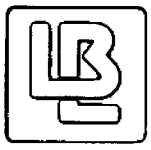
Title
CODATA TOOLS: PORTABLE SOFTWARE FOR SELF-DESCRIBING DATA FILES

Authors
Merrill, D.
McCarthy, J.L.

Publication Date
1982-12-01



Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

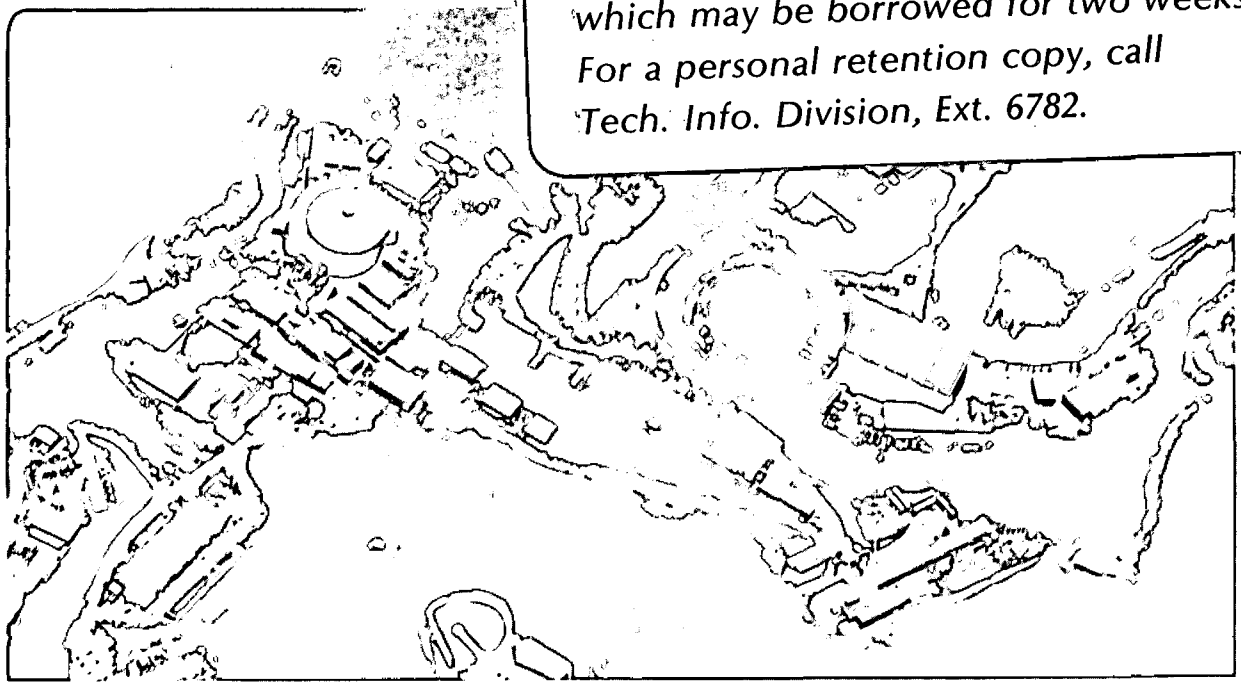
Computing Division

Presented at Computer Science and Statistics:
15th Symposium on the Interface, Houston, TX,
March 16-19, 1983; and published in the
Proceedings

CODATA TOOLS: PORTABLE SOFTWARE FOR
SELF-DESCRIBING DATA FILES

D. Merrill and J.L. McCarthy

December 1982



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

LBL-15441

CODATA TOOLS: PORTABLE SOFTWARE FOR SELF-DESCRIBING DATA FILES*

Deane Merrill and John L. McCarthy

Computer Science Department
University of California
Lawrence Berkeley Laboratory
Berkeley, California 94720

December 1982

Codata Tools: Portable Software for Self-Describing Data Files*

Deane Merrill and John L. McCarthy

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

This paper describes the Codata tools, a set of programs which read, write, and restructure self-describing Codata (common data format) files. These tools manipulate both data and data description, so that the the output of any operation is itself a Codata file. Semantics of results and descriptions of derived Codata files are inherited from descriptions of input Codata files. Following the Software Tools philosophy, the Codata tools are modular - each tool performs a specific limited task. They follow the UNIX and Software Tools conventions of standard input and output. The output of any module can automatically serve as the input of another, and they can be "pipelined" or "chained" together. Codata tools can be used to extract specified rows and/or columns from a file, to sort a file, to perform relational joins, to perform tabulations by aggregating on common key values, and to perform other operations. The Codata tools are written in RATFOR (a transportable FORTRAN preprocessor), and can be easily adapted to run on any computer where the Software Tools have been implemented. Work is currently under way on substantial enhancements to the Codata file format and the Codata tools to provide for more efficient physical storage formats, more complex data structures, and more extensive, open-ended data description.

