*International Tables for Crystallography* (2006). Vol. G, Section 5.1.2, pp. 481–483.

# 5.1. General considerations in programming CIF applications

BY H. J. BERNSTEIN

## 5.1.1. Introduction

There are many ways to create new 'CIF-aware' applications and to adapt existing applications to make them CIF-aware. This chapter reviews general considerations in programming CIF-aware applications, ranging from leaving an application CIF-unaware and relying on external filter utilities to do the job, through engineering an existing application to directly read and write CIFs, to writing a new CIF-aware application from scratch. The adaptation of applications to CIF does not happen in isolation. There are many other data representations and metadata frameworks relevant to crystallography. In Chapter 1.1, the CIF format was placed in the historical context of the development of data representation and metadata languages. In this chapter, we deal with that context from the perspective of software design.

The major issues in making an application CIF-aware are:

(1) Are CIFs to be read?
  (i) Are these CIFs produced externally?
  (ii) Does the organization of information required by the application conform to the organization of information specified in the relevant dictionaries?
  (iii) Is maximal performance in reading important?
(2) Are CIFs to be written?
  (i) Are these CIFs to be used externally?
  (ii) Does the organization of information used by the application internally conform to the organization of information specified by the relevant dictionaries?
  (iii) Is maximal performance in writing important?

Reading a CIF is a much more complex task than writing a CIF. Two equally valid CIF presentations of exactly the same information may be ordered differently. An application that reads a CIF must be prepared to accept tags and columns of tables in any order. The same order independence means that an application may write out tags and columns of tables in any convenient order, simplifying the design of CIF write logic.

When CIFs are only to be used internally, it is tempting to adjust the format to fit the application, *e.g.* by imposing order dependence. However, caution is advised if there is any possibility that such an application might eventually have to deal with CIFs coming from or going to other groups. In designing the CIF interface for an application, it is prudent to assume that the read logic will eventually have to deal with externally produced CIFs and that CIFs produced by the write logic will eventually be processed by software at other sites.

If performance is not a major issue, then it can be easy to make an application CIF-aware simply by the use of external filter programs. However, when performance *is* an issue, it becomes necessary to integrate the CIF reading and writing logic with the application. This can be done with reasonable efficiency by the use of existing CIF-aware libraries, but such libraries can impose a cost by their use of private internal data structures to hold the information from a CIF. Integrated design from scratch may be needed for maximal performance.

Creating or adapting software for CIF is an example of creating or adapting software for an agreed format, a format to be adhered to in the creation of multiple applications. There are different levels of 'agreement'. The agreement may apply to the representation of data, the representation of information about data ('metadata') or both (making a 'data framework'). The agreement may loosely specify the style of presentation of information or may specify details of presentation with great precision, or anything in between. The effort one needs to make in adapting an application to an agreed format depends to a large part on the level of agreement. If the agreed format is sufficiently detailed, one can comply strictly with the agreed format as a standard.

If one's goals are the widest possible interchange of data and the longest possible survival time of data in archives, it is important to achieve strict adherence to the agreed format as a standard. If one's goals are shorter-term and do not raise issues of interchange with independent groups, use of an agreed format as a checklist or style guide may help to avoid redundant effort. Even when one has longer-term goals, system requirements may not mesh with the agreed format and the development of new formats may be needed. CIF, as an agreed format, involves specification of metadata for a specific data format and of ontologies of domain-specific definitions that could be used not only in CIF but also in other formats. The use of an agreed format as a base can help to avoid redundant effort in this case as well. Within the domain of small-molecule crystallography, CIF has achieved its powerful impact on crystallographic publication and archiving by being used as a strict standard both for the data format and for definitions of terms. Within other domains or for certain applications other approaches may be appropriate. For example, it has proven productive to make use of CIF data names within XML (Bray *et al.*, 1998) formatted documents (Bernstein & Bernstein, 2002).

## 5.1.2. Background

There have been many efforts at creating agreed formats for data to be used in crystallography (see Chapter 1.1). We need to consider how software has been created to make use of such formats, especially software to make use of CIF.

