```
C The following common block corresponds to a
C structure defined in the C header, which is written
C to by routine 'cifsiv'. In order to correctly write
C to this common block, 'cifsiv' should be called
C with a *third* argument which will always be
C 'blockbeg'.
    REAL BLOCKBEG
    CHARACTER*84  ERRORMES
    INTEGER ERRORNUM
    CHARACTER*84 mylabel(50)
    REAL*8 atrat(50)
    REAL*8 atratesd(50)
    REAL*8 atsiteu(6,500)
    REAL*8 atsiteuesd(6,500)
    CHARACTER*84 extmet
    REAL*8 reftot(2)
    REAL*8 reftotesd(2)
    COMMON/CIFCMN/BLOCKBEG,ERRORMES,ERRORNUM,mylabel,
   *atrat,atratesd,atsiteu,atsiteuesd,extmet,reftot,
   *reftotesd
```

Fig. 5.3.7.7. Fortran include file forcif.inc for an application built by *BuildSiv* from the augmented DDL dictionary of Fig. 5.3.7.4.

```
      PROGRAM FORGET
      include 'forcif.inc'
      call cifsiv("tbshort.cif","tbal03",blockbeg)
      do i = 1,4
        write(*,*) mylabel(i), atsiteu(1,i),
     *             atsiteu(2,i)
      enddo
      write(*,*) reftot(1)
      write(*,*) extmet
      end
```

Fig. 5.3.7.8. An example Fortran program designed to read CIF data as tagged in the augmented DDL dictionary of Fig. 5.3.7.4.

always called 'BLOCKBEG'. The input subroutine is thus called from within a Fortran program by a line of the type

`CALL CIFSIV(`*FILE*`,` *BLOCK*`,` `BLOCKBEG)`

where *FILE* and *BLOCK* are, respectively, the name of the input file and data block.

Fig. 5.3.7.7 is an example Fortran include file generated by *BuildSiv* and Fig. 5.3.7.8 is an example application incorporating this file. As with the C examples, the CIF data to be read are those specified in the dictionary augmented according to Fig. 5.3.7.4.

It may be noted that the C header file generated by the Fortran implementation of *BuildSiv* (and which is used directly by the C object file produced) is callable by any other C program or subroutine. The Fortran common block is represented by a C structure named *cifcmnptr*, so that the variable names are stored within that structure and must be addressed through the C → operator. That is, an additional C routine compiled in with the Fortran example program of Fig. 5.3.7.7 would refer to the variable holding the value of the input `_refine_ls_extinction_method` as `(char *)cifcmnptr->extmet`.

5.3.8. Tools for mmCIF

The complex relationships between the components of a macromolecular structure at various levels of detail are richly described by the data names in the mmCIF dictionary, but their number and complexity demand more heavyweight tools for proper handling. Input/output for small-molecule or inorganic structures can often be handled by a simple CIF parse and identification of the desired components of one or a few looped data structures. For macromolecules, multiple categories must be loaded simultaneously, and the integrity of relationships between items in the different categories must be properly maintained. For this reason, the most effective tools for mmCIF-based applications have high-level interactions with the mmCIF or related dictionaries, and necessarily involve more complex data manipulations.

In this section are discussed three software systems that are available for work with macromolecular structures: *CIFOBJ* and related libraries, which provide a long-established and complete application program interface (API) to dictionaries and data files; *OpenMMS*, an exciting development allowing abstract data representations (based on the mmCIF dictionary definitions) to be exchanged between applications using an intermediate middleware layer; and *mmLib*, which is a Python toolkit for biomolecular structure applications. These latter two may come closer to the area of domain-specific applications than most of the generic tools we have discussed in this chapter. However, they demonstrate how the abstract data model represented by the mmCIF dictionaries can effectively be imported into a diverse range of programming environments.

5.3.8.1. *CIFOBJ* and related libraries

Early in the development of the mmCIF dictionary, the Nucleic Acid Database at Rutgers University (Berman *et al.*, 1992) created a number of CIF libraries and utilities to underpin data-processing activities. Much of this development work was carried across when the curatorship of the Protein Data Bank was transferred to the Research Collaboratory for Structural Bioinformatics (RCSB; Berman *et al.*, 2002), and the software provides the engine for many of the robust and industrial-strength database operations of these organizations.