1. Introduction

In this paper we discuss a simple self-describing file format and associated software developed and used extensively in the Lawrence Berkeley Laboratory (LBL) Computer Science and Mathematics (CSAM) Department since early 1978. The file format is known as "Codata" (common data format) and the associated software tools are known as the "Codata Tools" [MERR81]. These developments have been closely associated with that of SEEDIS (LBL's Socio-Economic Environmental Demographic Information System), but the potential applications are much more general [COMP82, MCCA82B]. The Codata format and tools are sufficiently simple and useful that they have been used extensively outside SEEDIS. One purpose of this paper is to make them more generally known and available to other researchers.

Another purpose of this paper is to point out the need for and importance of tools for statistical data management and analysis that both *use* and *produce* self-describing data files. Statistical package programs pioneered the use of self-describing data files over the past two decades. But with the notable recent exception of System S [BECK78],

they have been deficient in several important respects:

- most of them use *internal* self-describing files, but do not *produce* such files as a routine part of any given data analysis or manipulation routine (hence it is often difficult, if not impossible to make the output of one routine easily available as input for another)
- those that do produce "savefiles" of some kind do not do so in terms of a neutral, non-procedural interchange format, but rather in terms of their own idiosyncratic procedural language.
- they do not provide facilities to easily expand or operate on the meta-data or data description portions of self-describing files

The remainder of this introductory section describes the SEEDIS Project and motivations that led to development of the Codata File Format and Codata Tools. Section 2 explains the general requirements, design principles, and implementation strategies that underlie the Codata File Format and the Codata Tools. Section 3 describes the Codata Format in some detail, while Section 4 describes the Codata Tools and summarizes the specific operations which they perform. Section 5 describes new enhancements to the Codata File Format and Codata Tools which are currently being designed and implemented. Section 6 is a concluding summary.

1.1. Background: The SEEDIS Project

In the early 1970's, LBL contracted with the Department of Labor, the Bureau of the Census, and the National Technical Information Service to produce the Urban Atlas Metropolitan Map series and a number of Manpower Planning reports for use in the Department of Labor Employment and Training Administration (then the Manpower Administration). Important resources resulting from this effort were databases from the 1970 Census of Population, geographic base map files of the 35,000 census tracts defined by the 1970 Census, and associated software for managing and displaying these data. Under supplementary funding from the Army Corps of Engineers, other socioeconomic databases such as the City-County data book and the Census of Agriculture were acquired and made available for automatic retrieval by remote users.

In 1975 the name SEEDIS (Socio-economic Environmental Demographic Information System) was invented to describe what gradually evolved into an integrated system combining information retrieval, data analysis, and display capabilities in a user-friendly environment. A prototype system was implemented on LBL's CDC computer system. Beginning in 1976, the PAREP (Populations at Risk to Environmental Pollution) project began adding environmental and health-related data bases in order to analyze the relationships between human health and air pollution [MERR82]. But further data development was hampered by the lack of adequate data management tools. Even relatively simple data manipulation tasks required the writing of special-purpose programs.

* This work was supported by the Office of Health and Environmental Research and the Office of Basic Energy Sciences of the U.S. Department of Energy under Contract DE-AC03-76SF00098; and the Department of Labor, Employment and Training Administration under Interagency Agreement No. 06-2063-36.

1.2. Statistical Data Management Problems

By the late 1970's it was evident that further development of SEEDIS required more systematic data management. Motivating factors included data documentation requirements, the need for different applications programs to share data without being constrained to a single physical file format, and the need to carry out simple data manipulation operations without writing special-purpose programs.

Data documentation was a primary concern. Important data files were being developed whose usefulness would outlive the projects for which they were originally acquired. "Codebooks" were frequently inadequate and idiosyncratic. People who had originally worked with given data files usually forgot important undocumented details once they stopped working with a file on a regular basis. Since documentation was almost always separate from the data, it was easily misplaced or destroyed.

Second, with continuing addition of new data files, applications programs needed to be independent of minor changes in physical file formats. Without such data independence, adding a new data field or changing the length or position of a data field might require every program using that file to be modified.

