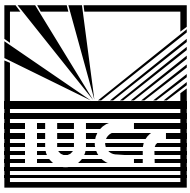


COMCIFS Dictionary Writing Workshop
A Satellite Workshop to the XXVI IUCr Congress

Organised by COMCIFS, the IUCr Committee for the Maintenance of the CIF Standard

The workshop will provide an introduction to writing and working with CIF dictionaries. Topics covered will include an introduction to CIF dictionaries, writing individual definitions for existing dictionaries, writing new dictionaries for topic areas, creating dREL methods, and writing software that uses CIF dictionaries.

The CIF dictionaries underpin the crystallographic data management framework that has been so successful over the last 30 years. The IUCr benefits as this workshop aims to grow the pool of competent CIF dictionary authors, which is more important than ever as the previous generation of CIF authors moves toward retirement.

COMCIFS Dictionary Writing Workshop

A Satellite Workshop to the XXVI IUCr Congress

Introduction

Welcome to the 2023 Dictionary Writing Workshop organised by the IUCr Committee for the Maintenance of the CIF Standard (COMCIFS).

CIF, the Crystallographic Information Framework, was developed by the International Union of Crystallography in 1991 initially as a standard file format for the exchange of data between crystallographic software applications, journals and databases. The formal definitions of terms that could be used in these files were collected in so-called 'dictionaries', and the terms and their attributes were stored in machine-parsable form using the same syntax as the data files. This meant that applications that could read the data files could also read the associated dictionaries, and so it was possible to develop validating programs that could handle extensions of the original standard by parsing and validating against new dictionaries or extensions of existing ones. In time, these definitions were also deployed in binary or otherwise formatted files that carried the same content, and so CIF became a framework for standards development and not just another file format.

This is the third workshop to be organised by COMCIFS to support the next-generation Crystallographic Information Framework. The first, held at the University of Warwick, UK, in 2013, introduced the extended CIF2 syntax and the prototype dictionary definition language (DDLm) that was then under development for more complex methods-based validation. The second, held in Hyderabad, India, in 2017, explored the implementation of CIF in various file formats, and developed the skills for constructing dictionaries for new domains using both DDLm and the relational dictionary definition language DDL2 prevalent in macromolecular structural science. The current workshop will refine DDLm skills at a time when the IUCr will reissue the canonical CIF dictionaries maintained by COMCIFS in conformance with the now mature DDLm standard.

This workshop will develop the skills needed for developing and working with CIF dictionaries. Included in this booklet are drafts of chapters in the forthcoming revised edition of the reference volume *International Tables for Crystallography, Volume G: Definition and exchange of crystallographic data*, which participants are encouraged to review and comment on.

Contents

Provisional timetable

Timings may change according to the needs and wishes of participants.

9.25am	Opening (James Hester)
9.30am	Introduction to the CIF Ontology (Brian McMahon)
10.00am	The relational underpinnings of CIF (James Hester)
10.45am	**Morning tea**
11.00am	Structure of a CIF dictionary (Brian McMahon)
11.30am	Dictionary creation exercise (James Hester)
12.30pm	**Lunch**
2.00pm	Dictionary creation exercise – conclusion
2.30pm	Using Github (Matthew Rowles)
	Using Github: some simple workflows
4.00pm	**Afternoon tea**
4.30pm	Strategies for writing CIF software
5.00pm	Discussion

This is a draft of a forthcoming chapter of *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data*, 2nd edition (in preparation).

3.1. Creating and expanding CIF dictionaries

By James R. Hester[1] and Brian McMahon[2]

3.1.1. Introduction

Much of the power and usefulness of the Crystallographic Information Framework (CIF) arises from the existence of a comprehensive set of curated data dictionaries that define all data items commonly used in the field. These are the dictionaries that are presented in Part 4 of this volume, and the contents of which are described and annotated in this Part. We call this corpus of curated dictionaries the 'CIF ontology', using a term which has become widespread in data science to describe a structured description of concepts, terminologies and relationships within a scientific discipline. The information contained in a data file (in any format) is expressed in terms of these data items. A data item consists of one or more values associated with a data name, or tag, which is a unique character string that is the key to the definition of the data value in the dictionary. The data name may appear explicitly in a data file, as it does in CIF-format files, or values in a file may be associated with data names through a specification external to the data file.

A data definition may include information such as a text description of the quantity, its physical units, the range within which valid values must lie, the names of other data items that are related to the data item and so on. Placing this information in a dictionary file, rather than in the data file itself, has a number of important advantages. First, it encourages the standardization of unique tags for data items, which is an essential step towards the seamless and unambiguous exchange of information. Curated dictionaries also facilitate a globally accepted understanding of what each data item is, and thus ensure that different data files using the same tags have a consistent interpretation.

This chapter will discuss general principles behind the design of CIF dictionaries, including important considerations when defining data names. It is directed at users who wish to compile new data definitions, whether acting autonomously to develop a novel area of study or accommodate a new software package, or as part of a community effort to increase the scope of the CIF ontology (see Section 4.1.2.2). It will describe how to construct dictionaries, how to extend existing dictionaries, and how local extensions may be built and used in ways that do not conflict with the need for community standards. Chapter 4.1 will describe the infrastructure maintained by the International Union of Crystallography (IUCr) for developing and distributing curated dictionaries under the aegis of its Committee for the Maintenance of the CIF Standard (COMCIFS). Dictionaries of data definitions specific to biological macromolecular structures are maintained separately by the Worldwide Protein Data Bank (wwPDB).

3.1.1.1. DDL versions

Ideally, compatibility between the data dictionaries originating from specific subdisciplines would be ensured by the adoption of the same attribute sets for data items. However, at this point in the evolution of the CIF standard, two slightly different attribute sets have become established. These are expressed in two versions of the dictionary definition language, DDLm and DDL2 (detailed in Sections 2.4.2 and 2.4.3, respectively). The core data items in crystallography must of course be accessible across the field, and so there are two formulations of the dictionary of core items, one in each DDL version.

An older DDL, DDL1, is now deprecated. Most dictionaries outside the domain of biological macromolecular crystallography were originally written in DDL1, but have now been rewritten in DDLm. DDL1 should not be used for new dictionaries. Information about DDL1 is available in Chapter 8.3.

Section 2.4.1.1 ('The dictionary data model') should be consulted for an introduction to the terminology and the relational data model used by the dictionaries. The current chapter focuses on the creation of DDLm dictionaries, although the general approach may be helpful for writing DDL2 dictionaries as well. Section 2.4.2 and Chapter 4.13 may be consulted as a reference for the individual meanings of the DDLm attributes.

3.1.2. Creating new dictionaries

3.1.2.1. Introduction

Dictionaries are generally associated with distinct areas of endeavour, are created and curated with the involvement of area specialists, and are likely to involve dozens of new data names. A dictionary will often be associated with a distinct set of software tools and

[1] James R. Hester, Australian Nuclear Science and Technology Organisation, New Illawarra Road, Lucas Heights, NSW 2234, Australia; [2] Brian McMahon, International Union of Crystallography, 5 Abbey Square, Chester CH1 2HU, England.

may be driven by a need to standardize communication between these software tools. Where dictionaries build on other dictionaries, they are likely to involve addition of new key data names to existing categories [see Section 2.4.1.1 and item (iii) of Section 3.1.2.2 below], and changes to the way that some data names are calculated. For example, the powder dictionary adds a phase (constituent) identifier to many categories, and the method of calculation of structure factors differs substantially from the core dictionary. If all proposed data names are either additions to existing categories or no new key data names are added to existing categories, addition of data names to existing dictionaries should be considered instead.

The dictionary development strategy introduced in tutorial form in Section 3.1.2.3 below begins with a very general approach that may be employed in the construction of any relational ontology, either within or outside the CIF paradigm.

3.1.2.2. Types of data names

For convenience in the following discussion, we classify data items into four different but overlapping types: derived, observational, keys and identifiers. Derived and observational data names are mutually exclusive classifications.

(i) Derived. *Derived* data items have values that can be derived using the values of other data items and well-known constants and relationships. Definitions for these data items should include enough information to allow their values to be re-derived.

(ii) Observational. If the value of a data item cannot be derived from other values, it is classed as *observational*. Examples of such data items include equipment settings, lists of experimenters, and raw measurements. The observational status of a data item may change if subsequent data items are defined that allow derivation of that data item: the 'observational' status of a data item is simply a practical distinction to aid dictionary construction.

(iii) Keys. In the relational model (Section 2.4.1.1) *key* data items are those whose combined values allow a unique row of a table to be referenced. All tables have a key, because a key consisting of all data items in the table will always identify a particular row (as rows may not be repeated in the relational model). However, in most cases a subset of the tabulated values suffices to identify any table row uniquely. A key data item may be observational, derived, or an identifier, or may be created especially to serve as a key – so-called 'synthetic keys'. In the simplest case, a single data item, often an identifier (see below) can act as a key.

(iv) Identifiers. *Identifiers* serve to identify particular instances of a class of things or concepts. The actual values of 'identifier' data items are ideally irrelevant, as the values are only used to distinguish or label different members. Examples of identifier data items include measurement points, serial numbers, atom sites, sample numbers and run numbers. Identifiers are also used to group objects which themselves have identifiers, in which case the identifier both labels the group and organises the individuals into the group: for example, a molecule may be assigned an identifier, and the constituent atom labels, which are themselves identifiers, are associated with that identifier to group them into the molecule.

While numbers are often used for identifier data values because it is easy to pick a unique value if we label sequentially, their numerical properties should not be used; if an identifier value contains more information beyond uniqueness, for example, it is used in calculations, then we run the risk that a situation will arise where the same value should be assigned to distinct items, and so our values can no longer serve as identifiers. For example, we may decide to identify image frames in a data collection by numbering sequentially from zero, with each frame corresponding to a small uniform change in a sample orientation axis. If we use the image number multiplied by the axis step to get the axis value, we can no longer cope with a situation where the same orientation was recollected.

3.1.2.3. A dictionary development strategy

Dictionary design is fundamentally about the distribution of a list of potentially linked data items across one or more tables. The following strategy is applicable for the development of any relational model; however the CIF terminology of data names (table column names) and categories (tables or relations) has been followed for simplicity. Exchange of text files, ideally mediated through a modern version-control system, will simplify management of the development process.

Although this strategy is laid out as a linear sequence of steps, later steps will often inspire revisions to previous decisions, so the process is better viewed as iterative.

3.1.2.3.1. Step 1. Data granularity

Data always appear in containers. These containers may be files, or logical divisions within files. Containers may themselves be collected into other containers,

for example files within a directory. Section 3.1.4.1 discusses the significance and use of containers in more detail.

A CIF dictionary describes the contents of the lowest-level container, while controlling aggregation of containers into larger collections. Container contents are defined by the items within them that may only take a single value: for example, a data block may contain information about a single compound, or a single experiment, or a single set of experimental conditions. Containers are then aggregated when describing multiple compounds, or experiments, or experimental conditions.

For this first step, decide what the topic or topics of a single data block will be. Where the new dictionary will build on existing dictionaries, this topic choice will already be partially determined. For example, the core CIF dictionaries describe a data container that includes information from an experiment conducted at a single wavelength from a single sample, using a single atomic model, so any dictionaries importing the CIF core dictionary are restricted in the same way.

3.1.2.3.2. Step 2. Develop a graph of data names

For simplicity 'data names' are referred to here, but at this stage focusing on the concepts is recommended, to avoid unnecessary disagreement about the meaning of a particular term. Where such disagreements arise, it is likely that several overlapping meanings need to be disentangled and assigned separate names.

Firstly, under a heading corresponding to the 'topic' of your data block chosen in the previous step, list short phrases or words describing concepts that relate to that topic. Particular attention should be paid to writing a plural noun where more than one of the labelled concept could be associated with the headline concept. For example, if the headline topic is 'a scientific presentation', then an associated concept would be the plural 'authors', as there could be multiple authors.

Next, repeat the process described in the previous paragraph, this time treating each of the plural nouns as a new heading. Do the same for the singular nouns, in those cases where additional information is necessary to properly describe the concept. For example, if 'room' is associated with 'presentation', then additional information for 'room' might include 'location', 'capacity', and 'equipment'. Continue this iterative process until no new concepts can be added. Ways of generating new data names are listed at the end of this section.

The result of the process outlined above can be represented as a graph with lines connecting the headline topic to the associated phrases or words. Plurals and singular nouns with extra information form new nodes in this graph. Leaf nodes are those nodes that only have a single line connecting them with the rest of the graph. This graph representation will be referred to below.

The following are useful ways to generate potential data names for inclusion in the graph:

(i) Identify relevant objects and their properties. For example, 'an experimenter' may have properties 'name', 'address', 'role', 'photograph'.

(ii) Nouns in data name definitions (see next step) are often sources of more objects for the previous step.

(iii) Locate identifiers and consider whether finer classification of the identified objects would reduce duplication. For example, instead of labelling each measured point in a sequence of scans with a unique identifier, it is likely to be more practical to label each point with both a scan identifier and an identifier for that point that is unique only within the scan. By doing this, all data names whose values do not change in a single scan can be tabulated by scan identifier instead of being repeated for every measurement point.

Such 'group' identifiers become useful if each identifier is expected to have many values in a given data file *and* there are properties that are fixed for each value of the identifier. For example, within a single scan the scan step or certain axis settings might be constant. In other cases the need to group items together arises from calculations that use a group of values, for example describing the Fourier transform of an XAFS scan requires that a scan identifier be created and assigned to each point in the scan so that the points for each Fourier transform can be identified.

(iv) Look at the data files that are already used in the field and identify the concepts used in them. Note that every scientifically useful value in a data file can and should be assigned to a data name.

(v) Finally, go through the list of data names and identify all the derived data names. The remaining data names are observational. Note which of these observational items are identifiers.

3.1.2.3.3. Step 3. Consolidate the graph

Check the graph of data names for repeated concepts. These can be consolidated by simply removing all but one of the duplications and redirecting lines to the remaining node. For example, when describing a con-

ference, a session might have a list of presenters, and a list of chairpersons, and so the initial graph might include a link between session and presenters, and session and chairs. However, both presenters and chairpersons are people. Therefore a single node, perhaps called 'person', has two links to 'session'; one identifies the chairpersons, and the other the presenters.

3.1.2.3.4. Step 4. Link categories using key data names

Each of the non-leaf nodes in the graph representation created in the previous step will eventually become a CIF category. In order for a node to be a proper category, it is sufficient that each line connecting nodes is a mathematical function, that is, if an arrow points from A to B, then a particular 'a' in A determines a particular 'b' in B. Therefore, as a first step, every line in the graph from the previous step should become an arrow, representing a function. Where this is not possible a separate 'glue' table needs to be created (see below). Each of these arrows corresponds to a data name in the source category. Note that the target of the arrow is not the data name, but the arrow itself. This is evident from our previous example of two links between a conference session and a list of people: one of those links would be the 'presenter' data name and one of those links would be the 'chair' data name.

In order to decide how a particular 'a' in 'A' is identified, so that it can be mapped to a particular 'b' in 'B', data names forming the key that uniquely determine an item in that category must be chosen or created. These key data names of a category, and the data names within the category that are used to refer to other categories, together form the basic structure of the dictionary.

In many cases the role of key data name will be performed by pre-existing identifiers for items in the category. For example, atomic elements have standard names that serve to identify that element. Before choosing key data names in this way, the possibility that they could fail to uniquely identify an item in the category should be considered carefully; a first name/family name combination would not uniquely identify an author if the pool of possible authors is reasonably large. In such cases, it is sufficient to invent a new data name whose sole purpose is to act as a key data name; in the above example 'author id' might be created to uniquely identify an author. Similarly, for a list of measurements the key data name might be a 'measurement identifier'.

Identifiers may also be created in order to reduce the number of key data names – so-called 'synthetic' identifiers. For example, in order to identify an atomic bond, two atomic sites in the asymmetric unit, two symmetry transformations applied to those sites, and potentially cell translations in all three directions for each atom should be specified, resulting in ten key data names. Any other categories referring to bonds must define ten of their own data names in order to link to the category listing the bonds. If a synthetic 'bond id' identifier is created, those other categories can identify bonds much more concisely.

As mentioned above, transforming a line in our graph into an arrow representing a function may not be possible. For example, while authors are associated with scientific papers, it is not possible to determine a unique paper given an author, nor is it usually possible to determine a unique author given a paper. In such cases an 'associative table' is created, where the two key columns take values from each of the items that we are trying to link. In our author example, one column would identify scientific papers, while another column identifies an author. The complete author list of a paper is then specified by repeating the paper identifier and providing a different author identifier on each row.

At the end of this step each non-leaf node should have a set of key data names assigned to it that allow items within it to be uniquely identified. Similarly, if a category 'A' refers to another category 'B', data names will have been created in 'A' to allow 'A' to refer to 'B' by giving the values of key data names within 'B'.

3.1.2.3.5. Adjust data names to be computationally useful

An important reason for writing CIF dictionaries is to convey information in a way that is manipulable by computer. The collection of data names should now be adjusted with reference to the following points.

(i) Any data name that has values that are free text strings (*e.g.* 'sample description') is placing information out of reach of reliable automated processing. Therefore, where information that could be used in computation appears in free or formatted text, the data name definition should be changed (potentially creating additional data names) so that values are either numeric or drawn from a set of strings.

For example, instead of a data name 'location', with a description of position in an experimental measurement chain left up to the software author, values of 'monitor': before the sample; 'detector': after sample; 'foil': after sample and calibration foil might be chosen, allowing software to be written to automatically determine the appropriate detector data to process.

(ii) A situation sometimes arises where multiple data names refer to multiple occurrences of a single concept, for example, 'gas 1', 'gas 2' to refer to the component gases in a gas mixture. Where there is no in-principle restriction to the number of such component names (for example, there may be three or more gases possible in a mixture), these names should be transformed by creating an identifier that labels each specific combination of the components.

To carry out this procedure, a new key data name that will be used to identify combinations of values for these duplicate data names is created. A second key data name is created that draws from the values of the original duplicated data names. Instead of the original data names, a single data name is created that refers to the data name that identifies particular combinations. Example 3.1.2.1 shows how this transformation appears in practice.

As a result of this transformation, an arbitrary number of component items can be accommodated. Note that this transformation is not necessary where multiplicity is intrinsically restricted, such as for the two atoms at each end of a bond or the three atoms defining a bond angle.

(iii) Commonly-occurring combinations of values should be bundled together. Where a set of data names is expected to take the same set of values, a separate identifier can be assigned to each set of values and these values are then replaced by a data name holding the value of that identifier (see Example 3.1.2.2).

(iv) Choose specific units. Unit choices at data file creation time create extra work for the file reading software in anticipating every possible unit that is appropriate. If the community has not converged on a particular unit, a second definition differing only in the unit used should be created.

(v) Avoid software-specific names. Any data name that refers to the input or output of a software package calculation has value in proportion to the number of people with access to the version of the software in question, or to the extent to which the software setting/output can be linked to specific calculations through documentation or source code. Given this, the value of such data names is likely to decline rapidly over time. Therefore, where such data names appear in the provisional list, they should be rephrased in non-software-specific terms. For example, instead of 'multiplicity as calculated by package *XYZ* Version 1.2', write 'the number of special positions divided by the number of general positions'.

(vi) Avoid instrument-specific names. Any data name whose definition refers to the configuration of an instrument in a way that is insufficient to enable reproduction in a different lab or through independent modelling is unlikely to be of use outside the lab that produced it. So, instead of 'Position of motor mom', 'monochromator takeoff angle' could be written. Of course, a large

Example 3.1.2.1. *Replacing repetitive data names with an associative table: an example using ion chamber gas mixtures.*

This example considers ion chamber detectors used at synchrotrons, which adjust sensitivity to the X-ray beam running through them by adjusting the gas or mix of gases in them. Each of these ion chamber detectors must be described, including recording the gas mixture information.

Initial data names are: 'gas mix', the mixture of gases in an ion chamber, in format *element-percent-element-percent*; 'detector length', length of the ion chamber; and 'location', the location of the ion chamber relative to sample and foil; with key data name 'detector id'. A tabulation using these data names might be:

detector id	gas mix	detector length	location
BB25	He-50-N-50	5	monitor
XYZ	Ar-100	5	detector
Old-G	Ar-100	10	foil

As noted in point (i) of Section 3.1.2.3.5, the gas mix definition embeds data items into the value, essentially making them unavailable elsewhere in our ontology. To remedy this, we create data names 'first gas', 'first gas percent', 'second gas' and 'second gas percent' (leaving out the other two columns for now):

detector id	first gas	first gas %	second gas	second gas %
BB25	He	50	N	50
XYZ	Ar	100	.	.
Old-G	Ar	100	.	.

Now we are in the situation described by point (ii). The gases and gas percentages are of the same type (with the same key data name), and in a situation where three or more gases are used we would need to define new data names. Therefore we create a new table that will describe gas mixes. First, we create an identifier 'gas mix id' and replace the original identifier data names 'first/second gas' by 'gas', which will have a value of 'gas mix id', dropping also 'first/second gas percent' as this information will now be added in our new table. Apart from 'gas mix id', the new table will require 'percentage' and 'gas name'. Now, given a detector, it is sufficient for us to nominate the gas mix id to completely identify the gas components. We can now tabulate all of our mixes in an associative table:

gas name	gas mix id	gas percentage
Ar	C	100
He	A	50
N	A	50

And we can now describe our detectors as follows:

detector name	detector gas mix id	detector length	location
BB25	A	5	monitor
XYZ	C	5	detector
Old-G	C	10	foil

As a result of this transformation, we can describe an arbitrary number of gases in a detector; that is, our data description has acquired robustness against future changes.

Example 3.1.2.2. *Grouping commonly-occurring values.*

Suppose that we plan to store a set of raw data images using 'raw image id', 'encoding type' and 'compression type' as key data names, using 'compressed data' to hold the data:

raw image id	encoding type	compression type	compressed data
image_1	"signed 16-bit integer"	none	AABRAAAAAAAA…
image_2	"signed 16-bit integer"	none	AABRAAAAAAAA…
…	…	…	…
image_656	"signed 32-bit integer"	none	AAZBQSr1sKTbq7bg…
image_657	"signed 32-bit integer"	none	AAZBOeOe9HTdMdDg…
…	…	…	…
image_4206	"signed 32-bit integer"	packed	8RJ1vKqAvxYDMD6…
image_4207	"signed 32-bit integer"	packed	r/tgsjMZoL0AEc4KigE…

However, we expect only one or two possible alternative encodings. Therefore, only a few combinations of 'compression type' and 'encoding type' will be present in any given collection of images, and the same combinations are likely to be repeated many, many times if we expect hundreds of images. So we create a new key identifier 'byte array construction id' and make this the key data name for 'encoding type' and 'compression type':

byte array construction id	encoding type	compression type
construct_1	"signed 16-bit integer"	none
construct_2	"signed 32-bit integer"	none
construct_3	"signed 32-bit integer"	packed

We add 'construction id' as a key data name for 'compressed data' in place of 'compression type' and 'encoding type'. Now we can list the few combinations of compression and encoding against 'construction id', and match the appropriate value of 'construction id' with 'raw image id' and 'compressed data'.

raw image id	byte array construction id	compressed data
image_1	construct_1	AABRAAAAAAAAAAAA…
image_2	construct_1	AABRAAAAAAAAAAAA…
…	…	…
image_656	construct_2	AAZBQSr1sKNTbq7bg…
image_657	construct_2	AAZBOeOe9HITdMdDg…
…	…	…
image_4206	construct_3	8RJ1vKqAvx9YDMD6…
image_4207	construct_3	r/tgssjMZobL90AEXc4KigE…

In this case we have saved one column of repetition.

facility may choose to create a dictionary for in-house use in which case such definitions might be sufficient for internal purposes when combined with local knowledge.

