International Tables for Crystallography (2006). Vol. G, Section 5.3.7.1, pp. 518–519.

5.3.7.1. ZINC: an interface to CIF for standard Unix tools

Unix and derivative operating systems provide a convenient environment for rapid prototyping of programs acting on textual data. The convenience arises from the facility to chain applications together in a 'pipeline', where the output from an application may be passed directly to the input channel of another application without needing intermediate disk files for storage; and from the very rich set of utilities supplied with the typical Unix command shell, which permit files to be concatenated, split, compared, searched, stream-edited and otherwise transformed.

Many users are familiar with these utilities and can rapidly develop prototype or short-lived applications of great power by chaining them together as required. There is a temptation to use such techniques to manipulate CIFs, which as ASCII files are well suited to this. However, there are some features of the CIF syntax that are at variance with the conventional Unix idiom of storing data tags and their associated values within a basic unit of a *line* (*i.e.* a sequence of characters terminated by an end-of-line character code). Although CIFs are built from ASCII-character-populated lines, data values may be placed on a different line from that containing the parent data name (this is almost always the case for looped lists).

5.3.7.1.1. *Description of the ZINC format*

The ZINC format (Stampf, 1994) was developed to transform CIF data into an isomorphous format suitable for manipulation by standard Unix utilities.

The manner in which Unix textual utilities work suggests that an appropriate working format is one in which every individual CIF data value is available within a single-line textual record. The record must also contain information about the context of the value, conveyed through: the name of the data block in which the data value occurs; its associated data name (from which the meaning of the data value is inferred); and for recurrent data (*i.e.* values in a looped list) an indication of the list in which the data value occurs and a counter of its current occurrence in that list. In practice, each record is structured as a data line containing five TAB-separated fields in the order

blockcode   name   index   value   list-id

where *blockcode* is the name of the CIF data block (the leading `data_` string is omitted); *name* is the data name; *index* is a zero-based index of the number of occurrences of the data name within a loop (with a null value if the data occur outside a loop); *value* is the data value itself; text strings extending over several lines are collapsed into a single line with the replacement of the end-of-line character by the sequence `\n`; and *list-id* is a list identifier, stored as the data name within the list that sorts earliest (because it is a purely syntactic transformation, the utility does not consult a dictionary file for the correct `_list_reference` identifying token).

Comments in the CIF are also stored, to permit regeneration of the original file by an inverse transformation; and because it is often convenient to read such interpolations, especially in interactive activities with CIFs of which the user has no prior knowledge. Comments are stored with a value of '`(`' in the data-name field. They are also numbered (starting from zero) in the index field of the ZINC record.

Most details of the ZINC transformation are illustrated by the simple example in CIF format shown in Fig. 5.3.7.1(*a*). This file transformed to ZINC format would appear on a display terminal as illustrated in Fig. 5.3.7.1(*b*). However, the spacing is deceptive, since typical display terminals convert TAB characters to a variable number of spaces. A more accurate (albeit less legible) representation of the output is given in Fig. 5.3.7.1(*c*).

```
# A simple CIF

data_object

# description of a simple
# polygon
  _name
;
triangle
;
  loop_
      _x _y
      0.0      0.0
      1.0      0.0
      0.0      1.0

      _num_sides 3
```

(*a*)

```
      (       0       # A simple CIF
object (       1       # description of a simple
object (       2       # polygon
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
```

(*b*)

```
␣(␣0␣# A simple CIF␣
object␣(␣1␣# description of a simple␣
object␣(␣2␣# polygon␣
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␣
```

(*c*)

Fig. 5.3.7.1. ZINC transformation of CIF. (*a*) is a sample file in CIF format. (*b*) shows the output to a display terminal when this file is transformed by *cifZinc*. (*c*) The same output as (*b*), but with TAB characters represented by special graphical characters.

Note the following points: the data-block name is initially null; each comment line is numbered in sequence from zero; the index field is null for data names that are not within looped lists.