Third, a number of existing data analysis and display programs required a standard communications file format in order to communicate with one another simply and effectively. For example, CHART, used to make bar charts, line graphs, etc. and CARTE, a thematic mapping program, required different forms of input files. Even the simplest task, such as making a bar graph and a choropleth map from the same data, could not be performed without writing a program.

The situation clearly called for data standards and data management tools, but there were few statistical packages, let alone data management packages, that could run on LBL's CDC computers and the locally developed BKY operating system. Most of the available statistical and database packages lacked sufficient capacity for databases larger than a few megabytes. Statistical package programs did not provide an efficient means for local applications to communicate with one another. Available data management software was slow, complex, and expensive -- particularly for interactive applications -- and it lacked adequate facilities for data description and documentation.

2. Statistical Data Management Requirements

The problems described above did not justify acquisition or development of a full-featured database management system. Data files were archival, with very infrequent updates. Updates did not have to be made by multiple users simultaneously. Security requirements were minimal.

On the other hand, it was clear that the system had to be relatively flexible to permit continuing addition of new files. It also had to be able to handle very large numbers of records (cases), attributes (variables), and databases for different types of entities (primarily geographic areas such as states, counties, census tracts, etc.).

2.1. Design Principles for Codata Files

The solution that SEEDIS project staff adopted was to design and implement a simple, standard, self-describing file format plus a set of tools to perform basic operations on such files. Both the file format and associated tools would be as modular as possible, in order to provide flexibility and to permit further gradual and evolutionary change.

Based on experience with statistical packages, database management software, and applications programs for graphical display, project staff decided that the standard SEEDIS file format should be self-describing. That is, data files would contain essential descriptive information such as data types, missing data conventions, labels, etc. A standard library of input-output routines would be implemented to read and write such files, and applications could use such routines to access both data and data descriptions for a given file. Some additional standards were agreed upon as part of the implementation strategy, as follows:

- data and description would be stored as ASCII text in lines no longer than 132 characters, in order to permit easy inspection and modification of files with standard text-editing tools;
- descriptive items would be stored in "{keyword} = {value}" format, for ease of reading and understandability;
- data would be stored in fixed-format fields to permit easy access via conventional fixed-format read and write routines as well as via special Codata input-output subroutines.

2.2. Codata Tools Development

Beginning in 1979, the Codata Tools were originally designed to satisfy the data management requirements of the PAREP project, including sorting, row and/or column selection, and relational joins. They proved useful enough that the user interface was substantially enhanced, in imitation of the LBL Software Tools and the UNIX operating system. UNIX input-output conventions were adopted, and documentation was brought into conformity with UNIX/Software Tools formats.

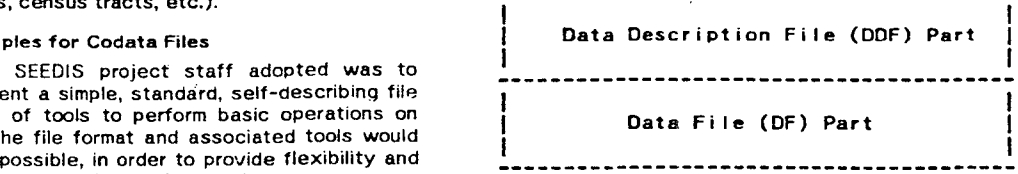
At the same time, SEEDIS was being adapted to a network environment of VAX 11/780 computers running the VMS operating system. Applications modules to perform such operations as the reading and writing of computer-independent compressed data records, information retrieval (by geographic area), user-prompted data entry, on-line browsing of data dictionaries, extraction of selected variables from different databases, data transformation, table construction, and thematic mapping were linked via a menu-driven command monitor and the underlying Codata interchange file format.

Experience with the new integrated system suggested that SEEDIS itself required a rather general set of transformation routines for dealing with Codata files. The Codata Tools were further developed and incorporated into SEEDIS at points where data transformations were required, for example in combining data being extracted from more than one data source. Between 1979 and 1982, The standard Codata file format and input-output routines to read and write such files became the nucleus of a new and substantially more integrated SEEDIS system.

3. The Codata format

Codata files are fixed-format ASCII text files containing two logical parts: the data description file (DDF), and the data file (DF), as pictured schematically in Exhibit 1.