3.1.2.3.6. Step 6. Finalise the text definitions

A text definition for each of the items identified in the previous steps is written, conveying unambiguously to a human reader the following three things:

(i) a description of the data name understandable to a scientist working in the field;

(ii) the nature of the data values (*e.g.* arbitrary identifier, real number, text, vector, integer, 'true'/'false', image) chosen from the list of DDLm types (Table 2.4.2.1);

(iii) how to interpret this data name given the values of the varying key data names.

Well-defined terminology from the field and references to literature should be used to keep the definition short.

At this point overlooked data names may become apparent. Nouns in the definitions that do not correspond to any data name usually indicate objects that have been overlooked. Identifiers are often associated with indefinite nouns, for example 'an image', 'a measurement', or 'an experimenter'.

3.1.2.3.7. Step 7: Naming

Data names in CIF are generally constructed in the form `_<category>.<object>`, where `<category>` is the category name and `<object>` is an arbitrary identifier. This naming strategy is advantageous as (i) data names that are closely related will often be close when listed alphabetically, and (ii) it will be easy for a human reader to recognise which key data names a given data name is related to.

Names should be chosen for each of the categories found in the earlier steps, and then `<object>` parts for each data name created. Abbreviated names should generally be avoided as they are potentially confusing for human readers: 'temp' may be short for 'temperature' or 'temporary'. However, although the CIF 2.0 specification lifts the earlier 75-character length restriction on data names (Bernstein *et al.*, 2016), many data names in the original CIF dictionaries used abbreviated terms to stay within that limit. If it is desired to use abbreviations to extend or parallel existing data names, such abbreviations should be consistent with those already in use (Table 3.1.2.1).

CIF has the restriction that a data name may only appear once in a data block; therefore where a key data name appears in multiple categories, a distinct name is created for each category and its relationship to the other data names indicated using the parent–child DDLm attributes. In terms of the graph representation described above, the parent key data name is the data name in the category at the destination of the arrow on the graph.

Where a dictionary is created solely to serve a single application, no further considerations are necessary. However, if it is to be used alongside other dictionaries, some care must be taken to avoid possible name clashes. Section 4.1.5.1 describes the use of a `[local]_` prefix convention to ensure that no attempt is made by public applications to interpret these data names, and Section 4.1.5.2 describes a procedure for reserving other prefix components to avoid name clashes. Section 3.1.5 of this chapter addresses the case where name clashes can occur, *e.g.* by the merging of dictionaries from different subdisciplines.

Table 3.1.2.1. Abbreviations in CIF data names

Terms for which abbreviations are defined are sometimes found unabbreviated.

Abbreviation	Term	Abbreviation	Term	Abbreviation	Term
abbrev	abbreviation	eqn	equation	oper	operation
abs	absolute (configuration, not structure)	esd	standard uncertainty (estimated standard deviation) (*see* su)	org	organism
absorpt	absorption			orient	orientation
alt	alternative	expt	experiment	origx	orthogonal coordinate matrix (PDB files)
amp	amplitude	exptl	experimental	os	operating system
AN	accession number	fom	figure of merit	param	parameter
anal	analyser	fract	fractional	pd	powder diffraction
aniso	anisotropic*	Fsqd	F squared	PDB	Protein Data Bank
anisotrop	anisotropic*	gen	generation	PDF	Powder Diffraction File
anom	anomalous	gen	generator	perp	perpendicular
ASTM	American Society for Testing and Materials	gen	genetic	phos	phosphate
asym	asymmetric	geom	geometric	pk	peak
atten	attenuation	H-M	Hermann–Mauguin	polarisn	polarization
au	arbitrary units	ha	heavy atom	poly	polymer
auth	author	hbond	hydrogen bond	pos	position
av	average	hist	history	prep	preparation
ax	axial	horiz	horizontal	proc	processed
B	B form of atomic displacement parameter (a.d.p.)	I	intensity	prof	profile
		ICSD	Inorganic Crystal Structure Database	prot	protein
backgd	background*	id	identifier	ptnr	partner
beg	begin	illum	illumination	publ	publication
bg	background*	imag	imaginary	R	agreement index
biol	biology	inc	increment	rad	radius
bkg	background*	incl	include	recd	received
bond	bonding	info	information	recip	reciprocal
Bsol	B form of a.d.p. for solvent	instr	instrument	ref	reference
calc	calculated	Int	international	refine	refinement
calib	calibration (pd)	ISBN	International Standard Book Number	refln	reflection
cartn	Cartesian	iso	isotropic	reflns	reflections
CAS	*Chemical Abstracts* Service	iso	isomorphous	res	resolution
char	characterization (pd)	ISSN	International Standard Serial Number	restr	restraints
chem	chemical	IUCr	International Union of Crystallography	rev	revision
chir	chirality	IUPAC	International Union of Pure and Applied Chemistry	Rmerge	agreement index of merging
clust	cluster			rms	root mean square
coef	coefficient	len	length	rot	rotation
com	common	lim	limit	S	goodness of fit
comp	component	loc	lack of closure	samp	sample
conc	concentration	ls	least squares	scat	scattering factor
conf	conformation	max	maximum	seq	sequence
config	configuration	MDF	Metals Data File	sigI	$\sigma(I)$*
conform	conformant	meanI	mean intensity	sigmaI	$\sigma(I)$*
conn	connectivity	meas	measured	sint	$\sin\theta$
cons	constant	mid	middle (between max and min)	sint/lambda	$\sin(\theta)/\lambda$*
CSD	Cambridge Structural Database	min	minimum	sol	solvent
db	database	mod	modification	spec	specimen
defn	definition	mods	modifications	src	source
detc	detector	mon	monomer	std	standard
der	derivative	monochr	monochromator (pd)*	stol	$\sin(\theta)/\lambda$*
dev	standard deviation	mono	monochromator (pd)*	struct	structure
dict	dictionary	nat	natural	su	standard uncertainty
dif	difference*	NBS	National Bureau of Standards (now National Institute of Standards and Technology)	suppl	supplementary
diff	difference*			sys	systematic
diffr	diffractometer			tbar	mean path length
diffrn	diffraction	NCA	number of connected atoms	temp	temperature
displace	displacement	ncs	noncrystallographic symmetry	tor	torsion angle
dist	distance	netI	net intensity	tran	transformation*
divg	divergence	NH	number of connected hydrogen atoms	transf	transformation*
dom	domain	nha	non-hydrogen atoms	transform	transformation*
dtime	deadtime	norm	normal	tvect	translation vector (PDB files)
ens	ensemble	nst	nonstandard	vert	vertical
eq	equatorial*	nucl	nucleic acid	wR	weighted agreement index
equat	equatorial*	num	number	wt	weight
equiv	equivalent	obs	observed		

* Terms with multiple definitions.

3.1.2.3.8. Step 8: child categories

The DDLm language allows a category to be the child of another category (Section 2.4.2.1.2). A child category is essentially a single category that has been split into two or more separate categories for efficiency. This is done when one or more of the data names in the category will potentially have missing or null values for a significant subset of the rows in the category. For example it is often the case that only a subset of atoms has refined anisotropic displacement parameters. The child category contains only those data names whose values are expected to be present or absent as a group.

When constructing the dictionary, a child category is created by splitting a category into two categories, with the leaf nodes divided between the two categories. The key data names are duplicated, with the child category versions becoming the children of the parent category versions. The child category is that category which is expected to have fewer rows, as described in the previous paragraph.

3.1.2.3.9. Summary

At the conclusion of the steps outlined above, a complete set of proto-definitions for a CIF dictionary will have been prepared. At the final stage, these definitions are rewritten in the DDLm language (Section 2.4.2). As a general guide, the decisions made in the previous steps are encoded as follows:

(i) Categories for which only one value of all of their key data names are permitted in a data block (see 3.1.2.3.1) have **_definition.class** of 'Set', otherwise it is 'Loop'.
(ii) Data names whose values are drawn from the values of another data name (typically identifiers) have **_name.linked_item_id** set to that parent data name.
(iii) The text description is given in **_description.text**.
(iv) The data name is given in **_definition.id**.
(v) The type is given in **_type.contents** and **_type.container**.
(vi) Derived items have **_type.source** of 'Derived'.
(vii) Key data names are listed under **_category_key.name**. Including key data names in 'Set' category definitions allows multi-data-block data sets to be covered (see Section 3.1.4.1).
(viii) Key data names also have a **_type.purpose** of 'Key'.
(ix) The category is given in **_name.category_id**, and the object part in **_name.object_id**.

Further attributes may be required depending on the particular type of the data name.

3.1.3. Considerations when writing data definitions

A general strategy for creating data definitions has been outlined in Section 3.1.2.3. Here we provide some concrete examples that illustrate how the strategy has been applied in existing canonical dictionaries, and we discuss some of the subtleties that can arise in practice.

3.1.3.1. Definitions of single quantities

Example 3.1.3.1 is the core dictionary definition of the data name for the ambient pressure during the experiment. The fact that this is a single value (as opposed to a vector or matrix quantity) is expressed by assigning the value 'Single' to its container type.

The type of the associated data value ('Real' to indicate a real number) is specified, together with indications of its status through use of the attributes **_type.purpose** and **_type.source**, indicating that it is a numerical value that has been recorded by measurement or derivation. The allowed numeric range is specified ('0.0:' indicates that it may be any non-negative real number) and the physical units of the quantity are given.

Although data names are usually constructed by concatenation of the category name with a specific object identifier, these components must be explicitly identified using the **_name.category_id** and **_name.object_id** attributes. It is also good practice for tracing the provenance of data to record the date at which the definition was last updated. Permissible alternative forms of the data name are listed as aliases.

Example 3.1.3.1. *A simple definition of a data item describing a physical quantity.*

```
save_diffrn.ambient_pressure
  _definition.id   '_diffrn.ambient_pressure'
  _alias.definition_id
                   '_diffrn_ambient_pressure'
  _definition.update   2023-01-13
  _description.text
;
   Mean hydrostatic pressure at which
   intensities were measured.
;
  _name.category_id     diffrn
  _name.object_id       ambient_pressure
  _type.purpose         Measurand
  _type.source          Recorded
  _type.container       Single
  _type.contents        Real
  _enumeration.range    0.0:
  _units.code           kilopascals
save_
```

Example 3.1.3.2. *A definition of a data item describing the standard uncertainty associated with another item.*

```
save_diffrn.ambient_pressure_su
  _definition.id   '_diffrn.ambient_pressure_su'

  loop_
    _alias.definition_id
       '_diffrn_ambient_pressure_su'
       '_diffrn.ambient_pressure_esd'

  _definition.update   2021-03-03
  _description.text
;
  Standard uncertainty of the mean hydrostatic
  pressure at which intensities were measured.
;
  _name.category_id    diffrn
  _name.object_id      ambient_pressure_su
  _name.linked_item_id
                '_diffrn.ambient_pressure'
  _type.purpose        SU
  _type.source         Recorded
  _type.container      Single
  _type.contents       Real
  _units.code          kilopascals
save_
```

The **_description.text** attribute is a concise human-readable documentation of the meaning associated with the data name.

Note that as an experimentally recorded value (purpose 'Measurand'), the recorded value may (and ideally will) have a standard uncertainty (s.u.) that may be appended in parentheses in CIF-format files. A separate data item must be defined to hold a s.u., in this case **_diffrn.ambient_pressure_su**. The definition of this linked property should have a **_type.purpose** value of 'SU' and a **_name.linked_item_id** value of the data name to which it relates (see Example 3.1.3.2).

A further real-world consideration is that the experimental quantity (in this case the ambient pressure) may not have been recorded directly, but is known to be within some range. To accommodate this possibility, additional data items have been defined (**_diffrn.ambient_pressure_gt** and **_diffrn.ambient_pressure_lt**), the 'gt' and 'lt' suffixes indicating 'greater than' and 'less than' limits respectively. As estimated values, these are assigned a **_type.purpose** value of 'Number' and so do not have associated standard uncertainties.

Example 3.1.3.3 is taken from the powder dictionary and illustrates a data item that can have only one of a limited set of values. This data item indicates the geometry of the experiment. The associated data value is of type 'Code' and may legally take only one of the two possible values listed. Note the type source value of 'Assigned' to indicate that this is a parameter chosen to determine the course of an experiment.

Example 3.1.3.3. *A data item that can take only one of a discrete set of allowed values.*

```
save_pd_spec.mount_mode
  _definition.id       '_pd_spec.mount_mode'
  _alias.definition_id '_pd_spec_mount_mode'
  _definition.update   2014-06-20
  _description.text
;
  A code describing the beam path through
  the specimen.
;
  _name.category_id    pd_spec
  _name.object_id      mount_mode
  _type.purpose        Encode
  _type.source         Assigned
  _type.container      Single
  _type.contents       Code

  loop_
      _enumeration_set.state
         reflection
         transmission
save_
```

3.1.3.1.1. Looped data

The attributes of looped data items, such as their physical units or valid numerical values, are defined in exactly the same way as for non-looped data. The relationships between different looped data items are determined by the definition of the category to which they belong.

Consider the following example listing of some three-dimensional atom-site coordinates and displacement parameters.

```
loop_
_atom_site.label
_atom_site.fract_x
_atom_site.fract_y
_atom_site.fract_z
_atom_site.U_iso_or_equiv
_atom_site.thermal_displace_type
 O1  .4154(4)  .56990(10)  .3026000    .0600(10)  Uani
 C2  .5630(5)  .5087(2)    .32460(10)  .060(2)    Uani
 C3  .5350(5)  .4920(2)    .39970(10)  .0480(10)  Uani
 N4  .3570(3)  .55580(10)  .4167000    .0390(10)  Uani
 C5  .3000(5)  .6122(2)    .35810(10)  .0450(10)  Uani

loop_
_atom_site_aniso.label
_atom_site_aniso.U_11
_atom_site_aniso.U_22
_atom_site_aniso.U_33
_atom_site_aniso.U_12
_atom_site_aniso.U_13
_atom_site_aniso.U_23
O1  .071(1)  .076(1)  .0342(9)  .008(1)   .0051(9)
-.0030(9)
C2  .060(2)  .072(2)  .047(1)   .002(2)   .013(1)
-.009(1)
C3  .038(1)  .060(2)  .044(1)   .007(1)   .001(1)
-.005(1)
N4  .037(1)  .048(1)  .0325(9)  .0025(9)  .0011(9)
-.0011(9)
C5  .043(1)  .060(1)  .032(1)   .001(1)  -.001(1)
.001(1)
```

```
loop_
_geom_bond.atom_site_label_1
_geom_bond.atom_site_label_2
_geom_bond.distance
   O1  C2   1.342(4)
   O1  C5   1.439(3)
   C2  C3   1.512(4)
   C2  O21  1.199(4)
```

These loops, or tables of values, are properties of atom sites, each identified by a label such as O1. The definition of the ATOM_SITE category makes clear that the items in the category should be looped together, and that **_atom_site.label** is the key data name that ensures uniqueness of each table row (Example 3.1.3.4).

Example 3.1.3.4. *Extract from category definition showing the expected loop structure and category key.*

```
save_ATOM_SITE
  _definition.id        ATOM_SITE
  _definition.scope     Category
  _definition.class     Loop
  _definition.update    2023-02-03
  _description.text
;
  The CATEGORY of data items used to describe
  atom site information used in
  crystallographic structure studies.
;
  _name.category_id     ATOM
  _name.object_id       ATOM_SITE
  _category_key.name    '_atom_site.label'
save_
```

The anisotropic atomic displacement parameters are properties associated with atom sites, and could legitimately be appended to each row of the ATOM_SITE table. However, they have traditionally been presented in journals as separate tables, and this presentational aspect is facilitated by creating the ATOM_SITE_ANISO category as a child of the ATOM_SITE category, as shown in Example 3.1.3.5.

Example 3.1.3.5. *Definition of a child category, which can be presented as a standalone loop structure or folded into its parent category.*

```
save_ATOM_SITE_ANISO
  _definition.id        ATOM_SITE_ANISO
  _definition.scope     Category
  _definition.class     Loop
  _definition.update    2023-01-16
  _description.text
;
  The CATEGORY of data items used to describe
  the anisotropic atomic displacement
  parameters of the atomic sites in a crystal
  structure.
;
  _name.category_id     ATOM_SITE
  _name.object_id       ATOM_SITE_ANISO
  _category_key.name    '_atom_site_aniso.label'
save_
```

The key data name for this category (which is **_atom_site_aniso.label**) differs from that of the parent ATOM_SITE category, but is understood to be equivalent to **_atom_site.label** through the category parent–child relationship.

3.1.4. Describing data in multiple blocks

In the simplest case, a complete data set is contained within a single data block where all data names are described in a single dictionary. In more complex data sets information about an experiment or model may be distributed over multiple data objects. For example, calibration information may be provided as a separate measurement in a separate file, while having an important, machine-actionable relationship to the primary data and therefore forming part of the complete data set. The following sections describe how dictionaries can be created and used to describe data spread over multiple files or objects.

3.1.4.1. Foundations

The following discussion is arranged around the concept of a 'data container'. A data container encapsulates a collection of data items, and may include nested data containers. Typical data containers include files, levels in logical hierarchies within a single file, and directories in filesystems. When using the CIF format, a data container is either a save frame or a data block. A 'data set' is defined as the top level in this container hierarchy, encompassing all nested containers. The formal relationship of each data container to the complete data set is defined below using 'projection' and 'scoping' operations.

The complete data set may always be modelled as a set of relational tables. These tables may be algorithmically decomposed into data containers using 'projection', where the operation of 'projecting over <data name>' is defined as choosing in turn each value of <data name>, and retaining only those rows in any tables for which <data name> takes the chosen value. This operation extends to all child data names of <data name> taking that same value, and any tables not containing <data name> or its children remain unchanged. When this procedure is repeated for every value of <data name>, a series of non-overlapping, internally consistent sub-tables is produced. Each of these subtable collections corresponds to a separate data container. The projection operation can be recursively applied to each of the collections, using different data names for the projection, to replicate further levels of encapsulation. It follows that any data container that

does not constitute the entirety of a data set is a projection of the complete data set over one or more data names.

The encapsulation provided by data containers allows the use of scoping semantics to simplify the data container contents. In programming languages, the 'scope' of a variable refers to the region of code where the value of that variable is well-defined and accessible. By analogy, the 'scope' of a single-valued data name is defined as the data container within which it appears. Thus, when all the values of a data name in a table are identical, that data item can be provided instead as a single value outside the table but within the container, and its value for every row of that table remains unambiguous. When combining this scoping rule with the projection operation described above, both the projected data name and its children can be left out of the projected tables and replaced by a key–value pair consisting of the parent data name and value. Figure 3.1.4.1 illustrates how projection and scoping are used to distribute table contents between separate data blocks.

Dataset descriptions often evolve from a starting point that describes common cases encapsulated in a single data container. Thus, the original starting point for the core CIF dictionary was the description of a diffraction experiment conducted at a single wavelength on a single sample at a single set of diffraction conditions, contained within a single data block. Description of data sets containing more complex data, such as data collected at multiple wavelengths, then requires that either the specification of the original data block contents is expanded, with all data continuing to appear in a single data container, or that multiple data containers are permitted. The latter choice is equivalent to preserving the implicit data projection and scoping that applied to the original data container, and is preferable in practice so that software designed for the original data description will continue to correctly interpret the component data blocks of the whole data set.

Consideration of Figure 3.1.4.1 reveals that, in order to reconstruct the complete data set from constituent containers, it is sufficient to append each relational table to its equivalents in the other data containers after reinstating any key data names that have been dropped by the scoping rule. This procedure is iterated until all projected data names have been dealt with and the full tables for the complete data set have been reconstructed. In practice, a data set may have been divided into separate containers for space or handling reasons, and actually reconstituting it would be unwieldy. The reconstruction procedure described here is thus not a requirement for software to construct large internal tables; rather, any operations that the software performs should be consistent with such a data structure.

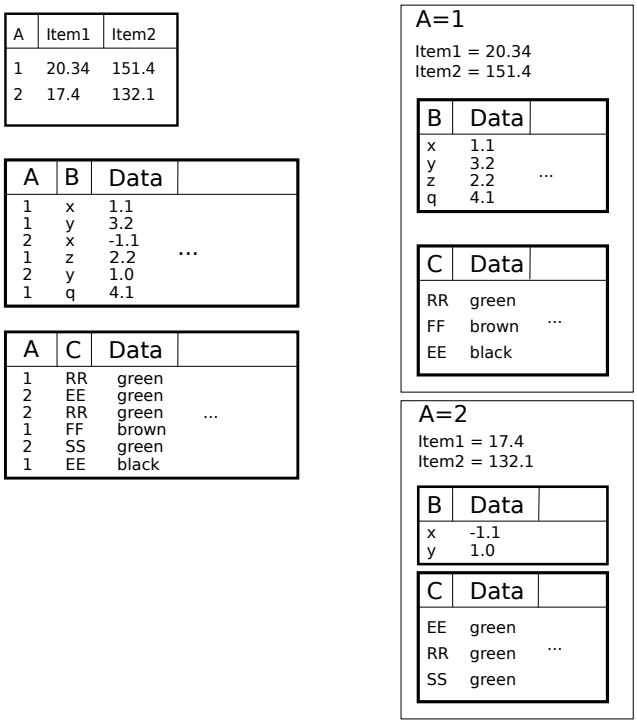


Fig. 3.1.4.1. Using projection and scoping to split a set of tables into data blocks. The right-hand side of the diagram is the result of projecting the tables in the left-hand side over the values of column 'A', resulting in two data blocks, one for each value of 'A'. Values from the top-most table can be presented as key–value pairs in the projected data blocks and column 'A' is dropped as values for each row of column 'A' are known for each data block. The original tables can be reconstituted as long as the implicit presence of column 'A' in all tables is specified. In a CIF context, columns labelled 'A' would be given distinct names and the relationship with other columns labelled 'A' indicated using **`_name.linked_item_id`**.

3.1.4.2. DDLm attributes for expanded data sets

The DDLm dictionary language used in IUCr dictionaries divides categories into single-valued 'Set' categories and multi-valued 'Loop' categories, where 'single-valued' refers to the number of values in the lowest-level data container. It follows from the previous section that each of these 'Set' categories is the result of projecting a notional 'Loop' category over one or more of the Loop category's key data names to obtain a single row, putting separate values of the key data name into separate data blocks, and then dropping any child data names from related Loop categories using the scoping rule. DDLm descriptions of multi-container data make the projection and scoping relationship explicit by adding key data names to the 'Set' categories and defining children of these key data names that were previously only implicitly present in Loop categories.

Simply redefining 'Set' categories to become 'Loop' categories is not recommended when dealing with an expanded data description. In this case, legacy software

is likely to incorrectly process data block contents: for example, density would be incorrectly calculated when each atom occurs more than once in the ATOM_SITE list owing to inclusion of atom positions from several different compounds. To minimise the possibility of such errors, a special CIF data name **_audit.schema** has been introduced. Non-default values of this dataname within a data block indicate that data names from one or more 'Set' categories in the core dictionary now appear in loops, with additional key data names added to other loops as described above. Each official **_audit.schema** value corresponds to a particular collection of these looped 'Set' categories. In general, it is desirable to restrict the number of such 'schema' in order to simplify software development, instead distributing complex data sets over multiple data blocks. Nevertheless, **_audit.schema** remains important for dealing with legacy data set descriptions and differences between the ways in which domains conventionally group their data. In particular, the mmCIF and imgCIF dictionaries each assume distinct collections of 'Set' categories that differ from core CIF.

