Using the Centinel Data Format to Decouple Data Creation from Data Processing in Scientific Programs

Clarence Lehman[1] **and Adrienne Keen**[2]
[1]University of Minnesota, 123 Snyder Hall, Saint Paul, MN 55108, USA
[2]London School of Hygiene and Tropical Medicine, Keppel St., London WC1E 7HT, UK

"Software is hard. It's harder than anything else I've ever had to do."
—Donald Knuth, 2002

Abstract— *Multi-dimensional numerical arrays are a staple of many scientific computer programs, where processing may be intricate but where data structures can be simple. Data for these arrays may be read into the program from text files assembled in advance, often laboriously from multiple sources or from large-scale databases. Notwithstanding simplicity in the structure of such files, their multi-dimensional nature and the very regularity of their data makes it difficult or impossible to know by inspection that they are assembled exactly as required by the processing programs. Moreover, data errors inadvertently may appear through unintended alteration of some parts of a file while other parts intentionally are being edited. Verifying the correctness of scientific programs is hindered by such difficulties. Here we describe how we have applied the Centinel archival data format to such problems. Centinel (1) provides a format that can be read without difficulty by both people and computers, (2) keeps all metadata locally in the same files as the data themselves, and (3) optionally protects the data with error correcting codes on each row, from the time the data are prepared until they are finally processed. In addition, we show how we have used the Centinel format to produce prototypes of large datasets for initial program testing before the actual data have been prepared. This effort is one step in the uncompromising process of ensuring that complex scientific programs rigorously perform the tasks they are intended to do.*

Keywords: data and metadata, code and metacode, scientific programming, software validation, database, data archives

1. Introduction

Consider the following two files made available to a computer program, each containing an 8×8 matrix of hypothetical average temperature measurements at points along a latitude line and at times throughout a season. The program expects distances to be represented in successive matrix rows and time in successive columns. In this example, the temperatures increase north to south (top to bottom in the array) and also increase as the season progresses (left to right in the array).

The two files below are identical, just with rows and columns transposed. Suppose one is correct and the other is not. Once read by the program, the data act as parameters, so that verifying the correctness of the program includes knowing the correctness of this data. The question is, how can one verify by inspection which of the two matrices has the correct format and will generate legitimate results in the program?

Input File 1:

18.9	20.7	22.4	24.9	21.7	24.0	23.1	24.9
21.1	24.0	24.2	21.6	25.0	23.0	23.8	24.2
21.7	20.9	24.8	22.7	26.4	25.0	24.8	28.4
23.9	21.7	22.9	24.1	27.0	25.8	28.3	27.6
23.8	22.5	24.8	26.3	25.1	26.9	29.6	29.1
24.3	27.0	25.3	26.2	26.0	25.9	27.0	29.9
25.1	27.2	26.3	28.8	27.4	28.3	29.7	28.6
27.7	27.5	28.2	25.8	29.5	26.5	27.3	30.6

Input File 2:

18.9	21.1	21.7	23.9	23.8	24.3	25.1	27.7
20.7	24.0	20.9	21.7	22.5	27.0	27.2	27.5
22.4	24.2	24.8	22.9	24.8	25.3	26.3	28.2
24.9	21.6	22.7	24.1	26.3	26.2	28.8	25.8
21.7	25.0	26.4	27.0	25.1	26.0	27.4	29.5
24.0	23.0	25.0	25.8	26.9	25.9	28.3	26.5
23.1	23.8	24.8	28.3	29.6	27.0	29.7	27.3
24.9	24.2	28.4	27.6	29.1	29.9	28.6	30.6

The answer is simple. One cannot. Without digging deeper into the processes that created the data files, one cannot know whether rows in the file represent distance and columns represent time, or vice versa. Nor can one be certain of what temperature units are represented. Celsius is plausible if this is a temperate region, but Fahrenheit is equally plausible if this is the subarctic. How the axes are scaled and other basic information about the data are also missing. In the absence

of such information, data development becomes undesirably coupled with software development.

