Lawrence Berkeley Laboratory

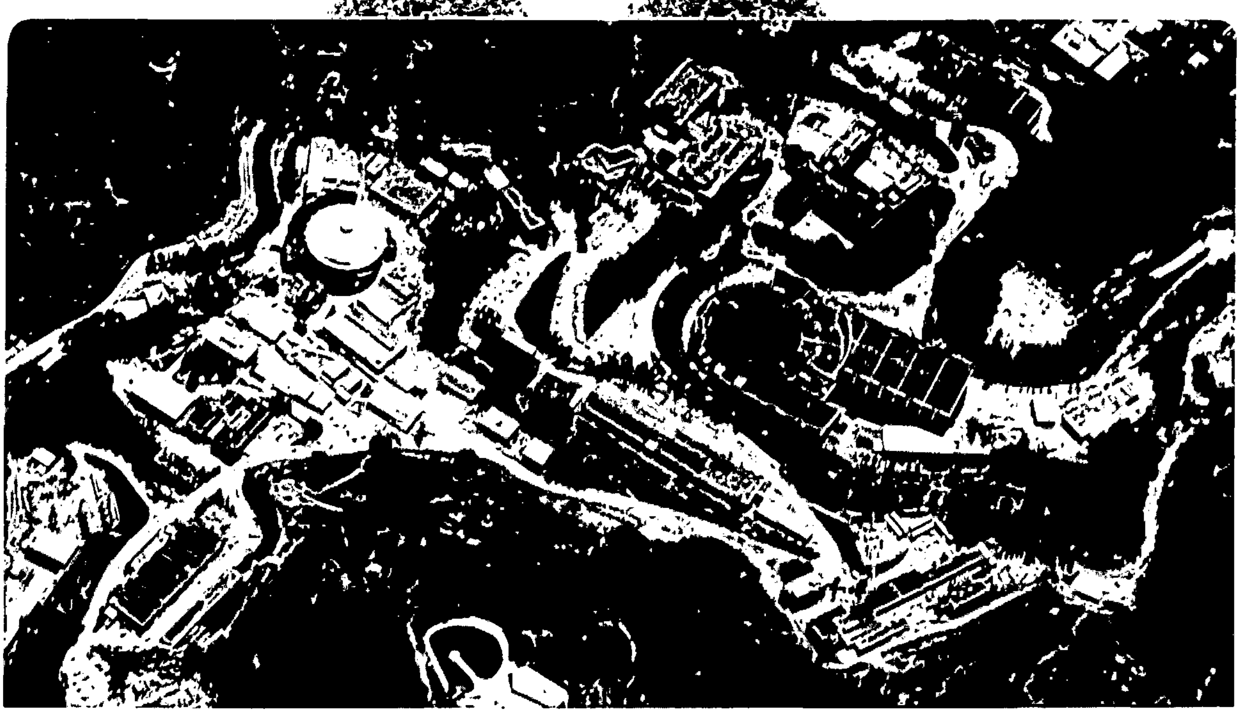
UNIVERSITY OF CALIFORNIA

Computing Division

ENHANCEMENTS TO THE CODATA DATA DEFINITION LANGUAGE

J.L. McCarthy

February 1982



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

DISCLAIMER

LBL-14083

ENHANCEMENTS TO THE CODATA DATA DEFINITION LANGUAGE

John L. McCarthy

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

February 1982

This work was supported in part by the Director, Office of Energy Research, Office of Basic Energy Sciences, Engineering, Mathematical, and Geosciences Division of the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

ENHANCEMENTS TO THE CODATA DATA DEFINITION LANGUAGE

by

John L. McCarthy

February 20, 1982 version

Author John L. McCarthy
Computer Science and Mathematics Department
Lawrence Berkeley Laboratory, 50B-3238
University of California
Berkeley, CA 94720
(415) 486-5307 (fts) 451-5307

Access tbl ~john/seedis/redesign/specs/codata | vtroff -mes

Files lblf:~john/seedis/redesign/specs/codata, macros, ex1, ex2, ex3

Version February 20, 1982

Status Preliminary Draft *Please mark up and return*
I would appreciate any comments – in writing if possible, verbally if necessary, and as soon as you can in any case.

Distribution

SEEDIS (2/18): Gey, Holmes, Merrill, Healey, Benson, Marcus, Yen, Burkhart, Schroeder, Kwok, Eades, Graves, Sventek

CSAM (2/18): Shoshani, Hall, Johnston, Kreps, Hawthorn, Chan, Eggers, Olken, Wong, Kuo, Selvin, Johnson, Sacks

PNL: [Burnett, Thomas, Littlefield, Carr, Nicholson]

Florida: [Su, Batory, Navathe]

Wisconsin: [DeWitt, Boral]

BLS: [Stevens, Weiss, Weeks]

Other: [Johnson, Einowski, Stinnett, Marks, Hammond, Becker, Klensin, Teitel, Dintelman, Kobayashi, Sato, Shanks, Ruderman, Sack, Guertin, Keefer, Winkler, Simes]

This memo outlines a number of proposed enhancements to the current codata data definition language (DDL) for SEEDIS [MCCA 81A]. At present, the codata DDL is used in data definition files (DDF's) to describe both fixed format, character-image codata files and binary format "SEEDIS Compressed" data files.

The basic enhancement proposed here is that we make the process of adding and changing metadata definitions easier (and less "hardwired") than it has been in the past. We also propose specific enhancements to facilitate handling 1980 Census Project data files, including multidimensional data elements, dimension descriptions, and differentiated labeling components. Such enhancements are intended to be compatible with a more extensive set of metadata outlined in a previous memo [MCCA 81B], as well as with other physical storage formats that may be incorporated in SEEDIS [EGGE 82] in the future, including multiply occuring data elements, transposed files, etc.

Since not everyone reading this memo will be familiar with codata concepts, the first section describes the current codata format and data definition language and the closely related "SEEDIS Compressed" format. Subsequent sections discuss proposed enhancements and implementation strategy.

Current Codata Data Definition Language

Codata is a shorthand term for the "common data format", which SEEDIS Project staff developed in 1978 to provide a self-describing interchange format for data used by different program modules [MERR 81]. The codata data definition language (DDL) presently is used to describe two different types of physical data storage implementions:

- fixed format, eye-readable codata files
- variable record-length, binary "SEEDIS Compressed" data files

In both cases, a data set consists of two logical components -- a data definition file (DDF) and a data file (DF). The logical data view is that of a table (i.e., a rectangular array or flat file) with a fixed number of rows and columns. Data are arranged so that each logical record is a row of the table and contains all the attributes (data elements or columns) of an named entity (e.g., "Alameda County", "person number 2037"), as well as a row label or stub plus any keys necessary for data access and matching. The number of logical records (rows) in a data file is equal to the number of entities, and the number of columns is equal to the number of data elements in each logical record (row).