The CIF framework does not include a comprehensive standard for specifying which data containers belong to a particular data set. Instead, a number of facilities are available to help determine the validity and completeness of a given group of data blocks. Firstly, each data block may be assigned a universally unique identifier (UUID) using the **_audit.block_id** data name, following which data blocks can refer to other data blocks forming part of the data set at the time that it was created using data names from the AUDIT_LINK category. Secondly, the relational structure of CIF data containers allows the consistency of any given data container aggregation to be checked independently of these audit data names. This is because each data container contains slices of an overall set of tables, and so after reconstruction of these tables from the individual containers key data names should either never repeat their combined values, or else if they do repeat combined values all other values in the same row for the same columns must be identical.

3.1.4.3. Expanding dictionaries to describe data spread over multiple data blocks

As discussed above, enhancing a dictionary to allow data spread over multiple data blocks is equivalent to adding key data names to one or more Set categories, with attendant addition of child data names to the relevant pre-existing Loop categories. Such a dictionary will usually build on definitions found in a base dictionary designed for a single data block. The following steps outline an approach to creating such a dictionary.

(i) The 'Head' category of the new dictionary should contain an import statement (Section 2.4.2.2.4) that imports the base dictionary with 'mode'='Full'. This has the semantic effect of replacing the 'Head' category in the imported dictionary with the 'Head' category of the importing dictionary, creating a composite dictionary that contains all of the old definitions, with some category definitions rewritten according to the steps below. If any definitions are overwritten (for example, a category has new key data names added), the **_import_details.if_dupl** flag in the import specification must be set to 'Replace'.

(ii) Definitions adding to pre-existing categories or developing new categories are created as usual.

(iii) Definitions of new key data names for any 'Set' categories are added.

(iv) Definitions for those Set categories from the previous step are rewritten with the **_category_key.name** attribute added pointing to the key data name defined in the previous step.

(v) Child data names of the data names defined in step (iii) are defined for all categories in the base dictionary that implicitly assumed a single value of any data name in the relevant 'Set' categories.

(vi) Category definitions for each of the Loop categories with a new child data name from the previous step are rewritten to include this new data name in the list of category keys.

3.1.4.4. Examples

3.1.4.4.1. Symmetry dictionary

The symmetry extension dictionary (see Chapter 3.12) adjusts definitions to permit multiple space groups to be used within a single data set. As a consequence, any categories that assume a single space group must have their category key expanded to include a child data name of the new space group identifier data name, **_space_group.id**. Such child data names are defined and added to categories SPACE_GROUP_SYMOP, SPACE_GROUP_WYCKOFF, CELL, REFLN, MODEL_SITE, GEOM_ANGLE and CELL_MEASUREMENT in the extension dictionary. As the information is distributed over several data blocks, the value of **_space_group.id** can be stated as a key–value pair within each data block and then the child data names in the above categories need not be included owing to the scoping rules, and **_audit.schema** takes its default value as the core CIF set of unlooped categories is preserved.

3.1.4.4.2. Modulated and composite structures dictionary

The modulated and composite structures dictionary (Chapter 3.6) includes definitions for describing struc-

tures formed from multiple components. When the component structures are listed within separate data blocks, **_cell_subsystem.code** takes a single value for each data block and the default **_audit.schema** value is used.

3.1.4.4.3. Powder diffraction dictionary

Powder diffraction projects routinely collect and model data from samples containing mixtures of compounds. In powder diffraction, each of these constituents is called a 'phase'. The final model often includes calculated reflection intensities for peaks from each phase, but the REFLN category from the core dictionary (see Section 3.2.3.2.1) assumes that a single *hkl* value is sufficient to identify a reflection and so cannot accommodate the multiple instances of the same reflection that might be required when multiple phases are present in the REFLN loop. The powder dictionary therefore extends the core dictionary by adding a new key data name to the REFLN category that is linked to the powder phase identifier. Thus, each set of reflections (together with structural information) for each phase can either be listed in separate data blocks containing different values for **_pd_phase.id**, or listed together in a single data block with **_audit.schema** set to the appropriate non-default value, typically 'Custom'.

3.1.4.4.4. Advanced example: DDL dictionaries as data files

A dictionary save frame contains data about data names. A dictionary is a self-consistent collection of these data containers. By the arguments above, each save frame is necessarily a projection of some category over a key data name. In the case of DDL dictionaries, that key data name is the DDL attribute used to state the data name being defined [**_definition.id** (DDLm) or **_item.name** (DDL2)]. In keeping with the scoping rule, child data names of this attribute need not be included in any loops appearing in the definition: thus the DDLm attribute dictionary does not include these child attributes, whereas DDL2 explicitly defines them. So, for example, **_enumerated_set.item_name** is an attribute that could appear in DDL2 ENUMERATED_SET loops with a value that is always the item being defined, but it does not appear in definitions in the published dictionaries as its value is unambiguous.

Following the procedure described in Section 3.1.4.1, a DDLm dictionary can be transformed into a single, large table for which the data name is the key. DDL2 dictionaries transform instead into two separate tables as they use separate categories for category and item names.

All dictionaries within an ontology may be further aggregated into a single set of tables by recognising that each dictionary projects the DICTIONARY category over its key data name.

3.1.5. Combining and extending existing dictionaries

A full description of some data sets requires drawing on data names from more than one dictionary. For example, a data file describing the data and results from a powder diffraction experiment on a modulated structure using an imaging detector would use data names from the modulated structures dictionary, the powder diffraction dictionary, and the image CIF dictionary. In the simplest case, dictionaries may be combined by simply associating each of them with the data file (see Section 3.1.6). In more complex cases, an overall dictionary using the import mechanism (Section 2.4.2.2.4) must be created.

Directly using data names from multiple dictionaries in the data file is appropriate when no categories or data names have been explicitly or implicitly redefined in any of the dictionaries. This condition is fulfilled if (i) any common categories have the same key data names, and (ii) the data file contains no data names defined in more than one of the source dictionaries. Typically this situation arises where the domains of the dictionaries are largely separate; for example, the image dictionary covers raw data and the core dictionary covers reduced and processed data, with only the DIFFRN category in common. Note that dictionaries that conform to the same **_audit.schema** value (see Section 3.1.4.2) will automatically satisfy criterion (i) above. Where the above criteria cannot be met, a new dictionary must be built through importation of the component dictionaries and addition or replacement of definitions.

In the context of data manipulation the 'meaning' of a data name can be divided into two aspects. The first aspect relates to how values of this data name are used, or the 'downstream' meaning of the data name. The second aspect relates to how values of this data name are derived, or the 'upstream' meaning. COMCIFS requires consistency in the downstream meaning of data names defined in dictionaries that it administers. New dictionaries may not, therefore, redefine a data name in a way that would render existing uses of values of that data name incorrect. Thus, while the way in which **_refln.F_calc** (the calculated structure factor) is determined will vary depending on the model, **_refln.F_calc** values found in data files may be used and interpreted identically regardless of the model (for example, to calculate a difference density). Similarly, a number of approaches exist for deriving observed intensity for a *hkl* spot from raw diffraction data, but that intensity, once derived, is used identically regardless of

derivation method in downstream calculations.

From the above discussion it follows that new dictionaries change only the 'upstream' meaning of one or more imported data names. This change usually arises from a change in the model underlying derived values, whether owing to a change in experimental technique or an enhancement of the structural model. Historically, changes in the upstream meaning of data names have not always been documented in dictionaries owing to the unchanged downstream use of the values found in data files; however, documenting variations in the upstream meaning of a data name is important for enabling reproducible science.

The dREL language (chapter 2.5) has been introduced as a way of capturing the upstream meaning of data names in a machine-readable way. Where a dREL method exists for a data name, a convenient way of determining whether or not a redefinition is required in a new dictionary is to check whether the dREL method needs to be changed. Importantly, dREL methods do not require changes where new key data names have been added to a category that simply expose relationships that were previously implicit. For example, although the powder diffraction dictionary adds **`_pd_phase.id`** as a new key data name to a number of categories, the dREL methods that access items in these categories remain identical for those dREL methods that are located in categories that have the same new key data name. This is described in detail in point (ii) of Section 2.5.2.3. In other words, because many categories in the single-crystal dictionary already implicitly described a particular compound (or 'phase' in powder diffraction terminology), making this dependency explicit by adding a phase identifier to the category does not change the substance of any calculations and therefore does not trigger the need to redefine every data name in those categories.

3.1.5.1. Creating a combined dictionary

Given the previous discussion one possible procedure for constructing a dictionary that extends existing dictionaries is as follows:

(i) Create a Head category that imports all the necessary existing dictionaries (see Section 2.4.2.2.4). It is not necessary to repeat imports of dictionaries that are imported within other imported dictionaries. Note that most dictionaries already import the core dictionary.
(ii) Add definitions for any new categories and their data names
(iii) Add definitions for any new non-key data names belonging to existing categories
(iv) Where an existing category has acquired key data names, define the expanded **`_category_key.name`** in a new definition for that category and define the new key data name.
(v) Repeat the previous step for any categories that refer to the expanded existing category until no such categories are left. A category references another category either when key data names share common parents, or **`_name.linked_item_id`** refers to a data name in that category.
(vi) Determine any pre-existing derived data names whose derivation methods are different relative to the imported dictionaries. Provide new definitions for each of these data names, explaining the new derivation method.

When data spread over multiple blocks is anticipated, the above procedure can usefully be combined with the procedure described in section 3.1.4.3.

3.1.6. Linking data to dictionaries

Software that attempts to validate a data file against the relevant CIF dictionaries needs to be able to identify and locate those dictionaries. This is sometimes achievable only through the context in which a data file has been acquired. For example, files provided with a powder diffraction paper are likely to be using the powder dictionary, files in a magnetic database use the magnetic dictionary *etc*. Sometimes a suitable heuristic exists for guidance; for example, the presence of **`_pd_*`** names in a data file indicates the powder dictionary.

However, formal machine-readability or adherence to the FAIR principles (Findability, Accessibility, Interoperability and Reusability) of data management (Wilkinson *et al.*, 2016) requires an explicit link between data names and dictionaries. The core CIF dictionary provides the AUDIT_CONFORM category to provide such a link and allow the identification and retrieval of the dictionaries relevant to any data file (see Section 3.1.6.1 below).

Where a simple listing of dictionaries is not sufficient, perhaps because of conflicting definitions, it may be necessary to create an extension using the methods of Section 3.1.5.

A dictionary merging protocol was described in the previous edition of this volume to resolve potential conflicts (McMahon, 2005), but is not appropriate for the handling of DDLm dictionaries.

3.1.6.1. Identification of dictionaries relevant to a data file

To permit automatic validation against the dictionaries used in constructing a CIF data file, the file should declare within each of its data blocks the names, version numbers and, where appropriate, locations of the

canonical and local dictionaries that contain definitions of the data names used in that block. The relevant identifiers are the items **_audit_conform.dict_name**, ***.dict_version** and ***.dict_location**, defined in the core dictionary.

The values of the items **_audit_conform.dict_name** and **_audit_conform.dict_version** are character strings that match the values of the **_dictionary.title** and **_dictionary.version** identifiers in the dictionary that defines the relevant data names. Validation against the latest version of a dictionary should always be sufficient, since every effort is made to ensure that a dictionary evolves only by extension, not by revising or removing parts of previous versions of the dictionary. Nevertheless, including **_audit_conform.dict_version** is encouraged: it can be useful to confirm which version of the dictionary the CIF was initially validated against.

The data item **_audit_conform.dict_location** may be used to specify a file name or uniform resource locator (URL). However, a file name on a single computer or network will be of use only to an application with the same view of the local file system, and so is not portable. A URL may be a better indicator of the location of a dictionary file on the Internet, but can go out of date as server names, addresses and file-system organization change over time. The preferred method for locating a dictionary file is to make use of a dynamic register, as described in Section 4.1.2.3. Nevertheless, **_audit_conform.dict_location** remains a valid data item that may be of legitimate use, particularly in managing local applications.

The following example demonstrates a statement of dictionary conformance in a data file describing a powder diffraction experiment with some additional local data items:

```
loop_
_audit_conform.dict_name
_audit_conform.dict_version
_audit_conform.dict_location
    cif_core.dic      2.3.1    .
    cif_pd.dic        1.0.1    .
    cif_local_my.dic  1.0
       /usr/local/dics/my_local_dictionary
```

It is clear that the location specified for the local dictionary is only meaningful for applications running on the same computer or network, and therefore the ability to validate against this local dictionary is not portable. On the other hand, it may be that the local data names used by the authors of this CIF are not intended to have meaning outside their own laboratory.

3.1.6.2. Locating a dictionary for validation

The following protocol applies to the creation and use of software designed to locate the dictionaries referenced by a data file and validate the data file against them. The protocol is necessary to address the issues that arise because dictionaries evolve through various audited versions, because not all dictionaries referenced by a data file may be accessible, and because data files might not in practice contain pointers to their associated dictionaries.

Software source code for applications that use CIF dictionaries to validate the contents of data files should be distributed with a copy of the most recent version of the register of dictionaries, and with the URL of the master copy hard-coded. Library utilities should be provided that permit local cacheing of the register file and the ability to download and replace the cached register at regular intervals. Individual dictionary files located and retrieved through the use of the register should also be cached locally, to guard against temporary unavailability of network resources.

Each CIF data file should contain a reference to one or more dictionary files against which the file may be validated. At the very least this will be **_audit_conform.dict_name** (*N*). The other data items **_audit_conform.dict_version** (*V*) and **_audit_conform.dict_location** (*L*) are optional. In the event that no dictionaries are specified, the default validation dictionary should be that identified as having *N* = CORE_DIC and *V* = '.' (*i.e.* the most recent version of the core dictionary). Since dictionaries are intended always to be extended, it is normally enough just to specify the name (and possibly the location).

A software application validating against CIF dictionaries should attempt to locate and validate against the referenced dictionaries in the order cited in the data file, according to the following procedure. The terms 'warning' and 'error' in this procedure are not necessarily messages to be delivered to a user. They may be handled as condition codes or return values delivered to calling procedures instead.

If *N*, *V* and *L* are all given, try to load the file from the location *L*, or a locally cached copy of the referenced file. If this fails, raise a warning. Then search the dictionary register for entries matching the given *N* and *V*. (An appropriate strategy would be to search a locally cached copy of the register, and to refresh that local copy with the latest version from the network if the search fails.) If a successful match is made, try to retrieve the file from the location given by the matching entry in the register (or a locally cached copy with the same *N* and *V* previously fetched from the location

specified in the register). If this fails, try to load files identified from the register with the same *N* but progressively older versions *V* (version numbering takes the form *n.m.l* . . ., where *n*, *m*, *l*, . . . are integers referring to progressively less significant revision levels). Version '.' (meaning the current version) should be accessed before any other numbered version. If this fails, raise a warning indicating that the specified dictionary could not be located.

If *N* and *V* but not *L* are given, try to load locally cached or master copies of the matching dictionary files from the location specified in the register file, in the order stated above, *viz*: (i) the version number *V* specified; (ii) the version with version number indicated as '.'; (iii) progressively older versions. Success in other than the first instance should be accompanied by a warning and an indication of the revision actually loaded.

If only *N* is given, try to load files identified in the register by (i) the version with version number indicated as '.'; (ii) progressively older versions.

If all efforts to load a referenced dictionary fail, the validation application should raise a warning.

If all efforts to load all referenced dictionaries fail, the validation application should raise an error.

For any dictionary file successfully loaded according to this protocol, the validation application must perform a consistency check by scanning the file for internal identifiers (`_dictionary.title`, `_dictionary.version`) and ensuring that they match the values of *N* and *V* (where *V* is not '.'). Failure in matching should raise an error.

3.1.7. References

Bernstein, H. J., Bollinger, J. C., Brown, I. D. *et al.* (2016). *Specification of the Crystallographic Information File format, version 2.0. J. Appl. Cryst.* **49**, 277–284. *https://doi.org/10.1107/S1600576715021871*.

COMCIFS (2006). *Terms of Reference.* https://www.iucr.org/resources/cif/comcifs/terms-of-reference

COMCIFS (2014). *IUCr Committee for the Maintenance of the CIF Standard (GitHub repositories).* https://github.com/COMCIFS

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. *https://doi.org/10.1107/S010876739101067X*.

McMahon, B. (2005). In *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data*, edited by S. R. Hall & B. McMahon, Section 3.1.9, pp. 88–89. Dordrecht: Springer, 1st ed.

Spadaccini, N. & Hall, S. R. (2012). *DDLm: A new dictionary definition language. J. Chem. Inf. Model.* **52**(8), 1907–1916. *https://doi.org/10.1021/ci300075z*.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. 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*.

This is a draft of a forthcoming chapter of *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data*, 2nd edition (in preparation).

4.1. Management and use of CIF dictionaries

By James R. Hester[1] and Brian McMahon[2]

4.1.1. Introduction

This part of the volume presents the definitions and most significant attributes of terms in the set of dictionaries over which the International Union of Crystallography (IUCr) exercises control. This canonical set forms the ontology that is the basis of the Crystallographic Information Framework.

Here we include in Chapter 4.10 the biological macromolecular CIF dictionary (mmCIF) that was originally developed under the aegis of the IUCr. This has subsequently been greatly expanded and enhanced with additional dictionaries covering biological structure determination and reporting, and the so-named mmCIF/PDBx family of dictionaries is now managed and maintained by the Worldwide Protein Databank, who provide a rich set of documentation resources on their website (https://mmcif.wwpdb.org), which should be consulted for current information. Nevertheless, the early version of the mmCIF dictionary and its description (Chapter 3.10) are retained here to provide an insight into the essentials of the macromolecular structure description and its relationship to the core CIF model.

This introductory chapter discusses the administrative machinery established by the IUCr to manage these canonical dictionaries. It describes how to locate, retrieve and handle single or aggregated dictionaries appropriate to different disciplinary needs. It establishes a high-level view of the different categories of definitions that contribute to the overall conceptual model of the subject. It concludes with guidance on developing local extensions to the framework if needed for specific purposes.

4.1.2. Administration of canonical dictionaries

4.1.2.1. The role of COMCIFS

The body known as COMCIFS (the Committee for the Maintenance of the CIF Standard) was established in 1993 at the XVI Congress of the IUCr in Beijing, China. At that time the only data definitions in use were those specified in the original CIF paper (Hall *et al.*, 1991). Its role was defined as ratifying extensions to the set of CIF approved data names and the commissioning and guidance of experts in other areas to construct new CIF dictionaries appropriate to their subject. It subsequently undertook responsibility for managing and extending the CIF format specification (Bernstein *et al.*, 2016), and the adoption and implementation of a methods-enabled dictionary definition language (DDLm) based on an initial proposal of Spadaccini & Hall (2012).

Its current terms of reference (COMCIFS, 2014) emphasize its primary role as maintaining the integrity of the Crystallographic Information Framework, and its responsibility to extend the crystallographic ontology into other related disciplines.

COMCIFS operates as an advisory subcommittee of the IUCr Executive Committee, to which it reports directly. As such, it represents a high level of authority invested by the Union in the standardization project that it represents.

The technical work for which COMCIFS was originally responsible is now largely undertaken by delegated dictionary maintenance groups or technical working groups (see Section 4.1.2.2).

The Committee now consists of a small core of voting members responsible for directing overall policy. In a spirit of openness, it welcomes a larger group of interested participants who act as observers, but are free to participate in the majority of COMCIFS discussions, which are conducted through email discussion lists and publicly archived on the IUCr web site.

4.1.2.2. Developing community standards

The core CIF dictionary was developed to describe concepts relevant to all areas of crystallography, as well as the central activity of single-crystal structure determination by diffraction methods. However, crystallography and closely neighbouring structural sciences cover a wide range of subdisciplines, each with different community norms and modes of practice. COMCIFS has sought to work with these various communities to initiate and ratify new subdiscipline-specific dictionaries.

[1] James R. Hester, Australian Nuclear Science and Technology Organisation, New Illawarra Road, Lucas Heights, NSW 2234, Australia; [2] Brian McMahon, International Union of Crystallography, 5 Abbey Square, Chester CH1 2HU, England.

Often these have been started or encouraged by IUCr Commissions, but sometimes they have been the initiative of individual research groups. Dictionary authoring efforts ideally include representatives of all stakeholders in the relevant community. A typical stakeholder group might include software authors, domain experts, publishers and database managers. A diverse author group ensures that any issues with the dictionary are detected quickly, and therefore barriers to subsequent adoption are eased. It is particularly helpful if target software applications can be modified to allow live testing of proposed new dictionary content as it is developed.

Early dictionary extensions were carried out in an *ad hoc* manner by specialist groups, usually collaborating by email. Once their dictionaries received formal COMCIFS approval, dictionary management groups were created with dedicated discussion lists. Archives of these list discussions are also on the IUCr web site.

Since 2014, COMCIFS has provided a shared standards development area on GitHub, a major code repository platform (COMCIFS, 2006), to allow collaborative work on CIF dictionaries and related software. This mode of operation has several advantages. It allows all workers on a common project to have a common view of the progress of the work, and new participants may join at any time and see a complete history of all work carried out to date. All changes are tracked automatically by the repository software, and retrieval of any prior version is possible. Versions of the dictionary or software code may be forked, allowing for development of different approaches. Issues may be raised and tracked in online structured discussion. Further, the ability to visit different repositories provides a useful survey of how different groups are approaching their specific projects.

4.1.2.3. Distribution infrastructure

COMCIFS retains the responsibility of trying to harmonize the treatment of similar data requirements in different dictionaries and to maintain maximum compatibility between data files originating from different subdisciplines. To achieve this, COMCIFS can officially approve dictionaries submitted to and reviewed by it. It is these approved dictionaries that are included in this volume. Provisional dictionaries may also be issued and used within the relevant community before formal approval is given.

The GitHub site described in Section 4.1.2.2 is an important platform for developing new dictionaries and extensions to existing dictionaries, and can usefully function as a source for provisional releases.

However, formally approved dictionaries are then published from the network services of the IUCr. The CIF section of the IUCr web site provides links to current and archived versions of approved dictionaries, with commentaries and change logs.

Since the dictionaries are machine-readable resources, it is of course useful for software to be able to download them directly from a known uniform resource locator (URL). The dictionaries are currently distributed over two network protocols, ftp and https. The use of the File Transfer Protocol (ftp) goes back to the release of the original dictionary in 1991, when that was the standard means of transferring files across the then still relatively young academic Internet. When the World Wide Web was launched around 1994, web browsers rapidly became the application of choice for retrieving distributed information. For many years, most popular web browsers supported ftp natively, so that the user could visit a URL with an ftp scheme (*e.g.* ftp://ftp.iucr.org/pub/cifdics/cif_core.dic) and view the contents immediately in the browser. However, support for this protocol was dropped from most browsers by the early 2020s, and so the IUCr now also allows for transport over the secure hypertext transmission protocol (https).

COMCIFS maintains a register of dictionaries known to it, including the identifying name and version strings within those dictionaries. In addition to COMCIFS-approved dictionaries, there are a number of dictionaries used internally by other organizations or users that are known to be properly constructed, so that this register has the potential to be a central resolver for any public dictionary. The register includes the location of each dictionary, expressed as ftp: or https: based URLs. The location of the register is https://www.iucr.org/__data/iucr/cif/dictionaries/cifdic.register and (to maintain compatibility with ftp-based applications) ftp://ftp.iucr.org/pub/cifdics/cifdic.register.