The problems are ameliorated but not solved with "database connectors"—software to access databases from within processing programs. Careful discipline beyond the basic requirements of the database is needed at every step to guarantee that the encoding of the data is known, data transformations are specified, units are clear, and a variety of other items are documented that can otherwise remain underspecified.

Associated mistakes can be spectacular. An unmanned spacecraft vanished in 1999 after a ten-month interplanetary journey, breaking into pieces and burning in the Martian atmosphere in part because some of the units expected by the program did not match those provided in the data (conflicts between English and metric systems) [1]. "Our inability to recognize and correct this simple error has had major implications," according to then-director of JPL, Edward Stone [2]. Results in other scientific programs may be less spectacular but of equal or greater moment. Simulations informing national programs for vaccination and disease control, for instance, or estimating potential climate change from biophysical parameters, can affect millions of people.

In this paper we illustrate the problem and its solution with basic software we developed to connect data that is stored in the Centinal archival format [3]. This software may be used directly in C programs or transcribed to serve other languages. The principles apply to programs that connect to any database.

2. Methods and results

2.1 Problem details

The two sample matrices above are an idealization of an actual situation we confronted in a large-scale scientific simulation developed by one of us (A.K., mathematical model for tuberculosis in the UK [4]). The first version of the simulation program had a standard input specification, represented below in a C-like programming language. The plausible correctness of the program can be verified by inspection.

```
define N 8
float a[N][N];  //Celsius array, a[g][t].
for (g=0;  g<N; g++)
  for (t=0;  t<N; t++)                                  (1)
    if (scanf("%f",  &a[g][t]) < 1)
      ExitMsg(1);
```

The input (File 1) can also be inspected—eight lines with eight numbers on each—which matches the program above. The doubly nested loop reads each number on a line into the t dimension, then reads subsequent lines into the g dimension of the array a. Inspection of the code shows that the input file cannot overflow the array, and that missing or non-numeric values will be detected and the subroutine *ExitMsg* will be notified to handle them, typically by issuing an error message and terminating the operation.

However, the reason we said *plausible correctness* is that one cannot know by inspection of the data and the code that the order of the loops is correct, nor that the units are indeed Celsius as the program expects. The danger is easy to identify in this basic example, but the dimensionality of arrays in practice commonly grows to five or more and the dangers of undetected errors compound.

2.2 A basic solution

We sought general ways of decoupling the processes of (1) creating the data and verifying the correctness of the created data, and (2) writing the computer program and verifying the correctness of the program's code. Our solution was simple in concept and not difficult to accomplish. We inserted a "decoupling step" between the data and the program, with two components: (1) computer- and human-readable metadata maintained within the file and (2) software that processes not only the data but parts of the metadata as well. Below is an example of File 1 in Centinel format.

Centinel Version of File 1:

```
2976573 Dataset: Seasonal omega-transformed temperatures.
6519832 Description: This is purely a sample dataset constructed
0823811   for illustration. The data are quite imaginary.
3097624 Label a:  Average temperature over time t, location g,
6421009           in degrees Celsius, omega-transformed.
2347567 Label t:  Time, two-week intervals from March 21.
2785463           (0=Mar21–Apr03, 7=Jun27–Jul10)
1127554 Label g:  Geographic location, half-degree quadrangles
5437743           from the 45th parallel north centered on
8620815           the 100th meridian west.
6584390           (0=45.0–45.5°N, 7=49.0–49.5°N)
9307204 |g |a:t=0 |a:t=1 |a:t=2 |a:t=3 |a:t=4 |a:t=5 |a:t=6 |a:t=7
8217764 |7 |18.9 |20.7 |22.4 |24.9 |21.7 |24.0 |23.1 |24.9
6802135 |6 |21.1 |24.0 |24.2 |21.6 |25.0 |23.0 |23.8 |24.2
1493093 |5 |21.7 |20.9 |24.8 |22.7 |26.4 |25.0 |24.8 |28.4
7564407 |4 |23.9 |21.7 |22.9 |24.1 |27.0 |25.8 |28.3 |27.6
4186572 |3 |23.8 |22.5 |24.8 |26.3 |25.1 |26.9 |29.6 |29.1
3622154 |2 |24.3 |27.0 |25.3 |26.2 |26.0 |25.9 |27.0 |29.9
5894658 |1 |25.1 |27.2 |26.3 |28.8 |27.4 |28.3 |29.7 |28.6
9717717 |0 |27.7 |27.5 |28.2 |25.8 |29.5 |26.5 |27.3 |30.6
```