The basic structure of metadata elements in the DDF is :

<keyword> = <value>

with one "keyword=value" pair per unit record (line). At present, all keywords may be abbreviated to the first letter, so the current procedure to read such metadata is basically to scan a physical unit record (line) for the first non-blank character. This data description is always stored in text form. No metadata field may cross physical unit record (line) boundaries. 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.

Supported keywords and syntax are described in the next two subsections. Keywords that are used with only one of the two types of data storage formats -- SEEDIS compressed files or codata files -- are so indicated in brackets. Names imbedded in angle brackets (e.g., <name>) are used to indicate values that must be provided by the user. When there are only a limited number of keywords permitted, those are separated by vertical bars (e.g., "alpha|decimal|integer").

Present Metadata Elements

Global (File Level)

This first set of metadata items only can appear in the initial, global section of a DDF, and they apply to the file as a whole.

FILE=*<text string>...*<text string>

This is simply a textual description of the file. * can be any user defined delimiter.

MODE=compressed

[SEEDIS COMPRESSED FILES ONLY]

VAX=<data file location | ddf file location>
[SEEDIS COMPRESSED DDF or NDX FILES]

NDE=<integer>
Number of data elements or columns

AREAS=<integer>
Number of rows

CARD_LENGTH=<integer>
Length of unit record in characters (max 132). At present, this gives the maximum record length for both DF and DDF in codata files, and for SEEDIS compressed DDF"s.

RECORD_SIZE=<integer>
Number of bytes in each physical block [in NDX files describing COMPRESSED FILES ONLY]

ACCESS=direct
Type of access method used [for NDX file describing COMPRESSED FILES ONLY]

MISSING=<low end real value> <high end real value>
Range of values to be considered as "missing data" in calculations.

END_DDF
Indicates end of DDF.

Global or Element Level

The following items apply to all data elements in the file if they appear at this point in the global section of the DDF. They can also repeat within each individual data element definition in order to override the global defaults.

TYPE=<Alpha|Integer|Decimal>
Decimal points must be explicit for fixed format codata files.

USE=<Stub|Data|Key>
If the data element use is key, the data element name must match one of the type A keys currently defined in SEEDIS (about 80 at present; most of them geographic -- e.g., FIPS.STATE)

SCALE_FACTOR=<real number>
Number by which stored values are multiplied to yield displayed values [COMPRESSED FILES ONLY]. Codata files created from compressed files are scaled prior to output, since codata files presently do not have a scale factor.

* <comment>
text for comments -- can appear at any point in the ddf

Repeated for Each Data Element (Column)

This group of metadata items recurs for each data element in a file.

DE=<name>
Data element name

START=<integer>
Starting character position in a logical record [CODATA FILES ONLY]

LENGTH=<integer>
length in characters [for compressed data files, this directive pertains to output string length only]

POSITION=<integer>
Sequential data element number [in DDF for COMPRESSED FILES ONLY]. These are these generated automatically by the routines that create a compressed file from a codata file.

HEADER=;<line 1>;<line 2>;<line 3>;
The semi-colon can be any user-defined break character to separate portions of a header or label that should be output on different multiple lines; alternatively, the user may specify multiple "Header =;<text>;" type directives. Currently, the codata tools use whatever character immediately follows the equals sign as the break character.

Present Data Files

The second logical component of a data set is the data file. As mentioned above, the same DDL is used to describe either of two different kinds of physical storage formats for data files -- fixed record-length, line-image codata files and variable record-length, binary SEEDIS compressed files. Although this paper is concerned only with the logical data definition language, it may be helpful to put things in context with a brief summary of the two types of physical data storage formats which the DDL currently supports.

Codata Data Files

In codata files, both the data definition file (DDF) and the data file (DF) reside on a single physical file, with information stored in character representation within fixed-length logical records as defined in the DDF. The DF is the subset of fixed length records following the "END DDF" directive; it contains data as defined in the DDF. As in the DDF, data fields in the DF cannot cross unit record boundaries. For numeric data, missing or suppressed values can be indicated by the global MISSING directive or simply by leaving the field blank (blanks are not interpreted as zeros on input, as in FORTRAN). Codata write routines automatically reformat fields in the DF to right justify numeric data.

SEEDIS Compressed Data Files

In SEEDIS Compressed files, the DDF is maintained in a separate physical file comprised of logical records with 132 column fixed length records and character representation -- just a slightly specialized form of codata DDF. Data is stored in binary form on a separate file [EGGE 82]. In addition, every SEEDIS Compressed file has two types of associated ancillary files. The DDX is a special type of file which serves as an index to the separate DDF; it contains data element names and their corresponding byte offset locations in the DDF. The NDX is a special codata file which contains a list (usually sorted) of KEY values for all records, and which serves as an index to the binary data file. Each key value set has a block number pointer to the starting position of the corresponding compressed record in the binary data file plus a byte count for the record. Since each compressed record starts on a block boundary this makes it easy to calculate how many blocks or bytes need to be read for a given record. There may be multiple NDX files for a single data file, and in some cases, the NDX and data files may be broken down into subsets (e.g., state groupings of county-level records). [A more complete discussion of this physical storage implementation and possible enhancements will be the subject of another paper]

Sample Codata File

The example below illustrates current useage of the existing DDL in a standard, fixed format codata file (DDF followed by the DF it describes).

```
FILE=*sample codata file*
NDE=4
AREAS=4
CARD LENGTH=40
TYPE=d
USE=d
* this is a sample data base
DE=FIPS.STATE
   TYPE=i
   USE=key
   START=1
   LENGTH=3
DE=area.name
   TYPE=a
   USE=stub
   START=4
   LENGTH=10
DE=population
   START=14
   LENGTH=8
DE=pop.density
   HEADER=;total population;per;
   HEADER=;square mile;
   START=23
   LENGTH=5
END DDF
 1alabama        10000  5.32
 4arizona       310012  25.1
 6california22000000 170.5
 9washington      4000  23.8
```

Primary Proposed Codata DDL Enhancements

One of the great strengths of the current codata implementation is its relative simplicity of structure, grammar, and syntax (with the exception of a few minor problems which should be easy to correct). Most people find codata files relatively easy to understand and use. In considering possible enhancements, we should try to preserve this simplicity as much as possible. At the very least, users should be able to continue using the present simple structures and syntax for simple data files. More complicated metadata should be optional.

