COMCIFS Dictionary Writing Workshop

A Companion Workshop to the 24th Congress and General Assembly of the International Union of Crystallography

Supporting materials

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

The Dictionary Writing Workshop will provide participants with the skills to create high-quality dictionary definitions and complete data dictionaries suitable either for inclusion within the CIF/mmCIF framework or as standalone dictionaries for use within other data frameworks, such as NeXus. Participants will be guided during practical sessions towards the goal of producing a complete dictionary or set of additional definitions in a scientific domain of interest to them.

Aims:

At the end of the workshop, participants will be able to:

- Understand the role that dictionaries play in data specifications
- Understand how datanames are stored in a variety of data formats (including 3-column ASCII/CIF/NeXus)
- Construct a high-quality dataname definition
- Construct a DDL2/m domain dictionary potentially building on previously-existing dictionaries and/or previously-existing data standards

Participants wishing to construct a dictionary for a particular domain are encouraged to bring a wish list of the items for inclusion in the dictionary to the workshop, and will be guided throughout the workshop in constructing their particular dictionary.

Prerequisites: There are no specific requirements. In particular, no programming or CIF experience is assumed.

Contents

Components of a Data Transfer Framework

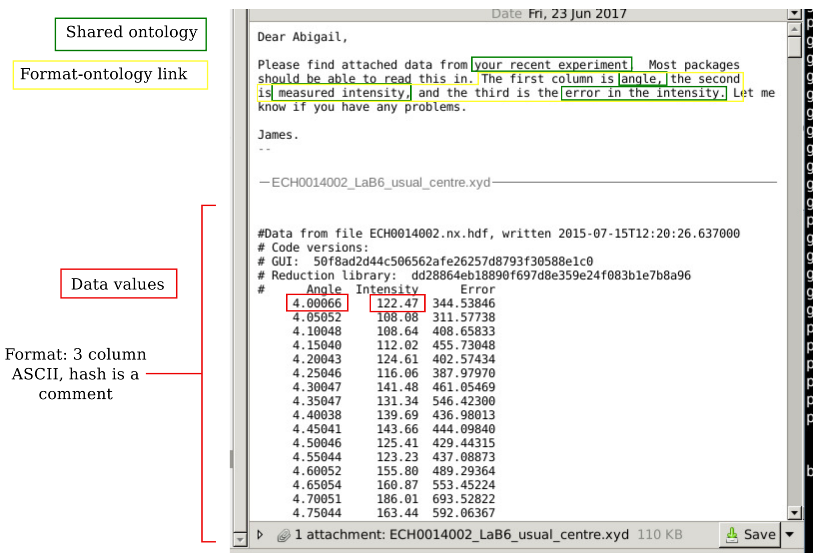
Any system for transferring data, whether it is a three-column ASCII file attached to an email or a CIF file, can be broken into four parts:

1. Data values: the values to be transferred. Values may be anything that we can plausibly transfer *via* computer, including numbers, text, vectors, sets, lists and images.

2. An ontology: a shared understanding of the meanings of the data values. This can be accomplished by attaching values to *data names*. The data name is then used to communicate how the one or more associated values are to be interpreted. The data names are collected into a data name *dictionary*, describing the ontology.

3. One or more file formats: structured arrangements of data values.

4. Linkage between a format and an ontology: a description of how the data names and values from the ontology are encoded in a given format.


Shared ontology
Format-ontology link
Data values
Format: 3 column ASCII, hash is a comment
Date Fri, 23 Jun 2017
—ECH0014002_LaB6_usual_centre.xyd
1 attachment: ECH0014002_LaB6_usual_centre.xyd 110 KB
Save

```
Dear Abigail,

Please find attached data from your recent experiment.  Most packages
should be able to read this in. The first column is angle, the second
is measured intensity, and the third is the error in the intensity. Let me
know if you have any problems.

James.
--
```

```
#Data from file ECH0014002.nx.hdf, written 2015-07-15T12:20:26.637000
# Code versions:
# GUI:  50f8ad2d44c506562afe26257d8793f30588e1c0
# Reduction library:  dd28864eb18890f697d8e359e24f083b1e7b8a96
#       Angle  Intensity       Error
      4.00066     122.47  344.53846
      4.05052     108.08  311.57738
      4.10048     108.64  408.65833
      4.15040     112.02  455.73048
      4.20043     124.61  402.57434
      4.25046     116.06  387.97970
      4.30047     141.48  461.05469
      4.35047     131.34  546.42300
      4.40038     139.69  436.98013
      4.45041     143.66  444.09840
      4.50046     125.41  429.44315
      4.55044     123.23  437.08873
      4.60052     155.80  489.29364
      4.65054     160.87  553.45224
      4.70051     186.01  693.52822
      4.75044     163.44  592.06367
```

Figure 1: The four data transfer components in an email message

The absence of any one of the above four components must lead to a failure in data communication. No data values means no data; no shared ontology means that the sender and receiver would understand the data differently; no format means no electronic encoding of the data; and no linkage means no way of encoding the data values in the format in a mutually understandable way.

The ontology and format–ontology linkage are often only informally or partially specified, relying on a common store of scientific knowledge and obvious meanings of shorthand data names. This approach is reasonably effective in some contexts. A more formal approach describes the ontology using a set of standard attributes in a machine-readable file. Later in this course we will show how to use CIF tools to write such a formal ontology.

Note that the ontology may be cleanly separated from the particular format used to encode the data values. This is in a sense obvious when we consider that observations and results should not depend on the form in which they are communicated. The first part of this course focuses only on describing the data names (that is, developing the ontology). The resulting ontology is applicable to any format.

Data names and their relationships

As described above, each of our data values belongs to a data name, which is a label for the concept that the value is an instance of. Examples include 'wavelength', 'measurement point', 'user name', 'sample composition'. Any data name will have multiple possible values – if a data name could only ever have one value, then there is no point including it in a data transfer framework, as the data file would not be providing any new information for this data name.

Context

Given that any data name could have multiple values, we need some further information to understand what distinguishes the values from one another. We could call this the 'context' of a data value. The context is described by assigning specific values to some set of data names $\{K\}$. If every value of the data names in $\{K\}$ is the same, then our target data name *must* always have the same value (or the logically equivalent statement: if the value of our data name changes, then at least one value of the data names in $\{K\}$ *must* be different). If this is not the case, we add further data names to $\{K\}$ until it is true. As a logical consequence, if any one of the values for the data names in $\{K\}$ is different, then our target data name *may* have a different value. We will call the list of data names $\{K\}$ the *key* data names for the data name that is defined.

The values taken by the data name are a mathematical mapping (function) of the values in $\{K\}$.

Exercise 1. Work out possible key data names for each data name in the following list.

'atomic position'

'atomic element'

'monochromator setting'

'observed intensity'

'calculated intensity'

Observations and Calculations

For convenience we now divide data names into 'observations' and 'calculations'. Calculated values will always be the same if the same input values are used, so the set of key data names for calculated values is just an exhaustive list of parameters for the calculation. Under 'observations' we include not just the particular quantities being measured, but also any descriptive information about the experiment. What are the key data names for such observational data names? Scanned quantities cannot satisfy our criteria. For example, if we measure at a series of energies and measure intensity at each energy point, we may think that our observed intensity values are distinguished from one another by the energy at which they were measured; however, the same value of energy could possibly lead to a different intensity on a subsequent measurement due to statistical variation, so it does not satisfy our criterion that identical values of the key data names *must* correspond to identical values of the defined data name. A key data name for observed quantities is therefore simply 'a measurement point', with arbitrary values that we assign (see below).

It is important not to confuse our model of the world, in which we expect the measured value to be determined by the values of some other parameters, with the actual experiment, in which a series of observations are made.

Making values of 'a measurement point' globally unique is cumbersome. We can add a further key data name, something like 'dataset id', to avoid this. If measurements are divided into a series of scans, with different settings for each scan, 'scan id' might also be a useful additional key data name.

Identifiers

We have introduced a third class of data names, those that are 'identifiers'. In our example above, these are the measurement points. For 'identifier' data names, the actual values are irrelevant, as the values are only used to distinguish or label different members. Examples of identifier data names include measurement points, serial numbers, atom sites, sample numbers and run numbers.

Warning:

While we often use numbers for identifier data values because it is easy to pick a unique one if we label sequentially, their numerical properties should not be used; if an identifier value is more than a simple unique value, 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 then fall into the trap of using 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, for whatever experimental reason.

When choosing identifiers, consider the human user and suggest a natural system of labeling in your definition – labels that are meaningful to humans are just as good as random strings, but the labels should never be manipulated in other definitions.

Unlike other data names, *identifiers do not always have key data names*. Identifiers can appear both as key and non-key data names in the ontology: for example, in our description of a structure an atom site may have 'element name' giving the element occupying that site. Elsewhere in our ontology we might have 'form factor', 'valence', 'isotope', which have 'element name' as the key data name. The values of the former 'element name' are drawn from the values of the latter. It is clearly important to distinguish these two uses of 'element name', as their interpretation is different: one is 'the element at a given atomic site', and the other is 'the element to which this valence/isotope/form factor relates' . For this reason the two distinct uses must be assigned different data names, for example 'atom site element' and 'element'.

Mathematically, identifiers are their own key data names.

The full interpretation might be 'the element at the atomic site in the structural solution for this dataset'

Summary

In order to define a data name we need to identify the key data names, the values that our data name can take, and how to use the values of the key data names to interpret values of the defined data name. A data name describing an observed fact could be defined as simply as ‘the value of A when the measurement was taken’, whereas a data name defining a derived quantity would need to identify all of the parameters of the calculation and the equations involved. References to published works for calculations may be sufficient as the target of our definition is a human reader (probably a scientist–programmer).

Practice questions:

Q 1. During a synchrotron experiment, monitor counts are recorded in a gas filled ion chamber. Which of the following data names are key data names for the counts measured in the ion chamber?

A: the gas pressure
B: the gas mixture
C: the ion chamber length
D: all of the above
E: none of the above

Q 2. During the same synchrotron experiment, variation of transmitted intensity as a sample is moved across the beam is recorded. The expected measured intensity is calculated following the experiment. Which of the following data names are key data names for this calculated intensity?

A: the monitor ion chamber measured intensity
B: the sample thickness
C: the detector voltage
D: all of the above
E: none of the above

Q 3. For efficiency, simultaneous intensity measurements from multi-pixel detectors are stored in a data file as a sequence of bytes that has a particular compression algorithm and integer encoding, which we have assigned data name ‘compressed image’. The particular choice of encoding and compression routine might vary within a single measurement sequence according to factors such as the range of values, maximum value, or detector module. What are the key data names for ‘compressed image’?

Answer: the byte sequence is processed data, so all parameters used in the processing are relevant. In this case the input is the raw data from every pixel, an encoding and a compression id. If any of these change, the byte sequence may be different, and given all of these, the byte sequence is fixed, so they fulfill our requirements for key data names. For brevity we would attach an identifier to each set of pixel data, and could call it ‘raw image id’.

Creating the ontology, step by step

Step 1: Brainstorm data names

Write down all of the concepts that might be included in a data file. For convenience, assign some short, descriptive names to them (these names may change later). Some data names may be implied by what is already in your list.

- Think in terms of objects and their properties, for example, 'an experimenter' may have properties 'name', 'address' 'role' 'photograph'.
- Look at the nouns in your definitions for indications of identifier data names.
- Locate identifiers and consider whether these could or should be classified more finely, just as we divided 'measurement id' into 'dataset id', 'scan id', 'measurement id' above. Such divisions are purely for convenience, and make sense if you expect each identifier to have many values in a given data file *and* you can think of relevant properties that are constant for each value of the identifier. For example, within a single scan the scan step or some orientation might be constant.
- Look at the data files that are already used in your field and extract data names from them. To locate data names, remember every scientifically useful value in a data file belongs to a data name. Examine the context of these values to find key data names. The context in a hierarchical structure typically consists of the nodes above the value of interest, and the values attached to the same node. Further context might be indicated in the specifications.

Step 2: Sharpen up the definitions

For each of your data names from Step 1, write a definition that conveys unambiguously to a human reader the following three things:

1. the nature of the data values (*e.g.* arbitrary identifier, real number, text, vector, integer, yes/no, image)
2. the key data names
3. how to interpret this data name given the values of the key data names

Add any further data names that you have overlooked. A classification into 'observations', 'calculations' and 'identifiers' may help, with identifiers often associated with indefinite nouns like 'an image' 'a measurement' 'an experimenter'. You could use well-defined terminology from your field and references to literature to keep your definition short.

Example:

Finding key data names. What are the key data names for 'an experimental role', which we have defined as 'the role performed by an experimenter during the experiment'?

'An experimental role' is observational, so {'measurement id', 'dataset id'} are key data names. Our definitional sentence includes nouns 'role' and 'experimenter', both of which could become identifiers. If we have a measurement and a person, do we have a single unique role identified? In other words, could one person perform two roles at once? If not (we did after all write '**the** role'), then {'measurement id', 'dataset id' and 'experimenter id'} are sufficient.

Exercise 2. Work out how to represent simultaneous roles. Possible roles might include {'principal', 'attending', 'experimenter', 'dogsbody', 'programmer', 'instrument responsible'} . See the ionisation chamber example below for ideas.

Step 3: Make your definitions computationally useful

Remember that an important reason for this work is to convey information in a way that is manipulable by computer.

1. Any data name that ends up having values that are free text strings (*e.g.* 'sample description') is essentially using the data file as a glorified word processor format, and has a much reduced value in automated settings. So look over your data names, and where you have quantitative information in free or formatted text, rework it into observational or calculated data names that take standard value types.

2. Where you have two or more identifier data names that refer to the same type of thing, with the same key data names, you should rework your ontology as follows. Create a new key data name that will be used to identify combinations of values for these duplicate data names (let's call it 'C'). Now create a second key data name that will take the values that your original data names were supposed to take. Finally, replace your duplicate data names by a single identifier data name that draws from the values of C. The same information is now representable in an extensible way. This technique could be described as creating an associative table.

Example:

Ion chambers used at synchrotrons have sensitivity to the X-ray beam running through them tuned by adjusting the gas or mix of gases in them. We wish to record this information in our data files.

Our starting definition is: **gas mix**, 'the mixture of gases in an ion chamber, in format element-percent-element-percent', with key data name 'detector id'; and other data names that also have 'detector id' as a key data name are 'detector length' and 'location'. If we tabulate this, we might have:

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 described in point (1) above, 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' '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 (2). 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. Following the prescription in Point (2) we create a new identifier **gas mix id** and replace the original identifier data names 'first/second gas' by **gas**. If we have a gas mix id and a gas, we can assign a percentage, so we make these two data names key data names for a new data name 'gas percentage' and drop 'first/second gas percent'. Now, given an ionisation chamber, it is sufficient for us to nominate the gas mix id to completely identify the gas components – but recall from the earlier 'element name' example that the gas mix id that has detector id as its key data name must have a different data name. 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 so 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

3. Where there are limited choices for the value of a data name, explicitly define each of these choices and assign a number or string to them. For example, instead of a data name 'location', with a description of position left up to the software author, you might define 'monitor': before the sample;

‘detector’: measure signal from sample; ‘foil’: measure signal after sample and calibration foil.

4. Bundle up commonly-occuring combinations of values. Where a series of data names are expected to take the same set of values, consider assigning a separate identifier to each set of values and replacing them with a pointer to this identifier.

This example adapted from the imgCIF dictionary

Example:

Consider the simple image ontology discussed in a previous question above. Our initial ontology uses ‘raw image id’, ‘encoding type’ and ‘compression type’ as key data names, using ‘compressed data’ to hold the data. 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 data file, 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’. 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’.

5. Units. Some file formats offer structures that allow the file writer to specify units. Avoid using these as they create extra work for the file reading software in anticipating every possible unit that is appropriate. Usually only one or two units are in common use, so choose and specify a unit in your definition. If the community has not converged on a particular unit, create a second definition that differs only in the unit used.

Tip:

Units. if you allow units to be specified in the data file instead of the definition of some data name A, you could be considered to be creating a new key data name ‘units for A’. One of these key data names will exist for every data name that takes units, and the definition for each of these key data names should list all possible values for the unit in question. This listing could be done explicitly and somewhat economically by referring to external standards, which has the downside that, if these standards are updated, your ontology will also ‘auto update’, whether you like it or not. This can be difficult for programmers who wish to track your ontology.

An alternative view is that your data file is simply providing a missing part of the ontological specification, which software can dynamically implement.

6. Software-specific names. Any data name that essentially refers to the input or output of some 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 occur, attempt to rephrase them in non-software-specific terms. Instead of ‘multiplicity as calculated by XYZ Version 1.2’, write ‘the number

of special positions divided by the number of general positions'.

7. Instrument-specific names. Similarly, 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 independent modelling is unlikely to be of use outside the lab that produced it. Instead of 'Position of motor mom' think 'monochromator takeoff angle'. Of course, a large facility may choose to create an ontology for in-house use in which case such definitions might be sufficient for internal purposes when combined with local knowledge.