Agreement on formats evolved from the earliest efforts at collaboration among research groups. Within crystallography, recognition of the need to use data formats as standards and to adapt applications to agreed formats, rather than to adapt formats to the caprices of particular applications or diffractometers or graphics engines, began in the late 1960s and early 1970s with the establishment of computerized data resources for the chemical and crystallographic community and the increasing availability of computer networks (Lykos, 1975). We will discuss three early data-resource efforts: the Cambridge Crystallographic Data Centre Structural Database File (CSD) (Allen *et al.*, 1973), the Brookhaven National Laboratory Protein Data Bank (PDB) (Bernstein *et al.*, 1977) and the NIH/EPA Chemical Information System (CIS) (Heller *et al.*, 1977). The differences and similarities among application development efforts related to these resources illustrate some of the issues

Affiliation: HERBERT J. BERNSTEIN, Department of Mathematics and Computer Science, Kramer Science Center, Dowling College, Idle Hour Blvd, Oakdale, NY 11769, USA.

that now face software developers working with CIF: conformance to agreed formats *versus* deviations from standards to improve performance, as well as cross-platform portability.

The Cambridge Crystallographic Data Centre was established in 1965 'to compile a database containing comprehensive information on small-molecule crystal structures, *i.e.* organics and metallo-organic compounds containing up to 500 non-H atoms, the structures of which had been determined by X-ray or neutron diffraction' (Allen, 2002). The Protein Data Bank was established at Brookhaven National Laboratory in 1971 as an archive of macromolecular structural information. The NIH/EPA Chemical Information System was established in 1975 as a confederation of databases including mass spectroscopy, NMR and the data from the CSD. The three resources, CSD, PDB and CIS, took different approaches to applications development. The CSD was an integrated software system centred on a database. Both the software and the database were distributed on magnetic tape for users to use on their local computers. The developers of the software had to be concerned with portability of the software across the multiple computer systems used by crystallographers, but retained control of the design of the retrieval software and a core suite of applications. The PDB was an archive, rather than a database. Some software and the data were distributed on magnetic tape, but the application development model was what would now be called 'open', with users and software developers taking the data and the PDB format specification and creating software that would do useful things with PDB entries. The CIS was a remotely accessed confederation of databases on a central computer. The developers of software for the CIS did not have to be concerned with cross-platform portability, or with changes in syntax or semantics of data files impacting on external software developers. Developers of software for the CSD and the PDB had to be concerned with strict compliance with the rules for the respective data formats, albeit on somewhat different timescales. Developers of software for the centralized CIS database could negotiate for immediate changes in the data format to improve performance of the relevant application.

The CSD had agreed internal formats (Cambridge Structural Database, 1978). However, as noted in Chapter 1.1, there were many different formats in use for small-molecule crystallography and related fields. One may conjecture that one of many causes for such divergence was the CCDC practice of acquiring much of its data from journals, after differences among data formats had been masked by the publication process. The transition from this Tower of Babel to CIF is described in Chapter 1.1, and that history will not be repeated here, but it is important to note that an application writer working in the domain of small-molecule crystallography still has to be aware of a wide variety of formats in addition to CIF.

In the beginning, the PDB went through a relatively rapid format change and then achieved a stable format for more than two decades. The PDB differed from the CSD in depending on user deposition of data prior to publication. The better a user conformed to PDB data-format conventions, the more efficiently could the data move from deposition to release. The initial standard PDB format (PDB, 1974) was derived from the format used in a popular refinement program of the day (Diamond, 1971) and used 132-character records identified by the character strings in the first six columns. Starting in 1976, the PDB spent more than a year (PDB, 1976*a*,*b*, 1977) converting to an 80-column format, extensions of which are still in use to this day. Many external programs were developed using this 80-column format and it has become a major *de facto* standard for macromolecular software applications. Most application packages producing crystallographic macromolecular structures made a gradual transition from having output options for producing 'Diamond format' to having output options for producing PDB format. Macromolecular applications working with other disciplines shared the small-molecule applications penchant for multiple formats.

The CIS, working in a completely closed, central service environment, had little direct impact on the formats to be used for applications. The CIS would acquire data from existing archives and databases and meld them into its master database. It would deliver its data as text on a CRT. Much of the impact of CIS data formats was to be restricted to its own internal application development.