CIFLIB (Westbrook *et al.*, 1997) was an early class library, no longer supported, that was developed to provide an API to macromolecular CIF data files and to the associated dictionaries (Chapters 3.6 and 4.5) and underlying dictionary definition language (DDL2) files (Chapter 2.6).

The RCSB Protein Data Bank now distributes object-oriented parsing tools (*CIFPARSE_OBJ*; Tosic & Westbrook, 2000) which fully support CIF data files and their underlying metadata descriptions in dictionaries and DDL2 attribute sets, and a comprehensive library of access methods for data and dictionary objects at category and item level.

The information infrastructure of the Protein Data Bank, built upon these tools, is discussed in Chapter 5.5. All the software produced for this purpose is distributed with full source under an open-source licence, to promote the development of mmCIF tools and to encourage interoperability with other software environments.

5.3.8.2. *OpenMMS*

Object classes represent the first stage in abstracting related data components. By building structured software modules that can manage the small-scale interactions between data components, the programmer can write more succinct code to handle the interactions between much higher-level data constructs. An API then permits third parties to handle the larger-scale objects without any need to know the internal workings of the class library. The next logical step is to present a standard set of 'objects' representing complete logical entities to any programmer for 'plug-and-play' incorporation into new applications.

The Life Sciences Research domain task force of the Object Management Group (OMG, 2001) is concerned with the development of standards for data exchange in biomolecular sciences, and in 2002 approved a macromolecular structure Corba specification. Corba (the common object request broker architecture) is a middleware architecture intended to serve just this purpose of providing access to standard objects representing discrete logical entities suitable for programmatic manipulation. Corba promotes interoperability across networked applications by separating entirely the API from the implementation of the underlying data objects. For applications such as the macromolecular structures database hosted by the Protein Data Bank, the attraction of networked interoperability is that information can be accessed through distributed and federated databases, and can be delivered on demand to any compatible software.

A Corba application comprises an interface definition language (IDL) and an API that together define access to a data structure that encapsulates the abstract representation of the objects and relationships relevant to a particular area of knowledge. In general terms, this data structure may be described as an 'ontology' (Westbrook & Bourne, 2000). The ontology adopted for macromolecular structure (MMS) data was based on the mmCIF dictionary following a submission by the Research Collaboratory for Structural Bioinformatics to a Request for Proposal (Greer, 2000).

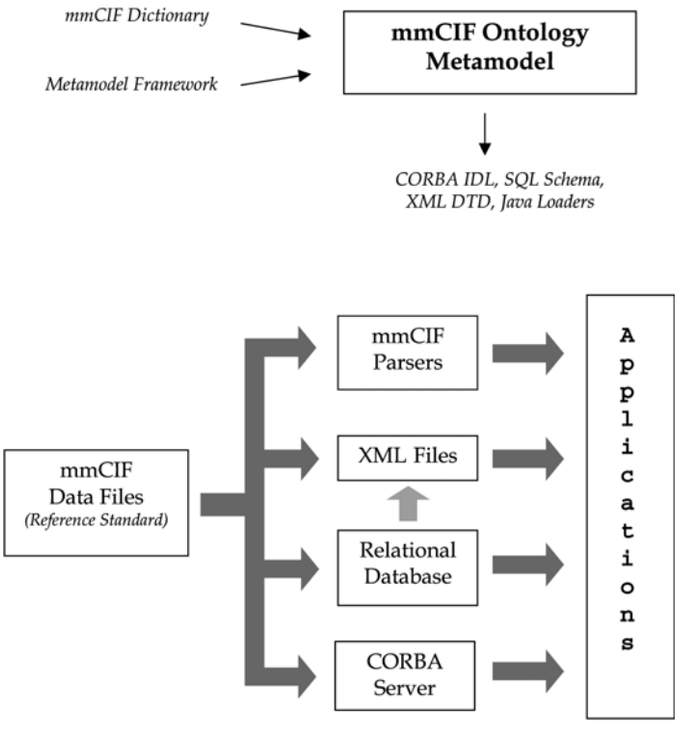

Fig. 5.3.8.1. The *OpenMMS* metamodel and data flow.

5.3.8.2.1. *The OpenMMS toolkit*

In practice, the ontology was developed in a 'metamodel' that combined the definitions and relationships between data items specified in the mmCIF dictionary with a generic metamodel framework. The metamodel extracts the information in the mmCIF dictionary but maintains it in a representation that is independent of the mmCIF STAR or any other file format. The standard building block of the metamodel is an *Entry* object, modelling a single macromolecular structure.