Step 4: Data blocks

At the completion of the previous step, your ontology has all the information necessary to use it for data transfer. We now draw out some important features of the ontology for practical use.

When you consider your data names, it is likely that some of them will have the same value over the entire data set that you wish to transfer (*e.g.* user names, beamline, equipment). If we were to actually record these in our data file for every measurement point, it would be a real waste of space. 'Data blocks' group our data values into blocks, and within each block these constant-valued data names are understood to apply to all data values within the block for which they are relevant according to the ontology, like global variables in programming.

Of course, the precise choice of constant data names depends on the experiment. Many current data transfer frameworks suppose a particular set of constant data names, and this assumption carries across to software. Explicit labelling of typical sets of constant data names will both aid software authors, and serve as reminder that all data names could conceivably take multiple values.

Exercise 3. Define at least two sets of constant-valued data names for your field.

Step 5: Categories

Group your data names so that data names in the same group have the same key data names. These groups of data names are called *categories* in CIF. If all the separate values of the key data names are listed in side-by-side columns, the corresponding values of the other data names in the category can be compactly tabulated together with them. Using this strategy, together with separately listing values for data names that do not change within a data block, leads to considerable space reduction when encoding values into files.

Step 6: Naming

It is organisationally useful to name the categories, and then name the data names within them using the category name as a prefix. In this case (i) data names that are closely related will often be close when listed alphabetically (ii) it will be easier for a human reader to recognise which key data names a given data name is related to.

Whether or not you choose to include the category in your name, you must eventually decide on permanent names for each of your draft data names. Short names are good for programmers, but potentially confusing – is 'temp' short for temperature or temporary? Whitespace is not an issue for modern programming languages, but in some contexts (*e.g.* operating system shell scripts) can be annoying.

Summary

You should now have a list of data names, with associated meanings that are unambiguous and fit for use in data transfer contexts. You have defined one or more data blocks.

Using the ontology

As discussed in the introduction, the ontology must be mated with one or more formats in order to transfer data. The format–ontology linkage should specify the data block type(s), the data names that are used, and how to find data for these data names in data files of the chosen format.

While format choice is outside the scope of this workshop, a few general points can be made about format selection:

1. the data values must be representable within the format. This is generally trivially possible, as any value can be represented as text, that is, a sequence of bytes with a specified text encoding, but extra work will be required if programming libraries for the format do not support encoding/decoding of a data value type.

2. the correspondence between each data value and its key data names' values must be representable. This requirement is met by any format that can put data values into ordered lists; in this case values at the same position in a list can be considered to correspond.

3. The format must be extensible to an arbitrary number of multiple-valued data names, to allow for future growth.

4. All other format considerations would be based on non-ontological criteria, such as software support, efficiency, or long-term archival support.

Exercise 4. Consider any data formats that you are familiar with. How well do they meet requirements (1)–(3), and your particular requirements for (4)?

Further topics

Aggregate calculations

Q: Give a key data name for data name 'average measured intensity'

Calculations that depend on a whole collection of data values, such as 'number of measurements', averages, observed uncertainties, and Fourier transforms, have key data names that identify whole sets of data. For a typical data block, there would be only one set of data and so an identifier for the whole data set could be left out of the data file because it is both arbitrary and single-valued. Its existence only becomes apparent when multiple data sets are handled, and some way of referring to a particular set of measurements is needed.

The individual 'observational' data names clearly have some relationship to this 'measurement set id'. A particular measurement can be derived from a measurement set by assigning some unique identifier to each member of each set (which could be our 'measurement id'), and then specifying a measurement set and the particular identifier.

A value that is the result of a Fourier transform will depend on a similar 'set id' that in many cases is also isomorphic to a dataset id. Explicit inclusion of this 'set id' would only become necessary when there are multiple runs of data requiring separate Fourier transformation and recording of the result prior to, for example, merging. Such merging also constitutes an aggregate function that might entail a new id if multiple separate merging processes are to be recorded. And so forth.

Adding and redefining data names

Adding new non-key data names is unproblematic. This is often the case for 'observational' data names, for example, providing a new data name to report humidity during data collection does not affect the intepretation of any other 'observational' information. Similarly, whole new categories (data names and their key data names) can be added with no effect on existing data names.

Once an ontology is published, the relationships between data names and their key data names become embodied in software that is then distributed and relied upon. If we change these relationships later, we risk silent misinterpretation of new data files by legacy software.

Adding new *key* data names to already-existing data names would, in theory, never happen as the context was supposed to have been completely defined when we selected our original key data names. However, as time goes on calculations are improved by the addition of new parameters, or models are expanded. For example, our original ontology for single-crystal crystallography might have listed model structure factor against key data name 'hkl'. When we expand this ontology to include incommensurate structures, we need to add the extra indices as additional key data names. We can avoid the software errors mentioned above by simply duplicating our original category with data names redefined to include the new key data names, but this has the drawback that any categories that referred to data names appearing in our original category will also need redefinition if the link is to be preserved.

A simple solution to this proliferation of data names that mean almost the same thing is to define a data name that identifies the model used. This is an additional key data name that is usually constant for a given data set. Such a data name should be defined when an ontology is first published, so that a check of its value is incorporated into all software.

The Relational Model

The ontological structure described in the previous section maps precisely onto the **Relational Model** used in relational databases. Insights from over 40 years of working with this model can be usefully applied to the structure of our ontology.

Codd, E. F. (1970). *Communications of the ACM. Classics*, **13** (6), 377–387. doi:10.1145/362384.362685

Tip:

This close correspondence to the relational model does not mean that the data file format must be a relational database or related structure. As discussed above, the ontology is applicable to any data file format, and the format should be chosen to be effective given the desired usage patterns (*e.g.* archiving, interchange, storage of massive datasets, centralised access, manual editing...)

Description of the relational model

A **relation** can be represented by a table satisfying a few conditions:

1. Columns can appear in any order.
2. Rows can appear in any order.
3. The values in a given column are drawn from the same **domain** (*i.e.* the same type).
4. No two rows contain completely identical values.

The column headings are called **attributes**. An m-row relation with n attributes is, in fact, just an m-element **set** of n-element **tuples**, where elements are indexed by attribute, not by column order. In the following the words 'tables', 'columns', 'headings' and 'rows' refer to the relational understanding of these words.

The set of attributes whose values taken together uniquely identify a row is called a **key**. Keys that are most relevant to us are:

Candidate key: a set of headings that taken together form a key.

Primary key: the set of headings that is actually used by other relations when referring to this relation.

Foreign key: a heading that is a primary key in a different relation. Note that the relation that has the foreign key is often called the child relation, and the relation for which the foreign key is a primary key is called the parent relation.

Correspondence to our ontology

Clearly, data names are attributes, categories are tables, and the key data names form the primary key. Identifiers appearing as non-key data names are usually foreign keys.

Insights from Relational Databases: Normalisation

Database normalisation aims to define and allocate headings amongst tables in order to meet certain goals. The goals that are relevant to us are:

- Remove redundancy.

  Each piece of information should appear in only one place to avoid the possibility of inconsistency and to save space.

- Minimise redesign when including new headings and tables.

  This is crucial for us as we cannot change the way in which data names are interpreted once the standard is embodied in distributed software.

Important normalisations that can be applied are:

- No embedded information within a data value. ‘unnormalised form’ $\rightarrow$ ‘first normal form’
- No repeated data names in a single relation (*e.g.* gas 1, gas 2)
- No non-key data names that depend on only a proper subset of the key data names. This means that any non-key data name value in a category requires knowledge of all key data name values in order to be determined or interpreted. This avoids repetition. For example, if a relation has a key composed of headings ‘K1’ and ‘K2’, and heading ‘C’ has a value that depends only on heading ‘K1’, if values for ‘K1’ are often repeated then values for ‘C’ will also be repeated, whereas if a separate table had ‘C’ tabulated against ‘K1’, this repetition would be avoided. ‘second normal form’
- No non-key data names depend on the key data names *via* a different non-key data name. Repetition is likely to be reduced if such non-key data names are moved to a separate table, where the non-key data name on which the others depend becomes the key data name, and a foreign key in the original table. ‘third normal form’

Normalisation techniques

The ionisation chamber example from part 1 is an example of a **many-to-many** relationship. Each ionisation chamber can have many gases, and each gas can be in many ionisation chambers. The standard way to express such relationships while maintaining a properly normalised database is through an **associative table**, as created in that example. **One-to-many** and **many-to-one** relationships are simply expressed by using a (key data name) identifier for the **many** and tabulating the **one** against it.

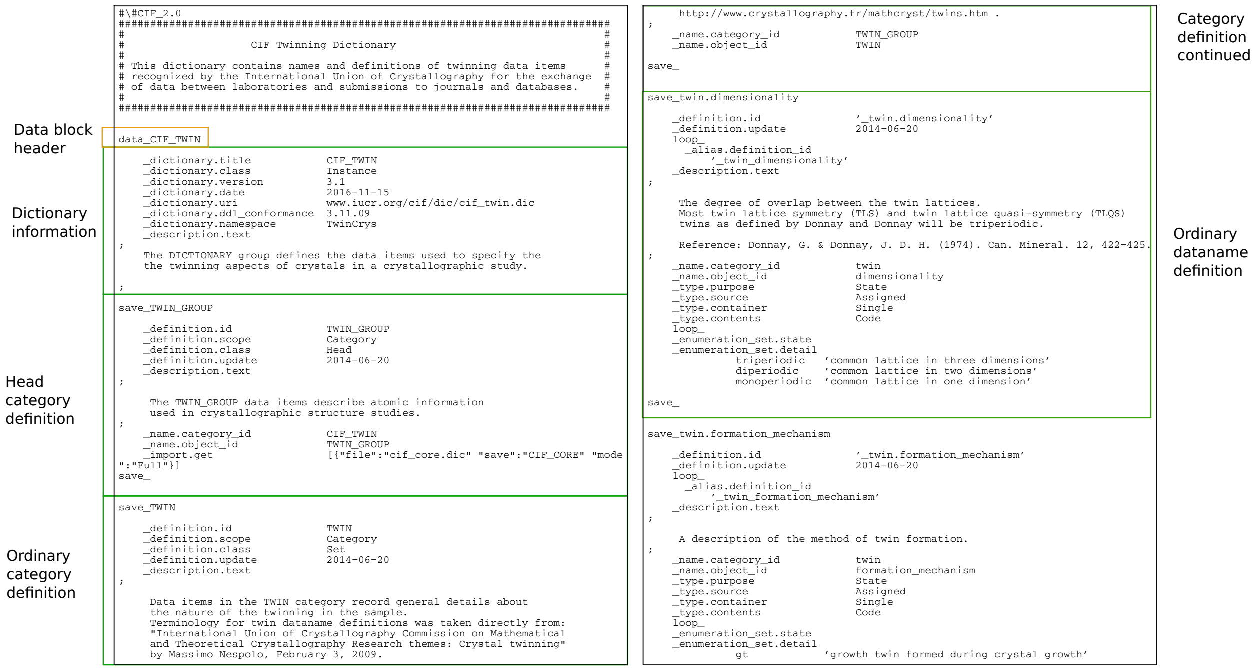
Creating formal dictionaries

The previous sections described how to create a scientific ontology that would underpin a fully-functional data transfer framework. That ontology was just a collection of plain-text data name definitions that satisfied a few basic criteria.

A CIF dictionary is a machine-readable version of that ontology. Just like the original human-readable version, the ontology expressed in a CIF dictionary describes data names that could appear in any format. Among other things, machine-readable ontologies are useful for:

- automated data checking. Relationships between data names can be checked for consistency in data files.
- assistance in constructing the ontology. Tools can ensure that all required information is provided and mutually consistent with other definitions
- assistance in accessing the ontology. Large ontologies benefit from automated tools that can perform searches, locate related data names, and show relationships between data names. The ontology can also be automatically transformed into different presentation formats, such as HTML or PDF.

The structure of a CIF dictionary


Data block header
Dictionary information
Head category definition
Ordinary category definition
Category definition continued
Ordinary dataname definition

```
#\#CIF_2.0
##############################################################################
#                                                                            #
#                    CIF Twinning Dictionary                                 #
#                                                                            #
# This dictionary contains names and definitions of twinning data items      #
# recognized by the International Union of Crystallography for the exchange  #
# of data between laboratories and submissions to journals and databases.    #
#                                                                            #
##############################################################################


data_CIF_TWIN

    _dictionary.title              CIF_TWIN
    _dictionary.class              Instance
    _dictionary.version            3.1
    _dictionary.date               2016-11-15
    _dictionary.uri                www.iucr.org/cif/dic/cif_twin.dic
    _dictionary.ddl_conformance    3.11.09
    _dictionary.namespace          TwinCrys
    _description.text
;
    The DICTIONARY group defines the data items used to specify the
    the twinning aspects of crystals in a crystallographic study.

;

save_TWIN_GROUP

    _definition.id                 TWIN_GROUP
    _definition.scope              Category
    _definition.class              Head
    _definition.update             2014-06-20
    _description.text
;

     The TWIN_GROUP data items describe atomic information
     used in crystallographic structure studies.
;
    _name.category_id              CIF_TWIN
    _name.object_id                TWIN_GROUP
    _import.get                    [{"file":"cif_core.dic" "save":"CIF_CORE" "mode
":"Full"}]
save_

save_TWIN

    _definition.id                 TWIN
    _definition.scope              Category
    _definition.class              Set
    _definition.update             2014-06-20
    _description.text
;

     Data items in the TWIN category record general details about
     the nature of the twinning in the sample.
     Terminology for twin dataname definitions was taken directly from:
     "International Union of Crystallography Commission on Mathematical
     and Theoretical Crystallography Research themes: Crystal twinning"
     by Massimo Nespolo, February 3, 2009.
     http://www.crystallography.fr/mathcryst/twins.htm .
;
    _name.category_id              TWIN_GROUP
    _name.object_id                TWIN

save_

save_twin.dimensionality

    _definition.id                 '_twin.dimensionality'
    _definition.update             2014-06-20
    loop_
      _alias.definition_id
          '_twin_dimensionality'
    _description.text
;

     The degree of overlap between the twin lattices.
     Most twin lattice symmetry (TLS) and twin lattice quasi-symmetry (TLQS)
     twins as defined by Donnay and Donnay will be triperiodic.

     Reference: Donnay, G. & Donnay, J. D. H. (1974). Can. Mineral. 12, 422-425.
;
    _name.category_id              twin
    _name.object_id                dimensionality
    _type.purpose                  State
    _type.source                   Assigned
    _type.container                Single
    _type.contents                 Code
    loop_
    _enumeration_set.state
    _enumeration_set.detail
              triperiodic   'common lattice in three dimensions'
              diperiodic    'common lattice in two dimensions'
              monoperiodic  'common lattice in one dimension'

save_

save_twin.formation_mechanism

    _definition.id                 '_twin.formation_mechanism'
    _definition.update             2014-06-20
    loop_
      _alias.definition_id
          '_twin_formation_mechanism'
    _description.text
;

     A description of the method of twin formation.
;
    _name.category_id              twin
    _name.object_id                formation_mechanism
    _type.purpose                  State
    _type.source                   Assigned
    _type.container                Single
    _type.contents                 Code
    loop_
    _enumeration_set.state
    _enumeration_set.detail
              gt            'growth twin formed during crystal growth'
```

Figure 2: General structure of a CIF dictionary

Syntactically, a CIF dictionary is a CIF file. It contains a single CIF data block, within which each data name definition is defined in a separate **save frame**. Just as for a normal CIF data file, all information in a dictionary is created by associating one or more values with textual tags. In a normal CIF data file, these tags are called 'data names'. To avoid ambiguity, the tags used in a dictionary are referred to as **attributes**. The collection of attributes that can appear in a dictionary is called the *Dictionary Definition Language* or

The 'CIF' dictionary could itself be written in any file format. It is written in CIF format both to save CIF users implementing or learning a separate format, and because of other positive attributes of the CIF format for standards work, such as easy readability, long-term support and easy transformation to a variety of presentations.

DDL. Two similar DDLs are in common use in the CIF framework, **DDL2** and **DDLm**. The earlier DDL1 is no longer used in new dictionaries.

Earlier we listed the core elements of a data name definition: (i) a description of the data values (ii) a list of the key data names (iii) a description of how the data values are interpreted given the values of the key data names. In a practical dictionary, further tags give audit and management information such as the date on which a definition was edited, other names for this data name, and version information for the dictionary as a whole.

Table 1: Selected DDLm attributes.