5.3.7.1.2. *ZINC-based utilities*

The purpose of ZINC is to form an intermediate stage in pipeline processes involving Unix tools, and therefore CIF data in ZINC format have only a transitory existence. The ZINC distribution package provides the complementary tools *cifZinc* and *zincCif* required to interconvert formats; and in addition a few sample applications are provided as examples for Unix programmers.

5.3.7.1.2.1. *cifZinc*

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

5.3.7.1.2.2. *zincCif*

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

5.3.7.1.2.3. *zincGrep*

The shell script *zincGrep* is the utility most requested by those seeing a CIF for the first time. It allows a regular-expression search of a ZINC-format file (or a CIF specified on the command line, which is converted to a ZINC-format file first) and reports the block name, data name, index and value. For example, if the file describing a triangle in the last section were called simple.cif, the command `zincGrep _name simple.cif` would produce

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

5.3.7.1.2.4. *cifdiff*

cifdiff is a C-shell script that takes two CIFs and lists the differences between them. Unlike the standard Unix utility *diff*, which compares files line-by-line, *cifdiff* can determine differences that are independent of reordering and white-space padding.

This script takes each CIF, converts it to a ZINC-format file, 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 the remainder in temporary files. It then runs the standard *diff* program against these reordered temporary 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.

Fig. 5.3.7.2 illustrates two CIFs that are very different in the presentation of their contents, but have only a small difference of substance in their content. Fig. 5.3.7.3 indicates the output from *cifdiff* that identifies the changed data.

```
data_sample
     _title  ’scatter graph’
     _description
; A collection of x, y coordinates of points
  drawn in specified colours
;
     loop_
         _x  _y  _colour
          0   0   red
          1   1   red
          2   4   red
          3   9   orange
          4   16  orange
          5   25  orange
      _status complete
```

(*a*)

```
data_sample
_status complete
loop_    _y  _x  _colour
          0   0   red
          1   1   red
          2   2   red
          9   3   orange
          16  4   orange
          25  5   orange
_title  ’scatter graph’ _description
; A collection of x, y coordinates of points
  drawn in specified colours
;
```

(*b*)

Fig. 5.3.7.2. Two example files in CIF format differing greatly in layout but little in content: (*a*) sample1.cif, (*b*) sample2.cif.

```
% cifdiff sample1.cif sample2.cif
18c18
< sample        _y      2       4
---
> sample        _y      2       2
```

Fig. 5.3.7.3. Output from *cifdiff* comparing files sample1.cif and sample2.cif of Fig. 5.3.7.2. The entries in each line comprise the data-block name, the variable name, the zero-based index of the occurrence of the value in a looped list and the value. Hence it is the *third* value of **`_y`** in the loop that has changed.

5.3.7.1.2.5. *zb*

zb is a small (less than 200 lines) Tcl/Tk program (Ousterhout, 1994) that provides a simple graphical front end to a ZINC-format file 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 in CIF format and converts them to ZINC format automatically.

5.3.7.1.2.6. *zincNl*

zincNl is a Perl script that 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 with *zb* above, it will automatically convert a CIF to a ZINC-format file if it needs to.

For a more substantial tool providing CIF input functions in Fortran and C, see the discussion below of *CifSieve* (Section 5.3.7.2).

5.3.7.1.2.7. *zincSubset*

zincSubset is another C-shell script which is very short but very useful. It allows a user to generate a custom subset of any ZINC-format file (or CIF) simply by listing the desired data blocks and data names. 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-format file itself (or standard input). It allows two options: ‘*-c*’ to remove comments and ‘*-v*’ to invert the sense of the search.

It converts the CIF into a ZINC-format file, uses the Unix *grep* program to search through the ZINC-format file for patterns that appear in the regular-expression file and pretty prints the result. For example, the command

```
zincSubset defs cif_core.dic > cifdic.defs
```

will produce a subset of the core CIF dictionary that contains only the names and definitions when the file named ‘defs’ contains two lines with the TAB-surrounded word ‘`_name`’ and the TAB-surrounded word ‘`_definition`’.

All the tools listed in this section operate by design in concert with each other, providing the opportunity for 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.