```
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.