Role	**Data names**
Naming	_alias.definition_id, **_definition.id**, **_name.object_id**
Describing Values	_enumeration.{def_index_id/default/mandatory/range}, _enumeration_default.{index/value}, **_enumeration_set.{detail/state}**, _type.{**container** / **contents** / dimension}, **_units.code**
Key data names	**_name.category_id** , **_category_key.name**
Interpretation	**_description**.{common/key_words/**text**}, _description_example.{case/detail}, _name.linked_item_id, _method.{purpose/expression}
Technical	_definition.{class/scope},_import.*
Management	_dictionary.{title/class/version/date/uri/ ddl_conformance/namespace/*}, _dictionary_audit.{version/date/revision}, _alias.{deprecation_date/dictionary_uri}, _definition.{replaced_by/**update**/xref_code}, _dictionary_xref.{code/date/format/name/uri}, _enumeration_set.{xref_code/xref_dictionary}, _type.{purpose/source}

Table 2: Selected DDL2 attributes

Role	**Data names**
Naming	**_item.name**, _item_aliases.alias_name, **_category.id**

... continued on next page

Table 2: Selected DDL2 attributes (... continued)

Role	**Data names**
Describing values	_item_enumeration.{value/detail},_item_default.value _item_range.minimum, _item_range.maximum _item_structure.{code/organisation}, _item_structure_list.{index/code/dimension} **_item_type.code**, **_item_units.code** _item_type_list.{code/detail/construct} _item_units_list.{code/detail}
Key data names	**_item.category_id** , **_category_key.name**
Interpretation	**_item_description.description**, **_category.description**, _category.examples _item_examples.{case/detail}, _item_linked.parent_name, _item_linked.child_name _item_methods.method_id _method_list.{id/detail/inline/language/code}

Management	_dictionary.{title/version/datablock_id}
	_datablock.id, _datablock.description
	_dictionary_history.{revision/update/version}

Differences between DDL2 and DDLm dictionaries

- DDL2 has a system for describing values in terms of regular expressions, which the dictionary author may customise, whereas DDLm dictionaries are restricted to the types provided in the DDLm attribute dictionary and do not use regular expressions. The same customisable approach is adopted by DDL2 for units.
- DDL2 dictionaries specify data name parent–child relationships (see below) in the topmost category definition, and optionally at the site of the child data name definition.
- DDL2 dictionaries restrict data names to single values per data block by relating them to an identifier that is defined to be single-valued, whereas DDLm uses a specific attribute for this purpose.
- DDL2 allows data transformation methods to be specified in any programming language, whereas DDLm requires environment-agnostic dREL code.
- DDLm has a well-specified protocol for combining dictionaries and using templates to simplify dictionary construction.
- DDLm assigns meaning to category parent–child relationships.

Understanding and composing DDLm dictionaries

Parent and child data names

Our earlier discussion supposed that key data names might appear in multiple categories. When data from these categories is presented in CIF loops, identical data names would appear in each loop. CIF, however, strictly forbids data names to appear more than once in a data block; therefore we must define distinct data names for each category in which a key data name appears. Note that our suggested *<category>.<name>* naming strategy automatically takes care of this for us.

We can arrange these data names into a parent–child hierarchy, where child data names are expected to take a subset of the values of the parent data names. In most situations, there will be only a single child level in this hierarchy.

Semantic structure

As discussed earlier, the data names in an ontology can be distributed into categories, within which every data name shares the same key data names or is itself a key data name for the non-key data names. CIF dictionaries describe each of these categories in their own save frames. Every data name definition states the category to which it belongs, and likewise every category has a single parent category. The category definition at the top of this hierarchy is called a 'Head' category.

The meaning of the category parent–child relationship is discussed later.

A category corresponds to the set of data names appearing in a loop within a CIF file.

Data blocks and CIF

Data names that are restricted to single values within a CIF data block are assigned to `Set` categories. All other data name definitions must belong to `Loop` categories.

The type of category is indicated by setting _definition.class to Set or Loop

The values of Set category key data names do not need to be provided in a data file, as these values are both arbitrary and unique by virtue of being the only value in the block. The same consideration applies to any child data names. Key data names from Set categories, and their children, can therefore be completely dropped from the CIF dictionary.

A mechanism exists in CIF to add back these key data names in supplementary dictionaries, should they be required.

Writing a DDLm dictionary from scratch

You may find it convenient to use a shorthand notation while developing the dictionary, that can be automatically transformed later to CIF. For example, writing:

```
#N my_data_name
#C my_category
#D this is the definition
#L value1: stuff, value2: more stuff, value3: no more
```

instead of

```
_definition.id '_my_category.my_data_name'
```

```
_name.object_id  my_data_name
_name.category_id my_category
_description.text 'This is the definition'
loop_
_enumeration_set.state
_enumeration_set.detail
    value1      stuff
    value2      'more stuff'
    value3      'no more'
```

Step 1. Name, and write a definition for, the Head category

The name of the Head category is purely for internal use. The `_definition.id` and `_name.object_id` attributes hold the name of the category. `_name.category_id` should be set to the value of the `_dictionary.title` attribute. Here is a template:

```
save_CIF_CORE

_definition.id CIF_CORE
_definition.scope Category
_definition.class Head
_definition.update 2014-06-18
_description.text
;
The CIF_CORE category contains the definitions of data items that
are common to all domains of crystallographic studies.
;
_name.category_id CIF_DIC   #_dictionary.title from enclosing block
_name.object_id CIF_CORE

save_
```

Step 2. Write the category definitions

You will need to assign the following attributes:

- `_definition.id`: the name of the category
- `_definition.scope`: `Category`
- `_definition.class`: `Set` or `Loop`. See above for explanations of these terms.
- `_definition.update`: the date the definition was written *yyyy-mm-dd*
- `_description.text`: a human-readable description of this category
- `_name.category_id`: the category this belongs to (usually the Head category)
- `_name.object_id`: the name of the category (again)
- `_category_key.name`: (possibly looped) the list of key data names for this category. Only necessary for `Loop` categories

Here is a simple Loop category definition:

```
save_DIFFRN_ATTENUATOR
_definition.id     DIFFRN_ATTENUATOR
_definition.scope  Category
_definition.class  Loop
_definition.update 2013-09-08
_description.text
;
The CATEGORY of data items which specify the attenuators used in the
diffraction source.
;
_name.category_id  DIFFRN
_name.object_id    DIFFRN_ATTENUATOR
_category.key_id   '_diffrn_attenuator.code'
loop_
   _category_key.name
       '_diffrn_attenuator.code'
save_
```

Step 3. Write the data name definitions

You will need to assign the following attributes:

- `_definition.id`: the data name, usually `<category>.<object>`
- `_definition.update`: the date the definition was written *yyyy-mm-dd*
- `_description.text`: a human-readable description of the data name
- `_name.category_id`: the category this belongs to
- `_name.object_id`: the name within the category
- `_type.container`: `List/Array/Matrix/Table` for compound data values, `Single` otherwise
- `_type.contents`: the nature of the individual values taken by this data name
- `_units.code`: the units for the data values. Leave out or put `none` if none
- `_type.source`: where values come from (optional but useful)
- `_type.purpose`: a classification of the values into some general classes. Useful information for dictionary tools

Some common situations covered by further attributes are:

- If your data name can take values from a restricted set, use `_enumeration_set.{state/detail}` to list and describe each option.
- If your data name corresponds to a key data name from another category that appears in this category, set `_name.linked_item_id` to that other data name and set `_type.purpose` to `Link`.

a link within a single category is also possible

- If your data name directly replaces another data name, assign the older name to `_alias.definition_id`

- If your data name is the standard uncertainty of another data name, set `_type.purpose` to `SU` and `_name.linked_item_id` to that other data name.
- If a specific value can be safely assumed when the data name is missing from a data file, specify this with `_enumeration.default`.
- If your data value is a list, array or matrix give the dimensions using `_type.dimension`. An asterisk (∗) can be used for arbitrary values.

```
save__diffrn_source.device
_definition.id 'diffrn_source.device'
 loop_
_alias.definition_id
    '_diffrn_radiation_source'
    '_diffrn_source.source'
_definition.update 2013-08-09
_description.text
;
Enumerated code for the device providing the source of radiation.
;
_name.category_id diffrn_source
_name.object_id device
_type.purpose State
_type.source Assigned
_type.container Single
_type.contents Code
 loop_
_enumeration_set.state
_enumeration_set.detail
tube                'sealed X-ray tube'
nuclear             'nuclear reactor'
spallation          'spallation source'
elect-micro         'electron microscope'
rot_anode           'rotating-anode X-ray tube'
synch                synchrotron
_enumeration.default tube
save_
```

Step 4. Create the enclosing data block

Assign the various `dictionary.*` attributes at the top of the data block. Arrange the save frames (definitions) in alphabetical order, unless you are sure some other order is more natural for a human reader searching for a particular definition.

Adding to a pre-existing DDLm dictionary

Examine the currently-existing categories in the dictionary to determine if any of them have matching keys. If not, follow step 2 to add new categories. Add your new data names as in Step 3 above.

Further DDLm dictionary enhancements

Imports

DDLm dictionaries allow groups of attributes to be imported from another dictionary using the `_import.get` attribute. The value for `_import.get` is a list of tables. Each table has a few keys that are set to guide the import process:

file: the file to import from

save: the name of the save frame to import from the file

mode: `Contents`: only the contents of the save frame are inserted; `Full`: the entire save frame and semantic children are included.

Importation is useful when many of the attributes for a set of data names are identical. In this case, the attributes are put into a save frame in a separate CIF 'template' dictionary, and the relevant save frame is simply imported into each of the definitions.

```
save__diffrn_standard_refln.index_h

_definition.id          '_diffrn_standard_refln.index_h'
_import.get             [{'save':Miller_index 'file':templ_attr.cif}]
_name.category_id       diffrn_standard_refln
_name.object_id         index_h

save_
```

The file `templ_attr.cif` contains save frame:

```
 save_Miller_index
_definition.update            2013-04-16
_description.text
;
The index of a reciprocal space vector.
;
_type.purpose                 Number
_type.source                  Recorded
_type.container               Single
_type.contents                Integer
_enumeration.range            -1000:1000
_units.code                   none
save_
```

Another use for this feature is to put long lists of enumerated items into separate files to avoid cluttering the main dictionary. Examples include element names, and units.

A separate use for imports is to specify dictionaries that your dictionary builds on. If your dictionary adds (or changes) data names from any categories in another dictionary, it is sufficient for the Head category of your dictionary to include an import of the Head category of the dictionary it builds upon. In this case, the import `mode` should be set to `Full`. Semantically, any definitions in the importing dictionary with the same `_definition.id` will replace definitions in the imported dictionary unless you have set the import key `if_dupl` to `ignore`.

```
save_PD_GROUP

_definition.id                PD_GROUP
_definition.scope             Category
_definition.class             Head
_definition.update            2014-06-20
_description.text
```

```
;
  Groups all of the categories of definitions in the powder
  diffraction study of materials.
;
_name.category_id            CIF_POW
_name.object_id              PD_GROUP
_import.get
         [{"file":"cif_core.dic" "save":"CIF_CORE" "mode":"Full"}]

save_
```

Child categories

In DDLm, Loop categories may be child categories not only of the Head category, but also of other Loop categories. A child category is obtained by splitting a single category into two parts, so that each part retains the same key data names, but the remaining data names differ. For most purposes, the two categories can be considered to be a single category that have been split for convenience, for example, when some subset of data names are likely to have undefined values for some of the key data name values. It is permissible for data files to present all of the data names from both categories in a single loop.

The combined category is obtained by a *left outer join* of the parent with the child.

Example:

The DDLm core CIF dictionary allows anisotropic displacement parameters to be presented in a separate loop, described in the `atom_site_aniso` category, which is a child category of `atom_site`. This allows those experiments for which many atoms do not have well-determined ADPs to save space in the *atom_site* listing.

Example:

The magnetic structures dictionary makes *atom_site_moment* a child category of *atom_site*, in recognition of the fact that for many structures only a few atomic sites will have associated moments. In such cases, the moments can be listed in a separate loop.

The child category after the split is the category that might take only a subset of the key data values. In the examples above, not all atom sites have to be listed in the *atom_site_moment* or *atom_site_aniso* loops, so these are the child categories.

dREL

Mathematical relationships between data names can be expressed using the `_method.expression` attribute. The value of this attribute is computer-parseable program code written in dREL. A dREL expression in a data name definition describes how a value for the data name is calculated from the values of other data names. The DDLm cif core dictionary contains many examples of dREL expressions. dREL is described in Spadaccini *et al.* (2012), *J. Chem. Inf. Model.* **52**, 1917–1925.

The advantage of dREL over concrete programming languages is that no CIF programming library or language environment is assumed, that is, dREL is environment-agnostic. As a result, in order to execute dREL code it must first be transformed to a concrete language, inserting the particular function calls required by the language and CIF API.

DDLm Dictionary extensions

_audit.schema

When writing the dictionary, you had to make a decision regarding which categories would allow only single-valued data names. This set of single-valued names would ideally satisfy the majority of data transfer scenarios in your discipline. For those users who need some (or all) of those data names to take multiple values (*e.g.* multiple samples per run), it is possible to define an extension dictionary. This dictionary `_imports` the original dictionary, and then redefines the relevant category to be a `Loop` category, creates and assigns the key data names for the category, and adds the child key data names to all affected categories.

Note that software written assuming the behaviour in the original dictionary is susceptible to behaving incorrectly if it does not know which dictionaries a given data file is written to conform to. At the same time, programmers are not necessarily prepared to read and parse entire dictionaries at runtime in order to check the interpretation of potentially only a few data names. The `_audit.schema` data name in core CIF has therefore been introduced as a shorthand flag that a data file is using a non-default set of single-valued data names. It is sufficient for software to check this and the following data name to ensure that the file contents are correctly understood.

_audit.formalism

Where a dictionary redefines one or more data names from the base dictionary, for example, by enhancing a structural model (magnetism, powder, modulated structures), the change in definition is flagged via the `_audit.formalism` tag.

Further reading

Constructing scientific ontologies

- Hester, J. R. (2016). ‘A robust, format-agnostic scientific data transfer framework’. *Data Science Journal*, **15**, 12, pp. 1–17, DOI:10.5334/dsj-2016-012
- Spivak, D. I. and Kent, R. E. (2012). ‘Ologs: a categorical framework for knowledge representation’. *PLoS One*, **7**, e24274

Relational model

- The imgCIF dictionary is an excellent, small-scale example of a nicely normalised ontology. Find it at: `ftp://ftp.iucr.org/cifdics/cif_img_1.3.2.dic.pdf` or *International Tables for Crystallography, Volume G* (ITVG) , Sections 3.7 and 4.6, and see the draft updated commentary for the revised edition of ITVG in Appendix 3 of this booklet.

- Wikipedia articles ‘Relational Model’ and ‘Database normalisation’ are good starting points.

Writing CIF dictionaries

- Spadaccini, N. and Hall, S. R. (2012). ‘DDLm: A new dictionary definition language’. *J. Chem. Inf. Model.* **52**, 1907–1916
- Spadaccini, N. Castelden, I. R., du Boulay, D. and Hall, S. R. (2012). ‘dREL: A relational expression language for dictionary methods’. *J. Chem. Inf. Model.* **52**, 1917–1925
- Westbrook, J. D., Berman, H. D. and Hall, S. R. (2005).‘Specification of a relational dictionary definition language (DDL2)’. *International Tables for Crystallography, Volume G*, pp. 61–72. Dordrecht: Springer
- The DDLm attribute dictionary (`http://github.com/COMCIFS/cif_core/blob/cif2-conversion/ddl.dic`)

Dictionaries for Complex Raw Data such as Images

Herbert J. Bernstein

The earlier parts of this workshop described ontologies in which the primary focus was on considerations of the data transfer framework and we have the luxury of requiring not just the ontology but the representation of each set of data to be human readable. That is not always possible in practice. Some data, such as diffraction images, can be too voluminous and complex to be efficiently representable as text. That does not stop us from creating a human-readable ontology, nor from using CIF for that purpose, but it forces us to find ways to relate CIF to binary data representations, such as NeXus/HDF5, and to non-relational models, such as XML and JSON. In this section we will explore the considerations involved in creating the CIF imgCIF dictionary and its relationship to the Crystallographic Binary Format (CBF) and other binary formats, as a model for creating ontologies to deal with complex raw data such as images.

What is Similar and What is Different

In many ways the process of designing an ontology for complex raw data such as images is similar to the process of designing any ontology. We have data values that we wish to label with data value names in an ontology and that we wish to present in some appropriate data formats. All the considerations that apply in designing a CIF dictionary for, say, small-molecule structures or for macromolecular structures apply. In particular, if we are to maintain relational databases of sets of data, it is very helpful to follow a relational model, organizing data into categories to that it will be relatively easy to search for particular sets of data, say, all the images collected at beamline ID-XX by user John Doe that related to NAG bound to lysozyme in the month of May 2017. From this point of view, all we really need is a good way to search the “metadata”, not the actual images.