From a suitable metamodel, it then becomes relatively straightforward to generate alternative expressions of the information to suit different access requirements. The *OpenMMS* toolkit (Greer *et al.*, 2002) was built using Java source code to generate a Corba interface, an SQL schema for relational database loading and an XML representation of macromolecular data sets (Fig. 5.3.8.1).

The toolkit contains an mmCIF parsing module capable of direct access to the underlying data archive of mmCIF data files. This is important, because the data files represent a common reference for all the derived representations. Any errors or discrepancies between the expressed forms of the Corba, XML or SQL representations are resolved against the standard mmCIF reference form.

The relational database supporting an SQL-92 compatible interface provides an appropriate API for many applications, particularly ones that require extensive string searches. The close relationship between the mmCIF data model and relational database models has already been described earlier in this volume (Chapter 2.6).

Advantages of the SQL interface are that it provides rapid access direct to the binary data storage representation and that individual components of a data set may be efficiently retrieved without the need to search sequentially through an entire entry.

This efficiency of access and the ability to retrieve individual MMS data elements from a remote server is best realized through the Corba interface, the primary purpose of which is indeed to facilitate such high-performance access.

The bulk exchange of data is addressed through the generation of XML files. XML is a simple, powerful and widely used standard for interchanging data, and its use for transporting macromolecular data obviates the need for target applications to build their own STAR parsers. However, the use of markup tags around every individual data element does make the files much larger than their mmCIF progenitors. This is not an insurmountable problem in large-scale application environments, but it can undermine the effectiveness of XML as a representation mechanism in such applications as web browsers. A possible approach to this could be to define different, less verbose, XML representations and populate these on demand from a database store, either by SQL or XML queries. This is not an approach that the current *OpenMMS* toolkit supports directly.

Fig. 5.3.8.2 is an extract from an XML data file generated from the PDB structure 1xy2. The XML uses a reserved name space *PDBx* conforming to the schema http://deposit.pdb.org/pdbML/pdbx-v0.905.xsd. Data tags map cleanly to the corresponding data names in the mmCIF dictionary formed by concatenating the XML element name with its parent category name. For example, the entry `<PDBx:length_a>27.080</PDBx:length_a>` included in the `<PDBx:cellCategory>` container tag can be directly translated to the corresponding mmCIF data item **`_cell.length_a 27.080`**. CIF data loops are represented by repeated instances of the XML tag representing the corresponding CIF data name (for example, the multiple `<PDBx:audit_author name>` tags are equivalent to a CIF **`loop_ _audit_author.name`** construct). Nonstandard items with a *pdbx_* prefix (*e.g.* `<PDBx:pdbx_description>` in the `<PDBx:entityCategory>` group) refer to private data names in the PDB extension dictionary (Appendix 3.6.2).