Most of the formats resulting from these early efforts were fixed-field, fixed-order formats. The result was that adapting an application to a data format was simple if the processing flow of the application conformed to the fixed order of the data format. Frequently, the data flow did conform. When the processing flow did not conform, it was necessary to create internal data structures or temporary files to allow the unfortunately timed arrival of data to be time-shifted until it was needed. In general, the heaviest burden was imposed on applications that needed to write data conforming to one of the agreed formats. As the complexity of such time-shifting processes increased, it became clear that the cleanest solution was to base an application on an internal database and to populate the database as the data were processed. When data were to be written by an application, the data could be extracted from the database in whatever order was required.

In the 1970s and early 1980s, such a procedure was a serious burden to place on an application. With limited memory and processor speeds, there was a strong argument for adapting agreed formats to the 'natural' processing flow, reducing or avoiding the need for an internal database. As the speed and size of computers have changed and as programming language and operating-system support for dynamic allocation of resources has improved, the need to have agreed formats driven by applications has become less pressing.

We need to understand three major thrusts in data representation: the development of markup languages, of data-representation frameworks and of database application support. Modern applications can benefit from all three.

#### 5.1.2.1. Markup languages

A markup language allows the raw text of a document to be annotated with interleaved 'markup' specifying layout information for the bracketed text. For document processing, the implicit assumption of the use of an internal database became formalized with the gradual adoption of agreed markup languages in the late 1980s and early 1990s [*e.g.* TEX (Knuth, 1986), SGML (ISO, 1986), RTF (Andrews, 1987), HTML (Berners-Lee, 1989)]. When used in this manner, such a language has the implicit ordering assumption of reading forward in the document. However, with modern demands for multidimensional layout and document reflow, applications managing such documents achieve the best performance and flexibility when they store the entire marked-up document in an internal data structure that allows random access to all the information.

#### 5.1.2.2. Data-representation frameworks

A data-representation framework provides the concepts for managing data and data about the management of data ('metadata'). Such frameworks may be based on programming languages or markup languages or built from scratch. They provide a mechanism for representing data (*e.g.* as data sets, graphs or trees)

and a mechanism for representing metadata (*e.g.* as dictionaries or schemas). Four are of particular importance in crystallography: CIF, ASN.1, HDF and XML.

As noted in Chapter 1.1, CIF was created to rationalize the publication process for small molecules. It combines a very simple tag–value data representation with a dictionary definition language (DDL) and well populated dictionaries. CIF is table-oriented, naturally row-based, has case-insensitive tags and allows two levels of nesting. CIF is order-independent and uses its dictionaries both to define the meanings of its tags and to parameterize its tags. It is interesting to note that, even though CIF is defined as order-independent, it effectively fills the role of an order-dependent markup language in the publication process. We will discuss this issue later in this chapter.

Abstract Syntax Notation One (ASN.1) (Dubuisson, 2000; ISO, 2002) was developed to provide a data framework for data communications, where great precision in the bit-by-bit layout of data to be seen by very different systems is needed. Although targeted for communications software, ASN.1 is suitable for any application requiring precise control of data structures and, as such, primarily supports the metadata of an application, rather than the data. ASN.1 can be compiled directly to C code. The resulting C code then supports the data of the application. ASN.1 notation found application in NCBI's macromolecular modelling database (Ohkawa *et al.*, 1995). ASN.1 has case-sensitive tags and allows case-insensitive variants. It manages order-dependent data structures in a mixed order-dependent/order-independent environment.

HDF (NCSA, 1993) is 'a machine-independent, self-describing, extendible file format for sharing scientific data in a heterogeneous computing environment, accompanied by a convenient, standardized, public domain I/O library and a comprehensive collection of high quality data manipulation and analysis interfaces and tools' (http://ssdoo.gsfc.nasa.gov/nost/formats/hdf.html). HDF was adopted by the Neutron and X-ray Data Format (NeXus) effort (Klosowski *et al.*, 1997). HDF allows the building of a complete data framework, representing both data and metadata. Two parallel threads of software development, focused on the management and exchange of raw data from area detectors, began in the mid-1990s: the Crystallographic Binary File (CBF) (Hammersley, 1997) and NeXus. The volumes of data involved were daunting and efficiency of storage was important. Therefore both proposed formats assumed a binary format. CBF was based on a combination of CIF-like ASCII headers with compressed binary images. NeXus was based on HDF. The first API for CBF was produced by Paul Ellis in 1998. CBF rapidly evolved into CBF/imgCIF with a complete DDL2 dictionary and a fully CIF-compliant API (Chapter 5.6). As of mid-2010, NeXus was still evolving (see http://www.nexusformat.org/).