The following sections propose a number of enhancements to the codata DDL, including some extensions and changes to the grammar as well as new metadata elements. Although none of the proposed enhancements calls for radical changes in the basic codata file design, some of them will require major software changes. Insofar as that is true, it would probably be a good idea to use the opportunity to also clean up some of the minor (but sometimes frustrating) problems with the current codata implementation.

Although a number of specific additions to the list of codata metadata elements are proposed below, the major change that we need to make to the codata DDL is not addition or change of particular metadata elements, but rather enhancement of our basic underlying ability to add or change metadata definitions quickly and easily. No set of metadata descriptions, however comprehensive, will be adequate for long. We will always be thinking of new things that would be useful to have, and ways that current definitions ought to be refined. We therefore need DDL facilities that will allow us to make such enhancements in the future without additional programming. With that general point in mind, let us turn to the specific DDL enhancements that are particularly needed for the 1980 Census Project.

Metadata Names and Aliases

As noted above, the present codata implementation requires that the first letter of each metadata element be unique, since that is the only part of the name that is currently examined by the program(s) that read DDF's. If we want to add a number of new metadata elements, it is awkward at best to constrain ourselves to names that have unique first letters. It would be a better idea to modify the routines that read metadata (principally CRDEF) to look at the entire metadata element name, and drop the requirement that first letters be unique.

Truncation or Explicit Aliases

The question arises whether we should permit truncation of names to unique strings or insist on designation of explicit "aliases" or synonyms wherever users wish to employ something other than the primary name. Our tentative recommendation is to require explicit aliases and not support truncation to unique strings.

In any case, we should support aliases or synonyms for metadata names (using only the first letter in effect accomplished the same thing in a less direct way). This will permit both long, descriptive names that help the user understand what each metadata element is, and short abbreviations to facilitate data entry and preparation of long data definition files. If we rewrite the program(s) that read metadata information, it shouldn't be too much additional work to incorporate code which permits several different synonyms or aliases to be used for a given metadata element name. For example,

DE = data_element = element = data_element_name

Codata write routines should continue to do some cosmetic reformatting of DDF's on output, such as converting aliases to primary names, indenting names to reflect nested metadata structures, etc.

Naming Conventions

Names should not contain embedded blanks. When separate words are required for clarity, underline characters can be used as connectors. We may not want to permit use of periods as simple connectors in names in order to reserve the period as a structure member operator (as it is in C and a number of database query languages). On the other hand, there are already a number of SEEDIS data element names containing periods, and people are rather used to using periods rather indiscriminately. Perhaps it would be better to use some less common character, such as "@" for the structure member operator.

Names of data elements and metadata elements need not be completely unique. They need only be unique within the data structure in which they appear. Thus a fully qualified data element name is preceded by all of the structures within which it is nested, ordered from highest to lowest, left to right.

It is proposed that each of these individual components can be up to 32 characters long. The total length of a fully qualified name, including all structure members connected by structure member operators, should not exceed 256 characters.

Long Data Elements and Continuation Conventions

Since the present codata standard does not permit any field to cross unit record boundaries, there has been no need for continuation conventions. In order to accommodate lengthy textual metadata items, such as descriptions, it might be a good idea to relax this requirement and permit character fields (but not numeric fields) to cross unit record boundaries. If so, there are two continuation conventions that we could honor and support.

First, routines which read metadata could look for an equal sign as the second token on each line. All situations where that is not the case could be recognized as continuation lines. If some pathological piece of text contains an equals sign at that point, the text would have to be surrounded by quotation marks to work correctly.

Second, as an alternative for those who wish to avoid the possibility of pathological cases, users could specify and use an explicit continuation character, such as "-", at the end of each line to be continued. The continuation character itself could be defined as one of the global (file level) metadata elements at the beginning of a ddf. The syntax might be:

continuation = <global continuation character for this codata file>

For maximum flexibility and consistency, the continuation character should not cause automatic insertion of white space when such lines are concatenated, so users would have to be careful to put such spaces where desired.

New Metadata Elements for Arrays

In order to handle 1980 Census Project data, it is almost imperative to have a means of describing and manipulating multidimensional tables. Otherwise we have to treat each individual cell of multidimensional census tables as a distinct data element. Since many census tables contain hundreds of cells, this would add substantially to the cost of preparating and maintaining DDF's, data dictionaries, etc.

Proposed syntax for multidimensional tables is described below. Syntax for data manipulation operations on arrays and subsets of arrays will be described in a separate paper. The minimum metadata information for an array element will be simply its name, cell_length, and either array_size or one or more dimension names.

structure = array | simple

This new metadata element within the global section or under a particular data element will denote an array with fixed dimensions and the same homogeneous type of data values in each of its cells. To begin with, arrays in codata files will be permitted to contain only homogeneous, fixed length cells (just as simple data elements can only be fixed length). The default value of the structure metadata element will be "structure=simple" for simple, single-valued data elements. At a later time we might incorporate "structure=complex" to describe more general structures consisting of arbitrary sets of data elements (some of which might themselves be arrays or structures, nested several levels deep). Eventually, we might also wish to add description of arrays whose components might themselves be complex data structures. To begin with, however, arrays could contain only simple, homogeneous sets of data values.

array_size = <n*m*...*r>

This gives the size of each dimension in an array-structured data element. n,m,...,r must be integers and must agree with information for each named dimension, if present. If this metadata element is not included for an element whose structure=array, it will be automatically calculated and inserted in the DDF, based on information in the named component dimensions.

cell_length = <integer>

the fixed length of each component cell in the array, used (in conjunction with card_length and array_size) to automatically calculate individual cell storage locations for the entire array. (Alternatively, one can use the length specification under the "cell" directive described below.)

cell = <subscript expression for a cell, range, or list of cells>

Any metadata item that can normally appear under a simple data element can appear below this directive to qualify certain cells or groups of cells in an array element (e.g., suppression flags and group headers for census data).

The subscript expression will contain a list of one or more sub-expressions, one for each dimension in the array, separated by commas. Each sub-expression will be either an integer, a pair of integers separated by a colon (to indicate a range); a colon (to indicate the entire range of values in that dimension), or a list of values and/or ranges separated by commas and surrounded by parentheses. The last (or rightmost) dimension will vary most rapidly, as in standard mathematical notation and PL/I (but just the opposite from FORTRAN). Subscripts will start at 1 (as in FORTRAN), rather than 0 (as in C). For example, cell = 3,(1,5:9) would refer to six values drawn from the third "row" of a two dimensional array.

The cell metadata element can occur multiple times to permit different information about different cells in an array.

dimension = <name>

this will describe a component dimension of the array. There may be multiple dimensions for a given array_element, and if there is more than one dimension, the sequence of dimensions will implicitly provide sequential dimension numbers for array notation, etc.