What is different is that, while we may only need to search through a bit more than 100,000 published macromolecular structures or almost 1,000,000 published small molecule structures, the number of diffraction images behind the published structures may easily be 2 to 5 orders of magnitude larger than the number of structures, and until we are certain which images will be needed in the future, we also may need to search through the images themselves, using both the metadata and the data itself to select and organize images for processing so we can see which images are really needed. This makes the choice of a particular format for the data something that has to be carefully considered and coupled very efficiently to the metadata format. As crystallography moves towards handling larger numbers of small crystals at increasingly brilliant light sources, we need to be able to quickly cluster images into sets that are appropriate to merge which is a process that demands access to the images, not just the data about the images. In addition, science is moving towards more multimodal experiments in which raw data and processed data from crystallography, cryo-EM and NMR all must be considered in relationship to one another.

That is why, as noted earlier in the workshop discussion, the imgCIF ontology has been carefully normalized – to facilitate database use, and to facilitate interactions in processing metadata from multiple sources, and to make it easy to find the necessary metadata to facilitate conversion of imgCIF/CBF data to and from other formats, such as NeXus/HDF5.

In addition, raw data differs from processed data, precisely in not yet having been processed. Therefore the results of that processing may cause us to re-evaluate what has been observed. Initial estimates of beam centres may need to be adjusted. The positions of detector modules may need to be refined, *etc.* Primary data may need to be replaced with derived data in multiple passes.

Further Reading

Bernstein, H. J. (2017), 'Classification and use of image data'. Draft Chapter 2.10 for future edition of *International Tables for Crystallography*, Volume G, presented as Appendix 3 of this booklet.

APPENDIX 1: DDLm dictionary. This is a synoptic listing of the DDLm dictionary, formatted in the style of *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data*, a second edition of which is in preparation.

DDLm dictionary

BY SYDNEY R. HALL, NICK SPADACCINI, JAMES R. HESTER, JOHN C. BOLLINGER AND ANTANAS VAITKUS

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*)
Dictionary URI in which the aliased definition belongs.

CATEGORY

The attributes used to specify the properties of a 'category' of data items.

`_category.key_id` (*Tag*)
Tag of a single data item in a Loop category which is the generic key to access other items in the category. The value of this item must be unique in order to provide unambiguous access to a packet (row) in the table of values. This may be assumed to be a function of the datanames listed in **`_category_key.name`**.

CATEGORY_KEY

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

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

Affiliations: SYDNEY R. HALL, Biomolecular, Biomedical & Chemical Sciences, Faculty Life & Physical Sciences, M310 University of Western Australia, Nedlands 6009, Australia; NICK SPADACCINI, The University of Western Australia, Crawley, 6009, Australia; JAMES R. HESTER, Australian Nuclear Science and Technology Organisation, New Illawarra Road, Lucas Heights, NSW 2234, Australia JOHN C. BOLLINGER, Department of Structural Biology, St Jude Children's Research Hospital, Memphis, Tennessee 38105, USA; ANTANAS VAITKUS, Institute of Biotechnology, Vilnius University, Vilnius, Lithuania.

`_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 datanames listed in this loop must be unique, so that unambiguous access to a packet (row) in the table of values is possible. The dataname associated with **`_category.key_id`** is only included in this loop if no other set of datanames can form a compound key.

DEFINITION

The attributes for classifying dictionary definitions.

`_definition.class` (*Code*)
The nature and the function of a definition or definitions.

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 must 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.
`Ref-loop`	A category containing one item that identifies the a category of items that is repeated in a sequence of save frames. The item, which is specifies as a as a Ref-table value (see type.container), is looped. This construction is for loop categories that contain child categories. If in the instance file, the child items have only one set of values, the Ref-loop item need not be used and child items need not be placed in a save frame.

`_definition.id` (*Code*)
Identifier name of the Item or Category definition contained within a save frame.

`_definition.replaced_by` (*Tag*)
A dataname that should be used instead of the defined dataname. The defined dataname is deprecated and should not be used.

`_definition.scope` (*Code*)
The extent to which a definition affects other definitions.

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.xref_code` (*Code*)
Code identifying the equivalent definition in the dictionary referenced by the DICTIONARY_XREF attributes.

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.

DESCRIPTION_EXAMPLE

The attributes of descriptive (non-machine parsable) examples of values of the defined items.
Category key(s): **`_description_example.case`**

`_description_example.case` (*Implied*)
An example case of the defined item.

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

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.
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.formalism` (*Text*)
The definitions contained in this dictionary are associated with the value of this attribute. Datanames 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 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_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.
Category key(s): **`_dictionary_valid.application`**

`_dictionary_valid.application` (*Code*[2])
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. This list signals the validity of applying the attributes given in **`_dictionary_valid.attributes`**.

`_dictionary_valid.attributes` (*Code*[])
A list of the attribute names and categories that are assessed for application in the item, category and dictionary definitions.

`_dictionary_valid.option` (*Code*)
Option codes for applicability of attributes in definitions.
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 data frame
`Category`	restriction applies to a category definition save frame
`Item`	restriction applies to an item definition save frame

DICTIONARY_XREF

Data items which are used to cross reference other dictionaries that have defined the same data items. Data items in this category are *not* used as attributes of individual data items.

Category key(s): `_dictionary_xref.code`

`_dictionary_xref.code` (*Code*)
A code identifying the cross-referenced dictionary.

`_dictionary_xref.date` (*Date*)
Date of the cross-referenced dictionary.

`_dictionary_xref.format` (*Text*)
Format of the cross referenced dictionary.

`_dictionary_xref.name` (*Text*)
The name and description of the cross-referenced dictionary.

`_dictionary_xref.uri` (*Uri*)
The source URI of the cross referenced dictionary data.

ENUMERATION

The attributes for restricting the values of defined data items.

`_enumeration.def_index_id` (*Tag*)
Specifies the data name 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.default` (*Implied*)
The default value for the defined item if it is not specified explicitly.

`_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.

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 values 'from:to' allowed for the defined item.

ENUMERATION_DEFAULT

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*)
Index key in the list of default values referenced by the value of `_enumeration.def_index_id`.

`_enumeration_default.value` (*Implied*)
Default enumeration value in the list referenced by the value of `_enumeration.def_index_id`. The reference index key is given by the `_enumeration_default.index` value.

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_set.xref_code` (*Code*)
Identity of the equivalent item in the dictionary referenced by the DICTIONARY_XREF attributes.

`_enumeration_set.xref_dictionary` (*Code*)
Code identifying the dictionary in the DICTIONARY_XREF list.

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 RFC3986 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 framecode 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.
The data value must be one of the following:

`Ignore`	ignore imported definitions if id conflict
`Replace`	replace existing with imported definitions
`Exit`	issue 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.
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 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. 'Contents' imports only the attributes found in the imported definition.
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 of source dictionary
`version`	version of source dictionary
`save`	save framecode of source definition
`mode`	mode for including save frames
`dupl`	option for duplicate entries
`miss`	option for missing duplicate entries

LOOP

Attributes for looped lists.

`_loop.level` (*Index*)
Specifies the level of the loop structure in which a defined item must reside if it is used in a looped list.

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.
The data value must be one of the following:

`Evaluation`	method evaluates an item from related item values
`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*)
Dataname 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 Measurement 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 container type of the defined data item value.
The data value must be one of the following:

`Single`	single value
`Multiple`	values as List or by boolean `,|&!*` or range `:` ops
`List`	ordered set of values. Elements need not be of same contents type.
`Array`	ordered set of numerical values. 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 across matrix objects.
`Table`	An unordered set of id:value elements

`_type.contents` (*Code*)

Syntax of the value elements within the container type. This may be a single enumerated code, or, in the case of a list, a comma-delimited sequence of codes, or, if there are alternate types, a boolean-linked (or range) sequence of codes. The typing of elements is determined by the replication of the minimum set of states declared. Where the definition is of a 'Table' container this attribute describes the construction of the value elements within those (Table) values. The CIF2 characterset 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].

The data value must be one of the following:

`Text`	case-sensitive sequence of CIF2 characters
`Code`	case-insensitive sequence of CIF2 characters containing no ASCII whitespace.
`Name`	case-insensitive sequence of ASCII alpha-numeric characters or underscore
`Tag`	case-insensitive CIF2 character sequence with leading underscore and no ASCII whitespace
`Uri`	A Uniform Resource Identifier per RFC 3986
`Date`	ISO standard date format $\langle yyyy \rangle$-$\langle mm \rangle$-$\langle dd \rangle$
`Version`	version digit string of the form $\langle major \rangle.\langle version \rangle.\langle update \rangle$
`Dimension`	integer limits of an Array/Matrix/List in square brackets
`Range`	inclusive range of numerical values min:max
`Count`	unsigned integer number
`Index`	unsigned non-zero integer number
`Integer`	positive or negative integer number
`Real`	floating-point real number
`Imag`	floating-point imaginary number
`Complex`	complex number $\langle R \rangle + j\langle I \rangle$
`Binary`	binary number $\backslash b\langle N \rangle$
`Hexadecimal`	hexadecimal number $\backslash x\langle N \rangle$
`Octal`	octal number $\backslash o\langle N \rangle$
`Symop`	a string composed of an integer optionally followed by an underscore or space and three or more digits
`Implied`	implied by the context of the attribute
`ByReference`	The contents have the same form as those of the attribute referenced by **`_type.contents_referenced_id`**.

Examples: '`Integer`' (content is a single or multiple integer(s)), '`Real,Integer`' (List elements of a real number and an integer), '`List(Real,Code)`' (List of Lists of a real number and a code), '`Text|Real`' (content is either text OR a real number)

`_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 or matrix of elements as a text string within bounding square brackets.

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 characterset definition.

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 $\langle yyyy \rangle$-$\langle mm \rangle$-$\langle dd \rangle$
`Uri`	a Uniform Resource Identifier string, per RFC 3986
`Version`	version digit string of the form $\langle major \rangle.\langle version \rangle.\langle update \rangle$
`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.

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), save frame or the URI of an external file.
`Extend`	**Used to extend the DDLm Reference Dictionary** Used in a definition, residing in the 'extensions' save frame of a domain dictionary, to specify a new enumeration state using an Evaluation method.
`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 may be used as a reference 'key' to identify a specific packet of items within the category.

`Link`	Used to type an item with a value that is drawn from the set of unique values for another item. The definition of this item must contain the attribute '**`_name.linked_item_id`** specifying the data name of the other item. The defined item is usually (but not always) a foreign key linking packets in this category to packets in another category.
`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. This value must be accompanied by its standard uncertainty (SU) value, expressed either as: 1) appended integers, in parentheses (), at the precision of the trailing digits, or 2) a separately defined item with the same name as the measurand item but with an additional suffix '**`_su`**'.
`SU`	Used to type an item with a numerical value that is the standard uncertainty of another dataname. The definition of an SU item must include the attribute '**`_name.linked_item_id`**' 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.

The data value must be one of the following:

`Recorded`	A value (numerical or otherwise) recorded by observation or measurement during the experimental collection of data. This item is **primitive**.
`Assigned`	A value (numerical or otherwise) 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 may be deemed **primitive**) or a particular choice in the way the data was analysed (and therefore may be considered **not primitive**).
`Related`	A value or tag used in the construction of looped lists of data. Typically identifying an item whose unique value is the reference key for a loop category and/or an item which as values in common with those of another loop category and is considered a Link between these lists.
`Derived`	A quantity derived from other data items within the domain instance. This item is **not primitive**.

UNITS

The attributes for specifying units of measure.

`_units.code` (*Code*)
A code which identifies the units of measurement.

APPENDIX 2: DDL2 dictionary. This is a synoptic listing of the DDL2 dictionary, originally published as Chapter 4.10 of *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data.* Dordrecht: Springer. Reproduced by permission. ©2005 International Union of Crystallography

4.10. DDL2 dictionary

BY J. D. WESTBROOK AND S. R. HALL

This is version 2.1.3 of the dictionary definition language (DDL2) that provides a machine-readable description of the attributes of data items in the mmCIF and related dictionaries (Chapters 4.5 to 4.7). This version of DDL is described in Chapter 2.6. The category groups in the DDL2 dictionary are: ddl_group (component categories of the macromolecular DDL); datablock_group (categories that describe the characteristics of data blocks); category_group (categories that describe the characteristics of categories); sub_category_group (categories that describe the characteristics of subcategories); item_group (categories that describe the characteristics of data items); dictionary_group (categories that describe the dictionary); and compliance_group (categories that are retained specifically for compliance with older versions of the DDL).

CATEGORY

Attributes defining the functionality for the entire category.
Category group(s): `ddl_group`
`category_group`
Category key(s): `_category.id`

*`_category.description` (*text*)
Text description of a category.
`[category]`

*`_category.id` (*idname*)
The identity of the data category. Data items may only be looped with items of the same category.
The following item(s) have an equivalent role in their respective categories:
`_category_examples.id`,
`_category_group.category_id`,
`_category_key.id`,
`_category_methods.category_id`,
`_item.category_id`. `[category]`

`_category.implicit_key`
An identifier that may be used to distinguish the contents of like categories between data blocks.

*`_category.mandatory_code` (*code*)
Whether the category must be specified in a dictionary.
`[category]`

CATEGORY_EXAMPLES

Example applications and descriptions of data items in this category.
Category key(s): `_category_examples.id`
`_category_examples.case`

*`_category_examples.case` (*text*)
A case of examples involving items in this category.
`[category_examples]`

`_category_examples.detail` (*text*)
A description of an example given in `_category_examples.case`.
`[category_examples]`

CATEGORY_GROUP

Provides a list of category groups to which the base category belongs.
Category group(s): `ddl_group`
`category_group`
Category key(s): `_category_group.id`
`_category_group.category_id`

CATEGORY_GROUP_LIST

This category provides the definition of each category group. A category group is a collection of related categories.
Category group(s): `ddl_group`
`category_group`
Category key(s): `_category_group_list.id`

*`_category_group_list.description` (*text*)
Text description of a category group.
`[category_group_list]`

*`_category_group_list.id` (*idname*)
The name of a category group.
The following item(s) have an equivalent role in their respective categories:
`_category_group.id`,
`_category_group_list.parent_id`. `[category_group_list]`

`_category_group_list.parent_id`
The name of the optional parent category group.

CATEGORY_KEY

This category holds a list of the item names that uniquely identify the elements of the category.
Category group(s): `ddl_group`
`category_group`
Category key(s): `_category_key.name`
`_category_key.id`

`_category_key.name`
The name of a data item that serves as a key identifier for the category (*e.g.* a component of the primary key).

CATEGORY_METHODS

Attributes specifying the association between categories and methods.
Category group(s): `ddl_group`
`category_group`
Category key(s): `_category_methods.method_id`
`_category_methods.category_id`

DATABLOCK

Attributes defining the characteristics of a data block.
Category group(s): `ddl_group`
`datablock_group`
Category key(s): `_datablock.id`

*`_datablock.description` (*text*)
Text description of the data block.
`[datablock]`

Affiliations: JOHN D. WESTBROOK, Protein Data Bank, Research Collaboratory for Structural Bioinformatics, Rutgers, The State University of New Jersey, Department of Chemistry and Chemical Biology, 610 Taylor Road, Piscataway, NJ, USA; SYDNEY R. HALL, School of Biomedical and Chemical Sciences, University of Western Australia, Crawley, Perth, WA 6009, Australia.

(*)`_datablock.id` (*code*)
The identity of the data block.
The following item(s) have an equivalent role in their respective categories:
`_datablock_methods.datablock_id`,
`_dictionary.datablock_id`,
`_category.implicit_key`. `[datablock]`

DATABLOCK_METHODS

Attributes specifying the association between data blocks and methods.
Category group(s): `ddl_group`
`datablock_group`
Category key(s): `_datablock_methods.method_id`
`_datablock_methods.datablock_id`

DICTIONARY

Attributes for specifying the dictionary title, version and datablock identifier.
Mandatory category.
Category group(s): `ddl_group`
`datablock_group`
`dictionary_group`
Category key(s): `_dictionary.datablock_id`

`_dictionary.datablock_id`
The identifier for the data block containing the dictionary.

*`_dictionary.title` (*char*)
Title identifier of the dictionary.
`[dictionary]`

`_dictionary.version`
A unique version identifier for the dictionary.