The IUCr makes every effort to retain published URLs indefinitely, but changes are sometimes forced by external circumstances (*e.g.* the dropping of native support for ftp transfer by browsers, or the expiry of a registered domain name). Consequently, since 2023, digital object identifiers (DOIs) have also been introduced for dictionaries. A DOI is a persistent identifier that can be resolved to an Internet location using a resolver service, thus allowing for changes in the end-point URL to be handled transparently within the resolver service. Registered DOIs are also included in the dictionary register. At the time of writing, the recommended resolver for these DOIs is provided by CrossRef; to link to a DOI such as 10.1107/cifdics_000001, one may prefix the address of the resolver service: https://doi.org/10.1107/cifdic_000001.

Table 4.1.2.1. CIF dictionary register (maintained as a CIF-format file). The https URLs have been abbreviated to fit into the column width. The elided part of the address is `__data/iucr/cif/dictionaries`.

```
data_validation_dictionaries
  loop_
    _cifdic_dictionary.name
    _cifdic_dictionary.version
    _cifdic_dictionary.DDL_compliance
    _cifdic_dictionary.reserved_prefix
    _cifdic_dictionary.date
    _cifdic_dictionary.URL
    _cifdic_dictionary.DOI
    _cifdic_dictionary.resource_type
    _cifdic_dictionary.description
#####################################################
# COMCIFS approved dictionaries                     #
#####################################################
cif_core.dic     .    1.4.1    .     .
  https://www.iucr.org/.../cif_core_2.4.5.dic
  10.1107/cifdic_000001  landing_page
  'Core CIF Dictionary'
          . . . . . .
cif_core.dic  2.3.1  1.4.1    .     2005-06-27
  ftp://ftp.iucr.org/pub/cifdics/cif_core_2.3.1.dic
  . .
  'Core CIF Dictionary as published in ITG edition 1'
          . . . . . .
cif_core.dic  3.2.0  4.1.0    .     2023-05-30
  https://www.iucr.org/.../cif_core_3.2.0.dic
  10.1107/cifdic_000002 dictionary
  'Core CIF Dictionary'
          . . . . . .
mmcif_std.dic    2.0.09   2.1.6  .     2005-06-27
  ftp://ftp.iucr.org/pub/cifdics/cif_mm_2.0.09.dic
  . .
  'Macromolecular CIF Dictionary (ITG edition 1)'
#####################################################
# Private dictionaries (re)distributed by the IUCr  #
#####################################################
          . . . . . .
cif_iucr.dic   1.2   1.4.1  .     2014-07-09
  https://www.iucr.org/.../cif_iucr_1.2.dic
  . .
  'IUCr private data items for journal publishing'
          . . . . . .
#####################################################
# DDL dictionaries                                  #
#####################################################
          . . . . . .
mmcif_ddl_2.1.6.dic    .   2.1.6  .     2004-04-15
  ftp://ftp.iucr.org/pub/cifdics/mmcif_ddl_2.1.6.dic
  . .
  'Relational (DDL2) dictionary definition language'

DDLm.dic    3.14.0    3.14.0   .     2019-09-25
  https://www.iucr.org/.../DDLm_3.14.0.dic
  . .
  'Methods dictionary definition language'
          . . . . . .
#####################################################
# Data items in the CIF dictionary register itself  #
#####################################################
cif_register.dic    .    1.4   .     .
  https://www.iucr.org/.../cif_register.dic
  . .
  'Data items used within the register of published CIF dicti
cif_register.dic    2.0    4.1.0   .     2023-06-20
  https://www.iucr.org/.../cif_register_2.0.dic
  . .
  'Data items used in CIF dictionary register' %
cif_register.dic    1.0    1.4   .     2005-06-24
  ftp://ftp.iucr.org/pub/cifdics/cif_register_1.0.dic
  . .
  'Data items used in CIF dictionary register' %
```

Table 4.1.2.2 shows some extracts from the current register. The data name `_cifdic_dictionary.resource_type` may take two values, 'dictionary' or 'landing_page'. The former provides a direct link to the machine-readable dictionary file. The latter is a link to an informational web page that describes the nature, purpose and version history of the dictionary. It also provides links to formatted versions of the dictionary (typeset PDF and hyperlinked HTML) and to relevant ancillary files.

By convention, an entry with resource type 'dictionary' that does not also have a specified version number will link to the current release version of the dictionary file.

4.1.3. An overview of the ontology

The collection of dictionaries approved by COMCIFS forms the crystallographic ontology expressed in the DDLm definition language. The collection of dictionaries maintained by the Worldwide Protein Data Bank forms a parallel ontology for biological structures expressed in DDL2. The conceptual framework of both ontologies is very similar, and is based on the notion of categories (which are formally equivalent to relational tables of objects and properties).

Fig. 4.1.3.1 is a very schematic 'map' of the CIF ontology maintained by COMCIFS. It sketches the families of categories in the various extension dictionaries, and indicates overlaps with the categories defined in the core dictionary. It also illustrates that mmCIF, the basis of the macromolecular structural ontology, is a superset of the core CIF dictionary.

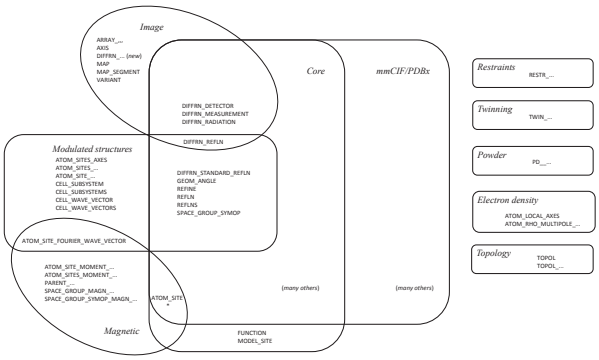


Fig. 4.1.3.1. Schematic map of ontological categories in the CIF dictionary collection. Categories are indicated in very abbreviated form; only categories in the core (and mmCIF) dictionaries that overlap with other extension dictionaries are indicated. The asterisk indicates that the magnetic dictionary defines a new category (ATOM_SITES_MOMENT) which is a proper child of the cited core category.

This figure demonstrates that, for the most part, the extension dictionaries define terms specific to a particular topic area. Only the image and modulated structures dictionaries add to a relatively small number of core categories.

This establishes that the core CIF dictionary supplies most of the information that might be needed to characterise any crystal structure, whether reported as a standalone model, or presented as a database entry or within a literature article. Hence, all these categories must be made available to a complete description of a structure additionally characterised by one or more of the extension dictionaries.

The formal way to achieve this is through the **_import.get** statement that appears in the 'Head' category of each extension dictionary (see Section 2.4.2.2.4). Fig. 4.1.3.2 shows how this is presented in the topology dictionary. See also Fig. 2.4.2.6 for an extended example (where the magnetic dictionary imports both the core and the modulated structures dictionaries) and an equivalent technique that uses the DDLm IMPORT_DETAILS category. The imported definitions are said to be 'reparented' to the Head category of the importing dictionary.

```
save_TOPOLOGY

    _definition.id                  TOPOLOGY
    _definition.scope               Category
    _definition.class               Head
    _description.text
;
    This category is the parent of all
    categories in the dictionary.
;
    _name.category_id               TOPOLOGY_CIF
    _name.object_id                 TOPOLOGY

    _import.get
    [{'file':cif_core.dic
        'mode':Full  'save':CIF_CORE}]
save_
```

Fig. 4.1.3.2. Different ways of describing imports.

In principle, each data block in a CIF data file should indicate the dictionaries that contain the definitions of the data names used in that block. This is done by using relevant items from the AUDIT_CONFORM category. Fig. 4.1.3.3 illustrates how this might appear in legacy data files (conformant to dictionaries constructed with the DDL1 formalism) and in data files using DDLm dictionaries. Note how, in the latter case, the core CIF dictionary is not included, because it is imported by the powder dictionary. The example also indicates the possibility of using 'local' dictionaries to make use of new definitions not found in the canonical set. Details of how to set these up are given in Section 4.1.5.

Notice that the location given for the local dictionary in this example is filesystem-based, and thus only meaningful for applications running on the same computer or network. This is legitimate, but means that the ability to validate against this local dictionary is not portable. On the other hand, it may be that the local data names used by the authors of this CIF are not intended to have meaning outside their own laboratory.

```
loop_
  _audit_conform_dict_name
  _audit_conform_dict_version
  _audit_conform_dict_location
    cif_core.dic       2.3.1    .
    cif_pd.dic         1.0.1    .
    cif_local_my.dic   1.0
          /usr/local/dics/my_local_dictionary
```

(*a*)

```
loop_
  _audit_conform.dict_name
  _audit_conform.dict_version
  _audit_conform.dict_DOI
  _audit_conform.dict_location
    cif_pd.dic         2.5.0
          10.1107/cifdic_00007   .
    cif_local_my.dic   1.0      .     .
          /usr/local/dics/my_local_dictionary
```

(*b*)

Fig. 4.1.3.3. The CIF dictionaries to which the data block conforms. (*a*) An example DDL1 statement in a legacy file. (*b*) The same example using DDLm formalism, with the additional ability to locate a dictionary by means of a digital object identifier.

Note that, since the AUDIT_CONFORM strategy resides in the core dictionary, there is a potential circularity: the dictionary that defines **_audit_conform.dict_*** data names needs to be loaded before those data names can be interpreted. It is not anticipated that this will cause significant problems in practice. Dictionary-aware validators may have the appropriate behaviour for taking action on AUDIT_CONFORM hard-coded as a bootstrapping procedure.

If a dictionary-aware program is dependent on loading dictionaries declared through the AUDIT_CONFORM mechanism, and these datanames are absent, then the appropriate default values should be

```
loop_
  _audit_conform.dict_name
  _audit_conform.dict_version
  _audit_conform.dict_location
    cif_core.dic        .
      ftp://ftp.iucr.org/pub/cifdics/cif_core.dic
```

where the core dictionary will be loaded; the version is not declared, so that the latest version will be fetched, and the use of an ftp-based location will also access the current version for direct download. The default location has been maintained since the early days of the CIF standard, but if transfer over the https protocol is preferred, an equivalent https location can be looked up in the dictionary register (Section 4.1.2.3).

This default procedure will load the most recent core dictionary in DDLm formalism, which will handle legacy data files based on DDL1 in a satisfactory way through data name aliasing.

4.1.4. Overview of the canonical CIF dictionaries

In this section we give a broad overview of the categories contained in each of the CIF dictionaries presented here in Part 4 of the volume. More details are given in the commentary chapters in Part 3, but the following summary may also help to orient the reader in finding relevant subject-specific data definitions, and in understanding better how the various dictionaries fit together.

This may also be considered an exercise in establishing what is needed to facilitate interoperability between related ontologies. Although these dictionaries explore different facets of crystallographic science, and so are in some sense all children of the 'core' CIF dictionary (so named to assert its central importance to the entire discipline), there is an opportunity to look at how the whole collection might interoperate with other, wider ontologies. Disciplines such as materials science, mineralogy and chemical structure are immediate examples that spring to mind, while the core dictionary already contains generic categories that are relevant to any field (author metadata, project sponsorship information *etc*.). We will not here develop strategies for wider integration – that is a future project of arbitrarily great complexity – but we present the category summaries as useful source material in seeking to build concordances or integration bridges with other subject ontologies.

4.1.4.1. The core CIF dictionary

Table 4.1.4.1 lists the categories defined in the core CIF dictionary (Chapter 4.2). With the introduction of formal parent–child category relationships, the core contents now have a clearer hierarchical structure. TFor convenience in discussion, the categories are arranged in six themes, marked in bold type in Table 4.1.4.1. A complete description of the concepts covered by this dictionary is given in Chapter 3.2.

The PUBLICATION categories have a broader compass than conventional literature publication, and include the AUDIT categories that carry general metadata about the project under which the CIF has been generated. These include details of authorship, financial sponsorship and other provenance metadata; information on links between data blocks in the same CIF; bibliographic details of resulting literature publications and database definitions; and possibly the entire discursive text of an associated publication. Most of these items are common to any scientific research project.

The EXPTL categories relate to experimental work carried out prior to diffraction measurements. While many of these are specific to crystallographic experiments (crystal preparation, space-group determination), it is clear that any other experimental science would have a parallel requirement to record preparation and set-up information about an experiment.

The DIFFRACTION categories handle the description of a diffraction experiment – still the most characteristic of crystallographic experiments – and its data collection and recording. These categories were originally developed for the typical single-crystal point-detector experiment current in the late 1980s, and have undergone considerable expansion with the emergence of new diffraction methodologies.

The STRUCTURE categories contain a variety of structure-related information, including (within the REFINE categories) information about the structure refinement. The definitions in the core dictionary are not sufficiently comprehensive to allow a re-refinement (some further information is provided through the restraints dictionary), but they do provide an indication of the methods and assumptions used to derive the reported structure. This is an essential requirement in meeting the FAIR principle (Wilkinson *et al.*, 2016) of replicability of published models.

The MODEL categories report details of the molecular or ionic structure derived from the inferred disposition of atoms in the crystal structure. Whereas the latter is represented in crystallographic coordinates, the geometry (and valence) information is directly interpretable in terms of a chemical model, easily exposing these aspects to non-crystallographic applications.

The single FUNCTION category currently provides functional relationships that ensure the integrity of intra-CIF relationships expressed with DDLm methods attributes. These data items are not generally expected to appear in data CIFs, but their appearance within the dictionary provides a strong assertion of the scientific relationships between reported data items, and so provides a strong basis for automated validation checks.

Table 4.1.4.1. Category hierarchy of the core CIF dictionary. The Head category is CIF_CORE. See text for a discussion of the grouping by theme, indicated in bold.

DIFFRACTION
- DIFFRN
  - CELL
    - CELL_MEASUREMENT
      - CELL_MEASUREMENT_REFLN
  - DIFFRN_ATTENUATOR
  - DIFFRN_DETECTOR
  - DIFFRN_MEASUREMENT
  - DIFFRN_ORIENT
    - DIFFRN_ORIENT_MATRIX
    - DIFFRN_ORIENT_REFLN
  - DIFFRN_RADIATION
    - DIFFRN_RADIATION_WAVELENGTH
  - DIFFRN_REFLN
  - DIFFRN_REFLNS
    - DIFFRN_REFLNS_CLASS
    - DIFFRN_REFLNS_TRANSF_MATRIX
  - DIFFRN_SCALE_GROUP
  - DIFFRN_SOURCE
  - DIFFRN_STANDARDS
    - DIFFRN_STANDARD_REFLN
  - REFLN
  - REFLNS
    - REFLNS_CLASS
    - REFLNS_SCALE
    - REFLNS_SHELL

EXPTL
- CHEMICAL
  - CHEMICAL_CONN_ATOM
  - CHEMICAL_CONN_BOND
  - CHEMICAL_FORMULA
- EXPTL_ABSORPT
- EXPTL_CRYSTAL
  - EXPTL_CRYSTAL_APPEARANCE
  - EXPTL_CRYSTAL_FACE
- SPACE_GROUP
  - SPACE_GROUP_GENERATOR
  - SPACE_GROUP_SYMOP
  - SPACE_GROUP_WYCKOFF

FUNCTION

MODEL
- GEOM
  - GEOM_ANGLE
  - GEOM_BOND
  - GEOM_CONTACT
  - GEOM_HBOND
  - GEOM_TORSION
- MODEL_SITE
- VALENCE
  - VALENCE_PARAM
  - VALENCE_REF

PUBLICATION
- AUDIT
  - AUDIT_AUTHOR
  - AUDIT_AUTHOR_ROLE
  - AUDIT_CONFORM
  - AUDIT_CONTACT_AUTHOR
  - AUDIT_LINK
  - AUDIT_SUPPORT
- CITATION
- CITATION_AUTHOR
- CITATION_EDITOR
- COMPUTING
- DATABASE
  - DATABASE_CODE
- DATABASE_RELATED
- DISPLAY
  - DISPLAY_COLOUR
- JOURNAL
  - JOURNAL_COEDITOR
  - JOURNAL_DATE
  - JOURNAL_INDEX
  - JOURNAL_TECHEDITOR
- PUBL
  - PUBL_AUTHOR
  - PUBL_BODY
  - PUBL_CONTACT_AUTHOR
  - PUBL_MANUSCRIPT
    - PUBL_MANUSCRIPT_INCL_EXTRA
  - PUBL_REQUESTED
  - PUBL_SECTION

STRUCTURE
- ATOM
  - ATOM_ANALYTICAL
    - ATOM_ANALYTICAL_MASS_LOSS
    - ATOM_ANALYTICAL_SOURCE
  - ATOM_SITE
    - ATOM_SITE_ANISO
  - ATOM_SITES
    - ATOM_SITES_CARTN_TRANSFORM
    - ATOM_SITES_FRACT_TRANSFORM
  - ATOM_TYPE
    - ATOM_TYPE_SCAT
- REFINE
  - REFINE_DIFF
  - REFINE_LS
    - REFINE_LS_CLASS

4.1.4.2. The restraints dictionary

Current extension dictionaries have rather 'flat' structures (*i.e.* do not have the same richness of hierarchical parent–child category relationships as the core). This largely reflects their focus on specific topics. The restraints dictionary (Chapter 4.3) provides several categories that describe constraints and restraints that can be applied within a structure refinement procedure (Table 4.1.4.2). These are described in detail in Chapter 3.3. Different software packages may restrain refinable parameters in many different ways, so it is not possible to provide definitions of all possible refinement strategies. Nevertheless, the structured description of the types of restraint applied provides guidance if attempts are made to re-refine the structure (particularly if the original software is again used). They also allow an experienced crystallographer to form a broad impression of the appropriateness of the refinement strategy used.

4.1.4.3. The twinning dictionary

This dictionary (Chapter 4.4) contains a small set of categories that describe twinning reported in a crystal

Table 4.1.4.2. Category hierarchy of the restraints CIF dictionary. The Head category is CIF_RSTR.

RESTR
RESTR_ANGLE
RESTR_DISTANCE
RESTR_DISTANCE_MIN
RESTR_EQUAL_ANGLE
RESTR_EQUAL_ANGLE_CLASS
RESTR_EQUAL_DISTANCE
RESTR_EQUAL_DISTANCE_CLASS
RESTR_EQUAL_TORSION
RESTR_EQUAL_TORSION_CLASS
RESTR_PARAMETER
RESTR_PARAMETER_CLASS
RESTR_PLANE
RESTR_PLANE_CLASS
RESTR_RIGID_BODY
RESTR_RIGID_BODY_CLASS
RESTR_TORSION
RESTR_U_ISO
RESTR_U_RIGID
RESTR_U_SIMILAR

Table 4.1.4.3. Category hierarchy of the twinning CIF dictionary. The Head category is TWIN_GROUP.

TWIN
  TWIN_INDIVIDUAL
  TWIN_REFLN

(Table 4.1.4.3). The TWIN_REFLN category partitions reflections according to the twin component with which they are associated, and so might be considered an interpretative extension of corresponding categories in the core. The remaining categories capture the methodology of twin assignment. See Chapter 3.4 for a detailed discussion.

4.1.4.4. The modulated structures dictionary

The modulated and incommensurate structures dictionary (Chapter 4.6) addresses the topic of aperiodicity in crystal structures from the viewpoints of superposition of distinct structural models, and projection into 3-space of higher-dimensional symmetries (see Chapter 3.6 for a detailed explanation). In consequence (Table 4.1.4.4), several core categories are extended by this dictionary, which also introduces some new categories that fit conceptually within core parent categories: the various ATOM_SITE_* and ATOM_SITES_* categories have parallels in the core dictionary (within the STRUCTURE theme), while the CELL_SUBSYSTEM* and CELL_WAVE_VECTOR* categories naturally belong within the core DIFFRACTION heading.

Table 4.1.4.4. Category hierarchy of the modulated structures CIF dictionary. The Head category is MS_GROUP. The symbol ¶ indicates extensions to categories already defined in the core CIF dictionary.

ATOM_SITES_AXES
ATOM_SITES_DISPLACE_FOURIER
ATOM_SITES_MODULATION
ATOM_SITES_ORTHO
ATOM_SITES_ROT_FOURIER
ATOM_SITE_DISPLACE_FOURIER
  ATOM_SITE_DISPLACE_FOURIER_PARAM
ATOM_SITE_DISPLACE_LEGENDRE
ATOM_SITE_DISPLACE_ORTHO
ATOM_SITE_DISPLACE_SPECIAL_FUNC
ATOM_SITE_DISPLACE_XHARM
ATOM_SITE_FOURIER_WAVE_VECTOR
ATOM_SITE_OCC_FOURIER
  ATOM_SITE_OCC_FOURIER_PARAM
ATOM_SITE_OCC_LEGENDRE
ATOM_SITE_OCC_ORTHO
ATOM_SITE_OCC_SPECIAL_FUNC
ATOM_SITE_OCC_XHARM
ATOM_SITE_PHASON
ATOM_SITE_ROT_FOURIER
  ATOM_SITE_ROT_FOURIER_PARAM
ATOM_SITE_ROT_LEGENDRE
ATOM_SITE_ROT_ORTHO
ATOM_SITE_ROT_SPECIAL_FUNC
ATOM_SITE_ROT_XHARM
ATOM_SITE_U_FOURIER
  ATOM_SITE_U_FOURIER_PARAM
ATOM_SITE_U_LEGENDRE
ATOM_SITE_U_ORTHO
ATOM_SITE_U_XHARM
CELL_SUBSYSTEM
CELL_SUBSYSTEMS
CELL_WAVE_VECTOR
CELL_WAVE_VECTORS
¶ DIFFRN_REFLN
¶ DIFFRN_REFLNS
¶ DIFFRN_STANDARD_REFLN
¶ GEOM_ANGLE
¶ REFINE
¶ REFLN
¶ REFLNS
¶ SPACE_GROUP_SYMOP

4.1.4.5. The electron density dictionary

The current electron density dictionary (Chapter 4.7) is restricted to a multipole expansion description of the electron density associated with atom sites in the crystal structure (Chapter 3.7). All the categories in this dictionary (Table 4.1.4.5) are associated with the properties of individual atom sites, including ATOM_LOCAL_AXES, which defines an atom-centric Cartesian coordinate system at each site.

Table 4.1.4.5. Category hierarchy of the electron density CIF dictionary. The Head category is RHO_GROUP.

ATOM_LOCAL_AXES
ATOM_RHO_MULTIPOLE
  ATOM_RHO_MULTIPOLE_COEFF
  ATOM_RHO_MULTIPOLE_KAPPA
  ATOM_RHO_MULTIPOLE_RADIAL_SLATER

4.1.4.6. The magnetic CIF dictionary

Magnetic properties of crystal structures are described by the magnetic CIF dictionary (Chapter 4.8). Table 4.1.4.6 shows the categories defined in this dictionary. Two (ATOM_SITE_MOMENT and ATOM_SITE_FOURIER_WAVEVECTOR) are extensions of the properties associated directly with atom sites in the core and modulated structures dictionaries respectively. As magnetic properties are often found in incommensurate structures (see Chapter 3.8), the magnetic CIF dictionary imports both the core and modulated structures dictionaries to provide a complete set of necessary definitions.

The remaining categories in this dictionary detail magnetic properties for both periodic and aperiodic structures.

Table 4.1.4.6. Category hierarchy of the magnetic CIF dictionary. The Head category is MAGNETIC. The symbol ¶ indicates children of categories already defined in the core CIF dictionary and the symbol § indicates children of categories defined in the modulated structures dictionary.