Metadata items concerning each dimension are described below. If there is no additional metadata information about that dimension immediately following a dimension directive, then such information should be located either

(a) in a previous dimension description, or
(b) in a global SEEDIS dimension description

The following new metadata elements could recur under each dimension directive.

category = <name>

component category of the dimension. Each dimension will have two or more categories, and categories will be implicitly numbered by sequential order. Additional information about each category will be described in descriptive metadata elements described below.

category_group = <name>

Name for a set of categories which may be grouped together for some purpose, such as higher level labels (for census data) or crosswalk procedures between one dimension and another similar (but not identical) dimension or value_label_set. Unlike categories and values in a dimension or value_label_set, category_groups would not have to be mutually exclusive or exhaustive.

category_group_component = <category name>

This multiply occuring item would provide the names of categories in a named group.

Descriptive Items for Various Metadata Levels

In order to provide finer gradations of descriptive information for labeling and documentation, we propose that databases, data_elements (including array elements), dimensions, value_label_sets, categories, category_groups, and data_element_groups each be permitted to have one or more of the following descriptive metadata elements:

alias = <name>
an alias or synonym to be used as an alternative name for the main name. There can be multiple instances of alias -- one for each alternate name.

occurrence_number = <integer>
this subscript or sequence number would normally be derived implicity from the sequential order of the item (element, dimension, etc.), but could be specified or requested explicitly.

header_label = <text string of up to 512 characters>
used to label output, etc. Should be as concise as possible, with abbreviations OK. May include information about universe (until other aspects of SEEDIS are revised), but should not duplicate information from "file" or other higher level headers -- since those headers will be concatenated on output (e.g., the database header "1980 Census STF1A" would be prepended to each data element header of that database, and category headers would get appended for individual cells of an array). May contain special intrafield break character to indicate default line separators.

class = <name>
One of a restricted set of names recognized by SEEDIS, used to indicate broad classes of dimensions, value_label_sets, entities, etc. (e.g., geography, time).

subject = <key word or phrase>
A multiply occuring metadata element, each of whose values will be chosen from a restricted vocabulary set of key words and phrases -- perhaps organized in hierarchical thesaurus form. These subject terms could be subsequently used to produce a cross-database subject index for all of SEEDIS.

group = <group name>
Name of a group (e.g., category_group, data_element_group) to which this item pertains (see above).

description = <up to twenty-three 78-character lines of text>
one screen of descriptive information -- as complete as possible. More lengthy text can be put in supplementary occurences of this metadata element.

note = <unlimited amount of text>
special information that should be highlighted or emphasized and printed along with any description

footnote = <integer or alpha code>
reference to one or more numbered footnotes kept elsewhere

comment = <unlimited amount of text> remarks relevant only for data installers, database administrators, etc. (not normally displayed to users). As at present, comments can occur at any point in the DDF.

If a header label were not specified for a given item, the name of the item would become the default header label. If the item were not named (e.g., a particular category in a dimension), the sequential occurence number would be used as the default name.

Since these descriptive metadata elements could occur at different levels of the DDF, there could be instances where the level to which something like "description" or "alias" pertains might be ambiguous. In such cases, the convention will be that the metadata element will pertain to the structure immediately above it in the DDF. For example, if a DDF contained the following:
element = tab12
structure = array
dimension = race
description = five major racial groups
the "description" would pertain to the dimension rather than the element.

Data Element Groups

Sooner or later it may be desireable to be able to reference two or more data elements as a named set -- in order to minimized redundant specifications, storage, etc. This could be done in several different ways. One of the easiest, at least to begin with, might be to have a metadata element called data_element_group, which could in turn contain two or more data_element_component items, as follows:

data_element_group = <name>
Name for a set of one or more data elements to be treated together.

data_element_component = <de name>
Name of a data element defined elsewhere in the DDF which is to be considered a member of this data_element_group.

Data element groups may also contain any of the set of descriptive metadata items discussed above.

Other Proposed Enhancements to Codata DDL

Since the original codata implementations in 1978, a number of individuals have made some experimental modifications to the data definition language and extensions to the standards outlined above (in fact, some of the "standards" outlined above, particularly those pertaining to SEEDIS Compressed files, were *ad hoc* extensions). Other enhancements were proposed in the original codata design specifications but not implemented universally. During the past year, Deane Merrill implemented several additional metadata elements and tested them in an experimental version. Most of these enhancements have not been incorporated throughout SEEDIS because doing so would have required changes in the calling sequences of a number of related programs.

Deane's experience provides some useful insights on the process of adding new metadata information in general. As he has pointed out, the process is currently a rather painful one, because each piece of metadata information is passed as a separate parameter in a FORTRAN subroutine call. Each time we add a new metadata element, all programs containing the call have to be changed and relinked. As a result, we have been quite reluctant to add new metadata information.

Other Global and Element Level Items

New metadata elements that Deane has experimented with are included below, along with a number of others that ought to be considered in the present round of enhancements. These metadata elements may appear at both the global and individual element levels. Those that appear in the global section of a DDF serve as default values for all elements unless explicitly overridden within an individual data element definition.

A number of these elements have values which are <expression>'s. As used here, an "expression" is an arithmetic expression which can contain constants, arithmetic operators, data element names, and perhaps function calls. If a data element name appears, the stored value of that data element from the same data record will be used (e.g., if "weight = population80" appears as part of the definition for a data element called "mortality70", where "population80" is the name of another data element in the same data record, then each value of "mortality70" will be weighted by the corresponding value of "population80" in each data record.

ddf_style = *1978*|1982
: differentiates old from new ddf's. Later we could add others (e.g., SPSS) to distinguish particular flavors of DDF's.

universe = <text>
: optional description of population universe to which one or more elements pertain (a standard census metadata item).

weight = <expression>
: a mathematical funtion referencing one or more other data elements, which could be used for automatic aggregation, disaggregation, etc.

missing = <low real value> <high real value>
: different missing data codes can apply to different elements, whereas the present codata standard recognizes "missing" only as a global, file-level directive.

error = <expression>
: In the future this could be used for automatic calculation of errors as data elements are displayed or used in computations.

scale factor = <real number>
: physically stored data are multiplied by this number prior to display or use in calculations. Can be used for standard conversion of data without recompression (e.g., miles to kilometers, one geographic coordinate system to another, etc.). This is currently implemented only for SEEDIS Compressed files.

suppression = <expression>
: the number of a suppression flag which pertains to this element (or cell of an array, as in census data), or the name of another data element containing suppression information. For the time being its use would be primarily descriptive, but it might be used in the future in various analysis and display modules.

value_label_set = <name>

This name would refer to and/or precede a set of items pertaining to labels for individual data values found in one or more data elements (or cells of an array). Its structure would be identical to the dimension metadata structure described above -- and the names would refer to a common pool of such structures, but it would be maintained as a separate type of metadata because a particular array element might have both dimensions and a value_label_set.

