ZINC - Galvanizing CIF to Work with UNIX

David R. Stampf
Protein Data Bank
Brookhaven National Laboratory
Upton, NY 11973

Introduction

Two of the stated goals of STAR[1] (and CIF[2]) were to define a self-defining data archiving format that would also be easy for a person to read and modify. This was designed at a time when many data formats where firmly rooted in their Fortran past with fixed format, column specific card images. In a number of ways, CIF was successful in making an easier to read format for people, but it still has a number of limitations both for human and machine accessibility. These include:

- CIF both allows and encourages (via an 80 column line length and the manner of representing tables) the placement of the data value a distance away from the data name. Thus even if you can locate the name of an object, its value need not be adjacent.

- The order of the data name-value pairs within a CIF is immaterial making it difficult to find data items or to compare CIFs.

- Table (loop) columns may be ordered differently from one CIF to another also making them difficult to compare and access.

- Lines have a maximum line length of 80 characters (making people who create any CIF count characters and face the prospect of the file being made invalid simply by passing it through a mailer or by cutting and pasting and having a tab expanded to an unspecified number of spaces)

- names have a limited length requiring some abbreviations and shortcuts in naming data items (as well as making people repeatedly count to 32 when creating a new dictionary)

All of these factors work against the human accessibility of the file. More serious however, is that the goal of human accessibility came at a time when it was becoming more important to have a computer accessible file due to the increasing volume of data that has to be represented (and searched) and the varying needs of users to see the data with differing views. So far, this has been addressed in a number of ways by creating program libraries that permit reasonable access to the contents of a CIF from Fortran, C, and C++ while still preserving the human accessibility of CIF. However, none of these by themselves helps provide access to CIFs from the higher languages and utilities found on most UNIX systems.

In this article, a data format called ZINC is described that is both isomorphic to CIF

---

[1]S.R.Hall, The STAR File: A New Format for Electronic Data Transfer and Archiving. *J Chem. Inf. Compt. Sci.* **31**, 326-333 (1992)
[2]S.R.Hall, F.H.Allen and I.D.Brown, The Crystallographic Information File (CIF): A New Standard Archive File for Crystallography. *Acta Cryst.* **A47**, 65 (1991)

(allowing the use of *all* existing CIFs, dictionaries and CIF tools) and is much easier for standard UNIX tools to access. In addition, filters that convert a CIF to a ZINC and a ZINC to a CIF are introduced, making ZINC ideal for pipelined as opposed to archival applications. By making a format that is more easily accessible to UNIX tools, not only can many of the above issues be adequately addressed, but the pool of programmers who can design new tools is dramatically increased. Furthermore it allows the CIF community to leverage an incredible assortment of tools that UNIX provides.

Requirements of UNIX Tools *vis-a-vis* CIF

Most UNIX tools (e.g., awk, grep, diff, perl) have the concept of the line (a string of bytes terminated by a newline) as an basic unit. Therefore, any transformation of a CIF should try to organize related information within lines. Also, many UNIX tools (e.g., perl, awk, sort, cut) also have the concept of lines being divided into fields separated by some specified character (typically ":" or white space) rather than fixed columns. Any new format would have to choose a reasonable field separator.

In CIF, data is often placed on a line separate from the data name — virtually always in the case of loops or long strings. This is partially to improve readability, but also to be within the 80 column limit. Therefore, any transformation of a CIF should allow for very long lines so that the entirety of a data value can be associated with the name of that object. In earlier versions of UNIX, the line length was often a limiting factor, but more recent versions of the standard utilities (e.g., see code developed by the GNU project on prep.ai.mit.edu) have addressed this issue.

In CIF, data items are permitted to span several lines. Any transformation of a CIF that tries to place all of the data value on a line with the data name, should have a way of representing a new line within the data item.

Finally, the contents of a CIF are described by a dictionary (which unfortunately is not necessarily referenced explicitly within the CIF!) and CIFs should be measured against this dictionary before being transformed. If we are to think in terms of a transformed CIF, then we can limit the domain of the CIFs that are handled to those that are, in fact, valid — essentially what is assumed in every case where data is passed to any other program.