DICTIONARY_HISTORY

Attributes for specifying the revision history of the dictionary.
Category group(s): `ddl_group`
`dictionary_group`
Category key(s): `_dictionary_history.version`

*`_dictionary_history.revision` (*text*)
Text description of the dictionary revision.
`[dictionary_history]`

*`_dictionary_history.update` (*yyyy-mm-dd*)
The date that the last dictionary revision took place.
`[dictionary_history]`

*`_dictionary_history.version` (*char*)
A unique version identifier for the dictionary revision.
The following item(s) have an equivalent role in their respective categories:
`_dictionary.version`. `[dictionary_history]`

ITEM

Attributes which describe the characteristics of a data item.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item.name`

*`_item.mandatory_code` (*code*)
Signals whether the defined item is mandatory for the proper description of its category.
The data value must be one of the following:

`yes`	required item in this category
`no`	optional item in this category
`implicit`	required item but may be determined from context

`[item]`

(*)`_item.name` (*name*)
Data name of the defined item.
The following item(s) have an equivalent role in their respective categories:
`_category_key.name`,
`_item_aliases.name`,
`_item_default.name`,
`_item_dependent.name`,
`_item_dependent.dependent_name`,
`_item_description.name`,
`_item_enumeration.name`,
`_item_examples.name`,
`_item_linked.child_name`,
`_item_linked.parent_name`,
`_item_methods.name`,
`_item_range.name`,
`_item_related.name`,
`_item_related.related_name`,
`_item_type.name`,
`_item_type_conditions.name`,
`_item_structure.name`,
`_item_sub_category.name`,
`_item_units.name`. `[item]`

ITEM_ALIASES

This category holds a list of possible alias names or synonyms for each data item. Each alias name is identified by the name and version of the dictionary to which it belongs.
Category key(s): `_item_aliases.alias_name`
`_item_aliases.dictionary`
`_item_aliases.version`

*`_item_aliases.alias_name` (*aliasname*)
Alias name for the data item.
`[item_aliases]`

*`_item_aliases.dictionary` (*char*)
The dictionary in which the alias name is defined.
`[item_aliases]`

*`_item_aliases.version` (*char*)
The version of the dictionary in which the alias name is defined.
`[item_aliases]`

ITEM_DEFAULT

Attributes specifying the default value for a data item.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_default.name`

`_item_default.value` (*any*)
The default value for the defined item if it is not specified explicitly. If a data value is not declared, the default is assumed to be the most likely or natural value.
`[item_default]`

ITEM_DEPENDENT

Attributes which identify other data items that must be specified for the defined data item to be valid.
Category key(s): `_item_dependent.name`
`_item_dependent.dependent_name`

`_item_dependent.dependent_name`
Data name of a dependent item.

ITEM_DESCRIPTION

This category holds the descriptions of each data item.
Mandatory category.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_description.name`
`_item_description.description`

`*_item_description.description` (*text*)
Text description of the defined data item.
`[item_description]`

ITEM_ENUMERATION

Attributes which specify the permitted enumeration of the items.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_enumeration.name`
`_item_enumeration.value`

`_item_enumeration.detail` (*text*)
A description of a permissible value for the defined item.
`[item_enumeration]`

`*_item_enumeration.value` (*any*)
A permissible value, character or number for the defined item.
`[item_enumeration]`

ITEM_EXAMPLES

Attributes for describing examples of applications of the data item.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_examples.name`
`_item_examples.case`

`_item_examples.case` (*text*)
An example application of the defined data item.
`[item_examples]`

`_item_examples.detail` (*text*)
A description of an example specified in `_item_examples.case`.
`[item_examples]`

ITEM_LINKED

Attributes which describe how equivalent data items are linked within categories and across different categories.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_linked.child_name`

`_item_linked.child_name`
Name of the child data item.

`_item_linked.parent_name`
Name of the parent data item.

ITEM_METHODS

Attributes specifying the association between data items and methods.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_methods.method_id`
`_item_methods.name`

ITEM_RANGE

The range of permissible values of a data item. When multiple ranges are specified, they are interpreted sequentially using a logical OR. To specify that an item value may be equal to a boundary value, specify an item range where the maximum and mimimum values equal the boundary value.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_range.name`
`_item_range.minimum`
`_item_range.maximum`

`_item_range.maximum` (*any*)
Maximum permissible value of a data item or the upper bound of a permissible range (maximum value > data value).
`[item_range]`

`_item_range.minimum` (*any*)
Minimum permissible value of a data item or the lower bound of a permissible range (minimum value < data value).
`[item_range]`

ITEM_RELATED

Attributes which specify recognized relationships between data items.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_related.name`
`_item_related.related_name`
`_item_related.function_code`

`*_item_related.function_code` (*code*)
The code for the type of relationship between the item identified by `_item_related.name` and the defined item.

'alternate' indicates that the item identified in `_item_related. related_name` is an alternative expression in terms of its application and attributes to the item in this definition. 'alternate_exclusive' indicates that the item identified in `_item_ related.related_name` is an alternative expression in terms of its application and attributes to the item in this definition. Only one of the alternative forms may be specified.

'convention' indicates that the item identified in `_item_ related.related_name` differs from the defined item only in terms of a convention in its expression.

'conversion_constant' indicates that the item identified in `_item_related.related_name` differs from the defined item only by a known constant. 'conversion_arbitrary' indicates that the item identified in `_item_related.related_name` differs from the defined item only by an arbitrary constant.

'replaces' indicates that the defined item replaces the item identified in `_item_related.related_name`. 'replacedby' indicates that the defined item is replaced by the item identified in `_item_related.related_name`.

'associated_value' indicates that the item identified in `_item_ related.related_name` is meaningful when associated with the defined item. 'associated_esd' indicates that the item identified in `_item_related.related_name` is the estimated standard deviation of the defined item. 'associated_error' indicates that the item identified in `_item_related.related_name` is the estimated error of the defined item.

The data value must be one of the following:

`alternate`	alternate form of the item
`alternate_exclusive`	mutually exclusive alternate form of the item
`convention`	depends on defined convention
`conversion_constant`	related by a known conversion factor
`conversion_arbitrary`	related by an arbitrary conversion factor

`replaces`	a replacement definition
`replacedby`	an obsolete definition
`associated_value`	a meaningful value when related to the item
`associated_esd`	an estimated standard deviation of the item
`associated_error`	an estimated error of the item

[item_related]

ITEM_STRUCTURE

This category holds the association between data items and named vector/matrix declarations.
Category group(s): **ddl_group**
item_group
Category key(s): **_item_structure.name**

***_item_structure.organization** (*code*)
Identifies whether the structure is defined in column- or row-major order. Only the unique elements of symmetric matrices are specified.
The data value must be one of the following:

`columnwise`	column-major order
`rowwise`	row-major order

ITEM_STRUCTURE_LIST

This category holds a description for each structure type.
Category group(s): **ddl_group**
item_group
Category key(s): **_item_structure_list.code**
_item_structure_list.index

***_item_structure_list.code** (*code*)
The name of the matrix/vector structure declaration.
The following item(s) have an equivalent role in their respective categories:
_item_structure.code. **[item_structure_list]**

***_item_structure_list.dimension** (*int*)
Identifies the length of this row or column of the structure.
The permitted range is $[1, \infty)$. **[item_structure_list]**

***_item_structure_list.index** (*int*)
Identifies the one-based index of a row or column of the structure.
The permitted range is $[1, \infty)$. **[item_structure_list]**

ITEM_SUB_CATEGORY

This category assigns data items to subcategories.
Category group(s): **sub_category_group**
item_group
Category key(s): **_item_sub_category.id**
_item_sub_category.name

ITEM_TYPE

Attributes for specifying the data-type code for each data item.
Category group(s): **ddl_group**
item_group
Category key(s): **_item_type.name**

ITEM_TYPE_CONDITIONS

Attributes for specifying additional conditions associated with the data type of the item.
Category group(s): **ddl_group**
item_group
compliance_group
Category key(s): **_item_type_conditions.name**

***_item_type_conditions.code** (*code*)
Codes defining conditions on the **_item_type.code** specification. 'esd' permits a number string to contain an appended standard deviation number enclosed within parentheses, *e.g.* 4.37(5). 'seq' permits data to be declared as a sequence of values separated by a comma <,> or a colon <:>. The sequence $v_1, v_2, v_3 \ldots$ signals that v_1, v_2, v_3 *etc.* are alternative values for the data item. The sequence $v_1{:}v_2$ signals that v_1 and v_2 are the boundary values of a continuous range of values. This mechanism was used to specify permitted ranges of an item in previous DDL versions. Combinations of alternate and range sequences are permitted.
The data value must be one of the following:

`none`	no extra conditions apply to this data item
`esd`	numbers may have esd values appended within parentheses
`seq`	data may be declared as a comma- or colon-separated sequence

[item_type_conditions]

ITEM_TYPE_LIST

Attributes which define each type code.
Category group(s): **ddl_group**
item_group
Category key(s): **_item_type_list.code**

***_item_type_list.code** (*code*)
The codes specifying the nature of the data value.
The following item(s) have an equivalent role in their respective categories:
_item_type.code. **[item_type_list]**

_item_type_list.construct (*text*)
When a data value can be defined as a pre-determined sequence of characters, optional characters or data names (for which the definition is also available), it is specified as a construction. The rules of construction conform to the the regular expression (REGEX) specifications detailed in IEEE (1991). Resolved data names for which **_item_type_list.construct** specifications exist are replaced by these constructions, otherwise the data-name string is not replaced.

Reference: IEEE (1991). *IEEE Standard for Information Technology – Portable Operating System Interface (POSIX) – Part 2: Shell and Utilities*, Vol. 1, IEEE Standard 1003.2-1992. New York: The Institute of Electrical Engineers.
Example: '`_year-_month-_day`' (typical construction for _date).
[item_type_list]

_item_type_list.detail (*text*)
An optional description of the data type.
[item_type_list]

***_item_type_list.primitive_code** (*code*)
The codes specifying the primitive type of the data value.
The data value must be one of the following:

`numb`	numerically interpretable string
`char`	character or text string (case-sensitive)
`uchar`	character or text string (case-insensitive)
`null`	for dictionary purposes only

[item_type_list]

ITEM_UNITS

Specifies the physical units in which data items are expressed.
Category group(s): **ddl_group**
item_group
Category key(s): **_item_units.name**

ITEM_UNITS_CONVERSION

Conversion factors between the various units of measure defined in the ITEM_UNITS_LIST category.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_units_conversion.from_code`
`_item_units_conversion.to_code`

*`_item_units_conversion.factor` (*any*)
The arithmetic operation required to convert between the unit systems: ⟨to_code⟩ = ⟨from_code⟩⟨operator⟩⟨factor⟩.
`[item_units_conversion]`

`_item_units_conversion.from_code`
The unit system on which the conversion operation is applied to produce the unit system specified in `_item_units_ conversion.to_code`: ⟨to_code⟩ = ⟨from_code⟩⟨operator⟩⟨factor⟩.

*`_item_units_conversion.operator` (*code*)
The arithmetic operator required to convert between the unit systems: ⟨to_code⟩= ⟨from_code⟩⟨operator⟩⟨factor⟩.
The data value must be one of the following:

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division

`[item_units_conversion]`

`_item_units_conversion.to_code`
The unit system produced after an operation is applied to the unit system specified by `_item_units_conversion.from_code`: ⟨to_code⟩ = ⟨from_code⟩⟨operator⟩⟨factor⟩.

ITEM_UNITS_LIST

Attributes which describe the physical units of measurement in which data items may be expressed.
Category group(s): `ddl_group`
`item_group`
Category key(s): `_item_units_list.code`

*`_item_units_list.code` (*code*)
The code specifying the name of the unit of measurement.
The following item(s) have an equivalent role in their respective categories:
`_item_units.code`,
`_item_units_conversion.from_code`,
`_item_units_conversion.to_code`. `[item_units_list]`

`_item_units_list.detail` (*text*)
A description of the unit of measurement.
`[item_units_list]`

METHOD_LIST

Attributes specifying the list of methods applicable to data items, subcategories and categories.
Category group(s): `ddl_group`
`item_group`
`category_group`
Category key(s): `_method_list.id`

*`_method_list.code` (*code*)
A code that describes the function of the method.
Examples: '`calculation`' (method to calculate the item), '`verification`' (method to verify the data item), '`cast`' (method to provide cast conversion), '`addition`' (method to define item + item), '`division`' (method to define item / item), '`multiplication`' (method to define item × item), '`equivalence`' (method to define item = item), '`other`' (miscellaneous method). `[method_list]`

`_method_list.detail` (*text*)
Description of application method in `_method_list.id`.
`[method_list]`

*`_method_list.id` (*idname*)
Unique identifier for each method listed.
The following item(s) have an equivalent role in their respective categories:
`_item_methods.method_id`,
`_category_methods.method_id`,
`_sub_category_methods.method_id`,
`_datablock_methods.method_id`. `[method_list]`

*`_method_list.inline` (*text*)
In-line text of a method associated with the data item.
`[method_list]`

*`_method_list.language` (*code*)
Language in which the method is expressed.
Examples: '`BNF`', '`C`', '`C++`', '`FORTRAN`', '`LISP`', '`PASCAL`', '`PERL`', '`TCL`', '`OTHER`'. `[method_list]`

SUB_CATEGORY

The purpose of a subcategory is to define an association between data items within a category and optionally to provide a method to validate the collection of items. For example, the subcategory named 'cartesian' might be applied to the data items for the coordinates *x*, *y* and *z*.
Category group(s): `ddl_group`
`sub_category_group`
Category key(s): `_sub_category.id`

*`_sub_category.description` (*text*)
Description of the subcategory.
`[sub_category]`

*`_sub_category.id` (*idname*)
The identity of the subcategory.
The following item(s) have an equivalent role in their respective categories:
`_sub_category_examples.id`,
`_sub_category_methods.sub_category_id`,
`_item_sub_category.id`. `[sub_category]`

SUB_CATEGORY_EXAMPLES

Example applications and descriptions of data items in this subcategory.
Category group(s): `ddl_group`
`sub_category_group`
Category key(s): `_sub_category_examples.id`
`_sub_category_examples.case`

*`_sub_category_examples.case` (*text*)
An example involving items in this subcategory.
`[sub_category_examples]`

`_sub_category_examples.detail` (*text*)
A description of an example given in `_sub_category_ examples.case`.
`[sub_category_examples]`

SUB_CATEGORY_METHODS

Attributes specifying the association between subcategories and methods.
Category group(s): `ddl_group`
`sub_category_group`
Category key(s): `_sub_category_methods.method_id`
`_sub_category_methods.sub_category_id`

APPENDIX 3: Classification and use of image data.

This is an early draft of chapter 2.10 of a new edition of *International Tables for Crystallography* Volume G (ITVG). In order to be able to cite portions of the original ITVG, the prefix **oITVG-** is used.

2.10. Classification and use of image data

BY H. J. BERNSTEIN

2.10.1. Introduction

This chapter describes the categories and organization of data items defined in the CBF/imgCIF dictionary. The classification of image data applies to both Crystallographic Binary File (CBF) and Image-supporting Crystallographic Information File (imgCIF) representations. An introduction to CBF data and construction is given in Chapter **oITVG-**2.3. Full details of the CBF/imgCIF dictionary are given in Chapter **oITVG-**4.6.

The main reason for introducing the new items defined in the CBF/imgCIF dictionary was to extend the mmCIF dictionary (Chapter **oITVG-**3.6) to allow the storage of synchrotron diffraction images. However, these items are also important in other fields that use binary image data, including the publication of articles, the creation of web pages and the production of movies.

Data categories in the CBF/imgCIF dictionary can describe one-, two- and three-dimensional array detectors that output data organized by time and/or wavelength. The categories defined at present support modular data that can be extended for future applications without having to make fundamental structural changes. For example, it is anticipated that additional data items will be needed soon to allow higher-dimensional data representations and more complex data structures; these should be accommodated easily.

The CBF/imgCIF dictionary consists of five groups of categories of data items: the ARRAY_DATA group, the AXIS group, the DIFFRN group, the MAP group. and the VARIANT group (Table 2.10.1.1). The DIFFRN group already exists in the mmCIF dictionary (Section **oITVG-**3.6.5.2; see also Section **oITVG-**3.2.2.2) and describes the diffraction data and their measurements. Definitions in the CBF/imgCIF dictionary extend and in some cases restate the definitions in the mmCIF dictionary.

The data categories defined in the CBF/imgCIF dictionary are described in this chapter. Table 2.10.1.1 lists the formal category groups declared in the dictionary and the sections of this chapter in which they are discussed. Each section is divided into subsections describing a single category or a small set of closely related categories. Within each subsection, the data names within the relevant categories are listed. Category keys, pointers to parent data items and aliases to data items in the mmCIF dictionary are indicated.

The data collected in an experiment are organized into scans. Each scan consists of one or more frames. Each frame consists of one or more data arrays. The logical data in the data arrays need to be described in terms of physical arrays of image elements.

Affiliation: HERBERT J. BERNSTEIN, School of Chemistry and Materials Science, College of Science, Rochester Institute of Technology, c/o National Synchrotron Light Source II, Building 745, Brookhaven National Laboratory, P.O. Box 5000, Upton, NY 11973-5000, USA.