Correcting Problems With The Current DDL

The most serious limitation of the current codata data definition language is the difficulty of adding new types of metadata or modifying old ones. The most important change we need to make is to make addition of new metadata easy in the future. Once that fundamental change is accomplished, we can improve the undifferentiated way in which descriptive information is handled, and the lack of facilities for data structures such as vectors and arrays, using some of the new metadata constructs outlined above.

In addition, although the basic structure and syntax of the codata DDL are quite simple and clean, there are a few current metadata elements and codata conventions that present some problems. Cleaning these up should not be particularly difficult, and doing so would be very much in keeping with the design goal of keeping things simple and general.

Special Characters for Breaks, Concatenation, etc.

As mentioned above, the codata tools currently use whatever character *immediately follows* the equal sign as a break character for the "FILE =" and "HEADER =" metadata elements -- this often leads to confusion if unwary users try to enter a one line header with no break character or leave a blank following the "=", etc. It would be less error prone if we made this intra-field break character an explicit metadata element which could be designated at the global or element level (perhaps something like "intra_field_break = #").

The original codata specifications also provided a mechanism to specify the character used for assignment of metadata values (normally "="). It might perhaps be useful to provide such a facility with something like "assignment_character = =", though this is certainly a lower priority item.

We also need a structure or concept delimiter that can be used to concatenate different levels of metadata information (e.g., data_base@data_element@header). This, too, could be specified at the beginning of the DDF itself, with something like "concatenation_operator = @".

Finally, if we are going to use special symbols to denote particular kinds of delimiters or operations, we also need an escape character to be able to use those symbols in simple text. Here, too, the escape character could be defined locally at the beginning of a DDF (e.g., "escape_character = \ ").

Upper and Lower Case Distinctions

The current implementation not only stores upper and lower case letters for text fields (which we definitely want to continue doing), but it also makes distinctions between named entities such as metadata names depending on case (which is not such a good idea). For example, names of data elements that are used as keys have to be capitalized to match corresponding names in a central SEEDIS list of recognized key names. It would be preferable to do automatic translation from upper to lower case for any such purposes of comparison. It should not make any difference whether someone uses an upper or lower case name for a metadata element, a data element name, etc.

Leading and Trailing Blanks

The current implementation forces codata users to right justify numeric input data or errors will result. It would seem preferable to permit placement of numeric data anywhere within a fixed field. Codata write routines should continue to reformat such data so that it gets right justified on output.

Metadata Specific to SEEDIS Compressed Files

The "mode = compressed" element is a good one. It could later be extended to include other physical storage formats, and perhaps even to permit individual data elements or subfile structures to have different modes, such as "transposed" within an over-all default mode, such as "sequential".

It is not a good idea to have a metadata element (such as VAX =) represent different things depending upon what kind of file it happens to be in. Nor is it a good idea to have metadata element names that do not describe the actual use of the element, at least in a rudimentary way. We therefore should change "VAX =" to two different metadata elements -- perhaps "data_file =<VMS path name>" and "ddf_file =<VMS path name>". "Record Size =" also seems to be a misnomer. While we could continue to recognize that old syntax, a preferable main name might be "blocksize =".

Handling of Input and Output Data

At present, each of the different versions of the codata write subroutines handle output in slightly different ways. Some insert leading zeros and/or blanks while others do not, some insert explicit decimal points while others do not, etc.

One immediate approach to this problem would be to decide on a standard default means of handling different types of data. A more general solution might be to incorporate two more metadata elements that could apply to individual data elements or sets of elements: input_processing_rules and output_processing_rules. The values of each of these metadata elements would be special types of expressions specifying how data should be converted and formatted on input to and output from the database. Scale factors and weights are, in fact, special cases of such general processing rules. Other examples might be conversion from integer to binary representation on input and formatting with two decimal places and an explicit decimal point on output. Input processing rules could also contain validation checks and different types of error handling specifications.

Missing Data

The current implementation only permits upper and lower bound specifications for a single range of missing data codes for numeric fields. There is not really any concept of missing data for character fields. In addition, the way in which missing data specifications interact with the scale factor specification is not perfectly straightforward.

Missing data specifications should apply to *stored* values, rather than to values after conversion via scale factors, weights, etc. There needs to be a low-level routine that checks for missing values prior to other operations, so that scale factors, etc. do not get applied to missing data values on output or use in computation routines.

We eventually should provide more general missing data specifications. It may be useful in some situations to distinguish between different types of missing data in character fields. It also might be desireable to have non-numeric characters specify missing data in numeric fields and/or to permit non-contiguous missing data codes, which could be handled by multiply occurring missing data directives. It also may be desireable to be able to distinguish between "missing" values and non-existent or null values for a particular instance of an individual data element occurence.

Record/Entity Type Designation

In our current implementation, each codata or compressed data file can contain only one record type -- i.e., the set of data elements described by the DDF. SEEDIS codata files currently must include a special "comment" line of the form "*LEVEL=<name>", where <name> specifies one of the geographic levels presently recognized by SEEDIS, such as "state" or "county80". SEEDIS compressed files can contain data from multiple levels, so they have no explicit designation of level, but records can only differ in terms of the types of entities to which they pertain -- e.g., counties, states, smsa's, tracts -- and not in terms of the data elements (including keys) each record contains. In fact, multiple NDX files can be used to "point to" different sets of records in a compressed data file -- in order to differentiate data from different levels of census data or to distinguish two slightly different definitions of the same level (e.g., county70 and county80) without redundant data storage.

For the time being (in the interest of simplicity), we will continue to restrict codata and SEEDIS compressed files to a single record type. Instead of designating "level" by means of a special kind of comment line, however, that information should be an explicit part of the DDF. For each separately indexed set of data records (e.g., counties, mcd's, places), there will be three metadata elements, as follows:

entity = <name>
Name of the geographic or other type of entity indexed -- equivalent to the current codata "*LEVEL=" specification.

key = <data_element_name>
A multiply occuring metadata element, one or more of which would give the complete set of data elements (with USE = key) required to uniquely identify an instance of this particular type of entity (e.g., fips.state for state, fips.state plus fips.county for counties, etc.).

ndx = <VMS file name>
Name of the NDX file used to index and identify individual instances of this particular type of entity.