Centinel files are ASCII text with three parts: (1) An optional column of numbers at the far left above, which represent error-correcting codes called "centinels." They guard each line against accidental alterations [3]. If the first character of a Centinel file is not a digit '0' to '9', then the column is not included and the file consists only of data and metadata. (2) Metadata, at the top of the example above and to the right of the column of centinels. Metadata describes the data to people and, in certain cases, to computer programs that may process parts of it. Metadata have "keyword–colon–data" format, with indented lines continuing the line above. The last line of metadata contains headings that define

the contents of each column of data. (3) Data, with data elements separated by vertical bars. In this case a column at the left defines the index for each row.

The column of numbers labeled 'g' defines the geographic location of each data element on the line, as described in the metadata above it. Each of the 64 data elements in the array is identified with its geographic location, in column 'g', and with its time, in the column headings marked 'a:t=0' through 'a:t=7'. Each such column heading contains the value of the label to the left of the colon ('a') indexed by the label to the right of the colon ('t') at the index specified to the right of the equal sign. Thus the value in the upper left corner of the data block is $a[g][t] = a[7][0] = 18.9$, the value immediately to its right is $a[7][1] = 20.7$, and so forth until the value in the lower right corner is $a[0][7] = 30.6$. In this way the file is self-defining and the following call to subroutine *Centinel* is sufficient to read it into the array.

```
define N 8
float a[N][N];  //Celsius array, a[g][t].
char b[] = "a[g=0~7][t=0~7]";                              (2)
if (Centinel(a, b, "omega.txt") ≠ 0) ExitMsg(1);
```

The second line in the code above specifies the array a and its indexes for the compiler, as before. The third line specifies the array and its indexes for the subroutine *Centinel*, which reads the file. Thus the second line says, "The array a has eight rows indexed by label g in the file and eight columns each indexed by label t in the file." The third line calls the subroutine *Centinel* to read the file. Its first parameter specifies the array to receive the data, in this case a, its second parameter defines the structure of the array and names the index values, and its third parameter is the name of the file to be read. Free source-code copies of the software are available from the authors upon request.

2.3 Equivalent transposed format

We have shown a sample matrix and its transposition, which could not be reliably distinguished, then showed how the first form of the matrix could be reliably represented. For completeness, below is the transposed form of the same matrix, which can also be read with the same call to the subroutine *Centinel*. No changes to the program are needed.

3515117	t	a:g=7	a:g=6	a:g=5	a:g=4	a:g=3	a:g=2	a:g=1	a:g=0
7125262	0	18.9	21.1	21.7	23.9	23.8	24.3	25.1	27.7
0961535	1	20.7	24.0	20.9	21.7	22.5	27.0	27.2	27.5
3303666	2	22.4	24.2	24.8	22.9	24.8	25.3	26.3	28.2
9369193	3	24.9	21.6	22.7	24.1	26.3	26.2	28.8	25.8
8881518	4	21.7	25.0	26.4	27.0	25.1	26.0	27.4	29.5
2627646	5	24.0	23.0	25.0	25.8	26.9	25.9	28.3	26.5
2756293	6	23.1	23.8	24.8	28.3	29.6	27.0	29.7	27.3
9655049	7	24.9	24.2	28.4	27.6	29.1	29.9	28.6	30.6

All lines of metadata but the heading line are identical and therefore not shown again here. Notice that the only differences in the remainder are in the labels on the heading line and in the column for 't', and in the centinels. Those are sufficient to allow the software to load the data into the proper locations of the program's array.

2.4 Equivalent relational format

Any format that properly specifies the data will work. In particular, an ordinary relational database format can be used with the subroutine *Centinel*, as depicted below. We have not used this format in our work nor in this explanation, however, since it is much less compact and therefore harder to examine visually.