Exhibit 1: Schematic Diagram of CODATA File



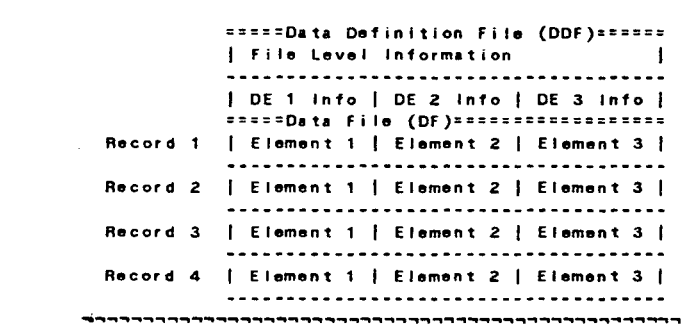
In Codata files, both the data definition file (DDF) and the data file (DF) reside in a single physical data file, with information stored in character representation within fixed-length logical records as defined in the DDF. The DDF includes file-level information, such as the number of data records (rows),

and the number of data elements (columns). It also contains information about each data element, including a short name, a descriptive label, data type (integer, floating point, or alphanumeric), field length, location in the record and other information. The data file (DF) portion of a Codata file is a simple "flat" file with fixed-length data elements (fields) in fixed-length records.

The primary virtue of the Codata format is its simplicity. Since both data and description are simple ASCII text, both can be read, written, and modified using either a text editor or formatted read and write statements from a programming language. Codata files are easy to read and understand, to transport between dissimilar computers, and to convert to other file formats. They can be easily printed, edited, or read by a user-written program. Most simple formatted data files can be converted to the Codata format simply by prepending a hand-edited DDF (data description file). Conversely, Codata tools are provided which will strip off the DDF and produce a FORTRAN format statement, for example to read a Codata file into a program like MINITAB or SPSS.

The logical view of a Codata file is that of a table (or flat file) with a fixed number of rows (records) and columns (fields), as pictured in exhibit 2.

Exhibit 2: Codata File Logical Structure



Data are arranged so that each logical row of the table contains all the attributes (data elements, columns, fields, or variables) of a named entity (e.g., Alameda County, person number 2037). Data elements include keys necessary for data access and matching, a row label or "stub", and ordinary numeric or alphabetic data values. The number of logical records (rows) is equal to the number of entities in the data file, and the number of columns is equal to the number of data elements in each record.

3.1. A Simple Codata File Example

Exhibit 3 shows a small example file in Codata format. The actual data are contained in the last six lines. The remainder of the file is meta-data, i.e. descriptive information about the data that can be used by programs and people. Explanatory comments in italics at the right are not part of the Codata file but simply for purposes of this illustration.

Exhibit 3: Example Codata File

```
NDE = 6                          file level information
AREAS = 3
CARDLENGTH = 30
MISSING = -999 -991
DE = FIPS.STATE                  1st data element information
  TYPE = A
  USE = K
  START = 1
  LENGTH = 2
  HEADER =#FIPS state code#
DE = STUB.GEO                    2nd data element information
  TYPE = A
  USE = S
  START = 3
  LENGTH = 5
  HEADER =#state name#
DE = POP80
  TYPE = I
  USE = D
  START = 8
  LENGTH = 9
  HEADER =#1980 population#
DE = POP70
  TYPE = I
  USE = D
  START = 17
  LENGTH = 9
  HEADER =#1970 population#
DE = POP60
  TYPE = I
  USE = D
  START = 31
  LENGTH = 9
  HEADER =#1960 population#
DE = LAND.AREA                   6th data element information
  TYPE = I
  USE = D
  START = 40
  LENGTH = 8
  HEADER =#land area#
  HEADER =#in square miles#
END DDF                          beginning of data (DF) part
19Iowa    2913808   2825368
  2757537    55941
48Texas  14229191  11198655
  9579677   262134
49Utah    1461037   1059273
      -999    82096
```

The first four lines constitute the global section of the DDF, and they apply to the file as a whole. The first specifies that the number of data elements (NDE) or columns in the file is six. The second specifies that the file contains three records (AREAS). CARDLENGTH specifies that physical records (lines) in the DDF and DF each contain a maximum of 30 characters of data. MISSING specifies that any numeric data value between -991 and -999 should be considered missing.

The second section of the DDF, through the "END DDF" line, contains a set of meta-data information for each data element (column) in the data file. DE is a unique identifier or name for each data element. TYPE specifies whether the data element values are alphanumeric (A), integer (I), or floating point (D). USE distinguishes between keys (K), row labels or "stubs" (S), and ordinary data elements (D). START gives the byte position for the data element (field) within each logical record. LENGTH specifies the number of bytes allocated to the data element. Each element can have one or

more HEADER lines containing descriptive information, for the use of application programs such as the SEEDIS thematic mapping program.