5.3.8.3. *mmLib*: a Python toolkit for bioinformatics applications

While the libraries developed for use within the Protein Data Bank provide powerful functionality, their very size and complexity make them inappropriate for some applications. Indeed, considerable effort may be needed to compile the C++ code on non-standard platforms. The *mmLib* toolkit (Painter & Merritt, 2004)

```
<?xml version="1.0" encoding="UTF-8" ?>
<PDBx:datablock  datablockName="1XY2"
 xmlns:PDBx="http://deposit.pdb.org/pdbML/pdbx-v0.905.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=
    "http://deposit.pdb.org/pdbML/pdbx-v0.905.xsd
        pdbx-v0.905.xsd">
 <PDBx:audit_authorCategory>
  <PDBx:audit_author name="Cooper, S."></PDBx:audit_author>
  <PDBx:audit_author name="Blundell, T.L.">
                                      </PDBx:audit_author>
  <PDBx:audit_author name="Pitts, J.E."></PDBx:audit_author>
  <PDBx:audit_author name="Wood, S.P."></PDBx:audit_author>
  <PDBx:audit_author name="Tickle, I.J."></PDBx:audit_author>
 </PDBx:audit_authorCategory>
 <PDBx:cellCategory>
  <PDBx:cell entry_id="1XY2">
   <PDBx:length_a>27.080</PDBx:length_a>
   <PDBx:length_b>9.060</PDBx:length_b>
   <PDBx:length_c>22.980</PDBx:length_c>
   <PDBx:angle_alpha>90.00</PDBx:angle_alpha>
   <PDBx:angle_beta>102.06</PDBx:angle_beta>
   <PDBx:angle_gamma>90.00</PDBx:angle_gamma>
   <PDBx:Z_PDB>4</PDBx:Z_PDB>
  </PDBx:cell>
 </PDBx:cellCategory>
 <PDBx:citationCategory>
  <PDBx:citation id="primary">
   <PDBx:title>Crystal structure analysis of
               deamino-oxytocin: conformational flexibility
               and receptor binding.</PDBx:title>
   <PDBx:journal_abbrev>Science</PDBx:journal_abbrev>
   <PDBx:journal_volume>232</PDBx:journal_volume>
   <PDBx:page_first>633</PDBx:page_first>
   <PDBx:page_last>636</PDBx:page_last>
   <PDBx:year>1986</PDBx:year>
   <PDBx:journal_id_ASTM>SCIEAS</PDBx:journal_id_ASTM>
   <PDBx:country>US</PDBx:country>
   <PDBx:journal_id_ISSN>0036-8075</PDBx:journal_id_ISSN>
   <PDBx:journal_id_CSD>0038</PDBx:journal_id_CSD>
  </PDBx:citation>
 </PDBx:citationCategory>
 <PDBx:computingCategory>
  <PDBx:computing entry_id="1XY2">
   <PDBx:structure_solution>SHELX</PDBx:structure_solution>
   <PDBx:structure_refinement>SHELX-76
                              </PDBx:structure_refinement>
  </PDBx:computing>
 </PDBx:computingCategory>
 <PDBx:database_2Category>
  <PDBx:database_2 database_id="PDB" database_code="1XY2">
                                        </PDBx:database_2>
 </PDBx:database_2Category>
 <PDBx:entityCategory>
  <PDBx:entity id="1">
   <PDBx:type>polymer</PDBx:type>
   <PDBx:src_method>man</PDBx:src_method>
   <PDBx:pdbx_description>OXYTOCIN</PDBx:pdbx_description>
   <PDBx:formula_weight>978.189</PDBx:formula_weight>
   <PDBx:pdbx_number_of_molecules>1
                           </PDBx:pdbx_number_of_molecules>
  </PDBx:entity>
  <PDBx:entity id="2">
   <PDBx:type>water</PDBx:type>
   <PDBx:src_method>nat</PDBx:src_method>
   <PDBx:pdbx_description>water</PDBx:pdbx_description>
   <PDBx:formula_weight>18.015</PDBx:formula_weight>
   <PDBx:pdbx_number_of_molecules>7
                           </PDBx:pdbx_number_of_molecules>
  </PDBx:entity>
 </PDBx:entityCategory>
```

Fig. 5.3.8.2. Sample XML output from the *OpenMMS* XML generator. Lines have been omitted or wrapped to fit the present column width.

addresses this by supplying a library of object-oriented routines implemented in Python (van Rossum, 1991) that are designed to integrate with existing or new applications in an easy way.

The objective of *mmLib* is to build a support platform to handle the increasingly rich data about macromolecular structure

Table 5.3.8.1. *The modules provided by the mmLib toolkit*

mmLib.mmCIF	mmCIF parser
mmLib.PDB	PDB format parser
mmLib.Library	Base chemical library
mmLib.Extensions.CCP4Library	Data retrieval from CCP4 monomer library
mmLib.Elements	Chemical data for elements
mmLib.AminoAcids	Chemical data for amino acids
mmLib.NucleicAcids	Chemical data for nucleic acids
mmLib.Structure	Macromolecular structure model
mmLib.GLViewer	OpenGL visualizer

```
import mmLib
from mmLib.FileLoader import LoadStructure, SaveStructure
    struct = LoadStructure(
                fil = cif,
                format = "PDB",
                build_properties = ("no_bonds",) )
    SaveStructure(
        fil = pdb,
        structure = struct,
        format = "CIF")
```

Fig. 5.3.8.3. A snippet of code illustrating mmCIF/PDB file format conversion with the *mmLib* toolkit.

available to structural biologists. Not only do applications need to be able to handle atomic positions and build appropriate three-dimensional structure representations; but links to and integration with information on sequence, homologous structures, and biochemical, genetic and medical form and function are also demanded from individual program systems. Since much of these data are available from external databases in a variety of formats, *mmLib* will not be restricted to the handling of files in a single format. Its initial release provides support for mmCIF, for the PDB format files that historically have been used for representation of macromolecular structures (Westbrook & Fitzgerald, 2003) and for the MTZ format used by the *CCP4* program suite (Collaborative Computational Project, Number 4, 1994).