2393973	g	t	a
9788239	0	0	27.7
3291521	0	1	27.5
0461845	0	2	28.2
8743319	0	3	25.8
2912616	0	4	29.5
5291316	0	5	26.5
6321936	0	6	27.3
1876497	0	7	30.6
4415933	1	0	25.1
2860027	1	1	27.2
	⋮	⋮	⋮
0270284	7	5	24.0
2154910	7	6	23.1
2712213	7	7	24.9

2.5 Over and under specification

The datafile may contain more data than the array contains. Data corresponding to array indexes that are out of bounds are ignored, as defined in the specifier b. An error indicator will be returned if requested. Also, any labels in the file that are not part of the array are ignored. These are "over-specified" files that contain more information than needed. They allow different parts of a single file to be loaded into different arrays, for example.

Files may also be "underspecified," in that they do not contain enough information to fill the array. For example, any of the three files above could be divided into eight separate files, one for each column of the matrix. When each was read, it would fill in only its column of the array. Multiple files may thus be combined into a single array—convenient for some organizations of data. Of course, in all cases care must be taken not to leave parts of the array undefined.

2.6 Prototyping

The datasets we have shown thus far have single integer indexes in each location. In addition, sequences and ranges of integers can be used in each location, for the purpose of prototyping. Often a program will be ready for partial testing before its data are fully available. We included basic prototyping in the *Centinel* algorithm to allow this.

A set of indexes can be a range of integers separated by a tilde, written 'n_1~n_2', where the n_i are integers, or a

sequence, written '$m_1, m_2, m_3, \ldots, m_k$', where the m_i are integers or ranges of integers. Here are some examples:

Specification	*Indexes represented*
1	1
0, 1	0 1
0~1	0 1
0, 3~9, 40~38, 2	0 3 4 5 6 7 8 9 40 39 38 2

The example below is related to an actual dataset we used, where a collection of probabilities, p, is indexed in four dimensions by a region $0 \le region \le 2$, a relative year $0 \le year \le 95$, a state $0 \le q \le 8$, and a class $0 \le c \le 3$. This is an array of $3 \cdot 96 \cdot 9 \cdot 4 = 10{,}368$ elements. When the data became available and completely encoded, each array element had its own distinct probability value, but in the meantime program development needed to continue. A file like the following, with appropriate additional descriptive metadata, sufficed for initial testing.

|region	|year	|q	|p:c=0	|p:c=1	|p:c=2	|p:c=3	
|0~2	|0~95	|0	|0	|0	|0	|0	(line 1)
|0	|0~95	|1~7	|0.80	|0.60	|0.79	|0.89	(line 2)
|1, 2	|0~95	|1~7	|0.84	|0.71	|0.88	|0.99	(line 3)
|0~2	|0~95	|8	|1	|1	|1	|1	(line 4)

When the above file is read, every subarray for $q = 0$ is set to zero (by line 1) and every subarray for $q = 8$ is set to one (by line 4). Of the remaining elements in the array, every subarray for $region = 0$ is set to the vector $0.80, 0.60, 0.79, 0.89$ for $c = 0, 1, 2, 3$ (by line 2), and the remainder is set to the vector $0.84, 0.71, 0.88, 0.99$, for the same values of c (by line 3). Thus the array can be filled initially with appropriate "placeholders." As data are developed, the file can be filled out and the program further tested, until all placeholders are withdrawn and the full 10,368 array entries are individually specified.

2.7 Error correction

The optional error-correcting codes represented by numbers to the left of the lines of data and metadata are "Hamming codes" [5], originally designed for 0–1 bits but redesigned in Centinel for symbols. They allow (1) any single-character error on a line to be corrected, (2) any double-character error to be detected, and (3) the overwhelming majority of multi-character errors also to be detected. The codes are created by the Centinel algorithm [3] or by a text editor that supports the Centinal algorithm.