In the example file of Exhibit 3, the six data elements (columns) are named FIPS.STATE, STUB.GEO, ... through LAND.AREA. FIPS.STATE and STUB.GEO are alphanumeric (type A); the others are integer (type I). FIPS.STATE is a key (use K); STUB.GEO is a row label or stub (use S); the others are ordinary data (use D). The column labels (headers) are indicated with each data element. FIPS.STATE starts in column 1 and occupies two characters. STUB.GEO starts in column 3 and occupies five characters. POP60 starts in column 31, which means the first position of the second line of each record (since the line length is 30 characters).

3.2. Data Definition Details

As shown in exhibit 3, the data description file (DDF) or meta-data part of a Codata file contains textual specifications in "{keyword} = {value}" format. This data description is stored in line-image form. No meta-data field can exceed the line length specified by CARDLENGTH, which is limited to a maximum of 132 characters. Blanks and upper-lower case distinctions are significant only within the text of HEADER lines. Keywords occurring before the first data element definition have global effect. That is, they hold for all data elements, unless specifically overridden by keyword definitions within the local environment of a data element definition.

3.3. Data File Details

The DF is the set of fixed length records following the "END DDF" line. As in the DDF, data fields in the DF cannot exceed the specified line length, and can never exceed 132 characters. Upper-lower case distinctions and blanks in alphanumeric fields of the DF *are* significant, unlike those of the DDF. The Codata Tools expect key elements (USE=KEY) to be alphanumeric (TYPE=A), without blanks. Missing data values (for example data suppressed for reasons of confidentiality) can be indicated by values defined in the MISSING line or by leaving the field blank (blanks are not interpreted as zeros on input, as in FORTRAN). Codata write routines automatically right-justify numeric data fields in the DF, preserving as many significant figures of information as possible. Exponential notation (e.g. 1.3E5) is not allowed.

A more complete description of the Codata format is included in [MERR 81].

3.4. DDF's for Other Data Files

The DDF syntax just described is a subset of a more general data definition language (DDL) being developed in connection with the SEEDIS project. In addition to the fixed-length, line-image, eye-readable codata files described here, the same DDL is used to describe variable-length, binary, compressed data files used for space-efficient storage of archival data. In both cases, a data set consists of two logical components -- a data definition file (DDF) and a data file (DF). Translation programs convert SEEDIS compressed files to Codata files and vice-versa. A discussion of the SEEDIS compressed data format is in preparation [GEY 83].

4. The Codata Tools

As stated in sections 1 and 2 above, development of the Codata format was primarily motivated by needs for data independence among SEEDIS applications programs. Development of the Codata Tools followed shortly thereafter due to the need for general data manipulation routines to deal with Codata files. The "tools" approach to data manipulation for SEEDIS was inspired and aided by the modular Software Tools implementation efforts that were already underway in connection with other projects at LBL,

4.1. Design Philosophy

The Codata Tools were designed and written in accordance with principles outlined by Kernighan and Plauger in *The Elements of Programming Style* [KERN74], and *Software Tools* [KERN76]. In addition, the Codata Tools utilize conventions and extensions to the original Software Tools that have been implemented at Lawrence Berkeley Laboratory [HALL81, SVEN82]. In particular, the following principles were observed:

1. Most of the Codata Tools have one input, one output, and perform a transformation on the data passing through. Such programs are known as "filters." The output from one program can be directly "pipelined" into another. For example, the command line

cocol <file1 countyname income | cosort income | lpr

extracts the variables "countyname" and "income" from file1, sorts the records in order of income, and prints the resulting file.

2. Each Codata Tool does a specific job. In the preceding example, variable extraction is performed by COCOL while sorting is performed by COSORT. The principle of modularity has been observed down through low-level subroutine calls.

3. On-line documentation in a standardized format is obtained by typing "coman <toolname>". Invoking a Codata Tool with a question mark (e.g. "cocol ?") causes a one-line reminder of the expected command syntax to be printed.

4. The Codata Tools are written in a simple extension of Fortran known as "Ratfor" (Rational Fortran). Fortran is widely available and well supported, but is a poor language for programming or describing programs. We avoid the major idiosyncrasies of Fortran, and hide the unavoidable ones in well- defined modules.

5. Input and output and low-level functions such as string manipulation are performed via standard Software Tools library subroutines.

As a result of points 4 and 5 above, the Codata Tools are easily transported to different computers and operating systems where the Software Tools primitive functions have been implemented.