ZINC

To merge requirements of UNIX tools with the realities of CIF format, the ZINC format has been created. Zinc is not an interchange format as CIF is, but rather a piping format, i.e., a format that makes the contents of a CIF accessible to UNIX utilities. Each data line of a ZINC file consists of 5 tab separated fields (this may be increased in the future to allow for referencing within the file) as shown:

```
block	name	index	value	loop-id
```

The first field is the name of the CIF data block (the data_ prefix is omitted) and is repeated on each line where appropriate. The second field is the name of the data item. The third field is an index specifier which is empty for non-looped data, and a zero based index for looped data and comments. The fourth field is the data item itself. For multiple line CIF data, newlines are replaced by the two characters "\n". The backslash character becomes the escape character throughout the ZINC format. The fifth field is a loop identifier, currently the name within the loop that sorts earliest, and is used to keep loops together when reformatting a ZINC. Comments that appear in a CIF are associated with the previous token and are also represented in the ZINC format.

For example, the following CIF illustrates most aspects involved in translating to a ZINC:

```
#
#     A simple CIF
#

data_object

#
#     polygon
#
  _name
;
triangle
;
  loop_
    _x _y
     0.0     0.0
     1.0     0.0
     0.0     1.0

     _num_sides 3
```

In ZINC, this would appear as:

```
          (                   #
          (                   #     A simple CIF
          (                   #
object    (                   #
object    (                   #     polygon
object    (                   #
object    _name               ;\ntriangle\n;
object    _x         0        0.0                 _x
object    _y         0        0.0                 _x
object    _x         1        1.0                 _x
object    _y         1        0.0                 _x
object    _x         2        0.0                 _x
object    _y         2        1.0                 _x
object    _num_sides          3
```

Note that comments "belong" to a data block and are represented with an open parenthesis for the data name. Initially, the data block name is defined to be the null string.

Any CIF can be converted into an equivalent ZINC, and the mapping back to CIF will preserve the information content of the file, but not necessarily the detailed placement of each token. (Comments are problematical. The decision has been made to associate the comments with the current data block.)

In order for ZINC to be useful, it is important that conversion between a CIF and a ZINC be fast. Therefore, it is assumed that in converting from a CIF to a ZINC, that the CIF was correct with respect to a dictionary, the DDL and the CIF standard. That is, it is left to other tools (compiler based tools) to validate a CIF, but once a CIF is validated, then it is assumed to remain correct. Testing has shown that the conversion of a CIF to a ZINC for a wide range of dictionaries and data files takes under 5 seconds on a low end workstation.

Existing Tools

A number of tools have been developed to support the user community in using the ZINC format and to access the information contained in a CIF. Most are simple and allow the users to modify the code to tackle new problems.

- **cifZinc**, takes a CIF name as a command line argument or the CIF itself from the standard input and produces a ZINC formatted file on the standard output. It has one option, "-c" that removes comments (which arguably have no place in a CIF).

- **zincCif** a perl script that takes a ZINC formatted file (again from the standard input or as a name in the command line) and pretty prints a the corresponding CIF to the standard output. Often, the pipeline "`cifZinc a.cif | zincCif > b.cif`" produces a more attractive CIF than the original.

- **zincGrep** is a shell script that is the utility most requested by those seeing CIF for the first time and allows a regular expression search of a ZINC (or a CIF specified on the command line which is converted to a ZINC first), and reports the block name, data name, index and value. For example, "zincGrep _name simple.cif" would produce

  ```
  object          _name           ;\ntriangle\n;
  ```

- **cifdiff** is a c-shell script that takes two CIFs and determines the difference between them. It is interesting to examine this script to see how well ZINC fits in with UNIX tools. The content of the script is:

  ```
  cifZinc $1 | sort -t\   +0 -1 +4 +1 -2 | \
                     gawk -F     '{print $1, $2, $3, $4 } \
                     > /tmp/$1.zinc
  cifZinc $2 | sort -t\   +0 -1 +4 +1 -2 | \
                     gawk -F     '{print $1, $2, $3, $4 } \
                     > /tmp/$2.zinc

  diff /tmp/$1.zinc /tmp/$2.zinc
  rm /tmp/$1.zinc /tmp/$2.zinc
  ```

  This script takes each CIF, converts it to a ZINC, then sorts it, first based on the data block name, then (keeping the loops together) on the data name. It then removes the last field (which is not part of the CIF), and stores them in temporary files. It then runs diff against these files. This is remarkably effective both in finding any differences and in providing the context (it names the block and data-name as well as the value) needed to understand the differences. It handles CIFs that have be re-ordered as well as individual loops that may have been rearranged. The script finally cleans up after itself.