As mentioned earlier, they guard against accidental modification of one piece of data while editing another. They also make printed copies of the data into reliable long-term storage media for archiving the data. Printed copies of the data can be scanned and verified long into the future, with no intervening migration or maintenance of the data necessary [3].

3. Discussion

3.1 Correctness of scientific programs

Writing software that works is one of the most difficult of human endeavors, and scientific software is at a special disadvantage. Whereas commercial and engineering software can be very complex, its desired behavior can be specified in advance. For example, if a spreadsheet operation is intended to produce the sum of a column of numbers, it is possible to determine whether it is actually doing so. That is, testing is possible. In scientific software, however, testing is often impossible. The program's behavior is often not known because the behavior of the natural system being simulated is not known. Indeed, the whole purpose of the simulation program is to determine how the system behaves.

One aspect among several is "correctness proving." [6] [7] [8] This topic has been well discussed but less well practiced. An essential part is partitioning the software into manageable pieces and documenting each piece so that its correctness can be verified. The ideas discussed in this paper are part of that process—because data read by the program as parameters become part of the program, the program's correctness in turn depends on the data's correctness. Thus the data must also be partitioned into manageable pieces and documented.

The goal is to restrict the range of attention to what can be understood by the human mind in one review session. In software, this can be accomplished by adding "metacode" to the code, describing, among other things, full entry and exit conditions for every module, no matter how small. For data, it can be accomplished by partitioning the data and encapsulating each partition with metadata, as described here.

3.2 Centinel and other forms

This approach can be applied to any database and any programming language that can connect with that database. However, methods such as we have described for partitioning the data into manageable pieces, for documenting it, and optionally for guarding it against unintended alteration, are important with any database. Column names and row names are not required by common spreadsheet software, and spreadsheets for important data are sometimes prepared with little more information than in the sample matrix files shown at the beginning of this paper.

Centinel files may be constructed directly with a text editor. More commonly they are assembled by collections of programs and scripting languages, from databases or from spreadsheets. When created from spreadsheets, column 1 of the spreadsheet can be used solely for metadata, with all actual data beginning in column 2. Then when the files are saved, for example as tab-separated text files, and after the tabs are translated to vertical bars, each actual data line will begin with a vertical bar. Centinel formats can thus be

transferred back and forth to spreadsheet programs without loss of data in either direction.

3.3 Database labels

It is useful to label data elements in the file so that they exactly match corresponding variable names in the program. Doing so means restricting labels to letters and digits, beginning with a letter, and possibly supplemented with optional characters such as underscores. That way no confusion will arise between variables in the program and labels in the database. There is a tendency to try to include metadata in the names of data elements in the database, especially with spreadsheets. For example, a spreadsheet column might be named "%cover-no litter". This is inadvisable for several reasons: (1) even a moderately large amount of metadata in the label is still insufficient to understand what the field really contains; (2) the label will need special characters such as period, hyphen, percent sign, and blank, which have special meanings in most programming languages; and (3) the long label induces a wide column, or alternatively forces part of the label to be hidden.

The approach we use and recommend here is to make a small distinct label, such as in this case "pcover", with metadata like, "Label pcover: Percentage of the area covered by the species in question, when viewed from directly above, relative to the area occupied by living plants (the area not occupied by leaf litter or bare soil)." Not much less than that amount of metadata is necessary for someone familiar with the data to understand what that data element represents, and that amount is too long for a label. Therefore, the better strategy is to use short data labels with ample metadata descriptions carried separately in the file.

3.4 Database metadata

In popular database management systems such as MySQL [9], metadata of the type we advocate can be added, though often not in the same file as the data. At the time of this writing, metadata elements that can be stored as comments in MySQL files are limited to one line of text each and thus are difficult to use for complete metadata. It is always possible to set up special tables to contain the metadata, but that presents other difficulties, for it is harder to maintain metadata when it is in a separate file.

We feel it is important to specify the metadata while creating the data. That is, after all, when the structure and meaning of the data are known. Writing it down then is only incremental time; writing it down later is re-creating a thought process that has already been completed once. Fine details of data and code evaporate from the mind with disappointing ease. Data structures should be defined as carefully as possible beforehand, although achieving good data structures, like good computer programs, can be an iterative process. The best practice is that documentation of the data be maintained at each step of the iteration.