Table 5.3.8.1 lists the main modules in the current release. *mmLib.mmCIF* and *mmLib.PDB* are read/write parsers for mmCIF and PDB format files, respectively, which handle file input and output in these formats, and provide support for inspection or modification of such file formats. They are typically used in conjunction with the *mmLib.FileLoader* component to populate the *mmLib.Structure* internal representation of the macromolecular structure. The high-level abstraction of such functionality allows for very succinct programmatic constructs. Fig. 5.3.8.3 illustrates this with a program snippet that (apart from the necessary system calls for file management) achieves the conversion of an mmCIF input file to a PDB format representation. This is sufficiently robust and lightweight to act as an input filter to software already designed for handling PDB format files.

mmLib.Structure represents the internal representation of a molecular structure and is implemented as an object hierarchy with four basic object classes: *Structure*, *Chain*, *Fragment* and *Atom*. The *Fragment* class has subclasses *AminoAcidResidue* and *NucleicAcidResidue*. In order to build a complete representation of a structure, the toolkit may need to load data from an input mmCIF or PDB format file, and also from standard data sets of properties of individual monomers and chemical elements; these standard libraries of chemical properties are provided by the *mmLib.Library* module. The core *mmLib* source includes a limited library of such chemical properties (accessible through the subclasses *mmLib.Elements*, *mmLib.AminoAcids* and *mmLib.NucleicAcids*)

and also provides support for the extensive CCP4 monomer library through the *mmLib.Extensions.CCP4Library*. The naming of this class expresses the intention that other standard data sources should be made accessible in the same way.

The CCP4 monomer library is in fact included with the software as a directory tree of small files in mmCIF format, which are loaded into the *Structure* object through the normal use of the toolkit's mmCIF parser.

mmLib.GLViewer is a module provided to support visualization programs using the OpenGL graphics environment. Although it does not by itself provide a stand-alone viewer, it can be incorporated into many common graphics application building environments. An example molecular viewer, *mmView*, is provided with the distribution as an example of an application using the GTK graphical user interface, a popular toolkit in Linux.

5.3.9. Concluding remarks

CIF is a domain-specific format that cannot attract the number of programmers that generic formats such as XML do. In spite of this, there is an impressive collection of programs available to support activities at many levels, from the single-line shell script needed to search for some desired content in a collection of CIFs, to the industrial-scale activities of major databases and publishing houses. As many examples as possible of the programs discussed in this chapter have been collected on the IUCr web site (http://www.iucr.org/iucr-top/cif/software). It is hoped that the contributions described here will inspire future generations of programmers to contribute to a growing and increasingly robust software collection to make the use of CIFs ever easier and more fruitful.

I am immensely grateful for the assistance, cooperation and involvement of the community of software authors who have contributed to this chapter in one way or another, and to all the programmers and developers who have been active through the cif-developers discussion list of the IUCr (http://www.iucr.org/iucr-top/lists/cif-developers) and in private discussions.

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., Johnson, O., Shields, G. P., Smith, B. R. & Towler, M. (2004). *CIF applications. XV. enCIFer: a program for viewing, editing and visualizing CIFs. J. Appl. Cryst.* **37**, 335–338.

Berman, H. M., Battistuz, T., Bhat, T. N., Bluhm, W. F., Bourne, P. E., Burkhardt, K., Feng, Z., Gilliland, G. L., Iype, L., Jain, S., Fagan, P., Marvin, J., Padilla, D., Ravichandran, V., Schneider, B., Thanki, N., Weissig, H., Westbrook, J. D. & Zardecki, C. (2002). *The Protein Data Bank. Acta Cryst.* D**58**, 899–907.

Berman, H. M., Olson, W. K., Beveridge, D. L., Westbrook, J., Gelbin, A., Demeny, T., Hsieh, S.-H., Srinivasan, A. R. & Schneider, B. (1992). *The Nucleic Acid Database: a comprehensive relational database of three-dimensional structures of nucleic acids. Biophys. J.* **63**, 751–759.

Bernstein, H. J. (1998). *cif2cif. CIF copy program.* http://www.iucr.org/iucr-top/cif/software/ciftbx/cif2cif.src/.

Bernstein, H. J. & Hall, S. R. (1998). *CIF applications. VII. CYCLOPS2: extending the validation of CIF data names*. *J. Appl. Cryst.* **31**, 278–281.