§ ATOM_SITE_FOURIER_WAVE_VECTOR
¶ ATOM_SITE_MOMENT
ATOM_SITE_MOMENT_FOURIER
  ATOM_SITE_MOMENT_FOURIER_PARAM
ATOM_SITE_MOMENT_SPECIAL_FUNC
ATOM_SITES_MOMENT_FOURIER
PARENT_PROPAGATION_VECTOR
PARENT_SPACE_GROUP
SPACE_GROUP_MAGN
SPACE_GROUP_MAGN_SSG_TRANSFORMS
SPACE_GROUP_MAGN_TRANSFORMS
SPACE_GROUP_SYMOP_MAGN_OG_CENTERING
SPACE_GROUP_SYMOP_MAGN_CENTERING
SPACE_GROUP_SYMOP_MAGN_OPERATION
SPACE_GROUP_SYMOP_MAGN_SSG_CENTERING
SPACE_GROUP_SYMOP_MAGN_SSG_OPERATION

4.1.4.7. The topology dictionary

The topology CIF dictionary (Chapter 4.9) describes the topological and connectivity properties of periodic nets, and is thus more of a mathematical abstraction of the underlying lattice (or lattices) of a crystal structure than its physical description (see Chapter 3.9). Table 4.1.4.7 lists the categories in this dictionary, which for the most part define abstract periodic nets. The TOPOL_ATOM category does allow for a mathematical lattice to be superimposed on or otherwise associated with a physical crystal structure, and the data item `_topol_atom.atom_label`, by matching a value of `_atom_site.label`, is the mechanism that links together the topological and physical descriptions of the crystal structure.

Table 4.1.4.7. Category hierarchy of the topology CIF dictionary. The Head category is TOPOLOGY.

TOPOL
TOPOL_ATOM
TOPOL_ENTANGL
TOPOL_LINK
TOPOL_NET
TOPOL_NODE
TOPOL_TILING

4.1.4.8. The macromolecular dictionary

The macromolecular CIF dictionary and its family of associated dictionaries used for biological structures is maintained by the Worldwide Protein Data Bank, and is not fully covered in this volume. Details of the central mmCIF as originally developed under the aegis of COMCIFS are found in Chapter 3.10, and the reference version of the dictionary is present in Chapter 4.10. Fig. 4.1.3.1 shows that the mmCIF dictionary was originally a proper superset of the core dictionary, though core categories such as FUNCTION and MODEL_SITE, which were developed at a later date to accommodate methods descriptions, do not appear in current versions of the expanded mmCIF/PDBx dictionary.

The handling of category relationships differs between DDLm and DDL2 dictionaries, the latter grouping related categories using a `_category_group.id` attribute. Table A3.10.1.1 shows the category structure of the reference mmCIF dictionary, arranged by the category group organisation within that dictionary.

4.1.4.9. The image dictionary

The image CIF dictionary (Chapter 4.11) is concerned with the raw image data collected in a diffraction

experiment (see Chapter 3.11 for an explanation of the way in which individual data arrays are organized into frames and scans, and how instrumental and sample coordinate axes are described). Conceptually, all its categories fit within the DIFFRACTION heading of the core dictionary. Table 4.1.4.8 lists the categories in the image dictionary, and identifies several that extend existing core categories.

Table 4.1.4.8. Category hierarchy of the img CIF dictionary. The symbol ¶ indicates extensions to categories already defined in the core CIF dictionary.

ARRAY_DATA
ARRAY_DATA_EXTERNAL_DATA
ARRAY_ELEMENT_SIZE
ARRAY_INTENSITIES
ARRAY_STRUCTURE
ARRAY_STRUCTURE_LIST
ARRAY_STRUCTURE_LIST_AXIS
ARRAY_STRUCTURE_LIST_SECTION
AXIS
DIFFRN_DATA_FRAME
¶ DIFFRN_DETECTOR
DIFFRN_DETECTOR_AXIS
DIFFRN_DETECTOR_ELEMENT
DIFFRN_FRAME_DATA
¶ DIFFRN_MEASUREMENT
DIFFRN_MEASUREMENT_AXIS
¶ DIFFRN_RADIATION
¶ DIFFRN_REFLN
DIFFRN_SCAN
DIFFRN_SCAN_AXIS
DIFFRN_SCAN_COLLECTION
DIFFRN_SCAN_FRAME
DIFFRN_SCAN_FRAME_AXIS
DIFFRN_SCAN_FRAME_MONITOR
MAP
MAP_SEGMENT
VARIANT

4.1.5. Extending the dictionaries to meet local requirements

We emphasise, as elsewhere in this volume, that the Crystallographic Information *Framework* is, strictly, format agnostic. That is, the definitions of specific data items, including their specific attributes (type, numeric range *etc.*) can be applied to a presentation of those data in any well-defined format (*e.g.* XML, JSON), so long as the data values can be unambiguously identified through association with the relevant data name. In the adoption and application of CIF-*format* files using data names defined in CIF dictionaries as a specific exchange mechanism, the crystallographic community has imposed on itself a particular discipline. This is not to be seen as a restriction but as a means to unambiguous and effective communication.

The existence and use of canonical dictionaries does not limit the contents of data files. A data file may contain arbitrary items not in the canonical dictionaries, as well as items formally defined in local dictionaries, which are intended for use only by certain software packages or in particular applications. The choice of which items to include in the data file depends on the capabilities of the applications that will use the data in the file. It is also influenced by the extent to which the author of the file wishes the data to be retrievable without ambiguity in the future.

It is therefore important to be able to include information in CIF data files that is only of local interest in a way that does not conflict with canonical or other public data names.

An author may define local data names in some completely informal manner; that is, there is no obligation to construct an attribute table in an external file that conforms to the style of the public dictionaries. Nevertheless, there are clear advantages to doing so: the author will benefit from standard software tools that validate data against dictionaries and the data names are more easily exported to the public domain if they subsequently become relevant to a wider community.

4.1.5.1. The [local]_ prefix

The string `[local]_` is *reserved* as a prefix to identify data names that will not appear in any public dictionary. (The left and right square brackets are included in this label.) Hence an author may construct private data names according to one of the following models, secure in the knowledge that the name will not appear in any global dictionary. With the older dictionary definition language DDL1 (now deprecated), a private data name will always have the form `_[local]_private_data_name`, while with DDLm and DDL2 the forms `_[local]_new_category_name.private_data_name` and `_existing_category_name.[local]_private_data_name` may be used. The first form is used for private data names in a category not already defined by a public dictionary; the second form permits the addition of local data names to an existing category.

While this convention guarantees that the new data name will not conflict with a public one, it cannot guarantee that it will not conflict with a local data name created by another author. Therefore these data names are appropriate only for testing purposes and not for release in data files that may be used by others.

Table 4.1.5.1. Reserved prefixes for private CIF data names

String	Reserved for the use of
acihd	Anorganisch-Chemisches Institut Heidelberg
aflow	*AFLOW* high-throughput Density Functional Calculations
amcsd	*American Mineralogist* Crystal Structure Database
anbf	Australian National Beamline Facility
asd	Active Site Database
B+S	Software developers Bernstein + Sons
BplusS	Software developers Bernstein + Sons (use with DDLm)
bruker	Bruker *AXS* software
ccdc	Cambridge Crystallographic Data Centre
CCP4	*CCP*4 program system
cgraph	Oxford Cryosystems *Crystallographica* package
chimerax	University of California San Francisco *ChimeraX*
cod	Crystallography Open Database
cifdic	Register of CIF dictionaries
crystmol	*CrystMol* package
csd	Cambridge Structural Database
dft	Density functional theory calculations (T. Björkman)
ebi	European Bioinformatics Institute
edchem	Edinburgh University Chemistry Department
gsas	*GSAS* powder refinement system
gsk	GSK (GlaxoSmithKline)
H5	CIF support of HDF5 and NeXus
iims	EBI project on integration of information about macromolecular structure
ISO	*ISOTROPY* software suite
itqb	Instituto de Tecnologia Química e Biológica da Universidade NOVA de Lisboa
iucr	IUCr journal use
LANL	Los Alamos National Laboratory
manchester	University of Manchester
mdb	Model Database (Glaxo)
montpellier	University of Montpellier
mp	Materials Project (MP) database
mpod	Material Properties Open Database
msd	EBI Molecular Structure Database Group
naned	Project NanED for 3D electron diffraction data
ndb	Nucleic Acids Database Project, Rutgers University
NIEHS	National Institute of Environmental Health Sciences
nomad	NOMAD Center of Excellence for theoretical material science calculations and structures
nottingham	University of Nottingham
NX	Nexus-related tags for CIF HDF5/NeXus integration
oxford	*CRYSTALS* package, University of Oxford
parvati	Validation and statistical summaries from *PARVATI* validation server
pdb	Protein Data Bank
pdbx	Protein Data Bank exchange dictionary
pdb2cif	Additions to mmCIF used by program *pdb2cif*
phenix	*PHENIX* software suite for macromolecular structures
prop	Properties (used in Material Properties Open Database)
publcif	*publCIF* editor
raman	Spectra obtained by Raman spectroscopy technique
rayonix	Information specific to Rayonix (Mar USA) instruments
rcsb	Research Collaboratory for Structural Bioinformatics
rotag	Rotamer software library *rotag*
shelx	*SHELXL* solution and refinement programs
solsa	SOLSA H2020 project (sonic drilling with automated mineralogy and chemistry)
SSAD	Sulfur SAD Database
sydney	University of Sydney
tcod	Theoretical Crystallography Open Database
TOPOS	*ToposPro* software
vrf	Validation reply form (IUCr/*Acta Crystallographica* use)
wdc	Entries in the World Directory of Crystallographers
xtal	*Xtal* program system

4.1.5.2. Reserved prefixes

To guarantee that locally devised data names may be placed without name conflict in interchange data files, authors may register a reserved character string for their sole use. As with the special prefix `[local]_` discussed in Section 4.1.5.1, the author's reserved prefix is simply an underscore-terminated string within the data name (*i.e.* it may not itself include an underscore character). For DDL1 applications it must be the first component of the data name; for DDLm and DDL2 applications it forms the first component of the data name if describing data names in a category not defined in the official dictionaries; or the first component after the full stop (category delimiter) if the local data name is an extension to an existing category.

Prefixes may be registered online through a web form at https://www.iucr.org/cgi-bin/cifreserve.pl. Table 4.1.5.1 gives a list of prefixes registered as of June 2023; this list will of course go out of date, but a current list will be maintained on the web at the address above.

An example of a data name incorporating a reserved prefix is the average (or Wilson) *B* factor reported by the Oxford *CRYSTALS* package as `_oxford_diffrn_Wilson_B_factor`.

4.1.5.3. Name spaces

The allocation of special prefixes as in Sections 4.1.5.1 and 4.1.5.2 above is a basic form of name-space allocation, because it gives authors the freedom to reproduce portions of otherwise standard data names within their own private constructions. This convention has been supported since the release of the original core dictionary.

DDLm has introduced `_dictionary.namespace` that allows formal name-space assignment within dictionaries (Section 2.4.2.2.3). At the time of writing this is used only in dREL methods [see item (iv) of Section 2.5.2.3], but it may provide a convenient mechanism for future development of dictionaries, especially in the context of cross-domain applications.

References

Bernstein, H. J., Bollinger, J. C., Brown, I. D. *et al.* (2016). *Specification of the Crystallographic Information File format, version 2.0. J. Appl. Cryst.* **49**, 277–284. *https://doi.org/10.1107/S1600576715021871*.

COMCIFS (2006). *Terms of Reference.* https://www.iucr.org/resources/cif/comcifs/terms-of-reference

COMCIFS (2014). *IUCr Committee for the Maintenance of the CIF Standard (GitHub repositories).* https://github.com/COMCIFS

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. *https://doi.org/10.1107/S010876739101067X*.

McMahon, B. (2005). In *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data*, edited by S. R. Hall & B. McMahon, Section 3.1.9, pp. 88–89. Dordrecht: Springer, 1st ed.

Spadaccini, N. & Hall, S. R. (2012). *DDLm: A new dictionary definition language*. *J. Chem. Inf. Model.* **52**(8), 1907–1916. *https://doi.org/10.1021/ci300075z*.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. 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*.

Appendices

APPENDIX 1: This is a concise listing of the DDLm dictionary, formatted in the style of *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data*, 2nd edition (in preparation).

DDLm dictionary

BY SYDNEY R. HALL[1], NICK SPADACCINI[1], JAMES R. HESTER[2], JOHN C. BOLLINGER[3] AND ANTANAS VAITKUS[4]

This dictionary contains the definitions of attributes that make up the DDLm dictionary definition language. It provides the meta meta data for all CIF dictionaries.

ATTRIBUTES

This category is parent of all other categories in the DDLm dictionary.

ALIAS

The attributes used to specify the aliased names of definitions.

Category key(s): **_alias.definition_id**

_alias.definition_id (*Tag*)

Identifier tag of an aliased definition.

_alias.deprecation_date (*Date*)

Date that the aliased tag was deprecated as a definition tag.

_alias.dictionary_uri (*Uri*)

Absolute URI of dictionary to which the aliased definition belongs.

CATEGORY_KEY

The attributes used to specify (possibly multiple) keys for a given category.

Category key(s): **_category_key.name**

_category_key.name (*Tag*)

A minimal list of tag(s) that together constitute a compound key to access other items in a Loop category. In other words, the combined values of the data items listed in this loop must be unique, so that unambiguous access to a packet (row) in the table of values is possible.

DEFINITION

The attributes for classifying dictionary definitions.

_definition.class (*Code*)

The nature and the function of a definition or definitions.

Where no value is given, the assumed value is '`Datum`'.

The data value must be one of the following:

`Attribute`	Item used as an attribute in the definition of other data items in DDLm dictionaries. These items never appear in data instance files.
`Functions`	Category of items that are transient function definitions used only in dREL methods scripts. These items never appear in data instance files.
`Datum`	Item defined in a domain-specific dictionary. These items appear only in data instance files.
`Head`	Category of items that is the parent of all other categories in the dictionary.
`Loop`	Category of items that in a data file may reside in a loop-list with a key item defined.
`Set`	Category of items that form a set (but not a loopable list). These items may be referenced as a class of items in a dREL methods expression.

_definition.id (*Code*)

Identifier name of the Item or Category being defined.

_definition.scope (*Code*)

The extent to which a definition affects other definitions.

Where no value is given, the assumed value is '`Item`'.

The data value must be one of the following:

`Dictionary`	Applies to all defined items in the dictionary.
`Category`	Applies to all defined items in the category.
`Item`	Applies to a single item definition.

_definition.update (*Date*)

The date that a definition was last changed.

DEFINITION_REPLACED

Attributes used to describe deprecated and replaced definitions.

Category key(s): **_definition_replaced.id**

_definition_replaced.by (*Tag*)

Name of the data item that should be used instead of the defined data item. The defined data item is deprecated and should not be used. A value of '.' signifies that the data item is deprecated, with no replacement.

_definition_replaced.id (*Code*)

An opaque identifier for the replacement.

[1] The University of Western Australia, 35 Stirling Highway, 6009 Perth, Australia; [2] Australian Nuclear Science and Technology Organisation, Locked Bag 2001, Kirrawee DC, NSW 2232, Australia [3] Department of Structural Biology, St Jude Children's Research Hospital, Memphis, Tennessee 38105, USA; [4] Institute of Biotechnology, Vilnius University, Vilnius, Lithuania.

DESCRIPTION

The attributes of descriptive (non-machine parsable) parts of definitions.

_description.common (*Text*)
Commonly-used identifying name for the item.

_description.key_words (*Text*)
List of key-words categorising the item.

_description.text (*Text*)
The text description of the defined item, category, or dictionary.

DESCRIPTION_EXAMPLE

Descriptive (non-machine parsable) examples of values of the defined items and categories.
Category key(s): **_description_example.case**

_description_example.case (*Implied*)
An example case of the defined item or category. Category example cases present data names and values as they would appear in a CIF-formatted file. Item example cases present values only, which inherit the enumeration range, enumeration set, container, dimension, content and purpose type constraints of the defining item.

_description_example.detail (*Text*)
A description of an example case for the defined item or category.

DICTIONARY

Attributes for identifying and registering the dictionary. The items in this category are *not* used as attributes of INDIVIDUAL data items.

_dictionary.class (*Code*)
The nature, or field of interest, of data items defined in the dictionary.
Where no value is given, the assumed value is '`Instance`'.
The data value must be one of the following:

`Reference`	DDLm reference attribute definitions.
`Instance`	Domain-specific data instance definitions.
`Template`	Domain-specific attribute/enumeration templates.
`Function`	Domain-specific method function scripts.

_dictionary.date (*Date*)
The date that the last dictionary revision took place.

_dictionary.ddl_conformance (*Version*)
The version number of the DDL dictionary that this dictionary conforms to.

_dictionary.DOI (*Text*)
The digital object identifier (DOI) of the dictionary.
Example: '`10.5555/12345678`'

_dictionary.formalism (*Text*)
The definitions contained in this dictionary are associated with the value of this attribute. Data items may only be redefined if the value of this attribute is also changed, and any such redefinitions must include the original behaviour as a particular case.

_dictionary.namespace (*Code*)
The namespace code that may be prefixed (with a trailing double colon '::') to an item tag defined in the defining dictionary when used in particular applications. Because tags must be unique, namespace codes are unlikely to be used in data files.

_dictionary.title (*Code*)
The common title of the dictionary. Will usually match the name attached to the **data_** statement of the dictionary file.

_dictionary.uri (*Uri*)
An absolute uniform resource identifier (URI) for this dictionary.

_dictionary.version (*Version*)
A unique version identifier for the dictionary.

DICTIONARY_AUDIT

Attributes for identifying and registering the dictionary. The items in this category are *not* used as attributes of individual data items.
Category key(s): **_dictionary_audit.version**

_dictionary_audit.date (*Date*)
The date of each dictionary revision.

_dictionary_audit.revision (*Text*)
A description of the revision applied for the **_dictionary_audit.version**.

_dictionary_audit.version (*Version*)
A unique version identifier for each revision of the dictionary.

DICTIONARY_AUTHOR

Attributes used to record the dictionary author information.
Category key(s): **_dictionary_author.id**

_dictionary_author.email (*Text*)

The electronic mail address of an author of the dictionary, in a form recognizable to international networks. The format of e-mail addresses is given in Section 3.4, *Address Specification*, of *Internet Message Format*, RFC 2822, P. Resnick (Editor), Network Standards Group, April 2001.

Examples: 'name@host.domain.country', 'bm@iucr.org'

_dictionary_author.id (*Word*)

Arbitrary identifier for this author.

_dictionary_author.id_ORCID (*Code*)

Identifier in the ORCID Registry of a publication author. ORCID is an open, non-profit, community-driven service to provide a registry of unique researcher identifiers (https://orcid.org/).

Example: '0000-0003-0391-0002'

_dictionary_author.name (*Text*)

The name of an author of this dictionary. The family name(s), followed by a comma and including any dynastic components, precedes the first name(s) or initial(s). For authors with only one name, provide the full name without abbreviation.

Examples: '''O'Neil, F.K.''', 'Yang, D.-L.', 'M\"uller, H.A.', 'Chandra'

DICTIONARY_VALID

Data items which are used to specify the contents of definitions in the dictionary in terms of the **_definition.scope** and the required and prohibited attributes. Validation rules described by data items in this category apply only to Reference and Instance dictionaries.

Category key(s): **_dictionary_valid.scope**
_dictionary_valid.option

_dictionary_valid.application (*Code*[2])

Deprecated. Provides the information identifying the definition scope (from the **_definition.scope** enumeration list) and the validity options (from the **_dictionary_valid.option** enumeration list), as a two element list.

_dictionary_valid.attributes (*Code*[])

A list of the attribute names and categories that are assessed for application in the item, category and dictionary definitions. A parent attribute category implicitly recursively includes all child categories.

_dictionary_valid.option (*Code*)

Option codes for applicability of attributes in definitions. Attributes not listed as 'Prohibited' for a given scope are allowed in that scope.

Where no value is given, the assumed value is 'Recommended'.

The data value must be one of the following:

Mandatory	Attribute must be present in definition frame.
Recommended	Attribute is usually in definition frame.
Prohibited	Attribute must not be used in definition frame.

_dictionary_valid.scope (*Code*)

The scope to which the specified restriction on usable attributes applies.

The data value must be one of the following:

Dictionary	Restriction applies to dictionary definition.
Category	Restriction applies to a category definition.
Item	Restriction applies to an item definition.

ENUMERATION

The attributes for restricting the values of defined data items.

_enumeration.def_index_id (*Tag*)

Deprecated. The **_enumeration.def_index_ids** data item should be used instead of this item. Specifies the data name of the item with a value used as an index to the DEFAULT enumeration list (in category ENUMERATION_DEFAULT) in order to select the default enumeration value for the defined item. The value of the identified data item must match one of the **_enumeration_default.index** values.

_enumeration.def_index_ids (*Tag*[])

Specifies the data names of items that are collectively used to identify the suitable default value in the ENUMERATION_DEFAULTS category loop. The values of these items are used to construct an ordered list which is checked against the values of the **_enumeration_defaults.index** attribute.

_enumeration.default (*Implied*)

The default value for the defined item if it is not specified explicitly. Value of this attribute inherits the enumeration range, enumeration set, container, dimension, content and purpose type constraints of the defining item.

_enumeration.mandatory (*Code*)

Yes or No flag on whether the enumerate states specified for an item in the current definition (in which item appears) *must* be used on instantiation.

Where no value is given, the assumed value is 'Yes'.

The data value must be one of the following:

Yes	Use of state is mandatory.
No	Use of state is unnecessary.

_enumeration.range (*Range*)

The inclusive range of numerical values allowed for the defined item. If the defined item has associated SU values, the reported data values may fall outside these limits.

Examples: '-4:10' (Values must be no less than -4 and no greater than 10.), '0:' (Values must be greater than or equal to 0.), ':3.1415' (Values must be less than or equal to 3.1415.)

ENUMERATION_DEFAULT

Deprecated. The ENUMERATION_DEFAULTS category should be used instead of this category. Loop of pre-determined default enumeration values indexed to a data item by the item **_enumeration.def_index_id**.

Category key(s): **_enumeration_default.index**

_enumeration_default.index (*Code*)

Deprecated. The **_enumeration_defaults.index** data item should be used instead of this item. Index key in the list default values referenced to by the value of **_enumeration.def_index_id**.

_enumeration_default.value (*Implied*)

Deprecated. The **_enumeration_defaults.value** data item should be used instead of this item. Default enumeration value in the list referenced by the value of **_enumeration.def_index_id**. The reference index key is given by the value of **_enumeration_default.index** value.

ENUMERATION_DEFAULTS

Loop of pre-determined default enumeration values indexed by a set of data items specified using the **_enumeration.def_index_ids** attribute.

Category key(s): **_enumeration_defaults.index**

_enumeration_defaults.index (*Inherited*[])

An index value, in the form of an ordered list, which allows the identification of the most suitable default value. The order of values in the list must correspond to the order of the related data item references in the **_enumeration.def_index_ids** attribute list. That is, the *n*th list element is assumed to represent a value that may be assigned to the data item referenced at the *n*th position of the **_enumeration.def_index_ids** attribute list value. The constituent values of the **_enumeration_defaults.index** attribute inherit the enumeration range, enumeration set, and the content type from the data item with which they are paired.

_enumeration_defaults.source_id (*Word*)

An identifier which corresponds to an entry in the ENUMERATION_SOURCE category loop. Used to provide a reference to the original source of a specific enumeration value.