3.5 Database connections versus files

Even when working with a large-scale database that can connect to the program, there is merit in creating files of the Centinel type for communication with the program. Those files fully document the data that will lead to conclusions drawn from the program, and can be used in supplemental material submitted with any publications that result. Recorded in Centinel format, they will be ready for long-term archiving, along with the scientific publication itself. (See example in Appendix.)

Too often, when such files are not created, subsequent changes in the dataset used to draw the conclusions will make it difficult for anyone, including the original authors, to replicate precisely the results. This can make it impossible to precisely compare former conclusions with new conclusions that may arise as conditions change, and may occasionally call into question the original results.

Many other considerations in constructing databases and documenting them lie beyond the scope of our purposes here, but appear in other publications [10] [11] [12].

3.6 Non-relational data

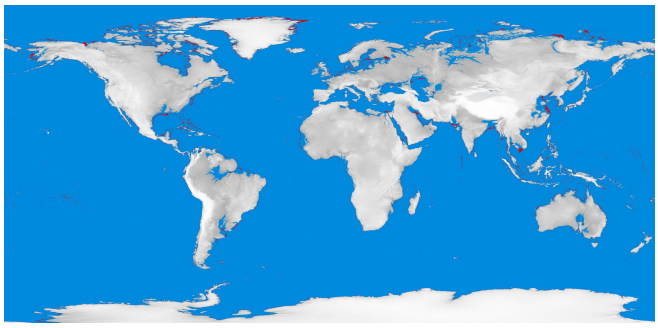
Up to this point we have emphasized ordinary scientific data as stored in relational databases, but any kind of data can be represented in the form we have described. That form allows the data to be written and read directly by simple computer programs and to take forms that adapt to various requirements. As an example, the Centinel format has been applied to large-scale photographic radar images, which can have 10^5 or more levels per spectral band and more than three spectral bands, and exceed the limits of simple image formats such as JPEG and PNG. An image can be represented as a rectangular array of colors, with each color being a set of numeric values. Below is an excerpt from a large array of satellite radar elevation measurements from public NASA databases.

Title:	Earth at maximal ice melt
Contents:	Pixel array, 4320 x 2160
Spectral bands:	3
Bits per band:	32
Wavelengths:	RGB standard
Resolution:	1/12 degree, latitude and longitude
Data source:	NASA STMR30 database
Produced by:	flood.c
Label Lat:	Latitude band in 1/12 degree resolution. Elevation in meters, Lat0=90N, Lat2160=90S.
Label Lon:	Longitude band in 1/12 degree resolution. Elevation in meters, Lon0=180W, Lon2160=0, Lon4320=180E

Lat	Lon0	Lon1	Lon2	Lon3	Lon4	Lon5	..
0	0,10,880	0,10,880	0,10,880	0,10,880	0,10,880	0,10,880	..
1	0,10,880	0,10,880	0,10,880	0,10,880	0,10,880	0,10,880	..
539	0,10,880	0,10,880	65.66	67.93	85.21	129.23	..
540	0,10,880	0,10,880	76.52	100.53	109.73	129.9	..
:							
2159	2605.37	2605.95	2606.52	2607.23	2607.79	2608.44	..

Ellipsis symbols in the example above (':' and '..') represent material in the file that is not shown here for brevity. Each data element consists of a string of comma-separated numbers following a vertical bar, each number specifying a spectral band. If there are fewer numbers than spectral bands, the last number is taken to be repeated. Thus single numbers represent monochromatic pixels. In this example colors were only used to represent blue water and red coastlines, the remainder representing elevations in meters as monochrome intensities.

The full file is approximately 80 MB uncompressed. Converted to a pixel image, it appears as the following map. As a point of interest, the data represent the results of a flood-fill algorithm estimating coastlines of the planet if all the glacial ice were distributed as water to the oceans.



The point of the example above is that data of many kinds can be treated by the methods we describe in this paper, beyond data that are usually considered relational database material.