Organization of DDX and NDX Files – A Digression

At present, DDX files are not themselves codata files. It probably would be a good idea if they were. In addition, neither DDX nor NDX files are necessarily sorted, and the routines that access them use serial search methods rather than binary search routines to locate individual values. This works well enough for small files and situations where access is essentially serial, but there are substantial performance penalties for the DDX or NDX files, especially in situations where there is no record instance for the value being sought (e.g., a geocode or data element name that is not in the NDX or DDX file) and search routines thus have to scan the entire file.

We should require that all NDX and DDX files be *sorted* codata files, and routines which access them should use binary search routines for optimal performance.

Variable Length (and Multiply Occurring) Data Fields

The original codata file design specified a means of describing a data file with variable length data fields, in addition to the fixed format standard that was implemented. Although implementation of a variable length field scheme may require too much effort for immediate implementation, some form of handling variable length fields certainly would be desirable as soon as we can manage it. In the 1978 codata specifications for variable length fields, physical record length remained fixed, as specified in the global "record length" metadata element, but fields could be separated by a DDF-specified break character (Break = <field terminator character>), rather than (or in addition to) starting position and length. Any conflicts between break, start, and length would result in error messages. Fields could not span physical records (lines), but logical records (rows) could contain different numbers of physical records (lines). New rows (logical records) would always begin at a new physical record.

Missing data could be specified by simply concatenating break characters as well as by blanks or explicit missing data codes. One exception to this rule would be if the break character were defined to be a blank; in such cases, only one delimiter would be counted no matter how many blanks were found.

There are also situations where it would be desirable to store multiply occuring values for a single data element. The current "header" metadata element is such a case. One simple means of doing this would be to use another DDF-specified break character to indicate multiple occurrences within an individual data field.

As we redesign other aspects of the codata specifications, we need to bear in mind the possibility of variable length and multiply occuring fields and groups of fields. If efficiency considerations are the primary barrier to implementation of variable length and multiply occuring fields, perhaps we could implement them in the context of a different ddf "style".

Summary of Proposed Enhancements

This section summarizes enhancements proposed above by means of a hierarchical diagram and two example ddf's incorporating the new proposed metadata elements.

Metadata Structure

As things presently stand, there are three implicit levels of hierarchy in the current codata ddl. At the first level, one can define a number of global parameters. At the second, or data element level, each de can have several associated types of metadata elements such as headers, type, etc., which repeat for each individual data element. Thus to uniquely identify a particular header, one must specify dename@header. In fact, since headers can occur multiple times for a given data element, there is a third implicit level, and a full specification would be dename@header@instance.

In order to accomodate some of the additional metadata elements proposed in this paper, we propose to permit up to nine levels in the hierarchy, so that one might have something like

element@dimension@category@label

Internally, this could be handled with unique numbers, rather than names, at each level.

Exhibit 1 below illustrates the proposed hierarchical structure for new and existing metadata elements. Existing metadata elements are shown in UPPER CASE, and proposed metadata elements are shown in lower case. The current name is sometimes shown as an alias rather than the primary name in cases where it seems that a new primary name would be clearer.

Proposed aliases or alternative synonyms for primary metadata element names are shown to the right of each name, separated by slashes. Special comments appear in *italics (or underlined)* to the right of any aliases.

Items that are the initial element in a metadata structure are shown in **boldface**, and items included within the structure are indented below the initial item. A set of items that is repeated for several different metadata levels is refered to by a {name in brackets} after its initial presentation in order to save space.

The letters "m" and "r" to the left are flags to indicate which elements are *r*equired (i.e., must occur) and which may occur *m*ultiple numbers of times in a given structure. Items are optional (i.e, they need not occur) unless shown as *r*equired. Items will only occur once in a given structure unless indicated as *m*ultiple. If an item is only required for codata files, a (c) follows the r; if it is only required for Seedis compressed files, a (s) follows the r.

Metadata elements flagged by an asterisk (*) at the left are those that deserve top priority for inclusion on the first round of enhancements.

For metadata elements whose values can only take on a restricted set of keywords, those keywords are shown following an equals sign (=), with the default value in italics (i.e., the value that will be assumed if the metadata element is not present in the input DDF). For other required metadata items where there is a default, the value that codata read/write routines would insert is shown in brackets.

Exhibit 1: Hierarchical Structure of Proposed Metadata Elements

flag	name[=*default*|value2|...]	/alias/alternate name(s)/

Database Metadata

m	comment	/*/
r	ddf_style=*1978*|1982	/style/
r	ddf_author	/author/
r	date_ddf_created [= *system*]	/create/
r	date_ddf_last_modified [= *system*]	/modified/
	assignment_character [= =]	/assignment/
	~~continuation_character [= -]~~	~~/continuation/~~
	escape_character [=]	/escape/
	inter_field_break [= ;]	/field_break/
	intra_field_break	/line_break/
	concatenation [= @]	
	MODE=*codata*|compressed	/storage_mode/
r	element_count	/NDE/number_of_de's/
r	record_count	/AREAS/records/
r(s)	data_file	/VAX/
r(s)	ddf_file	/VAX/
r(s)	blocksize	/RECORD_SIZE/
r(c)	record_length	/CARD_LENGTH/
r(s)	ACCESS=*sequential*|direct	
*r	**database**	/database_name/
	{*Description Set*}	
	{*Data Element Default Set*}	

Entity Metadata

rm	**entity**	/entity_type/
rm	key	
r	ndx	/index/
	{*Description Set*}	

{Description Set}

*	occurence_number [= *system*]	/subscript/
*m	alias	
*m	label	/HEADER/title/
*m	description	
*m	note	
*m	footnote	
m	subject	/keyword_phrase/
m	group	
	class	

{Data Element Default Set}

r	data_type=*alpha*|int|dec	/TYPE/
r	USE=*data*|key|stub|sort	
r	MISSING [= -999?]	
	SCALE_FACTOR [= 1.0]	
r	structure=*simple*|array|group	
	weight [= 1.0]	
	error	
	universe	
	valid_values	
*	suppression_flag	

Data Element Group Metadata

m	**data_element_group**	/degroup/
	{*Description Set*}	
m	data_element_component	/de_component/

Data Element Metadata

rm	**DATA_ELEMENT**	/de/element/
	{*Description Set*}	
	{*Data Element Default Set*}	
r(c)	START	
r(c)	LENGTH	
	value_label_set	
	{*Description Set*}	
m	**value**	/category/
	{*Description Set*}	