- **zb**, a small (< 200 lines) tcl/tk program that provides a simple GUI front end to a ZINC or CIF allowing the user to browse through the contents. Multiple files can be viewed simultaneously as can multiple data blocks on any X terminal. zb recognizes command line argument file names in the form *.cif as being a CIF and converts it to a ZINC automatically.

- **zincNl**, a perl script, takes a ZINC file and creates a FORTRAN compatible Namelist file allowing for easy access to any CIF by FORTRAN programs without the need for extensive I/O libraries or reprogramming. As zb above, it will automatically convert a CIF to a ZINC if it needs to.

- **zincSubset** is another C-Shell script that is very short but equally useful. It allows a user to generate a custom subset of any ZINC (CIF), simply by listing the data blocks and data names that she wishes to include. The script has two file arguments, the first of which specifies a file with regular expressions that specify what is to be included in the subset and the second of which is the ZINC file itself (or the standard input). It also allows two options, "-c" to remove comments and "-v" to invert the sense of the search. The business part of the script looks like:

  ```
  cifZinc $comment $2 | egrep -f $v $1 | zincCif
  ```

  where $comment and $v carry the optional choices mentioned earlier, $2 is the CIF and $1 is a file specifying the subset of data items required. It converts the CIF into a ZINC, greps through the ZINC for patterns that appear in a file, and pretty prints the resulting file. For example the command "`zincSubset defs mmcif94 > mmcif94.def`" will produce a subset of the mmcif dictionary that contains only the names and definitions given that the file named "defs" contains two lines with the tab surrounded word "name" and the tab surrounded word "definition".

All of these tools also operate in concert with each other providing the opportunity of generating increasingly complex tools. For example, to generate the FORTRAN namelist input file with only certain data items, a pipeline of zincSubset and zincNl will suffice. As more tools are developed, the range of applications will increase many-fold.

Advantages

At the outset, five features of CIF were identified which make the file difficult for both people and UNIX tools to deal with effectively. These are made much more manageable by ZINC.

- The data value and name are always adjacent allowing for the use of tools such as grep.
- Sorting a ZINC properly allows one to make diffs on two files.
- Tables and columns can be sorted by the column names making it easier for field based programs (such as gawk) to be able to interchange data.
- cifZinc is forgiving of line length problems, while zincCif is strict. Therefore, it permits one to clean up a CIF simply by ZINCing it then pretty-printing it.
- agrep may prove to be useful in searching ZINCs if the exact spelling of a data name is unknown and actually return very useful information.

Drawbacks

The ZINC format has some drawbacks. ZINC files are larger than CIFs and the long lines of ZINC make it inappropriate to use with most text editors and mailers — it is not for direct human access, and is probably better thought of as being part of the transfer mechanism of a pipeline where it is never stored to a disk. The ZINC version of the mmCIF dictionary is 15% larger than the original while the ZINC version of a typical PDB entry is 500% larger. This is balanced somewhat in that cifZinc and zincCif are extremely fast programs allowing the rapid conversion between formats. The conversion of the mmCIF dictionary from CIF to ZINC takes under 3 seconds, while the reverse conversion takes about 5 seconds.

Code

The formal definition of ZINC and the above mentioned programs are available on the BNL PDB file server, accessible via ftp, gopher or WWW. Others are invited to submit their favorite scripts that use ZINC.

Acknowledgments

Thanks to M. Levitt who let me know *exactly* what he was going to miss when the PDB is converted to CIF and J. Sussman who made me focus on what the difficulties with raw CIF really are.