4. Conclusions

By applying methods of judiciously organizing data and metadata, the processes of data development and software development can be separated. The consequence is data that are better defined, programs that are more often correct, and results that are replicable. Based on the problems and solutions discussed in this paper, we make the following suggestions and recommendations.

A) Use metadata to disentangle data construction from data usage, including data input to scientific programs.
B) Maintain data formats that people can read with ease and computers can access with simple algorithms.
C) Develop metadata concurrently with data collection.
D) Store metadata in the same files as the data themselves.
E) Use data prototyping to test programs before all data are available.
F) Resist the temptation to embed metadata within data labels. Keep labels simple.
G) Maintain archival copies as snapshots of evolving data—especially data used in reaching published scientific conclusions.
H) Include error-correcting codes in archival data to assure integrity independent of changing storage media.

This method has been practical and useful in reducing or eliminating data errors in large-scale simulations [4] and we recommend it for use and extension by others. Code for the functions described here and for related query and maintenance operations on the Centinel format is available free in compilable source files from the authors upon request.

5. Acknowledgements

We thank Eric Lind and Todd Lehman for helpful discussions and comments. The project was supported in part by a resident fellowship grant to C. Lehman from the UMN Institute on the Environment, by grants of computer time from the Minnesota Supercomputer Institute, and by doctoral research funding to A. Keen from the Modelling and Economics Unit at the Health Protection Agency, London.

6. Contributions

A. Keen wrote the simulation programs that inspired the present paper and prepared corresponding simulation data in the format explained here. C. Lehman coded the software for reading Centinel data into scientific programs. Both authors contributed to the development of the Centinel data format and the manuscript.

References

[1] R. A. Kerr, "More than missing metric doomed orbiter," *Science*, p. 207, 1999.
[2] D. Isbell, M. Hardin, and J. Underwood, "Mars climate orbiter team finds likely cause of loss," *JPL–NASA report, Release 99-113*, 1999.
[3] C. Lehman, S. Williams, and A. Keen, "The Centinel data format: Reliably communicating through time and place," *International Conference on Information and Knowledge Engineering, Proceedings*, vol. IKE 12, pp. 47–53, 2012.
[4] A. Keen, "Understanding tuberculosis dynamics in the United Kingdom using mathematical modelling," *Doctoral Thesis, London School of Hygiene and Tropical Medicine, University of London, 488 pp.*, 2013.
[5] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 26, pp. 147–160, 1950.
[6] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, pp. 576–585, 1969.
[7] E. W. Dijkstra, "A discipline of programming," *Prentice-Hall Series in Automatic Computation*, 1976.
[8] D. Jackson, "Alloy: A lightweight object modelling notation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, pp. 256–290, 2002.
[9] B. Schwartz, P. Zaitsev, and V. Tkachenko, "High performance MySQL: Optimization, backups, and replication," *3rd Ed., O'Reilly Media, 828 pp.*, 2012.
[10] P. A. Sharp, D. Kleppner, and committee, "Ensuring the integrity, accessibility, and stewardship of research data in the digital age," *National Academies Press, 180 pp.*, 2009.
[11] E. T. Borer, E. W. Seabloom, M. B. Jones, and M. Schildhauer, "Some simple guidelines for effective data management," *Bulletin of the Ecological Society of America*, vol. 90, pp. 205–214, 2009.
[12] D. Butler, "The future of electronic scientific literature," *Nature*, vol. 413, pp. 1–3, 2001.