Bluhm, W. (2000). *STAR (CIF) parser*. http://pdb.sdsc.edu/STAR/index.html.

Brown, I. D., Zabobonin, A. & Holt, B. (2004). *beCIF. Browser and editor for CIF.* Private communication.

Collaborative Computational Project, Number 4 (1994). *The CCP4 suite: programs for protein crystallography. Acta Cryst.* D**50**, 760–763.

Edgington, P. R. (1997). *HICCuP: High-Integrity CIF Checking using Python*. Cambridge: Cambridge Crystallographic Data Centre.

Greer, D. S. (2000). *Macromolecular structure RFP response*. Revised submission. http://openmms.sdsc.edu/OpenMMS-1.5.1_Std/openmms/docs/specs/lifesci_00-11-01.pdf.

Greer, D. S., Westbrook, J. D. & Bourne, P. E. (2002). *An ontology driven architecture for derived representations of macromolecular structure. Bioinformatics*, **18**, 1280–1281.

Hall, S. R. (1993). *CIF applications. III. CYCLOPS: for validating CIF data names*. *J. Appl. Cryst.* **26**, 480–481.

Hall, S. R., Allen, F. H. & Brown, I. D. (1991). *The Crystallographic Information File (CIF): a new standard archive file for crystallography. Acta Cryst.* A**47**, 655–685.

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.

Hester, J. R. (2006). *A validating CIF parser: PyCIFRW. J. Appl. Cryst.* **39**, 621–625.

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

Knuth, D. E. (1986). *The TEXbook*. *Computers and Typesetting*, Vol. A. Reading, MA: Addison-Wesley.

McMahon, B. (1993). *ciftex: translation utility from CIF to TEX.* ftp://ftp.iucr.org/pub/ciftex.tar.Z.

McMahon, B. (1998). *vcif: a utility to validate the syntax of a Crystallographic Information File.* http://www.iucr.org/iucr-top/cif/software/vcif/index.html.

OMG (2001). *Life Sciences Research Domain Task Force.* http://www.omg.org/lsr/.

Ousterhout, J. K. (1994). *Tcl and the Tk toolkit*. Reading, MA: Addison-Wesley.

Painter, J. & Merritt, E. A. (2004). *mmLib Python toolkit for manipulating annotated structural models of biological macromolecules. J. Appl. Cryst.* **37**, 174–178.

Patel, A. J. (2002). *Yapps: Yet Another Python Parser System.* http://theory.stanford.edu/∼amitp/yapps/.

Rossum, G. van (1991). *Python programming language*. http://www.python.org.

Spadaccini, N. & Hall, S. R. (1994). *Star_Base: accessing STAR File data*. *J. Chem. Inf. Comput. Sci.* **34**, 509–516.

Stampf, D. R. (1994). *ZINC: galvanizing CIF to work with UNIX.* Brookhaven: Protein Data Bank.

Toby, B. H. (2003). *CIF applications. XIII. CIFEDIT, a program for viewing and editing CIFs*. *J. Appl. Cryst.* **36**, 1288–1289.

Tosic, O. & Westbrook, J. D. (2000). *CIFParse. A library of access tools for mmCIF.* Reference guide. http://sw-tools.pdb.org/apps/CIFPARSE-OBJ/cifparse/index.html.

Wall, L., Schwartz, R. L., Christiansen, T. & Orwant, J. (2000). *Programming Perl*, 3rd ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc.

Westbrook, J. D. & Bourne, P. E. (2000). *STAR/mmCIF: an ontology for macromolecular structure*. *Bioinformatics*, **16**, 159–168.

Westbrook, J. & Fitzgerald, P. (2003). *The PDB format, mmCIF formats and other data formats. Structural bioinformatics*, edited by P. E. Bourne & H. Weissig, pp. 161–179. Hoboken, NJ: John Wiley & Sons, Inc.

Westbrook, J. D., Hsieh, S.-H. & Fitzgerald, P. M. D. (1997). *CIF applications. VI. CIFLIB: an application program interface to CIF dictionaries and data files. J. Appl. Cryst.* **30**, 79–83.

Westrip, S. P. (2004). *printCIF for Word*. http://www.iucr.org/iucr-top/cif/software/printCIFforWord/index.html.

Winn, M. (1998). *cif.el: an Emacs mode for CIF*. Daresbury Laboratory, Warrington, England.