4.2. Functions performed by the Codata Tools

Here we describe each of the Codata Tools individually, in order of increasing complexity. In this discussion, "{}" means "value of". For example "{file}" is an arbitrary file name, not literally "file". Arguments in square brackets "[]" are optional.

Following UNIX conventions, "<in" and ">out" indicate standard input and standard output, respectively. Including "<{file1}" in the command line causes input to be read from file {file1}; omitting it causes input to be read from the terminal. Including ">{file2}" in the command line causes output to be written to file {file2}; omitting it causes output to be written to the terminal.

4.2.1. Reformatting Functions

The first three tools perform simple reformatting functions, changing neither data values or their order in the file:

- **COCAT: check and reformat.**

cocat <in >out [{file}]

COCAT copies a Codata file to standard output. In general the output is not strictly identical to the input, as COCAT manipulates all data fields internally. Thus COCAT may be

used to check the validity of the input file format. If {file} is not specified, COCAT reads standard input.

• **COCL: change line length.**

cocl <in >out [{linelength}]

COCL changes the line length of a Codata file, reading from standard input and writing to standard output. Unused space between data fields is removed, and all data elements are repositioned. If {linelength} is unspecified, the output line length is equal to the line length of the input file.

• **CODDF: modify DDF attributes.**

coddf <in >out {modfile}

CODDF reads a Codata file from standard input and writes a new Codata file to standard output. Changes to the DDF are specified in {modfile}. The DF portion of the output Codata file corresponds to the revised DDF.

Columns can be renamed, modified, or added; but not deleted or reordered. New columns are added at the end; the corresponding data values are written as blank if type=A, or as a missing data value if type=I or type=D. For further details, consult [MERR81].

4.2.2. Interfaces to External Applications

The next three tools can be used to provide an interface between Codata files and external applications:

• **COSPLIT: partition into DF and DDF.**

cosplit <in >out [{areafile} [{ddffile}]]

COSPLIT reads a Codata file from standard input and writes the DF portion to standard output. One or two file names may be specified as optional arguments. The first file, if specified, receives a copy of the "AREAS={n}" line from the DDF. The second file, if specified, receives a copy of the remainder of the Codata DDF, up through the "END DDF" line.

• **DDFFMT: make FORTRAN format from DDF.**

ddffmt <in >out

DDFFMT reads a Codata file, or the DDF portion of a Codata file, from standard input. A corresponding Fortran format specification is written to standard output.

Type A, I, and D fields are respectively translated to "An", "Fn.0", and "Fn.0", where n is an integer. The output format specification includes surrounding parentheses "()". A new line is begun after every 60 characters.

• **CONAME: list data element names.**

coname <in >out [-h][{c}] [{file}]

CONAME reads a Codata file and writes the names of the columns (data elements) to standard output. If {file} is not specified, CONAME reads from standard input.

If the "-h" flag is present, data element headers are also written. If the "-h" flag is followed by a character "{c}", header break characters (normally "#") are changed to "{c}" on output.

4.2.3. Data Manipulation Functions

The five remaining tools perform data manipulation functions:

• **COSORT: sort rows.**

cosort <In >out [-d] [{colname} ...]

COSORT sorts the rows of a Codata file on standard input, and writes the resulting Codata file to standard output. The command line optionally specifies the names of columns whose values comprise the sort key. If no column names are specified, the sort key is the concatenation of all columns having use=K.

Rows are put out in ascending order of the sort key unless "-d" (decreasing order) is specified. In alphabetic (type=A) data elements, blanks are significant. Upper/lower case differences are ignored.

• **COCOL: select columns.**

cocol <in >out {dename}
...or cocol <in >out -{colfile}

COCOL reads a Codata file from standard input and writes a Codata file to standard output. Only specified columns are copied to the output file. The command line specifies either (a) a list of column names {dename} ..., or (b) the name of a file {colfile} containing the list of column names. In mode (b), a minus sign precedes the file name. Upper/lower case differences are ignored. In the output file columns appear in the order requested.

• **COROW: select rows.**

corow <in >out {keyvalue}
... or corow <in >out -{keyfile}

COROW reads a Codata file from standard input and writes a Codata file to standard output. Only specified rows are copied to the standard output. The command line specifies either (a) a list of key values {keyvalue} ..., or (b) the name of a file {keyfile} containing the list of key values. In mode (b), a minus sign precedes the file name.

In both mode (a) and (b), the key value of a row is the concatenation of all data elements having use=K and type=A.