Values must match those for the following item(s): ***_enumeration_source.id***

_enumeration_defaults.value (*Implied*)

The default enumeration value associated with a specific index value constructed using data items referenced by the **_enumeration.def_index_ids** attribute. The associated index value is provided using the **_enumeration_defaults.index** attribute.

ENUMERATION_SET

Attributes of data items which are used to define a set of unique pre-determined values.

Category key(s): **_enumeration_set.state**

_enumeration_set.detail (*Text*)

The meaning of the code (identified by **_enumeration_set.state**) in terms of the value of the quantity it describes.

_enumeration_set.state (*Text*)

Permitted value state for the defined item.

ENUMERATION_SOURCE

Attributes used to record the original sources of the default values enumerated using the ENUMERATION_DEFAULTS category.

Category key(s): **_enumeration_source.id**

Example 1 – There are two sources for the default values given in the enumeration. Entries marked with 'a' come from Table 4.4.4.1 in the 2nd edition of Volume C of the International Tables of Crystallography. Entries marked with 'b' come from Table 4.3.2.1 in the 1st edition of Volume H of the International Tables of Crystallography.

```
loop_
_enumeration_source.id
_enumeration_source.reference
a 'International Tables Vol. C Table 4.4.4.1 2nd ed.'
b 'International Tables Vol. H Table 4.3.2.1 1st ed.'

loop_
_enumeration_defaults.index
_enumeration_defaults.value
_enumeration_defaults.source_id
    [ H  ]  1  a
    [ H- ]  2  a
    [ He ]  2  b
    # ...
```

Example 2 – There is a single source for all default values given in the enumeration. All entries can be found in Table S4 of the material referenced by the given DOI.

```
_enumeration_source.reference
'https://doi.org/10.1107/S2053273322010944, Table S4'

loop_
_enumeration_defaults.index
_enumeration_defaults.value
    [ He ] 0.8733928
    [ Li ] 1.129319
    [ Be ] 1.592162
    # ...
```

_enumeration_source.id (*Word*)

A unique identifier of the source material from which the **_enumeration_defaults.value** attribute values were taken.

_enumeration_source.reference (*Text*)

Bibliographic information sufficient to identify the original source of the associated **_enumeration_defaults.value** attribute values.

IMPORT

Used to import the values of specific attributes from other dictionary definitions within and without the current dictionary.

_import.get (*ByReference*)

A list of tables of attributes defined individually in the category IMPORT_DETAILS, used to import definitions from other dictionaries.

IMPORT_DETAILS

Items in IMPORT_DETAILS describe individual attributes of an import operation.

Category key(s): **_import_details.order**

_import_details.file_id (*Uri*)

A URI reference as per RFC 3986 giving the location of the source dictionary. When a relative URI is used, the base URI for the URI reference is the **_dictionary.uri** of the importing dictionary.

_import_details.file_version (*Version*)

The required version number for **_dictionary.version** of the imported dictionary. Dictionaries with the same major version number are compatible. If absent or null, any version is permitted.

_import_details.frame_id (*Code*)

The save frame code of the definition frame to be imported.

_import_details.if_dupl (*Code*)

Code identifying the action taken if the requested definition block already exists within the importing dictionary in 'Full' mode, or an attribute exists in both the importing definition block and the requested definition block in 'Contents' mode.

Where no value is given, the assumed value is 'Exit'.

The data value must be one of the following:

Ignore	Ignore imported definitions if block identifiers match in 'Full' mode. Ignore imported attributes that match attributes already in the importing definition in 'Contents' mode. When importing in 'Contents' mode, if the ignored attribute belongs to a Loop category, all attributes from that category must be ignored to avoid loop mismatches.
Replace	Replace existing definitions with imported definitions if block identifiers match in 'Full' mode. When importing in 'Contents' mode, contents of the two save frames should be merged and any duplicate attributes replaced with those from the imported save frame. In case the replaced attribute belongs to a Loop category, all attributes from that category must first be removed from the importing save frame to avoid loop mismatches.
Exit	Issue an error exception and exit.

_import_details.if_miss (*Code*)

Code identifying the action taken if the requested definition block is missing from the source dictionary.

Where no value is given, the assumed value is 'Exit'.

The data value must be one of the following:

Ignore	Ignore import.
Exit	Issue error exception and exit.

_import_details.mode (*Code*)

Code identifying how the definition referenced by **_import_details.frame_id** is to be imported. 'Full' imports the entire definition together with any child definitions (in the case of categories) found in the target dictionary. The importing definition becomes the parent of the imported definition. As such, the 'Full' mode must only be used in category definitions. As a special case, a 'Head' category importing a 'Head' category is equivalent to importing all children of the imported 'Head' category as children of the importing 'Head' category. A 'Head' category can only be imported in 'Full' mode and only by another 'Head' category. 'Contents' imports only the attributes found in the imported definition.

Where no value is given, the assumed value is 'Contents'.

The data value must be one of the following:

Full	Import requested definition together with any child definitions.
Contents	Import contents of requested definition.

_import_details.order (*Integer*)

The order in which the import described by the referenced row should be executed.

_import_details.single (*Text*)

A Table mapping attributes defined individually in category IMPORT to their values; used to import definitions from other dictionaries.

_import_details.single_index (*Code*)

One of the indices permitted in the entries of values of attribute **_import_details.single**.

The data value must be one of the following:

file	URI reference as per RFC 3986 giving the location of the source dictionary.
version	Version of source dictionary.
save	Save frame code of source definition.
mode	Mode for including save frames.
dupl	Option for duplicate entries.
miss	Option for missing duplicate entries.

METHOD

Methods used for evaluating, validating and defining items.

Category key(s): **_method.purpose**

_method.expression (*Text*)

The method expression for the defined item.

_method.purpose (*Code*)

The purpose and scope of the method expression.

Where no value is given, the assumed value is 'Evaluation'.

The data value must be one of the following:

Evaluation	Method evaluates an item from related item values. Definitions of primitive data items should normally not contain methods of this type.
Definition	Method generates attribute value(s) in the definition.
Validation	Method compares an evaluation with existing item value.

NAME

Attributes for identifying items and item categories.

_name.category_id (*Name*)

The name of the category in which a category or item resides. For Head categories this is the **_dictionary.title** given in the enclosing data block.

_name.linked_item_id (*Tag*)

Data name of an equivalent item which has a common set of values, or, in the definition of a type SU item is the name of the associated measurand item to which the standard uncertainty applies.

_name.object_id (*Name*)

The object name of a category or name unique within the category or family of categories.

TYPE

Attributes which specify the 'typing' of data items.

_type.container (*Code*)

The structure of values for the defined data item.

Where no value is given, the assumed value is 'Single'.

The data value must be one of the following:

Single	Single value.
List	Ordered set of values. Elements need not be of same contents type.
Array	Ordered set of values of the same type. Operations across arrays are equivalent to operations across elements of the Array.
Matrix	Ordered set of numerical values for a tensor. Tensor operations such as dot and cross products, are valid cross matrix objects. A matrix with a single dimension is interpreted as a row or column vector as required.
Table	An unordered set of id:value elements.
Implied	**Applied ONLY in the DDLm Reference Dictionary.** The value structure is taken from **_type.container** in the definition in which the defined attribute appears.

_type.contents (*Code*)

Syntax of the value elements within the container type. Where the definition is of a 'List' or 'Array' type, this attribute describes the contents of each element. Where the definition is of a 'Table' container this attribute describes the construction of the value elements within those (Table) values. The CIF2 character set referenced below consists of the following Unicode code points: [U+0009], [U+000A], [U+000D], [U+0020-U+007E], [U+00A0-U+D7FF], [U+E000-U+FDCF], [U+FDF0-U+FFFD], [U+10000-U+1FFFD], [U+20000-U+2FFFD], [U+30000-U+3FFFD], [U+40000-U+4FFFD], [U+50000-U+5FFFD], [U+60000-U+6FFFD], [U+70000-U+7FFFD], [U+80000-U+8FFFD], [U+90000-U+9FFFD], [U+A0000-U+AFFFD], [U+B0000-U+BFFFD], [U+C0000-U+CFFFD], [U+D0000-U+DFFFD], [U+E0000-U+EFFFD], [U+F0000-U+FFFFD], [U+100000-U+10FFFD] Two case-insensitive strings are considered identical when they match under the Unicode canonical caseless matching algorithm. In all cases, 'whitespace' refers to ASCII whitespace only, that is [U+0009],[U+000A],[U+000D] and [U+0020]. Note that descriptions of text syntax are relevant only to those formats that encode data values as text.

Where no value is given, the assumed value is 'Text'.

The data value must be one of the following:

Text	Case-sensitive sequence of CIF2 characters.
Word	Case-sensitive sequence of CIF2 characters containing no ASCII whitespace.
Code	Case-insensitive sequence of CIF2 characters containing no ASCII whitespace.
Name	Case-insensitive sequence of ASCII alphanumeric characters or underscore.
Tag	Case-insensitive CIF2 character sequence with leading underscore and no ASCII whitespace.
Uri	Uniform Resource Identifier reference as defined in RFC 3986 Section 4.1.
Date	ISO standard date format <yyyy>-<mm>-<dd>. Use DateTime for all new dictionaries.
DateTime	A timestamp. Text formats must use date-time or full-date productions of RFC 3339 ABNF.
Version	Version number string that adheres to the formal grammar provided in the Semantic Versioning specification version 2.0.0. Version strings must take the general form of <major>.<minor>.<patch> and may also contain an optional postfix with additional information such as the pre-release identifier. Reference: https://semver.org/spec/v2.0.0.html
Dimension	Size of an Array/Matrix/List expressed as a text string. The text string itself consists of zero or more non-negative integers separated by commas placed within bounding square brackets. Empty square brackets represent a list of unknown size.
Range	Inclusive range of numerical values expressed using the min:max notation in which the smallest value 'min' and the largest value 'max' are separated by a colon character. If 'max' is omitted, then the range includes all values that are greater than or equal to 'min'. If 'min' is omitted, then the range includes all values that are less than or equal to 'max'.
Integer	A number from the set of all integers.

Real	Floating-point real number.
Imag	Floating-point imaginary number.
Complex	A complex number.
Symop	A string composed of an integer optionally followed by an underscore or space and three or more digits.
Implied	The contents are described by the **_type.contents** attribute in the definition in which the defined attribute appears.
ByReference	The contents have the same form as those of the attribute referenced by **_type.contents_referenced_id**.
Inherited	**Applied ONLY in the DDLm Reference Dictionary.** Intended to be used with List, Array, Matrix or Table containers which may simultaneously contain values of several content types. The content type of each value in such containers matches the content type of the data item to which it is directly related. Specific rules for relating data values to data items MUST be provided as human-readable text in the **_description.text** attribute of all data items that have the Inherited content type.

Example: 'Integer' (Content is a single or multiple integer(s).)

_type.contents_referenced_id (*Tag*)

The value of the **_definition.id** attribute of an attribute definition whose type is to be used also as the type of this item. Meaningful only when this item's **_type.contents** attribute has value 'ByReference'.

_type.dimension (*Dimension*)

The dimensions of a list, array or matrix of elements expressed as a text string. A Matrix with a single dimension is interpreted as a vector.

Examples: '[3,3]' (3x3 matrix of elements.), '[6]' (List of 6 elements.), '[]' (Unknown number of list elements.)

_type.indices (*Code*)

Used to specify the syntax construction of indices of the entries in the defined object when the defined object has 'Table' as its **_type.container** attribute. Values are a subset of the codes and constructions defined for attribute **_type.contents**, accounting for the fact that syntactically, indices are always case-sensitive quoted strings. Meaningful only when the defined item has **_type.container** 'Table'. See the definition for **_type.contents** for the character set definition.

Where no value is given, the assumed value is 'Text'.

The data value must be one of the following:

Text	A case-sensitive sequence of CIF2 characters.
Code	Case-insensitive sequence of CIF2 characters containing no ASCII whitespace.
Date	ISO date format <yyyy>-<mm>-<dd>.
Uri	A Uniform Resource Identifier string, per RFC 3986.
Version	Version digit string of the form <major>.<version>.<update>
ByReference	Indices have the same form as the contents of the attribute identified by **_type.indices_referenced_id**.

_type.indices_referenced_id (*Tag*)

The **_definition.id** attribute of a definition whose type describes the form and construction of the indices of entries in values of the present item. Meaningful only when the defined item's **_type.container** attribute has value 'Table', and its **_type.indices** attribute has value 'ByReference'.

_type.purpose (*Code*)

The primary purpose or function the defined data item serves in a dictionary or a specific data instance.

Where no value is given, the assumed value is 'Describe'.

The data value must be one of the following:

Import	**Applied ONLY in the DDLm Reference Dictionary.** Used to type the SPECIAL attribute '**_import.get**' that is present in dictionaries to instigate the importation of external dictionary definitions.
Method	**Applied ONLY in the DDLm Reference Dictionary.** Used to type the attribute '**_method.expression**' that is present in dictionary definitions to provide the text method expressing the defined item in terms of other defined items.
Audit	**Applied ONLY in the DDLm Reference Dictionary.** Used to type attributes employed to record the audit definition information (creation date, update version and cross reference codes) of items, categories and files.
Identify	**Applied ONLY in the DDLm Reference Dictionary.** Used to type attributes that identify an item tag (or part thereof) or external location.
Describe	Used to type items with values that are descriptive text intended for human interpretation.
Encode	Used to type items with values that are text or codes that are formatted to be machine parsable.
State	Used to type items with values that are restricted to codes present in their 'enumeration_set.state' lists.
Key	Used to type an item with a value that is unique within the looped list of these items, and does not contain encoded information.
Link	Used to type an item that acts as a foreign key between two categories. The definition of the item must additionally contain the attribute '**_name.linked_item_id**' specifying the data name of the item with unique values in the linked category. The values of the defined item are drawn from the set of values in the referenced item. Cross referencing items from the same category is allowed.
Composite	Used to type items with value strings composed of separate parts. These will usually need to be separated and parsed for complete interpretation and application.
Number	Used to type items that are numerical and exact (i.e. no standard uncertainty value).
Measurand	Used to type an item with a numerically estimated value that has been recorded by measurement or derivation. A data item definition for the standard uncertainty (SU) of this item must be provided in a separate definition with **_type.purpose** of 'SU'. The value of a measurand item should be accompanied by a value of its associated SU item, either: 1) integrated with the measurand value in a manner characteristic of the data format; or 2) as a separate, explicit value for the associated SU item. These alternatives are semantically equivalent.

SU	Used to type an item with a numerical value that is the standard uncertainty of another data item. The definition of an SU item must have the **_type.source** attribute set to 'Related' and must include the **_name.linked_item_id** attribute which explicitly identifies the associated measurand item. SU values must be non-negative.
Internal	Used to type items that serve only internal purposes of the dictionary in which they appear. The particular purpose served is not defined by this state.

_type.source (*Code*)

The origin or source of the defined data item, indicating by what recording process it has been added to the domain instance. All data items can be classified as primitive or non-primitive based on the origin of the data. Primitive data items record data that usually cannot be deduced without repeating the entire experiment such as measurements, observations (*e.g.* instrument settings) and decisions made in nonlinear processes. Non-primitive data items record derivable data that can be directly evaluated from other data.

Where no value is given, the assumed value is 'Assigned'.

The data value must be one of the following:

Recorded	Data value (numerical or otherwise) was recorded by observation or measurement during the experimental collection of data. Data items of this type are considered primitive.
Assigned	Data value (numerical or otherwise) was assigned as part of the data collection, analysis or modelling required for a specific domain instance. These assignments often represent a decision made that determines the course of the experiment (and therefore the data item may be deemed primitive) or a particular choice in the way the data was analysed (and therefore the data item may be considered non-primitive).
Related	Data item was added based on a relationship to another data item. This state indicates that the item was used to record the SU value of a related measurand item or that the item was used in the construction of looped lists of data. In the latter case, it typically identifies an item whose unique values are used as the reference key for a loop category and/or an item which has values in common with those of another loop category and is considered a Link between these lists. Data items of this type includes both primitive and non-primitive items.
Derived	Data item was derived from other data items within the domain instance. Data items of this type are considered non-primitive.

UNITS

The attributes for specifying units of measure.

_units.code (*Code*)

A code which identifies the units of measurement. The 'unspecified' code should only be used in cases when the units cannot be properly expressed in DDLm. A typical example of such situation is a List data item with constituent values that may have differing units.

The data value must be one of the following:

none	dimensionless – *e.g.* a ratio, factor, weight or scale
coulomb	electronic charge in coulombs
electron_volts	electronic charge in electron volts eV
metres	length 'metres (metres * 10^(0))'
centimetres	length 'centimetres (metres * 10^(-2))'
millimetres	length 'millimetres (metres * 10^(-3))'
micrometres	length 'micrometres (metres * 10^(-6))'
nanometres	length 'nanometres (metres * 10^(-9))'
angstroms	length 'angstroms (metres * 10^(-10))'
picometres	length 'picometres (metres * 10^(-12))'
femtometres	length 'femtometres (metres * 10^(-15))'
reciprocal_centimetres	per-length 'reciprocal centimetres (metres * 10^(-2)^-1)'
reciprocal_millimetres	per-length 'reciprocal millimetres (metres * 10^(-3)^-1)'
reciprocal_nanometres	per-length 'reciprocal nanometres (metres * 10^(-9)^-1)'
reciprocal_angstroms	per-length 'reciprocal angstroms (metres * 10^(-10)^-1)'
reciprocal_angstrom_squared	per-area 'reciprocal angstroms^2'
reciprocal_picometres	per-length 'reciprocal picometres (metres * 10^(-12)^-1)'
nanometre_squared	length_squared 'nanometres squared (metres * 10^(-9))^2'
angstrom_squared	length_squared 'angstroms squared (metres * 10^(-10))^2'
8pi_angstroms_squared	length_squared '8pi^2 * angstroms squared (metres * 10^(-10))^2'
picometre_squared	length_squared 'picometres squared (metres * 10^(-12))^2'
femtometre_squared	length_squared 'femtometres squared (metres * 10^(-15))^2'
nanometre_cubed	length_cubed 'nanometres cubed (metres * 10^(-9))^3'
angstrom_cubed	length_cubed 'angstroms cubed (metres * 10^(-10))^3'
picometre_cubed	length_cubed 'picometres cubed (metres * 10^(-12))^3'
grams_per_centimetre_cubed	density 'grams per cubic centimetre'
kilograms_per_metre_cubed	density 'kilograms per cubic metre'
megagrams_per_metre_cubed	density 'megagrams per cubic metre'
angstrom_cubed_per_dalton	density 'angstrom cubed per dalton'
kilopascals	pressure 'kilopascals'
gigapascals	pressure 'gigapascals'
hours	time 'hours'
minutes	time 'minutes'
seconds	time 'seconds'
microseconds	time 'microseconds'
degrees	angle 'degrees (of arc)'
cycles	phase 'angle in 360 degree arcs'
radians	angle 'radians'
degrees_squared	angle 'degrees (of arc)'

degree_per_minute	rotation_per-time 'degrees (of arc) per minute'
Celsius	temperature 'degrees (of temperature) Celsius'
kelvins	temperature 'temperature in kelvins'
kelvins_per_minute	cooling rate 'kelvins per minute'
electrons	electrons 'electrons'
electron_squared	electrons-squared 'electrons squared'
electrons_per_nanometre_cubed	electron-density 'electrons per nanometres cubed (electrons * (metres * 10^(-9))^(-3))'
electrons_per_angstrom_cubed	electron-density 'electrons per angstroms cubed (electrons * (metres * 10^(-10))^(-3))'
electrons_per_picometre_cubed	electron-density 'electrons per picometres cubed (electrons * (metres * 10^(-12))^(-3))'
dalton	standard atomic mass unit
pixels_per_millimetre	area resolution unit
pixels_per_element	area resolution unit
kilowatts	power 'kilowatts'
milliamperes	current 'milliamperes'
kilovolts	emf 'kilovolts'
volt_squared	emf 'volts squared'
Bohr_magnetons	magnetic moment
arbitrary	arbitrary 'arbitrary system of units'
counts_per_photon	measure of gain used in array detectors
counts	counts from a detector
'photons per second'	photons registered in one second

APPENDIX 2: This is an excerpt from a forthcoming draft chapter of *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data*, 2nd edition (in preparation).

Style guide for DDLm dictionaries

BY JAMES R. HESTER, ANTANAS VAITKUS AND BRIAN MCMAHON

Overview

The following rules describe the preferred layout of DDLm Reference and Instance dictionaries. Following these rules should allow generic dictionary manipulation software to ingest, semantically edit and re-output dictionaries with minimal irrelevant changes to whitespace.

These rules are not intended to apply to CIF data files or Template dictionaries.

These rules are not comprehensive, for example, they do not envisage table values that are semicolon-delimited. They should cover all situations typically encountered in DDLm dictionaries, and will be expanded as new situations arise.

Terminology

'Attribute' refers to a DDLm attribute (a 'data name' in CIF syntax terms). Columns are numbered from 1. 'Starting at column x' means that the first non-whitespace character (which may be a delimiter) appears in column x. 'Indent' refers to the number of whitespace characters preceding the first non-whitespace value. 'Special values' are the non-delimited question mark (`?`) and period (`.`) used in CIF syntax to denote Unknown and Null values, respectively.

Magic numbers

The following values are used in the description.

line_length	80
text_indent	4
text_prefix	>
value_col	35
value_indent	*text_indent* + *loop_step*
loop_indent	2
loop_align	10
loop_step	5
min_whitespace	2

1. Lines and padding

1. Lines are a maximum of *line_length* characters long. Multi-line character strings should be broken after the last whitespace character preceding this limit and trailing whitespace removed, unless rule 2.1.15 applies.

2. Unless rule 2.1.15 applies, data values with no internal whitespace that would overflow the line length limit if formatted according to the following rules should be presented in semicolon-delimited text fields with leading blank line, no indentation and folded, if necessary, so that the backslash appears in column *line_length*.

3. (No trailing whitespace) The last character in a line should not be whitespace.

4. Blank lines are inserted only as specified below. Blank lines do not accumulate, that is, there should be no sequences of more than one blank line.

5. All lines are terminated by a newline character (`\n`) as per CIF2 specifications.

6. Tab characters may not be used either as whitespace or within data values, unless part of the meaning of the data value.

7. No comments appear within, or after, the data block.

2. Value formatting

2.1 Text strings

In general multi-line text strings can include formatting like centering or ASCII equations. The rules below aim to minimise disruption to such formatting where present in the supplied value. Note also that rule 1.2 overrides indentation rules below.

1. Values that can be presented undelimited should not be delimited, unless rule 9 applies. Note that the literal question mark (`?`) and period (`.`) must always be delimited as otherwise they will be interpreted as special values.

2. Where a delimiter is necessary, the first delimiter in the following list that produces a syntactically correct CIF2 file should be used: single quote (`'`), double quote (`"`), triple-single-quote (`'''`), triple-double-quote (`"""`), semicolon (`\n;`).

3. Text fields containing newline characters are always semicolon-delimited.

4. If a text field contains the newline-semicolon sequence the text-prefix protocol is used with *text_prefix* as the prefix.

5. Each non-blank line of multi-line text fields not appearing as part of loops should contain *text_indent* spaces at the beginning. Tab characters must not be used for this purpose. Paragraphs are separated by a single blank line which must contain only a newline character. Lines may contain more than *text_indent* spaces at the beginning, for example for ASCII equations or centering purposes.

6. No tab characters may be used for formatting data values.

7. The first line of a semicolon-delimited text field should be blank, except for line folding and prefixing characters where necessary.

