Unless the application designer can be certain that externally produced CIFs will never be presented to the application, or will be filtered through a reordering filter such as *QUASAR* or *cif2cif*, working with CIFs in an order-dependent mode is a mistake.

Because of the importance of being able to accept CIFs written by any other application, which may have written its data in a totally different order than is expected, it is a good idea to make use of one of the existing libraries or APIs if possible, unless there is some pressing need to do things differently.

If a fresh design is needed, *e.g.* to achieve maximal performance in a time-critical application, it will be necessary to create a CIF parser to translate CIF documents into information in the internal data structures of the application. In doing this, the syntax specification of the CIF language given in Chapter 2.2 should be adhered to precisely. This result is most easily achieved if the code that does the parsing is generated as automatically as possible from the grammar of the language. Current 'industrial' practice in creating parsers is based on use of commonly available tools for lexical scanning of tokens and parsing of grammars based on *lex* (Lesk & Schmidt, 1975) and *yacc* (Johnson, 1975). Two accessible descendants of these programs are *flex* (by V. Paxson *et al.*) and *bison* (by R. Corbett *et al.*). See Fig. 5.1.3.5 for an example of *bison* data in building a CIF parser. Both *flex* and *bison* are available from the GNU project at http://www.gnu.org.

Neither *flex* nor *bison* is used directly by the final application. Each may be used to create code that becomes part of the application. For example, both are used by *CifSieve* to generate the code it produces. There is an important division of labour between *flex* and *bison*; *flex* is used to produce a lexicographic scanner, *i.e.* code that converts a string of characters into a sequence of 'tokens'. In CIF, the important tokens are such things as tags and values and reserved words such as `loop_`. Once tokens have been identified, responsibility passes to the code generated by *bison* to interpret. In practice, because of the complexities of context-sensitive management of white space to separate tokens and the small number of distinct token types, *flex* is not always used to generate the lexicographic scanner for a CIF parser. Instead, a hand-coded lexer might be used.

The parser generated by *bison* uses a token-based grammar and actions to be performed as tokens are recognized. There are two major alternatives to consider in the design: event-driven interaction with the application or building of a complete data structure to hold a representation of the CIF before interaction with the application. The advantage of the event-driven approach is that a full extra data structure does not have to be populated in order to access a few data items. The advantage of building a complete representation of the CIF is that the application does not have to be prepared for tags to appear in an arbitrary order.

5.1.4. Conclusion

Making CIF-aware applications is a demanding, but manageable, task. A software developer has the choice of using external filters, using existing libraries and APIs, or of building CIF infrastructure from scratch. The last choice presents an opportunity to tune the handling of CIFs to the needs of the application, but also presents the risk of creating code that does not conform to CIF specifications. One can never know for certain how a new application may be used in the future. If there is any doubt that an application built from scratch will conform to CIF specifications, prudence dictates that one should use filter programs or well tested libraries and APIs in preference to cutting corners in building an application from scratch.

We are grateful to Frances C. Bernstein for her helpful comments and suggestions.

References

Allen, F. H. (2002). *The Cambridge Structural Database: a quarter of a million crystal structures and rising. Acta Cryst.* B**58**, 380–388.

Allen, F. H., Kennard, O., Motherwell, W. D. S., Town, W. G. & Watson, D. G. (1973). *Cambridge Crystallographic Data Centre. II. Structural Data File. J. Chem. Doc.* **13**, 119–123.

Andrews, N. (1987). *Rich Text Format standard makes transferring text easier. Microsoft Syst. J.* **2**, 63–67.

Berners-Lee, T. (1989). *Information management: a proposal.* Internal Report. Geneva: CERN. http://www.w3.org/History/1989/proposal-msw.html.

Bernstein, F. C. & Bernstein, H. J. (1996). *Translating mmCIF data into PDB entries. Acta Cryst.* A**52** (Suppl.), C-576.

