definitions by use of a `[local]` or otherwise registered prefix, as discussed in Section 2.1.4.2.

Handling input CIF data is more complex, because of the free-form layout and ordering of data in an input file. If a suitable library is not used, the programmer may wish to use a filter program to preprocess the input into a normalized presentation that is easier to write native code to handle. Some programs to do this are discussed in Chapter 5.4.

1.1.4.2. General utility programs

Some developers may wish to write generic programs to reorder, validate or otherwise transform CIFs (*e.g* to convert to a different format). Such developers will need a detailed understanding of the relevant format specifications (Chapters 2.2, 2.3) and might also benefit from studying how edge cases are handled, for example by the CIF API (Chapter 5.2). If the goal is to transform CIF data into another syntactic representation, Chapter 2.6 might also provide some useful ideas.

Developers of applications that validate against dictionaries, or transform data based on relationships in the dictionaries (*e.g* to separate or merge data values and their standard uncertainties, or to convert matrices into lists of their individual components) should understand the DDL specifications in Chapter 2.4. They may also be interested in implementing relational methods expressed in the dREL language (Chapter 2.5).

1.1.4.3. System developer

In this context, a system developer is one who aims to provide an end-to-end workflow, with tools that can take full advantage of all existing CIF features. Such an ambitious programmer should read all the specification chapters of Part 2, and the introductory discussions of general principles in programming (Chapter 5.1), dictionary construction (Chapter 3.1) and dictionary maintenance (Chapter 4.1).

Plans to develop re-usable application programming interfaces or software libraries should be informed by consideration of the CIF API (Chapter 5.2) or existing libraries (Chapter 5.3). It may also be useful to read the other chapters in Part 5 to identify existing programs that may be incorporated into the developing pipeline, ported to a more convenient programming language, or that may provide ideas on end-user function and usability.

1.1.5. The database manager

Databases may be required to ingest CIF data from a variety of sources, and so must be able to handle input conformant with any format specification they support (Chapters 2.2, 2.3). They may wish to incorporate existing filter programs into their workflow (Chapter 5.4), but they may also need to write a complete system to handle ingest, validation and database loading. In such a case, they may wish to build their own application programming interface, perhaps informed by the approach of the reference CIF API (Chapter 5.2) or by some of the design decisions of existing CIF libraries (Chapter 5.3).

Validation programs may be built to check all the constraints and relations expressed in dictionaries through the DDL (Chapter 2.4) and dREL (Chapter 2.5) languages. For correct interpretation of the ingested data, the definitions presented in all the relevant dictionaries (Part 4) must be studied. Data for a complex structure or system may be spread across several data blocks within the same file, and Chapters 3.3 and 4.3 provide guidance on how this might be presented.

Approaches to validation and, indeed, overall design and control of workflows of existing databases are discussed in depth in the chapters of Part 7.

1.1.6. The non-crystallographer

Here we mean the software developer charged with implementing a pathway between crystallographic data and some other scientific domain with its own data representation standards. The introductory Chapter 1.2 will be useful in contextualizing the need for interoperable data management protocols, and in emphasizing the need to work within the FAIR principles (Wilkinson *et al.*, 2016).

A detailed understanding of the relevant CIF format specifications is needed (Chapters 2.2, 2.3), and some study of existing alternative syntaxes may be useful (Chapter 2.6).

The most significant task, however, is a detailed understanding of the semantic content associated with each data name. Here the developer must understand the formalism in which the CIF dictionaries are constructed (DDL, Chapter 2.4), and must carefully study the definition and attributes of any data items that will participate in the inter-domain traffic. Chapter 4.1 provides an overview of the entire CIF ontology, while the individual dictionaries contributing to this are presented in the remaining chapters of Part 4. Although the developer may not be interested in the scientific relevance of individual data items, it may still be useful to read the commentaries on each dictionary in Part 3 in order to understand restrictions on how they may be used (as well as amplifying some of the dictionary definitions for the benefit of the non-specialist).

The discussion of NeXus/HDF5 in Chapter 2.7 may be a useful case study of the interaction of crystallographic information with a more general multi-discipline standards and software environment.

1.1.7. The dictionary developer

The dictionary developer should have a reasonable understanding of the dictionary file format (Chapter 2.2) and dictionary definition languages (DDL, Chapter 2.4; dREL, Chapter 2.5) that define the dictionary formalism. When contributing to COMCIFS-managed dictionaries, familiarity with the layout style rules (Section 2.4.2.6) is also desirable.

Chapter 3.1 provides a detailed account of the general principles of dictionary writing, applicable both to local dictionaries and those intended for wider distribution under the aegis of the IUCr or wwPDB. Chapter 4.1 provides more details of integrating dictionaries into the ecosystem of canonical dictionaries managed for the IUCr by COMCIFS. This chapter also provides an overview of existing dictionaries, and how they compose an overall crystallographic ontology.

Chapter 3.3 and its companion dictionary Chapter 4.3 are important for understanding complex structures or systems that are described across several distinct data blocks within the same file.

Developers extending an existing dictionary will of course familiarize themselves with the existing dictionary content in Part 4, as well as the detailed commentary on each to be found in Part 3. Developers of new dictionaries may find useful the sections describing the design decisions behind each existing dictionary, also discussed in the chapters of Part 3.

References

Wilkinson, M. D., Dumontier, M., Aalbersberg, IJ. J. *et al.* (2016). *The FAIR Guiding Principles for scientific data management and stewardship*. *Sci. Data*, **3**, 160018. https://doi.org/10.1038/sdata.2016.18.