Table 2.10.1.1. *Category groups defined in the CBF/imgCIF dictionary*

Section	Category group	Subject covered
Experimental measurements		
2.10.2	ARRAY_DATA	Binary image data
2.10.3	AXIS	Axes required to specify the data collection
2.10.4	DIFFRN	Diffraction experiment
2.10.5	MAP	Map data
2.10.6	VARIANT	Variants of data values

The axes of the laboratory coordinate system needed to describe the physical positions of the image elements and the positioning of the specimen are given in the AXIS category. The axes used for the positioning systems for the specimen and the detector are constructed in the same laboratory coordinate system. The DIFFRN_DETECTOR_AXIS category relates detector elements to axes. The DIFFRN_MEASUREMENT_AXIS category relates goniometers to axes. The DIFFRN_SCAN_AXIS and DIFFRN_SCAN_ FRAME_AXIS categories relate scans to overall axis settings and individual frames to frame-by-frame axis settings, respectively.

The organization of the data in the collected arrays of data is given in the ARRAY_STRUCTURE_LIST and the ARRAY_STRUCTURE_LIST_SECTION categories and the physical settings of axes for the centres of pixels that correspond to data points are given in the ARRAY_STRUCTURE_LIST_AXIS category.

2.10.2. Binary image data

The seven categories that collectively define the relationship between the sequences of octets in arrays of binary data and the information in the images those octets represent are:

ARRAY_DATA group
The image data (§2.10.2.1)
  ARRAY_DATA
Array elements (§2.10.2.2)
  ARRAY_ELEMENT_SIZE
Intensities (§2.10.2.3)
  ARRAY_INTENSITIES
Organization and encoding of array data (§2.10.2.4)
  ARRAY_STRUCTURE
  ARRAY_STRUCTURE_LIST
  ARRAY_STRUCTURE_LIST_SECTION
  ARRAY_STRUCTURE_LIST_AXIS

2.10.2.1. The image data

Data items in this category are as follows:

ARRAY_DATA
```
• _array_data.array_id
        → _array_structure.id
• _array_data.binary_id
• _array_data.variant
        → _variant.variant
  _array_data.data
```
The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item.

Each value of the `_array_data.data` data item is a sequence of octets representing a binary image. `_array_data.array_id` and `_array_data.binary_id`, and, optionally,`_array_data.variant` taken together, uniquely identify each image. The value of `_array_data.array_id` is a pointer to `_array_structure.id` to provide the relationship between the sequence of octets and the logical structure of the image. Since multiple images may have the same logical structure, the purpose of `_array_data.binary_id` is to ensure that each image has a unique identifier. `_array_data.variant` allows multiple observations of the same image to be image to be recorded. For example, this can be the result of multiple refinements of the positions of detector elements.

2.10.2.2. Array elements

Data items in this category are as follows:

ARRAY_ELEMENT_SIZE

```
• _array_element_size.array_id
      → _array_structure.id
• _array_element_size.index
      → _array_structure_list.index
• _array_element_size.variant
      → _variant.variant
  _array_element_size.size
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item.

The value of the `_array_element_size.size` data item is a size in metres of an image element (a pixel or voxel). The direction of the measurement is given in each dimension by `_array_element_size.index`. The array structure specifying the organization of the dimensions is referenced by the value of `_array_element_size.array_id`, which is a pointer to `_array_structure.id`. The value of `_array_element_size.index` is a pointer to `_array_structure_list.index`. For data organized into rectangular arrays of pixels or voxels, this gives the spatial dimensions of the individual image elements. The value of `_array_element_size.variant` is a pointer to `_variant.variant` which may optionally be used to present variants of image element sizes due, say, to multiple measurements or refinements.

2.10.2.3. Intensities

Data items in this category are as follows:

ARRAY_INTENSITIES

```
• _array_intensities.array_id
      → _array_structure.id
• _array_intensities.binary_id
      → _array_data.binary_id
  _array_intensities.details
  _array_intensities.gain
  _array_intensities.gain_esd
  _array_intensities.linearity
  _array_intensities.offset
  _array_intensities.overload
  _array_intensities.pixel_fast_bin_size
  _array_intensities.pixel_slow_bin_size
  _array_intensities.pixel_binning_method
  _array_intensities.scaling
  _array_intensities.undefined_value
• _array_intensities.variant
      → _variant.variant
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item.

The relationship between the data values for individual image elements and the number of incident photons can be complex. The data items in the ARRAY_INTENSITIES category provide information about this relationship. The value of `_array_intensities.linearity` states the type of relationship, and the values of `_array_intensities.array_id` and `_array_intensities.binary_id` identify the array structure and the image being discussed. The other items are used in different ways depending on the relationship. If the value of `_array_intensities.linearity` is `raw`, then the image elements hold uninterpreted raw data values from the detector, *e.g.* for calibration. If the value of `_array_intensities.linearity` is `linear`, then the count in an image element is proportional to the incident number of photons by the value of `_array_intensities.gain`. The standard uncertainty (estimated standard deviation) of the gain is optionally given in `_array_intensities.gain_esd`. The value used for this should be estimated from a good understanding of the physical characteristics of the experimental apparatus. If the value of `_array_intensities.linearity` is `offset`, then the value of `_array_intensities.offset` should be added to the image element value. If the value of `_array_intensities.linearity` is `scaling`, `scaling_offset`, `sqrt_scaled` or `logarithmic_scaled`, the necessary scaling factor is given by the value of `_array_intensities.scaling`. In all cases, the scaling factor is applied to the image element value before the other operations are applied. In the first case, only simple scaling is used. In the second case, the value of `_array_intensities.offset` is added after scaling. In the third case, the scaled value is squared. In the final case, 10 is taken to the power given by the scaled value. Binning is recorded using `_array_intensities.pixel_fast_bin_size`, `_array_intensities.pixel_slow_bin_size`, and `_array_intensities.pixel_binning_method`. The optional value of `_array_intensities.variant` is available to present multiple observations or calculations of this data.

2.10.2.4. Organization and encoding of array data

Data items in these categories are as follows:

(*a*) ARRAY_STRUCTURE

```
• _array_structure.id
  _array_structure.byte_order
  _array_structure.compression_type
  _array_structure.compression_type_flag
  _array_structure.encoding_type
• _array_structure.variant
      → _variant.variant
```

(*b*) ARRAY_STRUCTURE_LIST

```
• _array_structure_list.array_id
• _array_structure_list.index
      → _array_structure.id
  _array_structure_list.axis_set_id
  _array_structure_list.dimension
  _array_structure_list.direction
  _array_structure_list.precedence
  • _array_structure_list.variant
      → _variant.variant
```

(*b*) ARRAY_STRUCTURE_LIST_SECTION

```
• _array_structure_list_section.id
• _array_structure_list_section.array_id
• _array_structure_list_section.index
      → _array_structure_list.id
```

```
  _array_structure_list_section.start
  _array_structure_list_section.stride
  _array_structure_list_section.end
• _array_structure_list_section.variant
        → _variant.variant
```

(*c*) ARRAY_STRUCTURE_LIST_AXIS

```
• _array_structure_list_axis.axis_id
        → _axis.id
• _array_structure_list_axis.axis_set_id
        → _array_structure_list.axis_set_id
  _array_structure_list_axis.angle
  _array_structure_list_axis.angle_increment
  _array_structure_list_axis.angular_pitch
  _array_structure_list_axis.displacement
  _array_structure_list_axis.displacement_increment
  _array_structure_list_axis.radial_pitch
• _array_structure_list_axis.variant
        → _variant.variant
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item.

The data items in the ARRAY_STRUCTURE category show how the stream of octets in a binary image is to be reorganized into words of an appropriate size. Each possible encoding is identified by a value of **_array_structure.id**. In most cases, large images will have been compressed. The type of compression used is given by **_array_structure.compression_type**. Once a stream of octets has been decompressed, it can be organized into words. The type of each word is given by the value of **_array_structure.encoding_type** and the order of mapping octets onto words, most significant octet first ('big-endian') or least significant octet first ('little-endian'), is given by the value of **_array_structure.byte_order**.

The data items in the ARRAY_STRUCTURE_LIST category show how the list of words defined by the ARRAY_STRUCTURE category should be organized into image arrays. The value of **_array_structure_list.array_id** is a pointer to **_array_structure.id**. Each dimension (row, column, sheet *etc.*) of the image is identified by an index, counting from 1, given by **_array_structure_list.index**.
The order of nesting of the indices is given by the values of **_array_structure_list.precedence**, with the index of precedence 1 varying most rapidly (*i.e.* having values stored sequentially). The direction of index change for increasing memory location is given by the value of **_array_structure_list.direction**. For a given index, the number of image elements in that direction is given by the value of **_array_structure_list.dimension**.

The data items in the ARRAY_STRUCTURE_LIST_SECTION category are used to define subsections of arrays. The value of **_array_structure_list_section.array_id** is a pointer to the **_array_structure.id** for which **_array_structure_list_section.id** identifies an array section. The start, stride and end of the section are defined by the values of **_array_structure_list_section.start**, **_array_structure_list_section.stride**, and **_array_structure_list_section.end**. For any array of **array_id**, **ARRAYID**, array section ids of the form
ARRAYID(start1:end1:stride1,start2:end2:stride2,...)
are defined by default.

Data items in the ARRAY_STRUCTURE_LIST_AXIS category describe the physical settings of sets of axes for the centres of pixels that correspond to data points described in the ARRAY_STRUCTURE_LIST category.

In the simplest cases, the physical increments of a single axis correspond to the increments of a single array index. More complex organizations (*e.g.* spiral scans) may require coupled motions along multiple axes.

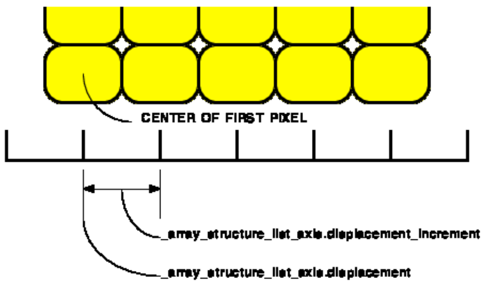

CENTER OF FIRST PIXEL
_array_structure_list_axis.displacement_increment
_array_structure_list_axis.displacement

Fig. 2.10.2.1. ARRAY_STRUCTURE_LIST specification of linearly organized image elements.

Note that a spiral scan uses two coupled axes, one for the angular direction and one for the radial direction. This differs from a cylindrical scan for which the two axes are not coupled into one set.

Multiple related axes are gathered together into sets. Each set is identified by the value of the axis set identifier, **_array_structure_list_axis.axis_set_id**, and each axis within a set is identified by the value of **_array_structure_list_axis.axis_id**. Each set given by a value of ***.axis_set_id** is linked to a corresponding value for **_array_structure_list.axis_set_id** to relate settings of the axes in the axis set to particular image elements in ARRAY_STRUCTURE_LIST.

If axes are all independent, no value need be given for **_array_structure_list_axis.axis_set_id**, which is then implicitly given the corresponding value of **_array_structure_list_axis.axis_id**. Each axis given by a value of **_array_structure_list_axis.axis_id** is linked to a corresponding value for **_axis.id** to provide a physical description of the axis. **_array_structure_list_axis.axis_id** and **_array_structure_list_axis.axis_set_id** together uniquely identify a row of data in an ARRAY_STRUCTURE_LIST AXIS table.

For the remaining data items, there are two important cases to consider: axes that step by Euclidean distance and axes that step by angle. Fig. 2.10.2.1 shows a portion of an array of image elements laid out on a rectangular grid. The starting point of an axis is specified in millimetres by the value of **_array_structure_list_axis.displacement** and the centre-to-centre distance between pixels is specified in millimetres by the value of **_array_structure_list_axis.displacement_increment**.

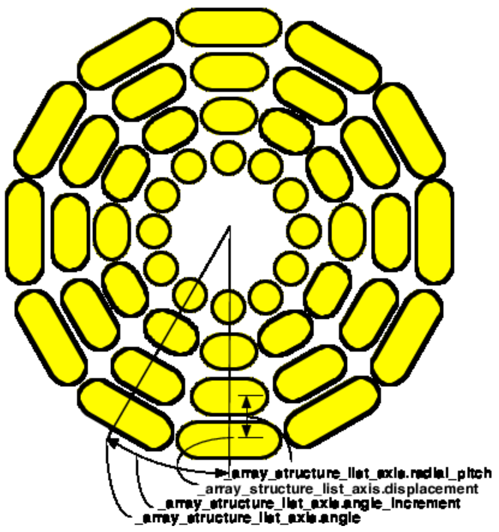

_array_structure_list_axis.radial_pitch
_array_structure_list_axis.displacement
_array_structure_list_axis.angle_increment
_array_structure_list_axis.angle

Fig. 2.10.2.2. ARRAY_STRUCTURE_LIST specification of 'constant-angle' image elements in a cylindrical scan. The angular and radial axes are independent. Note that outer-zone image elements are further apart, centre-to-centre, than inner-zone image elements.

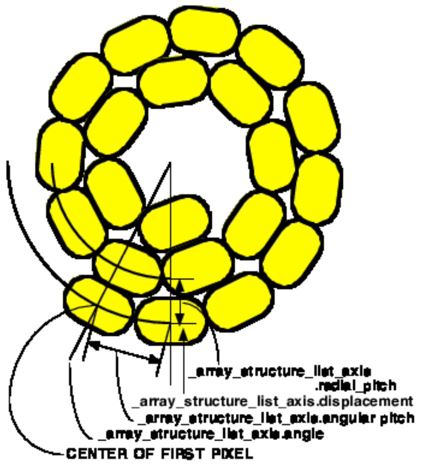

_array_structure_list_axis .radial_pitch
_array_structure_list_axis.displacement
_array_structure_list_axis.angular pitch
_array_structure_list_axis.angle
CENTER OF FIRST PIXEL

Fig. 2.10.2.3. ARRAY_STRUCTURE_LIST specification of 'constant-velocity' image elements in a cylindrical scan. The angular and radial axes are coupled. Note that outer-zone image elements are the same linear distance apart, centre-to-centre, as the inner-zone image elements.

Fig. 2.10.2.2 shows a portion of an array of image elements laid out in concentric cylinders. The starting point of the angular axis is specified in degrees by the value of `_array_structure_list_axis.angle` and the centre-to-centre angular distance between pixels is specified in degrees by the value of `_array_structure_list_axis.angle_increment`. The starting point of the radial axis is specified by the value of `_array_structure_list_axis.displacement` and the radial distance between cylinders of pixels is specified in millimetres by the value of `_array_structure_list_axis.radial_pitch`. Note that the image elements further from the centre are larger than the image elements closer to the centre.

Fig. 2.10.2.3 shows a portion of a spiral scan array in which the angular and radial axes are coupled. This example represents a 'constant-velocity' scan, in which the size of the image elements does not depend on the distance from the centre. The starting point of the angular axis is again specified in degrees by the value of `_array_structure_list_axis.angle`, but the centre-to-centre distance between pixels is specified in millimetres by the value of `_array_structure_list_axis.angular_pitch`. The coupled radial axis is handled in much the same way as for the uncoupled radial axis in the cylindrical array.

These examples show some of the more common two-dimensional data structures. By coupling an additional axis not in the plane of the first two, regular three-dimensional arrays of data can be represented without additional tags. The categories in the DIFFRN group allow arrays of data to be associated with frames and thereby with time and/or wavelength. More general data structures, for example ones based on dope vectors or hash tables, would require the definition of additional tags, but any data structure (see Aho *et al.*, 1987) that can be handled by a modern computer should be manageable within this framework.

Optional variants are permitted, but would not be commonly used in crystallographic experiments.

2.10.3. Axes

The category describing the axes required to specify the data collection is as follows:

AXIS group
  AXIS

Data items in this category are as follows:

```
AXIS
• _axis.equipment
• _axis.id
  _axis.depends_on
       → _axis.id
  _axis.equipment_component
  _axis.offset[1]
  _axis.offset[2]
  _axis.offset[3]
  _axis.rotation
  _axis.rotation_axis
  _axis.system
  _axis.type
  _axis.vector[1]
  _axis.vector[2]
  _axis.vector[3]
• _axis.variant
       → _variant.variant
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item.

Data items in the AXIS category record the information required to describe the goniometer, detector, source and other axes needed to specify a data collection. The location of each axis is specified by two vectors: the axis itself, given as a unit vector, and an offset to the base of the unit vector.

The vectors defining an axis are referenced to an appropriate coordinate system. The axis vector, itself, is a dimensionless unit vector. Where meaningful, the offset vector is given in millimetres. In coordinate systems not measured in metres, the offset is not specified and is taken as zero.

The available coordinate systems are:

The imgCIF standard laboratory coordinate system
The direct lattice (fractional atomic coordinates)
The orthogonal Cartesian coordinate system (real space)
The reciprocal lattice
An abstract orthogonal Cartesian coordinate frame

For consistency in this discussion, we call the three coordinate system axes X, Y and Z. This is appropriate for the img-

CIF standard laboratory coordinate system, and the last two Cartesian coordinate systems, but for the direct lattice, X corresponds to a, Y to b and Z to c, while for the reciprocal lattice, X corresponds to a^*, Y to b^* and Z to c^*.