Bernstein, F. C., Koetzle, T. F., Williams, G. J. B., Meyer, E. F. Jr, Brice, M. D., Rodgers, J. R., Kennard, O., Shimanouchi, T. & Tasumi, M. (1977). *The Protein Data Bank: a computer-based archival file for macromolecular structures. J. Mol. Biol.* **112**, 535–542.

Bernstein, H. J. (1997). *cif2cif – CIF copy program*. Bernstein + Sons, Bellport, NY, USA. Included in http://www.bernstein-plus-sons.com/software/ciftbx.

Bernstein, H. J. & Bernstein, F. C. (2002). *YAXDF and the interaction between CIF and XML. Acta Cryst.* A**58** (Suppl.), C257.

Bernstein, H. J., Bernstein, F. C. & Bourne, P. E. (1998). *CIF applications. VIII. pdb2cif: translating PDB entries into mmCIF format. J. Appl. Cryst.* **31**, 282–295. Software available from http://www.bernstein-plus-sons.com/software/pdb2cif.

Bray, T., Paoli, J. & Sperberg-McQueen, C. (1998). *Extensible Markup Language (XML)*. W3C recommendation 10-February-1998. http://www.w3.org/TR/1998/REC-xml-19980210.

Cambridge Structural Database (1978). *Cambridge Crystallographic Database User Manual*. Cambridge Crystallographic Data Centre, 12 Union Road, Cambridge, England.

Chang, W. & Bourne, P. E. (1998). *CIF applications. IX. A new approach for representing and manipulating STAR files. J. Appl. Cryst.* **31**, 505–509.

Diamond, R. (1971). *A real-space refinement procedure for proteins. Acta Cryst.* A**27**, 436–452.

Dubuisson, O. (2000). *ASN.1 – communication between heterogeneous systems*. San Francisco, CA: Morgan Kaufmann. (Translated from the French by P. Fouquart.)

Flack, H. D., Blanc, E. & Schwarzenbach, D. (1992). *DIFRAC, single-crystal diffractometer output-conversion software. J. Appl. Cryst.* **25**, 455–459.

Hall, S. R. (1993). *CIF applications. IV. CIFtbx: a tool box for manipulating CIFs. J. Appl. Cryst.* **26**, 482–494.

Hall, S. R. & Bernstein, H. J. (1996). *CIF applications. V. CIFtbx2: extended tool box for manipulating CIFs. J. Appl. Cryst.* **29**, 598–603.

Hall, S. R. & Sievers, R. (1993). *CIF applications. I. QUASAR: for extracting data from a CIF. J. Appl. Cryst.* **26**, 469–473.

Hammersley, A. P. (1997). *FIT2D: an introduction and overview*. ESRF Internal Report ESRF97HA02T. Grenoble: ESRF.

Heller, S. R., Milne, G. W. A. & Feldmann, R. J. (1977). *A computer-based chemical information system. Science*, **195**, 253–259.

Hester, J. R. & Okamura, F. P. (1998). *CIF applications. X. Automatic construction of CIF input functions: CifSieve. J. Appl. Cryst.* **31**, 965–968.

ISO (1986). ISO 8879. *Information processing – Text and office systems – Standard Generalized Markup Language (SGML).* Geneva: International Organization for Standardization.

ISO (2002). ISO/IEC 8824-1. *Abstract Syntax Notation One (ASN.1). Specification of basic notation.* Geneva: International Organization for Standardization.

Johnson, S. C. (1975). *YACC: Yet Another Compiler-Compiler*. Bell Laboratories Computing Science Technical Report No. 32. Bell Laboratories, Murray Hill, New Jersey, USA. (Also in UNIX Programmer's Manual, Supplementary Documents, 4.2 Berkeley Software Distribution, Virtual VAX-11 Version, March 1984.)

Keller, P. A. (1996). *A mmCIF toolbox for CCP4 applications*. *Acta Cryst.* A**52** (Suppl.), C-576.