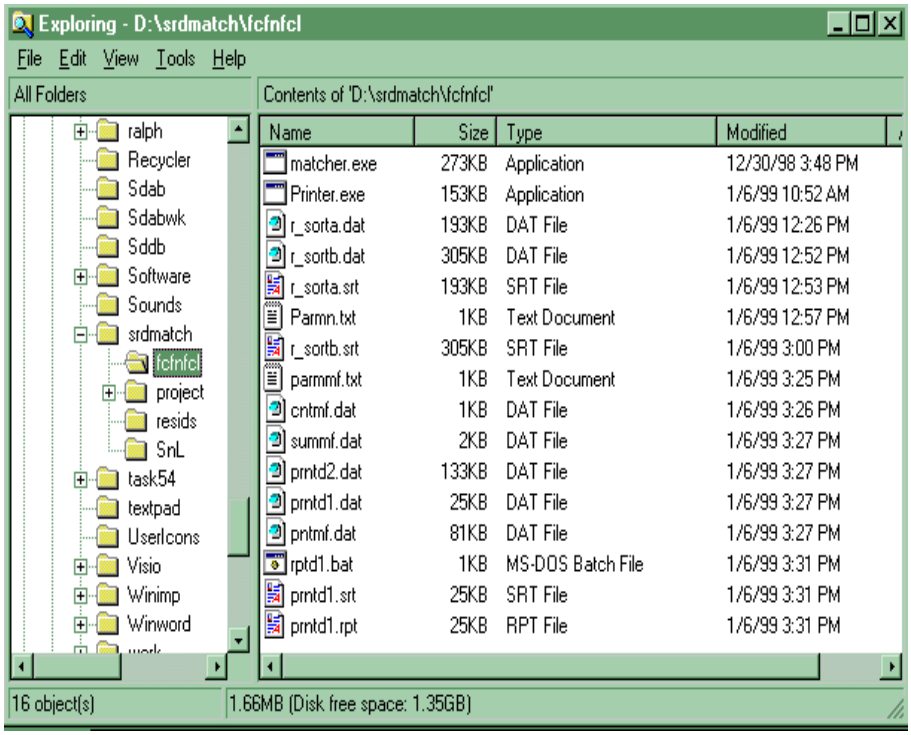
We created the file *cutoff.dat* as one line containing the cutoffs determined in the foregoing; that is:

```
3.0 1.0
```

Again, *parmn.txt* listed the input file names. The file *pntmf.dat*, produced by the matcher, was copied from the SnL folder. Then we ran RESID2.EXE to produce the smaller file (non-matches) to be re-blocked and re-matched. The outputs from RESID2.EXE were *r_sorta.dat* and *r_sortb.dat*.

A.6 Setting Up the Second Matching Run

First we copied *r_sorta.dat* and *r_sortb.dat* from the Resids folder to the fcfnfcl folder shown below. These files had to be sorted by first character of the first name with first character of last name. In File A these were in columns 174 (fcf) and 200 (fcl). In File B they were in 189 and 215. The sort produced *r_sorta.srt* and *r_sortb.srt.*



For the second match run, the parameters in *parmmf.txt* were:

```
2 10 0 1 0 0 0 1 0 214 229
fcf                 174 1 189 1 0
fcl                 200 1 215 1 0
last                200 13 215 13 0 uo 0.9799 0.0430
first               174 12 189 12 0 uo 0.9840 0.0078
mid                 187 12 202 12 0 uo  0.49 0.04
hsnm                102 5 117 5 0 c  0.8438 0.2354
strt                112 20 127 20 0 c  0.6630 0.3332
unit                146 5 161 5 0 c  0.10 0.01
boxn                163 5 178 5 0 c  0.7037 0.1353
city                71 17 81 17 0 c  0.6 0.4
state               93 2 103 2 0 c  0.5 0.5
zip                 96 5 106 5 0 c 0.5 0.5
2.2 -0.1
35.0 -14.0
```

A.7 Running the Matcher—Second Pass

Executing MATCHER.EXE produced the summary file, *summf.dat*, viz.:

```
Matching files [r_sorta.srt] with [r_sortb.srt].

No init file
Blocking Field:        fcf
Blocking Field:        fcl
Matching Field:       last   agree:   3.1263   disagree:  -3.8631
Matching Field:      first   agree:   4.8375   disagree:  -4.1273
Matching Field:        mid   agree:   2.5055   disagree:  -0.6325
Matching Field:       hsnm   agree:   1.2766   disagree:  -1.5882
Matching Field:       strt   agree:   0.6880   disagree:  -0.6824
Matching Field:       unit   agree:   2.3026   disagree:  -0.0953
Matching Field:       boxn   agree:   1.6489   disagree:  -1.0710
Matching Field:       city   agree:   0.4055   disagree:  -0.4055
Matching Field:      state   agree:   0.0000   disagree:   0.0000
Matching Field:        zip   agree:   0.0000   disagree:   0.0000


MATCHING SUMMARY STATISTICS

   919      records in the First file.
  1357      records in the Second file.
  5439      pairs of records compared.
    93      records skipped in the First File.
   188      records skipped in the Second File.
    64      matched pairs.
    44      clerical review pairs.
   717      unmatched records in the First file.
  1060      unmatched records in the Second file.
     0      blank records in the First file.
     0      blank records in the Second file.
   919      records read in the First file.
  1357      records read in the Second file.

     LARGEST BLOCKS:

      First       Second
          15           28
          13           26
          12           26
```

Notice in the summary that this second pass, looking only at the "residuals" which are non-matches in the first pass, picked up 64 new matches and 44 clerical review pairs.

As before, we sorted the *prntd1.dat* file by descending scores, and reformatted it using the PRINTER.EXE program, to produce *prntd1.rpt*.

This started off with the following cases:

```
+0012.8394   JP
   PFLUG          JO          ANN          1270     PEACHTREE BATTLE              ATLANTA          GA 30327
   PFLUG          JO          ANN          1270     PEACHTREE BATTLE              ATLANTA          GA 30327
```

And contained 61 matches down to:

```
+0004.4022   JA
   ASHTON         JOHN                      700          HINSDALE                  FORT COLLINS     CO 80526
   ASTIN          JOHN                                                         49698 LOS ANGELES      CA 90049
+0003.1472   TM
   MARVIN         TONY                     2862          YORKSHIRE                 JENSEN BEACH     FL 34957
   MARTIN         TONY                    10724          WILSHIRE  1406            LOS ANGELES      CA 90024
+0002.9271   HM
   MORGAN         HARRY                   13172          BOCA CANON                LOS ANGELES      CA 90049
   MORGAN         HENRY                   13172          BOCA CANON                LOS ANGELES      CA 90049
```

Here again probable matches become intermittent. Marvin looks good but the other two are questionable. Still it is possible in reviewing lower scores to see another match or two, for example:

```
+0001.3690   DS
   SWEENEY        DB                      25144          MALIBU                    MALIBU           CA 90265
   SWEENEY        D           B           25144          MALIBU                    MALIBU           CA 90265
```

This concluded the sample run.