XML is a simplified form of SGML, drawing on years of development of tools for SGML and HTML. XML is tree-oriented with case-sensitive entity names. It allows unlimited nesting and is order-dependent. Metadata are managed as a 'document type definition' (DTD), which provides minimal syntactic information, or as schemas, which allow for more detail and are more consistent with database conventions. In fields close to crystallography, the first effort at adopting XML was the chemical markup language (CML) (Murray-Rust & Rzepa, 1999). CML is intentionally imprecise in its ontology to allow for flexibility in development. The CSD and PDB have released their own XML representations (http://www.ccdc.cam.ac.uk/support/documentation/relibase/3_0/relibase_DPG/toc.html; http://pdbml.rcsb.org).

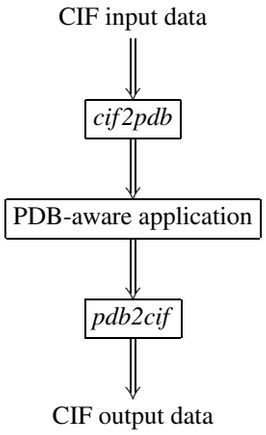


Fig. 5.1.3.1. Example of using filters to make a PDB-aware application CIF-aware.

It may seem from this discussion that the application designer faces an unmanageable variety of data frameworks in an unstable, evolving environment. To some extent this is true. Fortunately, however, there are signs of convergence on CIF dictionary-based ontologies and the use of transliterated CIFs. This means that an application adapted to CIF should be relatively easy to adapt to other data frameworks.

### 5.1.3. Strategies in designing a CIF-aware application

There are multiple strategies to consider when designing a CIF-aware application. One can use external filters. One can use existing CIF-aware libraries. One can write CIF-aware code from scratch.

#### 5.1.3.1. Working with filter utilities

One solution to making an existing application aware of a new data format is to leave the application unchanged and change the data instead. For almost all crystallographic formats other than CIF, the Swiss-army knife of conversion utilities is *Babel* (Walters & Stahl, 1994). *Babel* includes conversions to and from PDB format. Therefore, by the use of *cif2pdb* (Bernstein & Bernstein, 1996) and *pdb2cif* (Bernstein *et al.*, 1998) combined with *Babel*, many macromolecular applications can be made CIF-aware without changing their code (see Figs. 5.1.3.1 and 5.1.3.2). If the need is to extract mmCIF data from the output of a major application, the PDB provides *PDB_EXTRACT* (http://sw-tools.pdb.org/apps/PDB_EXTRACT/).

Creating a filter program to go from almost any small-molecule format to core CIF is easy. In many cases one need only insert the appropriate '`loop_`' headers. Creating a filter to go from CIF to a particular small-molecule format can be more challenging, because a CIF may have its data in any order. This can be resolved by use of *QUASAR* (Hall & Sievers, 1993) or *cif2cif* (Bernstein, 1997), which accept request lists specifying the order in which data are to be presented (see Fig. 5.1.3.3).

There are a significant and growing number of filter programs available. Several of them [*QUASAR*, *cif2cif*, *ciftex* (ftp://ftp.iucr.org/pub/ciftex.tar.Z) (to convert from CIF to TEX) and ZINC (Stampf, 1994) (to unroll CIFs for use by Unix utilities)] are discussed in Chapter 5.3. In addition there are *CIF2SX* by Louis J. Farrugia (http://www.chem.gla.ac.uk/~louis/software/utils/), to convert from CIF to *SHELXL* format, and *DIFRAC* (Flack *et al.*, 1992) to translate many diffractometer output formats to CIF. The program *cif2xml* (Bernstein & Bernstein, 2002) translates from CIF to XML and CML. The PDB