Array Metadata

*	array_size	
	cell_length	
m	**cell**	/cells/cell_specification/
	{*Description Set*}	
	{*Data Element Default Set*}	
	[any items under de may also appear here]	
*m	**dimension**	
	{*Description Set*}	
*m	**category**	/value/
	{*Description Set*}	
*m	**category_group**	/cgroup/
	{*Description Set*}	
m	category_group_component	/cgitem/

Exhibit 2: Example Census Codata File With Minimal Metadata

```
style = 1982
author = Deane Merrill
create = 2-feb-1982 18:13:32
modified = 2-feb-1982 18:45:16
database = stf1a.fragment
 MODE = codata
 NDE = 5
 AREAS = 5
 CARDLENGTH = 70
 MISSING = -21 -1
 label = 1980 U.S. Census of Population
 universe = U.S. Population, 15-apr-1980
entity = state
  key = fips.state
  ndx = disk$seedis001:[seedata.census80.stf1.county80]s44.ndx
DE = fips.state
  TYPE = alpha
  USE = key
  START = 1
  LENGTH = 2
DE = fips.county80
  TYPE = alpha
  USE = key
  START = 3
  LENGTH = 3
DE = stub.geo
  TYPE = alpha
  USE = stub
  START = 6
  LENGTH = 33
DE = tab12
  structure = array
  array_size = 5*4
  cell_length = 9
  TYPE = int
  USE = data
  START = 39
  universe = persons
  dimension = race1
    class = race
    category = total
    category = white
    category = black
    category = indian
    category = asianpi
  dimension = age2
    class = age
    category = under5
    category = 5to17
    category = 18to64
    category = over64
  cells = 1,:
    suppression = supflg01
  cells = 2,:
    suppression = supflg02
  cells = 3,:
```

```
      suppression = supflg03
    cells = 4,:
      suppression = supflg04
    cells = 5,:
      suppression = supflg05
  DE = tab13
    structure = array
    array_size = 3*4
    TYPE = int
    USE = data
    START = 238
    cell_length = 9
    universe = Persons of Spanish Origin
    dimension = race2
      class = race
      category = total
      category = white
      category = black
    dimension = age2
    cells = 1,:
      suppression = supflg06
    cells = 2,:
      suppression = supflg07
    cells = 3,:
      suppression = supflg08
END DDF
44001RI Bristol                                     2611    9541   28925
   5865    2566    9427   28587    5833       4      18
     87       8       5      11      23       0      18
     45     109      18      52     164     459      68
     50     156     428      67      -8      -8      -8
     -8
44003RI Kent                                        9435   32774   94384
  17570    9294   32346   93353   17465      55     149
    359      42       3      32     100      10      52
    178     383      35      89     284     630      64
     82     256     541      59      -8      -8      -8
     -8
44005RI Newport                                     5336   16871   49936
   9240    4797   15714   47241    8978     320     731
   1770     191      47      67     156      12      77
    194     390      26     116     299     754      96
     78     210     532      86       6      22      36
      2
44007RI Providence                                 33089  107955  345656
  84649   28696   97091  322242   82430    2365    6238
  13181    1272     146     413     844     150     270
    596    2100     209    1880    4140    9150     765
    998    2202    5226     559     126     175     454
     31
44009RI Washington                                  6221   19018   58480
   9598    6001   18342   56858    9431      59     197
    481      57      73     267     460      79      36
    121     423      23      75     149     427      46
     55     107     275      42      -8      -8      -8
     -8
```

Exhibit 3: Example Census Codata File With Full Metadata

```
style=1982
author = Deane Merrill
create = 2-feb-1982 19:26:25
modified = 2-feb-1982 20:12:14
continuation = -
escape = \
intra_field_break = ;
database = stf1a
   label = sample 1980 stf1a codata file
   note = level is county80 (1980 Census counties)
   comment = county area calculated from geographic base map
file. pop.density calculated with Query module
   MODE = codata
   NDE = 7
   AREAS = 5
   CARDLENGTH = 60
   TYPE = dec
   USE = data
   weight = 1.0
   MISSING =  -8
   error = 0.01
   universe = U.S. Population, 15-apr-1980
entity = county80
  key = fips.state
  key = fips.county80
  ndx = disk$seedis001:[seedata.census80.stf1.county80]s44.ndx
DE = fips.state
  TYPE = alpha
  USE = key
  START = 1
  LENGTH = 2
  label = FIPS state code
  description = 1980 FIPS state code
DE = fips.county80
  TYPE = alpha
  USE = key
  START = 3
  LENGTH = 3
  label = 1980 Census county code
  note = county codes are only unique within a state, so they must
be used in conjunction with state codes
DE = area_name
  TYPE = alpha
  USE = stub
  START = 6
  LENGTH = 35
  label = state abbreviation;and;1980 Census county name
DE = population_density
  TYPE = dec
  START = 41
  LENGTH = 8
  intra_field_break = #
  HEADER = total population#per
  HEADER = square mile
  note = calculated as stf1.tab1(1)/ctyarea.area.nickel (pop per
sq km)
```

```
   scale_factor = 2.59
     comment = value stored in this file is per sq km
     comment = value displayed will be per sq mi, 2.59 times as
large
   missing = -1
 DE = tab12
   TYPE = int
   START = 49
   cell_length = 9
   array_size = 5*4
   universe = persons
   dimension = race1
     alias = major races
     description = major racial groups, including hispanic
     class = race
     category = total
       comment = default label is category name
     category = white
     category = black
     category = indian
       label = American Indian, Eskimo, and Aleut
     category = asian_pi
       label = Asian and Pacific Islander
       description = In 100-percent tabulations, Asian and Pacific
Islander includes "Japanese", "Chinese,", "Filipino," "Korean,"
"Asian Indian," "Vietnamese," "Hawaiian," "Guamanian," and
"Samoan." In sample tabulations it includes these groups plus per-
sons who have a write-in entry of an Asian or Pacific Islander group
in the "Other" category.
       footnote = 4
     cell = 1,:
     comment = note how ":" indicates all values in that dimension
       suppression = supflg01
       missing = -1
     cell = 2,:
       suppression = supflg02
       missing = -2
     cell = 3,:
       suppression = supflg03
       missing = -3
     cell = 4,:
       suppression = supflg04
       missing = -4
     cell = 5,:
       suppression = supflg05
       missing = -5
   dimension = age2
     alias = age_4groups
     class = age
     category = under5
       label = Under 5 years
     category = 5to17
       label = 5 to 17 years
     category = 18to64
       label = 18 to 64 years
       group = adults
       comment = group_label can be printed out just above the
category under which it is listed
```

```
      category = over64
        label = 65 years and over
  DE = tab13
    structure = array
    array_size = 3*4
    alias = racebyage
    label = Race (3) by Age (4)
    TYPE = int
    START = 250
    cell_length = 9
    universe = Persons of Spanish Origin
    dimension = race2
      class = race
      category = total
      category = white
      category = black
    dimension = age2
      comment = age2 is defined above, so this merely refers back
    cells=1,1:4
      suppression = supflg06
      missing = -6
    cells=2,1:4
      suppression = supflg07
      missing = -7
    cells=3,1:
      suppression = supflg08
      missing = -8
  DE = tab15
    structure = array
    TYPE = int
    START = 370
    cell_length = 9
    label = Household Type and Relationship (9)
    comment = array_size will be calculated from dimensions
    universe = Persons
    note = Categories may not be mutually exclusive
    suppression = supflg01
    missing = -1
    dimension = hhtype_rel
      class = household_type_and_relationship
      category = householder
        group = family
      category = spouse
      category = other_relatives
        footnote = 5
      category = nonrelatives
        footnote = 6
        group = nonfamily
      category = male_householder
      category = female_householder
      category = nonrelatives
        footnote = 6
      category = inmate
        label = Inmate of institution
        group = group_quarters
      category = other
      category_group = family
        cgitem = householder, spouse, other_relatives
```

```
      category_group = nonfamily
        cgitem = male_householder
        cgitem = female_householder
        cgitem = nonrelatives
      category_group = group_quarters
        cgitem = inmate
        cgitem = other
END DDF
44001RI BRISTOL                                    358.336     2611
   9541    28925     5865     2566    9427   28587
   5833        4       18       87       8       5
     11       23        0       18      45     109
     18       52      164      459      68      50
    156      428       67       -8      -8      -8
     -8    12385    10681    18029     242    1129
   1914      412      804     1346
44003RI KENT                                        -1.0       9435
  32774    94384    17570     9294   32346   93353
  17465       55      149      359      42       3
     32      100       10       52     178     383
     35       89      284      630      64      82
    256      541       59       -8      -8      -8
     -8    41678    35119    60324    1141    4970
   7489     1809      981      652
44005RI NEWPORT                                    206.556     5336
  16871    49936     9240     4797   15714   47241
   8978      320      731     1770     191      47
     67      156       12       77     194     390
     26      116      299      754      96      78
    210      532       86        6      22      36
      2    20599    16959    29579     692    3255
   4392     1721      602     3584
44007RI PROVIDENCE                                 510.589    33089
 107955   345656    84649    28696   97091  322242
  82430     2365     6238    13181    1272     146
    413      844      150      270     596    2100
    209     1880     4140     9150     765     998
   2202     5226      559      126     175     454
     31   147434   116516   211335    4235   23736
  38528     9191     8799    11575
44009RI WASHINGTON                                 107.756     6221
  19018    58480     9598     6001   18342   56858
   9431       59      197      481      57      73
    267      460       79       36     121     423
     23       75      149      427      46      55
    107      275       42       -8      -8      -8
     -8    23032    19664    32903     913    3633
   4416   3040   1337   4379
```