For purposes of visualization, all the coordinate systems are taken as right-handed, *i.e.*, using the convention that the extended thumb of a right hand could point along the first (X) axis, the straightened pointer finger could point along the second (Y) axis and the middle finger folded inward could point along the third (Z) axis.

THE IMGCIF STANDARD LABORATORY COORDINATE SYSTEM

The imgCIF standard laboratory coordinate system is a right-handed orthogonal coordinate similar to the MOSFLM coordinate system, but imgCIF puts Z along the X-ray beam, rather than putting X along the X-ray beam as in MOSFLM.

The vectors for the imgCIF standard laboratory coordinate system form a right-handed Cartesian coordinate system with its origin in the sample or specimen.

The origin of the axis system should, if possible, be defined in terms of mechanically stable axes to be be both in the sample and in the beam. If the sample goniometer or other ample positioner has two axes the intersection of which defines a unique point at which the sample should be mounted to be bathed by the beam, that will be the origin of the axis system. If no such point is defined, then the midpoint of the line of intersection between the sample and the centre of the beam will define the origin. For this definition the sample positioning system will be set at its initial reference position for the experiment.

1 (X) The X axis is aligned to the mechanical axis pointing from the sample or specimen along the principal axis of the goniometer or sample positioning system if the sample positioning system has an axis that intersects the origin and which forms an angle of more than 22.5 degrees with the beam axis.

2 (Y) The Y axis completes an orthogonal right-handed system defined by the X axis and the Z axis (see below).

3 (Z) The Z axis is derived from the source axis which goes from the sample to the source.. The Z axis is the component of the source axis orthogonal to the X axis in the plane defined by the X axis and the source axis.

If the conditions for the X axis can be met, the coordinate system will be based on the goniometer or other sample positioning system and the beam and not on the orientation of the detector, gravity etc. The vectors necessary to specify all other axes are given by sets of three components in the order (X,Y,Z). If the axis involved is a rotation axis, it is right-handed, *i.e.* as one views the object to be rotated from the origin (the tail) of the unit vector, the rotation is clockwise. If a translation axis is specified, the direction of the unit vector specifies the sense of positive translation.

Note: This choice of coordinate system is similar to but significantly different from the choice in MOSFLM (Leslie & Powell, 2004). In MOSFLM, X is along the X-ray beam (the CBF/imgCIF Z axis) and Z is along the rotation axis.

In some experimental techniques, there is no goniometer or the principal axis of the goniometer is at a small acute angle with respect to the source axis. In such cases, other reference axes are needed to define a useful coordinate system. The order of priority in defining directions in such cases is to use the detector, then gravity, then north.

If the X axis cannot be defined as above, then the direction (not the origin) of the X axis should be parallel to the axis of the primary detector element corresponding to the most rapidly varying dimension of that detector element's data array, with its positive sense corresponding to increasing values of the index for that dimension. If the detector is such that such a direction cannot be defined (as with a point detector) or that direction forms an angle of less than 22.5 degrees with respect to the source axis, then the X axis should be chosen so that if the Y axis is chosen in the direction of gravity, and the Z axis is chosen to be along the source axis, a right-handed orthogonal coordinate system is chosen. In the case of a vertical source axis, as a last resort, the X axis should be chosen to point North.

All rotations are given in degrees and all translations are given in mm.

Axes may be dependent on one another. The X axis is the only goniometer axis the direction of which is strictly connected to the hardware. All other axes are specified by the positions they would assume when the axes upon which they depend are at their zero points.

When specifying detector axes, the axis is given to the beam centre. The location of the beam centre on the detector should be given in the DIFFRN_DETECTOR category in distortion-corrected millimetres from the (0,0) corner of the detector.

For convenience when describing detector element (module) placement, and optional mounting rotation axis and rotation angle may be specified. In such cases, the mounting rotation axis and angle of rotation around the mounting rotation axis are applied after applying the transformations upon which the given axis depends.

It should be noted that many different origins arise in the definition of an experiment. In particular, as noted above, it is necessary to specify the location of the beam centre on the detector in terms of the origin of the detector, which is, of course, not coincident with the centre of the sample.

The unit cell, reciprocal cell and crystallographic orthogonal Cartesian coordinate system are defined by the cell and the matrices in the ATOM_SITES category.

THE DIRECT LATTICE (FRACTIONAL COORDINATES)

The direct lattice coordinate system is a system of fractional coordinates aligned to the crystal, rather than to the laboratory. This is a natural coordinate system for maps and atomic coordinates. It is the simplest coordinate system in which to apply symmetry. The axes are determined by the cell edges, and are not necessarily orthogonal. This coordinate system is not uniquely defined and depends on the cell parameters in the CELL category and the settings chosen to index the crystal.

Molecules in a crystal studied by X-ray diffracraction are organized into a repeating regular array of unit cells. Each unit cell is defined by three vectors, a, b and c. To quote from Drenth,

"The choice of the unit cell is not unique and therefore, guidelines have been established for selecting the standard basis vectors and the origin. They are based on symmetry and metric considerations:

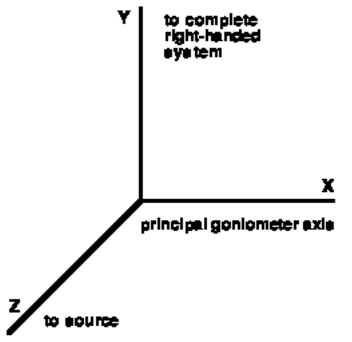


Fig. 2.10.3.1. AXIS laboratory coordinate system. The origin is centred in the specimen.

"(1) The axial system should be right handed.

"(2) The basis vectors should coincide as much as possible with directions of highest symmetry.

"(3) The cell taken should be the smallest one that satisfies condition (2)

"(4) Of all the lattice vectors, none is shorter than a.

"(5) Of those not directed along a, none is shorter than b.

"(6) Of those not lying in the ab plane, none is shorter than c.

"(7) The three angles between the basis vectors a, b and c are either all acute ($< 90\%$) or all obtuse ($\geq 90\%$)."

These rules do not produce a unique result that is stable under the assumption of experimental errors, and the the resulting cell may not be primitive.

In this coordinate system, the vector $(.5, .5, .5)$ is in the middle of the given unit cell.

Grid coordinates are an important variation on fractional coordinates used when working with maps. In imgCIF, the conversion from fractional to grid coordinates is implicit in the array indexing specified by `_array_structure_list.dimension`. Note that this implicit grid-coordinate scheme is 1-based, not zero-based, *i.e.* the origin of the cell for axes along the cell edges with no specified `_array_structure_list_axis.displacement` will have grid coordinates of (1,1,1), *i.e.* array indices of (1,1,1).

THE ORTHOGONAL CARTESIAN COORDINATE SYSTEM (REAL SPACE)

The orthogonal Cartesian coordinate system is a transformation of the direct lattice to the actual physical coordinates of atoms in space. It is similar to the laboratory coordinate system, but is anchored to and moves with the crystal, rather than being anchored to the laboratory. The transformation from fractional to orthogonal cartesian coordinates is given by the values of the `_atom_sites.Cartn_transf_matrix[i][j]` and `_atom_sites.Cartn_transf_vector[i]` data names. A common choice for the matrix of the transformation is given in the 1992 PDB format document

$$\begin{pmatrix} a & b\cos(\gamma) & c\cos(\beta) \\ 0 & b\sin(\gamma & c(\cos(\alpha) - \cos(\beta)\cos(\gamma))/\sin(\gamma) \\ 0 & 0 & V/(ab\sin(\gamma)) \end{pmatrix}$$

This is a convenient coordinate system in which to do fitting of models to maps and in which to understand the chemistry of a molecule.

THE RECIPROCAL LATTICE

The reciprocal lattice coordinate system is used for diffraction intensities. It is based on the reciprocal cell, the dual of the cell, in which reciprocal cell edges are derived from direct cell faces:

$$\begin{aligned} a^* &= bc\sin(\alpha)/V \\ b^* &= ac\sin(\beta)/V \\ c^* &= ab\sin(\gamma)/V \\ \cos(\alpha^*) &= (\cos(\beta)\cos(\gamma) - \cos(\alpha))/(\sin(\beta)\sin(\gamma)) \\ \cos(\beta^*) &= (\cos(\alpha\cos(\gamma) - \cos(\beta))/(\sin(\alpha)\sin(\gamma)) \\ \cos(\gamma^*) &= (\cos(\alpha)\cos(\beta) - \cos(\gamma))/(\sin(\alpha)\sin(\beta)) \end{aligned}$$

$$V = abc\Big[1 - \cos(\alpha)^2 - \cos(\beta)^2 - \cos(\gamma)^2 + 2\cos(\alpha)\cos(\beta)\cos(\gamma)\Big]^{1/2}$$

In this form the dimensions of the reciprocal lattice are in reciprocal ångstroms (Å^{-1}). A dimensionless form can be obtained by multiplying by the wavelength. Reflections are commonly indexed against this coordinate system as (h, k, l) triples.

DIFFERENCES BETWEEN NEXUS AND CBF COORDINATE FRAMES

The standard coordinate frame in NeXus is the McStas coordinate frame, in which the Z axis points in the direction of the incident beam, the X axis is orthogonal to the Z axis in the horizontal plane and pointing left as seen from the source and the Y axis points upwards. The origin is in the sample.

The standard coordinate frame in imgCIF/CBF aligns the X axis to the principal goniometer axis, chooses the Z axis to point from the sample into the beam. If the beam is not orthogonal to the X axis, the Z axis is the component of the vector points into the beam orthogonal to the X axis. The Y axis is chosen to complete a right-handed axis system.

Let us call the NeXus coordinate axes, X_{nx}, Y_{nx} and Z_{nx} and the imgCIF/CBF coordinate axes, X_{cbf}, Y_{cbf} and Z_{cbf} and the direction of gravity, *Gravity*. In order to translate a vector $v_{nx} = (x, y, z)$ from the NeXus coordinate system to the imgCIF coordinate system, we also need two additional axes, as unit vectors, $Gravity_{cbf}$ the downwards direction, and $Beam_{cbf}$, the direction of the beam *e.g.* $(0, 0, -1)$.

In practice, the beam is not necessarily perfectly horizontal, so Y_{nx} is not necessarily perfectly vertical. Therefore, in order to generate X_{nx}, Y_{nx} and Z_{nx} some care is needed. The cross product between two vectors a and b is a new vector c orthogonal to both a and b, chosen so that a, b, c is a right handed system. If a and b are orthogonal unit vectors, this right-handed system is an orthonormal coordinate system.

In the CBF coordinate frame, Z_{nx} is aligned to $Beam_{cbf}$

$$Z_{nx} = Beam_{cbf}$$

X_{nx} is defined as being horizontal at right angles to the beam, pointing to the left when seen from the source. Assuming the beam is not vertical, we can compute X_{nx} as the normalized cross product of the beam and gravity:

$$X_{nx} = (Beam_{cbf} \times Gravity_{cbf})/||Beam_{cbf} \times Gravity_{cbf}||$$

To see that this satisfies the constraint of being horizontal and pointing to the left, consider the case of $Beam = (0, 0, -1)$ and $Gravity = (0, 0, 1)$ then we would have $X_{nx} = (1, 0, 0)$ from the cross product above. The normalization is only necessary if the beam is not horizontal.

Finally Y_{nx} is computed as the cross product of the beam and X_{nx}, completing an orthonormal right-handed system with Y_{nx} pointing upwards

$$Y_{nx} = Beam_{cbf} \times X_{nx}$$

Then we know that in the imgCIF/CBF coordinate frame

$$v_{nx} = X.X_{nx} + Y.Y_{nx} + Z.Z_{nx}$$

Thus, given the imgCIF/CBF vectors for the true direction of the beam and the true direction of gravity, we have a linear transformation from the NeXus coordinate frame to the imgCIF/CBF coordinate frame. The origins of the two frames agree. The inverse linear transformation will transform a vector in the imgCIF/CBF coordinate frame into the NeXus coordinate frame.

In the common case in which the beam is orthogonal to the principal goniometer axis so that $Beam_cbf = (0, 0, -1)$ and the imgCIF/CBF Y axis points upwards, the transformation inverts the X and Z axes. In the other common case in which the beam is orthogonal to the principal goniometer axis and the imgCIF/CBF Y axis points downwards, the transformation inverts the Y and Z axes.

Each axis is uniquely identified by the values of `_axis.id` and of `_axis.equipment`. An axis may be a translation axis, a rotation axis or an axis for which the mode of motion is not relevant. The type of axis is specified by the value of `_axis.type`. The base of the axis is specified by the point in the laboratory coordinate system given by the values of `_axis.offset[1]`, `_axis.offset[2]` and `_axis.offset[3]`, and the direction of the axis from that base, as a dimensionless unit vector, is given by `_axis.vector[1]`, `_axis.vector[2]` and `_axis.vector[3]`. Axes can be refined from the data collected, so variants are relevant.

2.10.4. The diffraction experiment

The categories relating to the diffraction experiment are as follows:

DIFFRN group
Frames of data (§2.10.4.1)
DIFFRN_DATA_FRAME
The detector apparatus (§2.10.4.2)
DIFFRN_DETECTOR
DIFFRN_DETECTOR_AXIS
DIFFRN_DETECTOR_ELEMENT
Apparatus and instrumentation at the crystal (§2.10.4.3)
DIFFRN_MEASUREMENT
DIFFRN_MEASUREMENT_AXIS
The radiation source (§2.10.4.4)
DIFFRN_RADIATION
Intensity measurements (§2.10.4.5)
DIFFRN_REFLN
Diffraction scans (§2.10.4.6)
DIFFRN_SCAN
DIFFRN_SCAN_AXIS
DIFFRN_SCAN_FRAME
DIFFRN_SCAN_FRAME_AXIS

The CBF/imgCIF dictionary extends the mmCIF categories in the DIFFRN group, which are very similar to their corresponding categories in the core CIF dictionary. The DIFFRN group is introduced in the description of the core CIF dictionary in Section **oITVG-**3.2.2.2. Its use in the mmCIF dictionary is described in Section **oITVG-**3.6.5.2, from which we quote: 'The categories in the DIFFRN category group describe the diffraction experiment. Data items in the DIFFRN category itself can be used to give overall information about the experiment, such as the temperature and pressure. Examples of the other categories are DIFFRN_DETECTOR, which is used for describing the detector used for data collection, and DIFFRN_SOURCE, which is used to give details of the source of the radiation used in the experiment. Data items in the DIFFRN_REFLN category can be used to give information about the raw data and data items in the DIFFRN_REFLNS category can be used to give information about all the reflection data collectively.'

In this chapter we focus on the CBF/imgCIF extensions. Note that each category supports variants.

2.10.4.1. Frames of data

Data items in this category are as follows:

DIFFRN_DATA_FRAME
```
• _diffrn_data_frame.detector_element_id
        → _diffrn_detector_element.id
• _diffrn_data_frame.id
  _diffrn_data_frame.array_id
        → _array_structure.id
  _diffrn_data_frame.array_section_id
        → _array_structure_list_section.id
  _diffrn_data_frame.binary_id
        → _array_data.binary_id
  _diffrn_data_frame.center_fast
  _diffrn_data_frame.center_slow
  _diffrn_data_frame.center_units
  _diffrn_data_frame.detector_element_id
  _diffrn_data_frame.details
• _diffrn_data_frame.variant
        → _variant.variant
```

The bullet (●) indicates a category key. The arrow (→) is a reference to a parent data item.

Data items in the DIFFRN_DATA_FRAME category record details about each frame of data. An experiment may produce multiple frames of data and each frame may be constructed from data provided by multiple detector elements. Each complete frame of data is uniquely identified by the value of `_diffrn_data_frame.id`. The detector elements used are specified by values of `_diffrn_ data_frame.detector_element_id`, which forms the category key together with `_diffrn_data_frame.id`. `_diffrn_data_ frame.detector_element_id` is a pointer to `_diffrn_detector_ element.id` in the DIFFRN_DETECTOR_ELEMENT category. The structure of the data in the frame is completed by giving values for `_diffrn_data_frame.array_id` (a pointer to `_array_ structure.id`). The particular blocks of data in the frame are specified by giving values of `_diffrn_data_frame.binary_id` (a pointer to `_array_data.binary_id`).

The beam centre may be specified by the values of `_diffrn_data_frame.center_fast`, `_diffrn_data_frame.center _slow`, and `_diffrn_data_frame.center_units`. It is unusual in CIF to specify units in the data, but there is no community agreement on the use of millimetres, pixels or bins.

