

5. APPLICATIONS

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

```
# 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_\n;\ntriangle\n;\n
object_x_0_0_0_0_x
object_y_0_0_0_0_x
object_x_1_1_0_0_x
object_y_1_1_0_0_x
object_x_2_0_0_0_x
object_y_2_1_0_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.

variable number of spaces. A more accurate (albeit less legible) representation of the output is given in Fig. 5.3.7.1(c).

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).