• **COMRG: relational join of two files.**

comrg <in >out {file2} [{colname} ...]
... or comrg <in >out {file2} [-{colfile}]

COMRG performs a relational outer join of two Codata files which are on standard input and {file2} respectively. A resulting Codata file is written to standard output.

The rows in standard output correspond to those in standard input; row matching is performed for the keys defined in {file2}. The {file2} rows must be uniquely identified by columns having use=K and type=A. Standard input must contain corresponding columns.

Both standard input and {file2} must have been previously sorted (for example by COSORT) in ascending order of the values of these elements. (These elements need not have use=K in standard input, nor do they need to uniquely specify the rows of standard input.)

Each row of standard output corresponds to one row of standard input. All elements of standard input are copied unchanged. In addition, standard output contains additional columns corresponding to the non-key columns of {file2}. The unique keys occurring in {file2} determine which row of {file2} contributes to a row of standard output. A row of {file2} need not be used, or can be used more than once.

Optionally, the command line may specify either (a) a list of {file2} column names ({colfile} ...) or (b) the name of a file {colfile} containing a list of {file2} column names. In mode (b), a minus sign precedes the file name. If either option (a) or (b) is used, only the specified {file2} columns are written to standard output.

Other COMRG options are available to control the handling of missing data and the renaming of duplicate column labels. For further details, consult [MERR81].

• **COROAGG: aggregation on specified keys.**

coroagg <in >out [{col} ...] [-c{ctcol}] [-z]

COROAGG reads a Codata file from standard input and writes a Codata file to standard output. The output file contains the total of all the rows of the input file or (optionally) a number of subtotals.

If one or more {col} arguments are specified, these must correspond to type=A column names in standard input. In standard output, the specified {col} elements, and no others, have use=KEY.

Each row in standard output corresponds to a unique value of the specified {col} element(s). If no {col} arguments are specified, standard output contains a single row, with no use=KEY columns.

Standard input need not be sorted. Standard output is sorted in ascending order of the specified {col} elements.

If present, the optional argument {ctcol} preceded by "-c" is the name of a new column in standard output , which contains the number of input rows that contributed to the output row in question.

Numerical fields (type=I or type=D) are added. If an input data value is missing, a warning message is issued. Any missing data on input will produce a missing value in the corresponding sum, unless the "-z" option is specified. In this case missing input data values are assumed to be zero. Character (type=A) fields are copied to standard output if all contributing rows of the input file have the same value. Otherwise the corresponding output field is filled with blanks.

Other COROAGG options are available to provide weighting of each row by a specified variable, and to fill type=A fields with a character other than blanks. For further details, consult [MERR81].

4.2.4. Other Codata Tools

Other Codata tools, of a more experimental nature, are available. For complete documentation, consult [MERR81].

A notable omission is the lack of a Codata tool to calculate new data values (columns) as functions of other data values. A tool COCOCAL written to fill this need did not perform successfully. Though not following the documentation and input-output conventions of the other Codata tools, the QUERY module reads and writes Codata files and provides the required functionality in SEEDIS. QUERY uses the LANGPAC compiler generator to evaluate general algebraic and logical expressions.

4.3. Composite Example

One of the most powerful features of the Codata Tools is the idea, taken from the UNIX operating system, that output from one program can serve directly as input to another. The use of several Codata tools connected by "pipes" is illustrated in the following example. From the file listed in Exhibit 3, it is desired to list the three states in ascending order of 1980 population.

Example:

cocol <file2 stub.geo pop80 | cosort pop80 | cosplit >file20

Standard input (file2) is the same as shown in Exhibit 3.

Standard output (file20) contains:

```
Utah    1461037
Iowa    2913808
Texas  14229191
```

5. Limitations and Future Directions

As noted earlier, the primary virtue of the Codata format is its simplicity. Since 1978 it has served as the primary SEEDIS inter-module data interchange format and as a means for inexperienced users to get data in and out of SEEDIS. The Codata tools, with their extremely simple user interface, have been crucial in the development of major data files for many applications.

On the other hand, the present Codata Format and Codata Tools have inherent limitations which are being addressed in order to aid the future development of SEEDIS. Dynamic stack allocation will replace fixed-size arrays in order to efficiently handle large data files without repeated recompilation. New binary file formats are being developed in order to reduce the input-output and processing time required by the Codata tools. Input-output routines are being developed to read and write these new file formats; these will be used first in file translation utilities, and later linked directly to the major SEEDIS data manipulation and display modules. The data definition language (DDL) is being enhanced in order to provide economical description of large multi-dimensional data arrays and multiply-occurring and/or variable-length data elements [MCCA82A].