8. A newline character always follows the final semicolon of a semicolon-delimited text field.

9. Looped attributes should use the same delimiter for all values in the same column. Special values are exempt from this rule.

10. Category names in a category definition should be presented CAPITALISED in **_name.category_id**, **_name.object_id** and **_definition.id**.

11. Category and object names in data item definitions should be presented in 'canonical' case. Canonical case follows the rules of English capitalisation where the first letter is not considered to start a sentence. In particular:

(i) Proper names and place names (*e.g.* Wyckoff, Cambridge) and their abbreviations (*e.g.* 'H_M' for 'Hermann–Mauguin', 'Cartn', 'Lp_factor') are capitalised.
(ii) Symbols are capitalised according to crystallographic convention (*e.g.* `Uij`).
(iii) Initialisms are capitalised (*e.g.* CSD, IT for *International Tables*).

12. Case-insensitive data items should be output with a leading capital letter unless convention dictates otherwise.

13. Values of attributes drawn from enumerated states should be capitalised in the same way as the definition of that attribute.

14. Function names defined in DDLm Function categories are CamelCased.

15. If a character drawn from the set `#^*-=+~` appears 5 or more times sequentially (*e.g.* `^^^^^`) anywhere in a multi-line text value, the value is assumed to be preformatted. No line-length, prefixing or other alterations to the contents should be made.

2.2 Lists

No DDLm attributes are currently defined that require more than one level of nesting. If such attributes are defined, these rules will be extended.

1. The first and last values of a list are not separated from the delimiters by whitespace.

2. Each element of the list is separated by *min_whitespace* from the next element.

3. Where application of the rules for loop or attribute-value layout require an internal line break, the list should be presented as a multi-line compound object (see below).

4. These rules do not cover lists containing multi-line simple data values or lists with more than one level of nesting.

Examples

```
[112  128  144]
```

One level of nesting, can stay on single line

```
[[t.11 t.12 t.13] [t.21 t.22 t.23] [t.31 t.32 t.33]]
```

One level of nesting, can stay on a single line

```
_import.get ['file':templ_attr.cif 'save':aniso_UIJ]
```

2.3 Tables

No DDLm attributes are currently defined that require more than one level of nesting. If such attributes are defined, these guidelines will be extended.

1. Key:value pairs are presented with no internal whitespace around the `:` character.

2. The key is delimited by single quotes (`'`). If this is not possible, the rules for text strings (2.1) are followed.

3. Key:value pairs are separated by *min_whitespace*.

4. Keys appear in alphabetical order.

5. There is no whitespace between the opening and closing braces and the first/last key:value pair.

6. Where application of the rules for loop or attribute-value layout require an internal line break, the table should be presented as a multi-line compound object.

7. These rules do not cover tables containing multi-line simple data values or tables with more than one level of nesting.

Examples

```
'save':orient_matrix  'file':templ_attr.cif
```

One level of nesting:

```
['save':orient_matrix  'file':templ_attr.cif]
```

2.4 Multi-line compound object

A multi-line compound object is a list or table containing newlines. DDLm does not define attributes with more than one level of nesting. These rules will be extended if and when such items are defined. The indentation of the opening delimiter determined by rules (1) and (2) is labelled *object_indent*. Note that this refers to the number of whitespace characters preceding the opening delimiter, so the opening delimiter appears at column *object_indent* + 1. The intent of rule (1) is to minimise line breaks within any internal compound objects.

1. The opening delimiter is placed at the maximum of (*value_col*, the end of the previous value + *min_whitespace*), as long as any internal compound values would not exceed the line length when formatted as non-multi-line values according to the following rules.

2. Otherwise, the opening delimiter is placed at *value_indent* + 1 on a new line.

3. Each subsequent value is formatted according to the present rules until the final character of the next value would be beyond *line_length*.

4. The next value is placed on a new line indented by *object_indent* + *n*, where *n* is the nesting level.

5. A nested opening delimiter followed immediately by a primitive value is placed on a new line indented by *object_indent* + *n*, where n is the nesting level.

6. A closing delimiter immediately following a primitive value is placed on the same line.

7. Except when immediately following a primitive value, closing delimiters are placed on a separate line indented by the same amount as their corresponding opening delimiter.

8. A 'corresponding value' is either a list entry at the same position in each list of a list of lists, or a table value with the same key in a list of tables. Corresponding values must be vertically aligned on their first character such that a minimum spacing of *min_whitespace* is maintained, and at least one whitespace gap between each column is exactly *min_whitespace* for at least one row.

Examples

One level of nesting, but the nested data do not fit on a single line:

```
[
 [c.vector_a*c.vector_a  c.vector_a*c.vector_b  c.vector_a*c.vector_c]
 [c.vector_b*c.vector_a  c.vector_b*c.vector_b  c.vector_b*c.vector_c]
 [c.vector_c*c.vector_a  c.vector_c*c.vector_b  c.vector_c*c.vector_c]
]
```

Alignment of internal values, nested opening delimiter:

```
[
 'file':cif_core.dic  'save':CIF_CORE  'mode':Full
 'file':cif_ms.dic    'save':CIF_MS    'mode':Full
]
```

Internal value doesn't fit when starting at *value_col*, so must start at *value_indent*. Internal opening delimiter on new line:

```
    _import.get
         [
          "file":templ_attr.cif "save":Cromer_Mann_coeff
          "file":templ_enum.cif "save":Cromer_Mann_a1
         ]
```

Internal value fits using *value_col* as indent, but outer brackets are on separate lines by rule 5:

```
    _import.get                    [
                                    'file':templ_attr.cif  'save':Miller_index
                                   ]
```

Array item in loop starts at column 37 to maintain *min_whitespace*:

```
    loop_
```

```
_dictionary_valid.application
_dictionary_valid.attributes
   [Dictionary  Mandatory]    ['_dictionary.title'  '_dictionary.class'
                               '_dictionary.version'  '_dictionary.date'
                               '_dictionary.uri'
                               '_dictionary.ddl_conformance'
                               '_dictionary.namespace']
   [Dictionary  Recommended]  ['_description.text'
                               '_dictionary_audit.version'
                               '_dictionary_audit.date'
                               '_dictionary_audit.revision']
```

2.5 Enumeration ranges

Values of the **`_enumeration.range`** attribute should be expressed in a format that best reflects the content type of the defining item. That is, numeric range limits of data items with the 'Integer' content type should be formatted as integers while data items with the 'Real' content type should be formatted as floating-point real numbers. Additional formatting rules for enumeration ranges are provided in Section 2.5.1 and Section 2.5.2.

2.5.1. Integer ranges

Numeric range limits of data items with the 'Integer' content type should be expressed as integers that:

1. Do not include non-significant leading zeros, *e.g.* '7' instead of '007'.

2.Do not include a fractional part, *e.g.* '1' instead of '1.0'.

3. Do not include a trailing decimal separator, *e.g.* '2' instead of '2.'.

4. Do not include the '+' symbol, *e.g.* '42' instead of '+42'.

5. Do not include a signed zero, *e.g.* '0' instead of '+0' or '-0'.

The following regular expression may be used to check if a number adheres to the integer range limit formatting rules:

```
^
(
  0|( [-]?[1-9][0-9]* )
)
$
```

The regular expression above is formatted for readability using the additional syntax rules enabled by the '/x' Perl regular expression modifier (*e.g.* any unescaped whitespace symbols must be ignored).

Examples of properly formatted integer number ranges

```
1:230
0:
-8:8
```

2.5.2. Real number ranges

Numeric range limits of data items with the 'Real' content type should be expressed using floating-point real numbers that:

1. Include at least one digit before the decimal separator, *e.g.* '0.5' instead of '.5'.

2. Include at least one digit after the decimal separator, *e.g.* '7.0' instead of '7.' or '7'.

3. Include the smallest number of non-significant leading zeros that still satisfies other formatting rules, *e.g.* '0.25' instead of '000.25'.

4. Include the smallest number of non-significant trailing zeros that still satisfies other formatting rules, *e.g.* '13.0' instead of '13.000'.

5. Do not include the '+' symbol, *e.g.* '42.0' instead of '+42.0'.

6. Do not include a signed zero, *e.g.* '0.0' instead of '+0.0' or '-0.0'.

The following regular expression may be used to check if a number adheres to the real number range limit formatting rules:

```
^
(
  # Real number '0.0'.
  ( 0[.]0 ) |
  # All integer-like numbers, e.g. '-5.0'.
  ( [-]?([1-9][0-9]*)[.]0 ) |
  # All remaining floating-point numbers.
  ( [-]?(0|([1-9][0-9]*))[.]([0-9]*[1-9]) )
)
$
```

The regular expression above is formatted for readability using the additional syntax rules enabled by the '/x' Perl regular expression modifier (*e.g.* any unescaped whitespace symbols must be ignored).

Examples of properly formatted real number ranges

```
0.0:100.0
0.0:
:13.0
-180.0:180.0
-3.14:3.14
0.95:1.0
```

3. Data items

3.1 Attribute–value pairs

Note the following rule assumes that no DDLm attributes are longer than $value\_col - text\_indent - min\_whitespace$. The length of a value includes the delimiters. The rules for attribute–value pairs cover items from Set categories as well as items from single-packet Loop categories.

1. DDLm attributes appear lowercased at the beginning of a line after *text_indent* spaces.

2. A value with character length that is lesser or equal to $line\_length - value\_col + 1$ starts in column *value_col*.

3. A value with character length that is greater than $line\_length - value\_col + 1$ and lesser or equal to $line\_length - value\_indent + 1$ starts in column $value\_indent + 1$ of the next line.

4. A value with character length greater than $line\_length - value\_indent + 1$ is presented as a semicolon-delimited text string or as a multi-line compound object.

5. **_description.text** is always presented as a semicolon-delimited text string.

6. Attributes that take default values (as listed in 'ddl.dic') are not output, except:

(i) Those that participate in category keys
(ii) The following attributes from category TYPE: **_type.purpose**, **_type.source**, **_type.container**, **_type.contents**
(iii) Attributes used outside definitions (*e.g.* **_dictionary.class**)

Examples

```
_definition.id                '_alias.deprecation_date'
```

Maximum length value that can still appear on the same line (46 characters):

```
_description_example.case     'Quoted value with padding: 123456789A1234567'
```

Minimum length value that must appear on the next line (47 characters):

```
_description_example.case
    'Quoted value with padding: 123456789A12345678'
```

Maximum length value that can appear on the next line (72 characters):

```
_description_example.case
    'Quoted value with padding: 123456789A123456789B123456789C123456789D123'
```

Minimum length value that requires semicolon delimiters (75 characters):

```
_description_example.case
;
    Quoted value with padding: 123456789A123456789B123456789C123456789D1234
;
```

Long values with no internal whitespaces that fit into a single line should be presented without indentation as specified in rule 2.1:

```
_description_example.case
;
InChI=1S/C6H12O6/c7-1-2-3(8)4(9)5(10)6(11)12-2/h2-11H,1H2/t2-,3-,4+,5-,6?/m1/s1
;
```

Long values with no internal whitespaces that do not fit into a single line should be folded and presented without indentation as specified in rule 2.1:

```
_description_example.case
;\
InChI=1S/C40H60N10O12S2/c1-5-20(4)31-37(58)44-23(12-13-29(41)52)33(54)45-25(17-\
30(42)53)34(55)48-27(39(60)50-14-6-7-28(50)36(57)47-26(40(61)62)15-19(2)3)18-63\
-64-32(43)38(59)46-24(35(56)49-31)16-21-8-10-22(51)11-9-21/h8-11,19-20,23-28,31\
-32,51H,5-7,12-18,43H2,1-4H3,(H2,41,52)(H2,42,53)(H,44,58)(H,45,54)(H,46,59)(H,\
47,57)(H,48,55)(H,49,56)(H,61,62)/t20-,23+,24+,25?,26-,27-,28-,31+,32?/m1/s1
;
```

3.2 Loops

Loops consist of a series of packets. Corresponding items in each packet should be aligned in the output to form visual columns. To avoid confusion with 'column' in the sense of 'horizontal character position', these visual columns are called 'packet items' in the following. Note that loops in dictionaries rarely have more than 2 such packet items. The 'width' of a packet item is the width of the longest data value for the corresponding data name, including delimiters. The rules below are designed to make sure that packet items align on their first character, and that loops with only two packet items are readable.

1. A loop containing a single data name and single packet is presented as an attribute –value pair.

2. The lowercase **loop_** keyword appears on a new line after *text_indent* spaces and is preceded by a single blank line.

3. The n lowercase, looped attribute names appear on separate lines starting at column $text\_indent + loop\_indent + 1$.

4. Each packet starts on a new line. The final packet is followed by a single blank line.

5. The first character of the first value of a packet is placed in column *loop_align*.

6. Non-compound values that are longer than $line\_length - loop\_step$ characters are presented as semicolon-delimited text strings.

7. Semicolon-delimited text strings in loops are formatted as for section 2.1, except that they are indented so that the first non-blank, non-prefix character of each line aligns with the first alphabetic character of the data name header, that is, the first non-blank character appears in column $text\_indent + loop\_indent + 2$.

8. If the number of looped attributes $n > 1$, values in packets are separated by *min_whitespace* together with any whitespace remaining at the end of the line distributed evenly between the packet items. The following algorithm achieves this:

(i) Find largest integer p such that no data values before packet item p on the current line contain a new line and the sum of the widths of next p packet items, separated by *min_whitespace* is not greater than *line_length*. Call this *total_width*.
(ii) Calculate *remaining_whitespace* as floor$[(line_length - total_width)/(p-1)]$.
(iii) The start position of values for attribute number $d+1$ is start position of attribute d + width of data name d + *min_whitespace* + *remaining_whitespace* + 1.
(iv) If $p < n$, the next value is placed in column *loop_step* on a new line and procedure repeated from step 1.
(v) If any values for a data name contain a new line, data values following that data value start from step 4.
(vi) Notwithstanding (4), the starting column for multiline compound data values is that given in section 2.4.

9. If there are two values on a single line and the rules above would yield a starting column for the second value that is greater than *value_col*, the calculated value is replaced by *value_col* unless it would be separated by less than *min_whitespace* from the first value in the packet.

10. If there are two values in a packet and the second value would appear on a separate line, *loop_step* in rule 3.2.8(iv) above is replaced by $loop\_align + text\_indent$. If one of the values is semicolon-delimited and the other is not, the semicolon-delimited value has an internal indent of $loop\_align - 1$.

Examples

Alignment of semicolon-delimited text strings:

```
    loop_
      _enumeration_set.state
      _enumeration_set.detail
         Attribute
;
         Item used as an attribute in the definition
         of other data items in DDLm dictionaries.
         These items never appear in data instance files.
;
         Functions
;
         Category of items that are transient function
         definitions used only in dREL methods scripts.
         These items never appear in data instance files.
;
```

Alignment of semicolon-delimited text strings when both values are semicolon-delimited:

```
    loop_
      _description_example.case
      _description_example.detail
;
       Example 1 in the first semicolon delimited field.
;
;
       Detail 1 in the second semicolon delimited field.
;
;
       Example 2 in the first semicolon delimited field.
;
;
       Detail 2 in the second semicolon delimited field.
;
```

Alignment of single-line values:

```
    loop_
      _enumeration_set.state
      _enumeration_set.detail
         Dictionary             'applies to all defined items in the dictionary'
         Category               'applies to all defined items in the category'
         Item                   'applies to a single item definition'
```

4. Ordering

4.1 Front matter and definitions

1. The first line contains the CIF2.0 identifier with no trailing whitespace.

2. Between the first line and the data block header is an arbitrary multi-line comment, consisting of a series of lines commencing with a hash character. The comment-folding convention is not used.

3. A single blank line precedes the data block header.

4. The final character in the file is a new line (\n).

5. A single blank line follows the data block header.

6. **data_** is lowercase in the data block header.

7. The first definition is the 'Head' category.

8. A category is presented in order: category definition, followed by all data names in alphabetical order, followed by child categories.

9. Categories with the same parent category are presented in alphabetical order.

10. Notwithstanding (8), SU definitions always follow the definitions of their corresponding Measurand data names.

11. Notwithstanding (9), categories with **_definition.class** of 'Functions' appear after all other categories.

4.2 Layout of non-save-frame information

1. All non-looped attributes describing the dictionary appear before the first save frame, in the following order:

(i) **_dictionary.title**
(ii) **_dictionary.class**
(iii) **_dictionary.version**
(iv) **_dictionary.date**
(v) **_dictionary.uri**
(vi) **_dictionary.ddl_conformance**
(vii) **_dictionary.namespace**
(viii) **_description.text**

2. All looped attributes describing the dictionary are presented as loops appearing after the final save frame, in the following category order. Looped data names appear in the order provided in brackets.

(i) DICTIONARY_VALID (**scope**, **option**, **attributes**)
(ii) DICTIONARY_AUDIT (**version**, **date**, **revision**)

3. **_dictionary_audit.revision** is always presented as a semicolon-delimited text string.

4. Non-looped attributes not covered in rule 4.2.1 appear in alphabetical order after **_dictionary.namespace**.

5. Looped attributes not covered in rule 4.2.2 appear before DICTIONARY_VALID in alphabetical order of category, with data names in each loop provided in the order: key data names in alphabetical order, followed by other data names in alphabetical order.

4.3 Definition layout

1. One blank line appears before and after the save frame begin and end codes. The variable part of the save frame begin code is uppercase for categories and lowercase for all others.

2. **_import.get** attributes are separated by one blank line above and below.

3. IMPORT_DETAILS attributes are not used.

4. Attributes in a definition appear in the following order, where present. The names in brackets give the order in which attributes in the given category are presented.

(i) DEFINITION (**id**, **scope**, **class**)
(ii) DEFINITION_REPLACED (**id**, **by**)
(iii) ALIAS (**definition_id**)
(iv) **_definition.update**
(v) DESCRIPTION (**text**, **common**)
(vi) NAME (**category_id**, **object_id**, **linked_item_id**)
(vii) **_category_key.name**
(viii) TYPE (**purpose**, **source**, **container**, **dimension**, **contents**, **contents_referenced_id**, **indices**, **indices_referenced_id**)
(ix) ENUMERATION (**range**)
(x) ENUMERATION_SET (**state**, **detail**)
(xi) **_enumeration.default**
(xii) **_units.code**
(xiii) DESCRIPTION_EXAMPLE (**case**, **detail**)
(xiv) **_import.get**
(xv) METHOD (**purpose**, **expression**)

5. Any attributes not included in this list should be treated as if they appear in alphabetical order after the last item already listed for their (capitalised) categories above. If the category does not appear, the attributes are presented in alphabetical order of category and then **object_id** after DESCRIPTION_EXAMPLE.

5. Naming convention

1. Save frame code of a data item definition frame should be identical to the lowercase version of the **_definition.id** attribute value contained in the definition, with any leading underscores removed.

2. Save frame code of a category definition should be identical to the uppercase version of the **_definition.id** attribute value contained in the definition, with any leading underscores removed.

APPENDIX 3: Reproduced from the *IUCr Newsletter*.

Feature article *IUCr Newsletter* (2021), Vol. **29**, no. 4

30 Years of CIF

James Hester and Brian McMahon

A new paradigm in data characterization

Thirty years ago, in November 1991, *Acta Crystallographica Section C* published the first of a new category of articles, Regular Structural Papers. At the same time, an article appeared in *Acta Crystallographica Section A* describing a new standard archive file for crystallography. The two, of course, were not unconnected: the *Acta C* article was the first to be submitted using the new file format, CIF, described in the *Acta A* paper. Since that time, CIF has grown to be the accepted standard way to describe crystal and molecular structures derived from single-crystal X-ray diffraction experiments. Many journals will not publish such a structure unless it is accompanied by a CIF; and in many cases, the decision to publish will have included a technical review of the quality of the structure using the IUCr's automated *checkCIF* procedure.

However, CIF has grown far beyond its original design as a standard file format for single-crystal structure reports, and extensions of the original standard are found in an increasing number of crystallographic and other structural science applications. The acronym now stands for 'Crystallographic Information Framework', in recognition of its application across all of these fields, and many of the Commissions of the IUCr are actively engaged in extending its use within their disparate fields. In an age of data-driven science, crystallography has come to be seen as a pioneer in defining experimental and derived data with the precision and scope necessary to achieve the goals of the FAIR data management strategy – namely, to ensure that data are findable, accessible, interoperable and reusable.

Out of the blue?

As with most great ideas, CIF did not spring into the world fully-formed and with no history (Fig. 1). Of course, crystallography is a discipline blessed by an area of study that – to a first approximation – is inherently orderly and well defined (regular three-dimensional packing of atomic or molecular motifs), and that produces copious results from certain reasonably standard types of experiment (diffraction, whether of X-rays, electrons or neutrons). And classification of crystallographic properties, whether of mineral types, crystal habits, or space group symmetries, has been a key activity for decades, if not centuries.

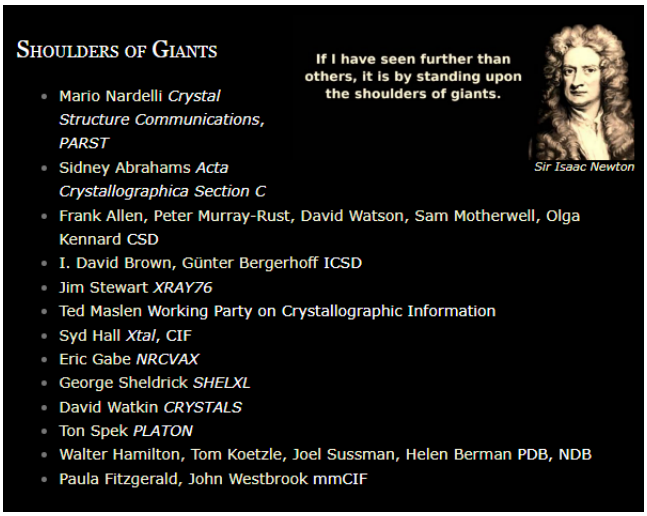


Figure 1. Selection of key contributors to the evolution of 'a new standard archive file for crystallography' that would revolutionize the reporting of crystal structures in databases and journals.

Even so, efforts towards capturing all the information necessary to repeat a crystal structure determination, especially when electronic computers became a significant research tool, go back further than many people realise. Fig. 1 captures some of the steps in the path towards the creation and early implementation of the CIF standard, based on the current authors' memories. Mario Nardelli, a distinguished and prolific structural chemist, launched the journal *Crystal Structure Communications* (1972–1983) at his home institution at University of Parma, and developed software (*Parma Structural Checking – PARST*) to ease the labour of checking the reported structures. When the journal was taken into the IUCr family of publications as *Acta Crystallographica Section C*, its rigorous standards of checking were adopted by the Editor-in-Chief, Sidney Abrahams, who formalised the requirements for information (what we would now call experimental metadata) needed to check the consistency and reasonableness of the reported structure. Similar requirements were emerging at that time within the relatively new structural databases, such as those developed for inorganic structures (ICSD) and organic/organometallic structures (CSD). Within the Cambridge Crystallographic Data Centre (CCDC), validation software (*UNIMOL*) was developed and subsequently used for the CSD and in early automated structure checking in IUCr journals.

Special mention should be made of David Brown, of McMaster University, who led the IUCr's first initiative towards a computer-readable standard file format – the Standard Crystallographic File Structure (SCFS)[1]. In the 1980s, when Fortran dominated scientific software development, any such standard was inevitably tied to Fortran input/output conventions (80-column records, fixed field lengths for different types of data); and by the late 1980s this was seen as lacking the flexibility to meet the increasingly complex requirements of increasingly capable and ambitious programming projects. Nevertheless, the SCFS project (informed by the requirements of the databases and journals) identified many of the discrete data items that needed to be transferred between different crystallographic programs, work that was to bear fruit in the development of the CIF core dictionary.