Implementation Strategies for Proposed Enhancements

While we discuss which new metadata elements and other changes ought to be added to the codata ddl, we also need to decide on a strategy for incorporating such enhancements. If we decide on the second strategy outlined below, we could begin implementation even before we decide on the final set of new metadata elements to be included in the enhanced codata DDL.

As I understand it, the software for handling metadata information (i.e., data definition files), is centralized in CRDEF and CWDEF, which read and write information for a single data element, plus a small amount of code in CRINIT and CWDEF, which do a special call to the former routines to read and write global, file-level information. There are at least three different versions of these programs, separately maintained by Bill Benson, Bob Healey, and Deane Merrill, that will have to be modified and/or replaced.

The choices are basically whether to modify the existing software or to rewrite the programs. Some of the considerations are outlined below.

Alternative #1 – Modify Existing Software

CRDEF and related routines currently pass information as separate parameters in calling sequences. We could follow the current form of these routines, and simply add more parameters to accomodate the new types of metadata. This might have the (dubious) virtue that if we added new parameters at the end of the sequence, calling programs might not have to be changed until they wanted to make use of the new information, and such changes could take place incrementally. But this would involve having different numbers of parameters in calling and called routines -- a dangerous practice which VMS FORTRAN may not even permit. As has been pointed out, moreover, the calling sequences are already too long, and once the current round of changes are incorporated it would be even more difficult to add new metadata elements. We may even run up against a limit on the length of a FORTRAN parameter list.

Alternative #2 – Redesign Calling Sequences

The preferable implementation strategy is to adopt a new type of calling sequence, as proposed by Rik Littlefield and others at PNL. The essence of this scheme would be to pass metadata information in a character array containing "name = value" strings. A common set of routines would be coded to make and crack these strings, so they would be the only things that would have to be changed as new metadata elements were added. This would make the gradual addition of new types of metadata *much* easier. It might even be possible to generate code for the common set of read routines using some kind of compiler compiler.

More detailed discussion of this new type of calling sequence will be presented in another memo (hopefully within the next few weeks).

Recursive Metadata Definition

One elegant way to make the definition of metadata elements in codata files more open-ended and flexible would be to permit definition of metadata elements in terms of a subset of "metadata primitives", arranged in a standard ddf format. In order to do this, we would need to implement variable length and multiply occuring data fields. The eye-readable version could be compiled into a more efficient form, perhaps using a compiler compiler to generate the ddf parser from the metadata data definition file.

If we did this, it would mean that the same subroutine calls used to request particular items of data could be used to request particular items of metadata -- a tidy and pleasing result. I hope to address this possibility in another paper, and perhaps to implement a prototype using SPIRES.

SELECTED REFERENCES

COMP 81 Computer Science and Mathematics Department, "SEEDIS Release Notes version 1.2," June 1981; version 1.3 (preliminary) October 1981

EGGE 81 Eggers, S., Olken, F., and Shoshani, A., "A Compression Technique for Large Statistical Databases," LBL-12353, February, 1981

EGGE 82 S. Eggers, F. Gey, R. Healey, H. Holmes, J. McCarthy, A. Shoshani, "Physical Database Research at the Lawrence Berkeley Laboratory," IEEE *Database Engineering Newsletter*, March, 1982 (forthcoming)

MARC 81 Marcus, A., "SEEDIS Graphic Design Manual" [in preparation]

MCCA 81A McCarthy, J., "The SEEDIS Project: A Summary Overview," PUB-424, September, 1981

MCCA 81B Mccarthy, J., "Metadata Tools," [draft internal document] July, 1981

MERR 81 Merrill, D., "CODATA Users' Manual," LBID-021, revised April 1981

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