6. Summary

The Codata tools, developed in connection with the SEEDIS project at Lawrence Berkeley Laboratory, are a set of programs which read, write, and restructure self-describing Codata (common data format) files. These tools manipulate both data and data description, so that the the output of any operation is itself a Codata file. The Codata tools can be used to extract specified rows and/or columns from a file, to sort a file, to perform relational joins, to perform tabulations by aggregating on common key values, and to perform other operations.

Codata files have two basic parts: data and data description. The data portion is a simple "flat" file with fixed-length data elements (fields) in fixed-length records. The data description portion consists of line images in "{name} = {value}" format. Codata files can be read, written, and modified using either a text editor or formatted READ and WRITE statements from a programming language. They are easy to read and understand, to transport between dissimilar computers, and to convert to other file formats.

Following the Software Tools philosophy, the Codata tools are modular - each tool performs a specific limited task. They follow the UNIX and Software Tools conventions of standard input and output, so that the output of one module can automatically serve as the input of another. They are written in RATFOR (a transportable FORTRAN preprocessor), and can be easily adapted to run on any computer where the Software Tools have been implemented.

Work is currently under way on substantial enhancements to the Codata file format and the Codata tools to provide for more complex data structures, more extensive, open-ended data description, and more efficient operation.

7. Acknowledgments

Carl Quong, head of the LBL Computer Science and Mathematics Department, is responsible for the stable and productive research environment in which this work was conducted. Among many others who made these results possible,

the authors wish to thank especially Fred Gey and Harvard Holmes for sustained interest and valuable suggestions. Bill Hogan's QUERY module fulfilled a need not met by the other Codata tools. Linda Wong wrote the Codata tool DDFFMT. Bill Benson, Peter Wood, and Bob Healey wrote the low-level read and write routines used by all the Codata tools and SEEDIS applications modules.

This work was supported by the Office of Health and Environmental Research and the Office of Basic Energy Sciences of the U.S. Department of Energy under Contract DE-AC03-763F00098; and the Department of Labor, Employment and Training Administration under Interagency Agreement No. 06-2063-36.

8. References

BECK78 Becker, R.A., and J.M. Chambers. Design and Implementation of the S System for Interactive Data Analysis, *Proc. I.E.E.E. Compsac78, (1978)*, pp. 626-629.

COMP82 Computer Science and Mathematics Department, SEEDIS Release Notes version 1.3, February, 1982; version 1.4 (preliminary), March, 1983.

GEY83 Gey, F., McCarthy, J.L. and Merrill, D.; Computer-Independent Data Compression: A Space-Efficient, Cost-Effective Storage Mechanism for Large Statistical Data Bases; Lawrence Berkeley Laboratory Report LBL-15824; submitted to the Second International Workshop on Statistical Database Management, Los Altos, California, 27-29 September 1983.

HALL81 Hall, D., Scherrer, D. and Sventek, J.; A Virtual Operating System, *Comm. ACM, Vol. 23 (1980), pp. 495-502.*

KERN74 Kernighan, B.W., and P.J. Plauger, *The Elements of Programming Style*, New York, N.Y.: McGraw-Hill, 1974.

KERN76 Kernighan, B.W., and P.J. Plauger, *Software Tools*, Reading, Mass: Addison-Wesley, 1976.

MCCA82A McCarthy, J. L., Enhancements to the Codata Data Definition Language Lawrence Berkeley Laboratory Report LBL-14083, February, 1982.

MCCA82B McCarthy, J. L., et al., The SEEDIS Project: A Summary Overview, Lawrence Berkeley Laboratory Report PUB-424, April, 1982.

MERR81 Merrill, D., CODATA Users' Manual, Lawrence Berkeley Laboratory Internal Document LBID-021, revised August 31, 1982.

MERR82 Merrill, D. and Selvin, S.; Populations at Risk to Environmental Pollution (PAREP): Project Overview, 1976-1982; Lawrence Berkeley Laboratory Report LBL-15321, December 1982. Included in An LBL Perspective on Statistical Database Management, H. Wong, editor, Lawrence Berkeley Laboratory Report LBL-15393, December 1982.

SVEN82 Sventek, J. et al, *Software Tools Virtual Operating System User's Manual* (VMS Version), Lawrence Berkeley Laboratory internal documentation, revised 1982.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720