Jim Stewart pioneered a type of structured information storage within the crystallographic software package *XRAY* that was further refined by Syd Hall (University of Western Australia) in *Xtal*, an early example of a collaborative project where crystallographic programmers around the world contribute separate modules to a growing library of software. Syd, Frank Allen of the CCDC and David Brown (Fig. 2) together developed CIF as a standard format in 1991, an outcome of the Working Party on Crystallographic Information that the IUCr had convened in 1987 under the leadership of Ted Maslen. Together with a lightweight extensible free-form structure, CIF was populated by the data items identified as essential to validation and structure characterization in databases and journals. Authors of leading structure refinement software of the time rapidly adopted this standard, giving the impetus for its universal adoption within the small-unit-cell community. Ton Spek developed a very powerful software tool that checked both the internal consistency of the reported structure and its chemical reasonableness, the latter informed by the wealth of data in the structural databases.

Meanwhile, the Protein Data Bank (PDB), founded in 1971 as a repository for biological macromolecular structures, was also feeling the limitations of the Fortran-style standard that it had adopted. The community embarked upon an extension of CIF – the macromolecular Crystallographic Information File – that could capture all the experimental data associated with a protein structure determination by X-ray diffraction, but that could also adequately describe the complex structure of intricately folded protein and nucleic acid molecules. To meet the needs of the new relational

Figure 2. David Brown, Syd Hall, Frank Allen: authors of the original CIF specification.

Timeline of Crystallographic Information

An updated version of the interactive timeline at http://www.iucr.org/resources/cif/comcifs/symposium-2013/timeline

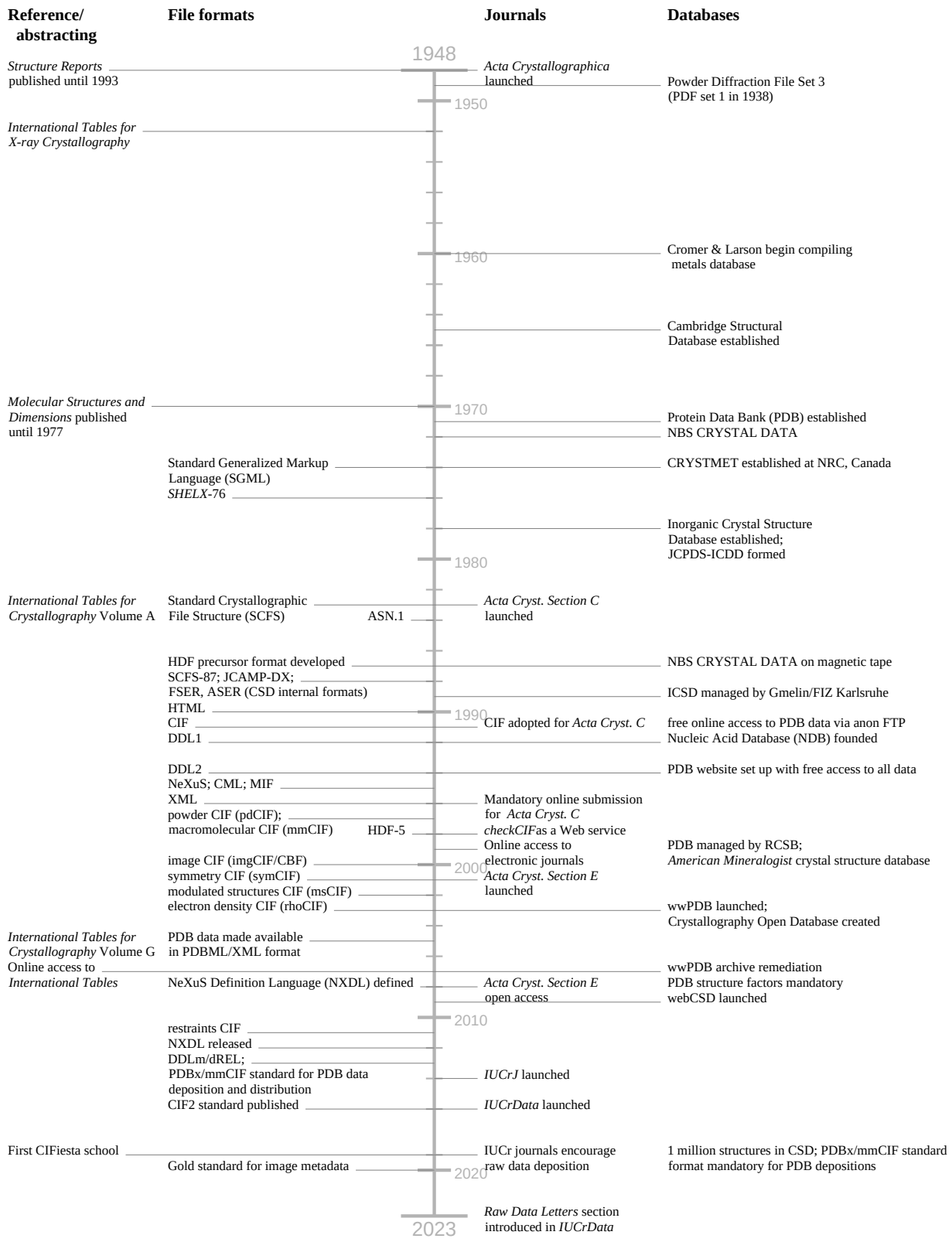


Figure 3. Timeline of developments in crystallographic information before and after the publication of the CIF standard.

database that was being designed for the PDB, mmCIF imposed more constraints on the relations between the data items that it defined. In a series of workshops[2], the mmCIF standard was refined to become a superset of the core dictionary with the additional attributes that made it formally equivalent to a relational database.

CIF and Crystallography

Many young researchers will have grown up (Fig. 3) with the idea of CIF as the natural way to publish single-crystal structure reports and import structural models; and they will be equally familiar with *checkCIF* as a validation tool and indicator of the completeness and precision of a structure determination.

While many journals require a CIF as supporting information, Fig. 4 shows the particular power of the complete integration of CIF in the publishing process that IUCr journals offers. In this interactive figure, the author has provided some alternative views of the molecule studied, highlighting different areas of interest. However, the reader can right-click into the main image to find a much larger menu of options, permitting visualization of the unit-cell packing, the crystal structure, or individual symmetry operations; and interrogation of the data for arbitrary geometric measurements.

However, researchers may have had fewer opportunities to use the small-unit-cell derivatives of CIF that have been developed during their lifetime, and that are gradually becoming established as ways to describe more complex types of structure. There are now separate extension dictionaries that cover such fields as: powder diffraction (embracing the different measurables of a wider range of instrumentation, the practice of using multiple data sets to fit a single crystallographic model, and the need to characterize different phases); modulated and composite structures (with the ability to assign superspace groups and modulation wave vectors); magnetic structures (describing both commensurate and incommensurate magnetic structures exhibiting long-range three-dimensional magnetic order); and the description of the topology of lattices and their relation to crystal structures (Fig. 5).

There are also small extension dictionaries describing aspects of twinning, multipole expansion of electron density, and structure refinement restraints and constraints. These are not very widely used, but are available as starting points for more detailed treatment when required. Many of the IUCr Commissions are also interested in developing CIF dictionaries to provide standard machine representations within their own spheres of interest.

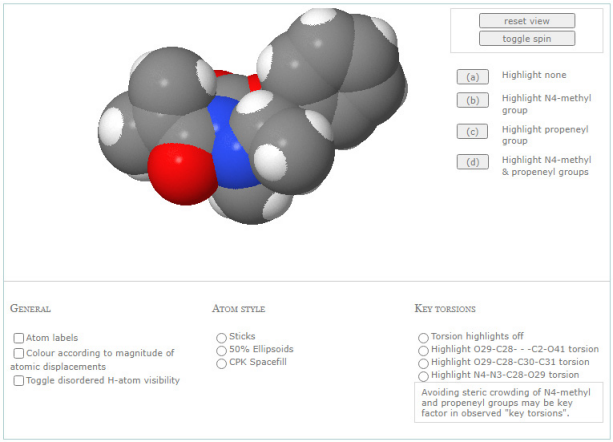


Figure 4. Slightly modified version of Fig. 2 of Knott *et al.* (2008)[3] demonstrating the ability to visualize and interrogate CIF data sets *in situ* within a research publication. The interactive version of this figure can be visited online at https://www.iucr.org/news/newsletter/volume-29/number-4/30-years-of-cif/interactive-figure. Interactive functionality provided by *Jsmol* [4].

In the area of structural biology, the Worldwide Protein Data Bank curates the mmCIF family of dictionaries, including PDBx (the main extension to mmCIF that tracks the developing field of protein crystallography) and a number of extension dictionaries relevant to NMR structure determinations, small-angle scattering, three-dimensional electron microscopy, integrative/hybrid methods, features of synchrotron radiation facilities and beamlines, and validation reports.

Figure 5. Logos of the CIF dictionaries under the curation of the IUCr Committee for the Maintenance of the CIF Standard (COMCIFS) published by 2021.

Another important development has been imgCIF, designed to capture diffraction image data and also the necessary experimental metadata allowing proper interpretation of the images captured. Since imgCIF and its binary data representation equivalent CBF (the Crystallographic Binary File) were developed in the late 1990s, data acquisition volumes and rates have increased so rapidly that this type of file format is no longer suitable for real-time data capture. Nevertheless, the data definitions of imgCIF have partly informed the NeXus macromolecular crystallography application definition (NXmx) that provides full metadata definitions in the HDF5 format increasingly used by beamlines.

This is a significant illustration that the real power of CIF lies in its definitions of concepts and quantities. While the concrete CIF file format is a useful information exchange mechanism, it is relatively easy to translate the format to other common standards, such as XML, the old PDB format, or – increasingly popular – JSON.

A simple catalogue of this plethora of dictionaries and their specific applications perhaps obscures the real importance that CIF has acquired during the course of its evolution. It now touches upon the complete workflow of a structure determination, from the capture of the experimental data, through its interpretation, modelling and publication, to worldwide dissemination in curated databases (Fig. 6). In data science applications, the collection of controlled vocabularies and interrelationships among detailed data definitions has come to be known as an 'ontology', and crystallography now has one of the most completely developed of any science. It is now possible to offer schools and workshops to early-career structural scientists where experimental best practice is developed with the full rigour of data characterization, interpretation and validation[5].

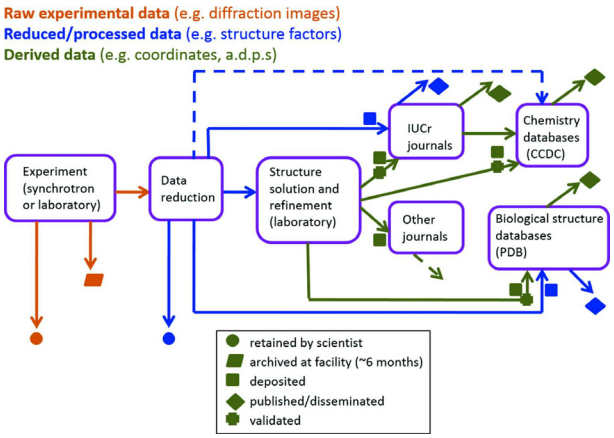


Figure 6. A coherent information flow in crystallography. CIF ontologies characterize data at every stage of the information processing life cycle, from experimental apparatus to published paper and curated database deposit.

Leading the way

The significance of CIF has been recognised in the information and data science communities, in the form of the 2006 Association of Learned and Professional Society Publishers Award for Publishing Innovation and the prestigious 2014 CODATA Prize to Professor Sydney Hall. In both cases the award citations were generous in their acclaim for what the CODATA judges called 'a momentous contribution'.

Yet despite the potential applicability of the approach to any field of science, there has been relatively little penetration in other scientific domains. Syd Hall intended CIF to be just one application of a general approach – 'STAR' (Self-Defining Text Archive and Retrieval) – which has also been used in small-scale pilots in chemistry (MIF, the molecular information file), quantum chemistry, and botany, and most successfully as the basis for the NMR structures database of BioMagResBank, one of the partners of the wwPDB[6].

There are also encouraging signs of novel projects in solid-state science that are inspired by CIF[7]. These are greatly to be welcomed, as a proliferation of domain ontologies in a common format will certainly lead to easier interoperability. It is likely that, despite its success within crystallography, many other disciplines consider the STAR/CIF approach as niche, and not sufficiently supported by the wider information and data science communities. Nevertheless, in our opinion as people who have been involved with CIF for a combined 45 years, STAR and CIF have much to offer, not least in the process of devising new ontologies. The basic file syntax is lightweight and clean, and the dictionary attribute sets (that is, the terms used to express definitions of concepts in machine-readable ways) are not unduly complex, and extensible where needed.

Relationships may be expressed in many different ways – early interest in the STAR File with its nested loops that are absent from CIF explored its suitability for populating object-–relational databases. Yet the more natural relational-database type structure that CIF more easily encapsulates is adequate for developing conceptual frameworks of sufficient complexity for most scientific purposes[8]. We respectfully encourage scientists who need to provide machine-readable descriptions of scientific data and metadata within their own discipline to consider the STAR/CIF approach as a useful starting point. Development into more complex web ontology frameworks such as OWL[9], if needed, can be left to a later implementation stage, once the essential definitions and relationships have been expressed in the manner of a CIF dictionary.

Constancy and change

As is apparent from the title of the 1991 CIF paper, one of its primary design goals was to archive data. As such, stability is crucial, both of the file format and of the dictionary definitions. Definitions should not be changed, as that could invalidate archived data sets. On the other hand, concepts do evolve, and the dictionaries will acquire new data names with their associated definitions. Where there is a clear shift in the way an existing concept is realised, there are mechanisms to mark an old data name as deprecated and to express its relationship to a new entry.

In fact, the core dictionary has grown rather little since its original release, reflecting the relative maturity of single-crystal structure determination. In marked contrast, the PDBx extension to the original mmCIF dictionary has raised the number of data names defined for the core biological macromolecular structure determination from around 1670 to over 6400, reflecting both the complexity of the structural description of proteins and nucleic acids and the explosive growth of the subject itself.

To accommodate new requirements in data management the CIF file format itself was revised in 2016[10]. Its main differences from version 1 were the adoption of Unicode as the native character set, the ability to represent all possible text strings, and simpler ways to represent vectors, matrices and other compound data structures. This was coincident with the adoption of a new formalism for dictionary definitions known as DDLm, a methods-capable dictionary definition language. The term 'methods' indicates that formal relationships between different data items can be expressed, evaluated and validated by machine-executable statements within the dictionary itself. This opens the way to more rigorous validation tools, and in principle brings closer the idea of a universal processing engine that can manipulate scientific entities by inputing a suitable dictionary – no domain-specific coding would be needed within such programs.

Again reflecting the maturity of single-crystal diffraction and the existence of established software that adequately handles the publication requirements of such structures, relatively few 'CIF2' files are yet found in the wild. However, we expect that they will be popular in newer areas of research, and we note, for example, the adoption of CIF2 in a novel Raman spectroscopy database[11].

The image we have chosen to introduce this article is, we feel, an appropriate metaphor for CIF at the end of its first three decades of deployment. From small beginnings, it has grown to a significant size, constantly propelled by a small core of developers and adopters. In the process, it may have acquired some grit and irregularities around the edge, and is now at a stage where even more effort will be needed, especially to gather up more material from the fresh 'snowfields' of new techniques, structural representations, and understanding.

However, most of the pioneers from the early days have now retired or otherwise withdrawn from the scene. If we are not to lose momentum, we need fresh young blood to keep the snowball rolling and growing ever larger. Please join in the fun!

Resources

Find out more about CIF:

- The IUCr CIF website
- CIF dictionaries maintained by COMCIFS
- CIF software for small-unit-cell structures
- PDBX/mmCIF dictionary resources maintained by wwPDB
- CIF software for macromolecular structures
- cif-developers mailing list
- Crystallographic Information and Data Management Symposium, U. Warwick, 2013
- Crystallographic Information Fiesta School, Naples 2019
- International Tables for Crystallography Volume G: Definition and exchange of crystallographic data (1st online edition 2006; a second edition is in preparation)

Notes and references

[1] Brown, I. D. (1988). Standard Crystallographic File Structure-87. *Acta Cryst.* A**44**, 232. DOI: https://doi.org/10.1107/S010876738700970X

[2] Fitzgerald, P., Berman, H., Bourne, P., McMahon, B., Watenpaugh, K. & Westbrook, J. (1996). The macromolecular CIF dictionary https://www.iucr.org/resources/commissions/crystallographic-computing/schools/school96/the-mmcif-dictionary

[3] Knott, S. A., Hitchcock, S. R. & Ferrence, G. M. (2008). (5*S*,6*S*)-4,5-Dimethyl-3-methylacryloyl-6-phenyl-1,3,4-oxadiazinan-2-one. *Acta Cryst.* E**64**, o1101. DOI: https://doi.org/10.1107/S1600536808013986

[4] *Jsmol* is the HTML5 modality of *Jmol*: an open-source Java viewer for chemical structures in 3D. http://www.jmol.org/ Under continual development by its principal developer, Robert M. Hanson, this tool provides innovative and insightful visualizations of structures described by many flavours of CIF, including macromolecular (mmCIF), incommensurate modulated (msCIF), magnetic (magCIF) and topological (topoCIF)

structures.

[5] The pioneering CIFiesta school, hosted by the Italian Crystallographic Association in Naples in 2019, has been described in the *IUCr Newsletter*. Teaching materials from the school are available at: https://www.iucr.org/resources/cif/comcifs/cifiesta-2019

[6] Hall, S.R. & McMahon, B. (2016). The Implementation and Evolution of STAR/CIF Ontologies: Interoperability and Preservation of Structured Data. *Data Science Journal*, **15**, p.3.

[7] Some examples of recent standards development efforts in solid-state science: Zainul Ihsan, A., Dessi, D., Alam, M., Sack, H. & Sandfeld, S. (2021). Steps towards a Dislocation Ontology for Crystalline Materials. *arXiv [cond-mat.mtrl-sci]*, arXiv:2106.15136; Evans, J. D., Bon, V., Senkovska, I. & Kaskel, S. (2021). A Universal Standard Archive File for Adsorption Data. *Langmuir*, **37**, 4222–4226. DOI: https://doi.org/10.1021/acs.langmuir.1c00122; Andersen, C. W., Armiento, R., Blokhin, E. *et al.* (2021). OPTIMADE, an API for exchanging materials data. *Sci Data* **8**, 217. DOI: https://doi.org/10.1038/s41597-021-00974-z; Cheung, K., Drennan, J. & Hunter, J. (2008). Towards an ontology for data-driven discovery of new materials. *AAAI Spring Symposium: Semantic Scientific Knowledge Integration*, pp. 9–14.

[8] Hester, J. (2016). A Robust, Format-Agnostic Scientific Data Transfer Framework. *Data Science Journal*, **15**, p.12. DOI: http://doi.org/10.5334/dsj-2016-012

[9] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. & Rudolph, S. (2012). OWL 2 Web Ontology Language Primer In: Tech. rep. W3C. Available at: https://www.w3.org/TR/2012/REC-owl2-primer-20121211.

[10] Bernstein, H. J., Bollinger, J. C., Brown, I. D., Grazulis, S., Hester, J. R., McMahon, B., Spadaccini, N., Westbrook, J. D. & Westrip, S. P. (2016). Specification of the Crystallographic Information File format, version 2.0. *J. Appl. Cryst.* **49**, 277–284. DOI: https://doi.org/10.1107/S1600576715021871

[11] El Mendili, Y., Vaitkus, A., Merkys, A., Grazulis, S., Chateigner, D., Mathevet, F., Gascoin, S., Petit, S., Bardeau, J.-F., Zanatta, M., Secchi, M., Mariotto, G., Kumar, A., Cassetta, M., Lutterotti, L., Borovin, E., Orberger, B., Simon, P., Hehlen, B. & Le Guen, M. (2019). Raman Open Database: first interconnected Raman–X-ray diffraction open-access resource for material identification. *J. Appl. Cryst.* **52**, 618–625. DOI: https://doi.org/10.1107/S1600576719004229

Appreciation

As we were working on this article, we were saddened to hear of the unexpected passing of John Westbrook (1957–2021), who did so much to develop the mmCIF/PDBx and related extension standards, and whose companionship along almost the entire route of CIF development will be greatly missed.

Brian McMahon is based at the IUCr offices in Chester, UK, and has been COMCIFS Secretary since 1993. James Hester works at ANSTO, Australia, and has been COMCIFS Chair since 2008.

25 October 2021

APPENDIX 4: Additional resources for learning about CIF dictionaries

CIF dictionaries maintained by COMCIFS

The CIF section of the IUCr web site, especially

- CIF home page https://www.iucr.org/resources/cif/
- CIF dictionaries section
  https://www.iucr.org/resources/cif/dictionaries

Standards development site on GitHub, especially

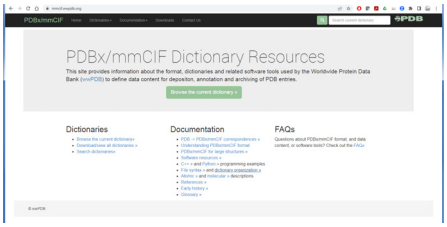
- IUCr Core CIF development repository (includes DDLm dictionary)
  https://github.com/COMCIFS/cif_core
- Dictionary Writing Workshop materials
  https://github.com/COMCIFS/Dictionary-Writing-Workshop

International Tables for Crystallography Volume G: Definition and exchange of crystallographic data

- https://it.iucr.org/G
- Online access is by institutional subscription to the entire *International Tables* series.
- Printed copies of the first edition (2005, slightly revised 2010) are available from Wiley, price £232
- Second-hand or discounted prices are sometimes available. Search on ISBN 1-4020-3138-6

The Second Edition, from which draft material has been included in this booklet, is in an advanced stage of preparation.

CIF dictionaries maintained by the Worldwide Protein Data Bank



The wwPDB Dictionary Resources website, especially

- PDBX/mmCIF home page https://mmcif.wwpdb.org/
- Dictionary organization tutorial
  https://mmcif.wwpdb.org/docs/tutorials/mechanics/pdbx-mmcif-dict-struct.html

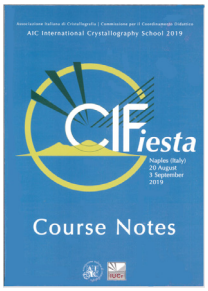
PDB-101 tutorials, especially

- Virtual Crash Course *Use PDB data to their full extent: Understanding PDBx/mmCIF*
  https://pdb101.rcsb.org/train/training-events/mmcif

General

Previous COMCIFS events at IUCr and regional meetings

- Crystallographic Information and Data Management Symposium
  (ECM 28, University of Warwick, UK, August 2013)
  https://www.iucr.org/resources/cif/comcifs/symposium-2013
- Crystallographic Information and Data Management Workshop
  (ECM 28, University of Warwick, UK, August 2013)
  https://www.iucr.org/resources/cif/comcifs/workshop-2013
- Dictionary Writing Workshop
  (IUCr XXIV, Hyderabad, India, August 2017)
  https://www.iucr.org/resources/cif/comcifs/workshop-2017



The Crystallographic Information Fiesta (AIC International School 2019)

- https://www.iucr.org/resources/cif/comcifs/cifiesta-2019
- http://www.iucr.org/news/newsletter/volume-27/number-4/cifiesta