7. Appendix

Below is a sample excerpt of a file in Centinel format, used as input to scientific analyses and showing the style of metadata and data specification. At the left of each line are the optional "centinels," error detecting and correcting codes that accompany the file as it is transferred across media, supplementing any such codes that may be part of specific computer media. Thus even printed copies of the file that may be retained indefinitely into the future can be subsequently scanned and the data recovered with the full reliability of any computer medium. Lines beginning with a vertical bar to the right of the centinel codes are data, in columnar format. Other lines are metadata, describing the data sufficiently well to be understood by a worker in the field who may be accessing the data from a remote place or time. Metadata have "keyword–colon–data" format, with indented lines continuing the line above. Keywords are chosen to fit the data and the needs of processing programs. For example, "Label" is used by query and other database management programs that process the Centinel format. An automatic summary line at the end guards against missing or duplicate lines.

```
  3255845646594753  Dataset:   Peatland dates and depths
  8969865226586934  By:        Art Dyke, Eville Gorham, Jan Janssens
  8286898747137843  Date:      September 15, 2012
  0000000000000000
  1314776168875326  Contents: Age, depth, and location data for North American
  8620213562356287    peatlands. Please consult the publication below for
  0901842416681217    details.
  0000000000000000
  5827730880685764  Publication: This is the archival dataset for "Long-Term
  1520774603075243    Carbon Sequestration in North American Peatlands," Gorham,
  7259216113317888    Lehman, Dyke, Clymo, and Janssens, Quaternary Science
  7505773863167388    Reviews, 2012, doi 10.1016/j.qsciref.2012.09.018.
  0000000000000000
  1884884179373648  Format: This file is recorded in Centinel format, which is
  1043670634366401    for immediate use and long term archiving. The numbers at
  6812166714427858    the left are error-correcting and error-detecting codes to
  4037101440275922    help ensure that inadvertent alterations of the file will
  7313760841625847    not go undetected. See Lehman, Williams, and Keen (2012),
  4508626531434354    "The Centinel Data Format: Reliably Communicating through
  6206860589148901    Time and Place," International Conference on Information
  1864531348064250    and Knowledge Engineering, IKE 12:47-53, Proceedings.
  0000000000000000
  7770288188098974  Label ID:     Unique identifier for the sample.
  0000000000000000
  7740581166254016  Label Lat:    Latitude, degrees north of the equator.
  6908751974650935                Negative is south latitude.
  0000000000000000
  6204302802774740  Label Lon:    Longitude, degrees east of the prime meridian.
  4064257255528647                Negative is west longitude.
  0000000000000000
  6239741042826543  Label Depth:  Depth of the peatland in centimeters.
  0000000000000000
  4325213022514926  Label CalBP:  Date of peatland initiation, calendar years
  2283760574804679                before present, reckoned as 1950. Calculated
  5705568955445847                using 2004 international calibration methods.
  0000000000000000
  4024078842187041  Label Line:   Serial line number.
  0000000000000000
  5384013588094368  |ID           |Lat      |Lon        |Depth   |CalBP   |Line
  8963026933143738  |A-1112       |41.5     |-113.5     | 707.5  |14367   |    1
  2355144908347452  |A-2143       |63.33    |-149.      |    .   |14046   |    2
  3055234629388956  |A-2147       |63.33    |-149.      |    .   | 6643   |    3
  4680358064467548  |A-2163       |63.33    |-152.      |  30.   | 1840   |    4
  6020894830527075  |A-219        |42.2     | -88.6     | 175.   |13713   |    5
  5124434961658461  |A-9338       |55.15    |-162.95    |    .   |10491   |    6
  4450355574122328  |AA-10925     |42.667   | -70.883   | 179.   |13760   |    7
  2609664440071317  |AA-20755     |68.02    |-158.73    |    .   |11903   |    8
  8704316082120318  |AA-20756     |68.02    |-158.73    | 300.   |10996   |    9
-2047 LINES OMITTED-|  :          |  :      |    :      |    :   |    :   |    :
  3187374887671188  |Y-2464       |45.08    | -71.08    |    .   |11618   |2056
  0513875673303111  |Y-416        |49.62    | -99.43    |    .   | 8878   |2057
  4553882162706867  |Y-418        |51.17    |-100.25    |  10.   | 1316   |2058
  6580606639747581  |Y-526        |40.025   | -82.975   |    .   |13351   |2059
  1002595983626604  |Y-527        |54.8     | -60.82    | 115.   | 4300   |2060
  4600813124741886  |Y-762        |46.02    | -61.565   |    .   |12618   |2061
  0000000000000000
  4314701862316530  Summary: 2061 data lines, 41 metadata lines, Centinel V2.
```