2.10.4.2. The detector apparatus

Data items in these categories are as follows:

(*a*) DIFFRN_DETECTOR
```
• _diffrn_detector.diffrn_id
        → _diffrn.id
• _diffrn_detector.id
  _diffrn_detector.details
  _diffrn_detector.detector
  _diffrn_detector.dtime
  _diffrn_detector.gain_setting
  _diffrn_detector.number_of_axes
  _diffrn_detector.type
  _diffrn_detector.layer_thickness
• _diffrn_detector.variant
        → _variant.variant
```

(*b*) DIFFRN_DETECTOR_AXIS
```
• _diffrn_detector_axis.axis_id
        → _axis.id
• _diffrn_detector_axis.detector_id
        → _diffrn_detector.id
• _diffrn_detector_axis.variant
        → _variant.variant
```

(*c*) DIFFRN_DETECTOR_ELEMENT
```
• _diffrn_detector_element.id
• _diffrn_detector_element.detector_id
        → _diffrn_detector.id
  _diffrn_detector_element.center[1]
  _diffrn_detector_element.center[2]
• _diffrn_detector_element.variant
        → _variant.variant
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item. Items in italics are defined in the mmCIF dictionary.

The DIFFRN_DETECTOR category is defined in the mmCIF dictionary (Section **oITVG-**3.6.5.2; see the detailed discussion in Section **oITVG-**3.2.2.2.4). The CBF/imgCIF dictionary restates the DIFFRN_DETECTOR category, adding new tags. Data items in the DIFFRN_DETECTOR category describe the detector used to measure the scattered radiation, including any analyser and post-sample collimation. In order to allow for multiple detectors, the category key has been extended to include `_diffrn_detector.id` to uniquely identify each detector. If there is only one detector, `_diffrn_detector.id` need not be specified, and it will implicitly default to the value of `_diffrn_detector.diffrn_id` (a pointer to `_diffrn.id` in the DIFFRN category in the mmCIF dictionary). The general class of detector is given by the value of `_diffrn_detector.detector` with the make and model given by the value of `_diffrn_detector.type`. Any special aspects of the detector not covered elsewhere are given by the value of `_diffrn_detector.details`. As in mmCIF, the value of `_diffrn_detector.dtime` gives the deadtime of the detector. Additional data items may need to be added in the future for complex inhomogeneous deadtime situations. In addition, the number of axes can be specified using `_diffrn_detector.number_of_axes`.

Data items in the DIFFRN_DETECTOR_AXIS category associate axes with detectors. Each axis is associated with a detector through the value of `_diffrn_detector_axis.detector_id` (a pointer to `_diffrn_detector.id`). The value of `*.axis_id` (a pointer to `_axis.id`) identifies an axis. Together `*.detector_id` and `*.axis_id` form the category key.

Data items in the DIFFRN_DETECTOR_ELEMENT category record details about the spatial layout and other characteristics of each element of a detector which may have multiple elements, giving the *X* and *Y* coordinates of the position of the beam centre relative to the lower left corner of each detector element. Each detector element is identified by the value of `_diffrn_detector_element.id` and the detector of which it is an element is identified by the value of `_diffrn_detector_element.detector_id` (a pointer to `_diffrn_detector.id`).

In most cases, it would be preferable to use the more detailed information provided in the ARRAY_STRUCTURE_LIST and ARRAY_STRUCTURE_LIST_AXIS categories rather than simply specifying the coordinates of the centre of the beam relative to the lower left corner of each detector element.

2.10.4.3. Apparatus and instrumentation at the crystal

Data items in these categories are as follows:

(*a*) DIFFRN_MEASUREMENT
```
• _diffrn_measurement.diffrn_id
        → _diffrn.id
• _diffrn_measurement.device
• _diffrn_measurement.id
  _diffrn_measurement.details
  _diffrn_measurement.device_details
  _diffrn_measurement.device_type
  _diffrn_measurement.method
  _diffrn_measurement.number_of_axes
  _diffrn_measurement.specimen_support
• _diffrn_measurement.variant
        → _variant.variant
```

(*b*) DIFFRN_MEASUREMENT_AXIS
```
• _diffrn_measurement_axis.axis_id
        → _axis.id
• _diffrn_measurement_axis.measurement_device
        → _diffrn_measurement.device
• _diffrn_measurement_axis.measurement_id
        → _diffrn_measurement.id
• _diffrn_measurement_axis.variant
        → _variant.variant
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item. Items in italics are defined in the mmCIF dictionary.

The DIFFRN_MEASUREMENT category is defined in the mmCIF dictionary (Section **oITVG-**3.6.5.2; see the detailed discussion in Section **oITVG-**3.2.2.2.3). The CBF/imgCIF dictionary restates the DIFFRN_MEASUREMENT category, adding new tags. Data items in the DIFFRN_MEASUREMENT category record details about the device used to orient and/or position the crystal during data measurement and the manner in which the diffraction data were measured. To allow for multiple measurement devices, `_diffrn_measurement.id` has been added to the category key. The number of axes is given by the value of `_diffrn_measurement.number_of_axes`. The axes should be described using entries in DIFFRN_MEASUREMENT_AXIS.

Data items in the DIFFRN_MEASUREMENT_AXIS category associate axes with goniometers, just as data items in the DIFFRN_DETECTOR_AXIS category associate axes with detectors.

2.10.4.4. The radiation source

Data items in this category are as follows:

DIFFRN_RADIATION
- ● *_diffrn_radiation.diffrn_id*
  - → _diffrn.id
- *_diffrn_radiation.collimation*
- _diffrn_radiation.div_x_source
- _diffrn_radiation.div_y_source
- _diffrn_radiation.div_x_y_source
- *_diffrn_radiation.filter_edge*
- *_diffrn_radiation.inhomogeneity*
- *_diffrn_radiation.monochromator*
- *_diffrn_radiation.polarisn_norm*
- *_diffrn_radiation.polarisn_norm_esd*
- *_diffrn_radiation.polarisn_ratio*
- *_diffrn_radiation.polarisn_ratio_esd*
- _diffrn_radiation.polarizn_source_norm
- _diffrn_radiation.polarizn_source_norm_esd
- _diffrn_radiation.polarizn_source_ratio
- _diffrn_radiation.polarizn_source_ratio_esd
- _diffrn_radiation.polarizn_Stokes_I
- _diffrn_radiation.polarizn_Stokes_I_esd
- _diffrn_radiation.polarizn_Stokes_Q
- _diffrn_radiation.polarizn_Stokes_Q_esd
- _diffrn_radiation.polarizn_Stokes_U
- _diffrn_radiation.polarizn_Stokes_U_esd
- _diffrn_radiation.polarizn_Stokes_V
- _diffrn_radiation.polarizn_Stokes_V_esd
- *_diffrn_radiation.probe*
- *_diffrn_radiation.type*
- *_diffrn_radiation.wavelength_id*
  - → _diffrn_radiation_wavelength.id
- *_diffrn_radiation.xray_symbol*
- ● _diffrn_radiation.variant
  - → _variant.variant

The bullet (●) indicates a category key. The arrow (→) is a reference to a parent data item. Items in italics are defined in the mmCIF dictionary.

The DIFFRN_RADIATION category is defined in the mmCIF dictionary (Section **oITVG-**3.6.5.2; see the detailed discussion in Section **oITVG-**3.2.2.2.2). The CBF/imgCIF dictionary adds the items

`_diffrn_radiation.div_x_source`, `*.div_y_source` and `*.div_x_y_source` to specify beam crossfire, and the items

`_diffrn_radiation.polarizn_source_norm` and `*.polarizn_source_ratio` to provide a definition of polarization relative to the laboratory coordinate system rather than relative to the diffraction plane. The value of the beam crossfire component

`_diffrn_radiation.div_x_source` is the mean deviation in degrees of the X-ray beam from being parallel to the X axis as it illuminates the sample. The value of the beam crossfire component

`_diffrn_radiation.div_y_source` is the mean deviation in degrees of the X-ray beam from being parallel to the Y axis as it illuminates the sample. The value of the beam crossfire component

`_diffrn_radiation.div_x_y_source` is the correlation of the X and Y components. The value of the normal component of the polarization

`_diffrn_radiation.polarizn_source_norm` is the angle in degrees, as viewed from the specimen, between the normal to the polarization plane and the laboratory Y axis as defined in the AXIS category. The dimensionless value of `_diffrn_radiation.polarisn_ratio` is the ratio $(I_p - I_n)/(I_p + I_n)$, where I_n is the intensity (amplitude squared) of the electric vector of the illumination of the sample normal to the polarization and I_p is the intensity of the electric vector of the illumination of the sample in the plane of polarization. With suitable choices of laboratory axes, the definitions conform to synchrotron conventions. See Chapter **oITVG-**4.6 for a detailed description of these items.

`_diffrn_radiation.polarizn_Stokes_I` is $I_p + I_n + I_{np}$, where I_p is the intensity (amplitude squared) of the electric vector in the plane of polarization, I_n is the intensity (amplitude squared) of the electric vector in the plane of the normal to the plane of polarization, and I_{np} is the intensity (amplitude squared) of the non-polarized (incoherent) electric vector. This is an average or other representative sample of the scan. This is the first of the Stokes polarization parameters, I, Q, U, V (also known as I, M, C, S). See (Berry & Gabrielse 1977) If the absolute intensity is not known, the value 1. is assumed for I, and all 4 Stokes parameters are dimensionless. When the absolute intensity is known,all 4 Stokes parameters are in units of Watts per square meter. Note that, if the polarized intensity $I_p + I_n$ is required, $(I_p + I_n)^2$ is the sum of $Q^2 + U^2 + V^2$.

`_diffrn_radiation.polarizn_Stokes_Q` is $(I_p - I_n)\cos(2*\theta)$, where I_p is the intensity (amplitude squared) of the electric vector in the plane of polarization, I_n is the intensity (amplitude squared) of the electric vector in the plane of the normal to the plane of polarization, and θ is the angle as viewed from the specimen, between the normal to the polarization plane and the laboratory Y axis as defined in the AXIS category.

`_diffrn_radiation.polarizn_Stokes_U` is $(I_p - I_n)*\sin(2*\theta)$.

`_diffrn_radiation.polarizn_Stokes_V` is $\pm 2 * \sqrt{I_p I_n}$, with a + sign for right-handed circular polarization.

2.10.4.5. Intensity measurements

Data items in this category are as follows:

DIFFRN_REFLN
- ● _diffrn_refln.frame_id
  - → _diffrn_data_frame.id
- ● *_diffrn_refln.id*
- ● *_diffrn_refln.diffrn_id*
- *_diffrn_refln.angle_chi*
- *_diffrn_refln.angle_kappa*
- *_diffrn_refln.angle_omega*
- *_diffrn_refln.angle_phi*
- *_diffrn_refln.angle_psi*
- *_diffrn_refln.angle_theta*
- *_diffrn_refln.attenuator_code*
- *_diffrn_refln.counts_bg_1*
- *_diffrn_refln.counts_bg_2*

```
  _diffrn_refln.counts_net
  _diffrn_refln.counts_peak
  _diffrn_refln.counts_total
  _diffrn_refln.detect_slit_horiz
  _diffrn_refln.detect_slit_vert
  _diffrn_refln.elapsed_time
  _diffrn_refln.index_h
  _diffrn_refln.index_k
  _diffrn_refln.index_l
  _diffrn_refln.intensity_net
  _diffrn_refln.intensity_sigma
  _diffrn_refln.scale_group_code
  _diffrn_refln.scan_mode
  _diffrn_refln.scan_mode_backgd
  _diffrn_refln.scan_rate
  _diffrn_refln.scan_time_backgd
  _diffrn_refln.scan_width
  _diffrn_refln.sint_over_lambda
  _diffrn_refln.standard_code
  _diffrn_refln.wavelength
  _diffrn_refln.wavelength_id
• _diffrn_refln.variant
        → _variant.variant
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item. Items in italics are defined in the mmCIF dictionary.

The DIFFRN_REFLN category is defined in the mmCIF dictionary (Section **oITVG-**3.6.5.2; see the detailed discussion in Section **oITVG-**3.2.2.2.2). Data items in the DIFFRN_REFLN category record details of the intensities measured in the diffraction data set identified by `_diffrn_refln.diffrn_id`. The CBF/imgCIF dictionary extends the key with `_diffrn_refln.frame_id` (a pointer to `_diffrn_data_frame.id`), so that multiple data sets may be recorded.

2.10.4.6. Diffraction scans

Data items in these categories are as follows:

(*a*) DIFFRN_SCAN
```
• _diffrn_scan.id
  _diffrn_scan.date_end
  _diffrn_scan.date_start
  _diffrn_scan.frame_id_start
        → _diffrn_data_frame.id
  _diffrn_scan.frame_id_end
        → _diffrn_data_frame.id
  _diffrn_scan.frames
  _diffrn_scan.integration_time
```

(*b*) DIFFRN_SCAN_AXIS
```
• _diffrn_scan_axis.axis_id
        → _axis.id
• _diffrn_scan_axis.scan_id
        → _diffrn_scan.id
  _diffrn_scan_axis.angle_start
  _diffrn_scan_axis.angle_range
  _diffrn_scan_axis.angle_increment
  _diffrn_scan_axis.angle_rstrt_incr
  _diffrn_scan_axis.displacement_start
  _diffrn_scan_axis.displacement_range
  _diffrn_scan_axis.displacement_increment
  _diffrn_scan_axis.displacement_rstrt_incr
• _diffrn_scan_axis.variant
        → _variant.variant
```

(*c*) DIFFRN_SCAN_FRAME
```
  _diffrn_scan_frame.date
• _diffrn_scan_frame.frame_id
        → _diffrn_data_frame.id
• _diffrn_scan_frame.scan_id
        → _diffrn_scan.id
  _diffrn_scan_frame.frame_number
  _diffrn_scan_frame.integration_time
  _diffrn_scan_frame.polarizn_Stokes_I
  _diffrn_scan_frame.polarizn_Stokes_I_esd
  _diffrn_scan_frame.polarizn_Stokes_Q
  _diffrn_scan_frame.polarizn_Stokes_Q_esd
  _diffrn_scan_frame.polarizn_Stokes_U
  _diffrn_scan_frame.polarizn_Stokes_U_esd
  _diffrn_scan_frame.polarizn_Stokes_V
  _diffrn_scan_frame.polarizn_Stokes_V_esd

• _diffrn_scan_frame.variant
        → _variant.variant
```

(*d*) DIFFRN_SCAN_FRAME_AXIS
```
• _diffrn_scan_frame_axis.axis_id
        → _axis.id
• _diffrn_scan_frame_axis.frame_id
        → _diffrn_data_frame.id
  _diffrn_scan_frame_axis.angle
  _diffrn_scan_frame_axis.angle_increment
  _diffrn_scan_frame_axis.angle_rstrt_incr

  _diffrn_scan_frame_axis.displacement
  _diffrn_scan_frame_axis.displacement_increment
  _diffrn_scan_frame_axis.displacement_rstrt_incr
• _diffrn_scan_frame_axis.variant
        → _variant.variant
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item.

Data items in the DIFFRN_SCAN category describe the parameters of one or more scans, relating axis positions to frames. Each scan is uniquely identified by the value of `_diffrn_scan.id`. The data items in this category give overall information for the scan.

The detailed frame-by-frame data are given in DIFFRN_SCAN_FRAME and DIFFRN_SCAN_FRAME_AXIS. The values of `_diffrn_scan.date_start` and `*.date_end` give the starting and ending time for a scan. The original definition of the *yyyy-mm-dd* data type, which includes date and time, has been extended in the CBF/imgCIF dictionary. This allows the seconds part of the time to include an optional decimal fraction. The approximate average integration time for each step of the scan is given by the value of `_diffrn_scan.integration_time`. The scan is tied to individual frame IDs by the values of `_diffrn_scan.frame_id_start` and `*.frame_id_end`. The number of frames in the scan is given by the value of `_diffrn_scan.frames`.

Data items in the DIFFRN_SCAN_AXIS category describe the settings of axes for particular scans. Unspecified axes are assumed to be at their zero points. The vector of each axis is not given here, because it is provided in the AXIS category. By making `_diffrn_scan_axis.scan_id` and `_diffrn_scan_axis.axis_id` keys of the DIFFRN_SCAN_AXIS category, an arbitrary number of scanning and fixed axes can be specified for a scan. The value of `_diffrn_scan_axis.scan_id` (a pointer to `_diffrn_scan.id`) identifies the scan and the values of `_diffrn_scan_axis.axis_id` (a pointer to `_axis.id`) associate particular axes with that scan. The steps of each axis are specified by `*_start`, `*_range`, `*_increment` and `*_rstrt_incr` values for angles or for displacements. The `*_start` value is the setting of the relevant axis at the start of the scan. The `*_range` value is the total change in the axis setting through the scan. The `*_increment` value is the increment in the axis setting for each step of the scan. The `*_rstrt_incr` value is the increment in the axis setting after each step of the scan.

Data items in the DIFFRN_SCAN_FRAME category describe the relationship of particular frames to scans. The value of `_diffrn_scan_frame.frame_id` (a pointer to `_diffrn_ data_frame.id`) identifies the frame. The value of `_diffrn_scan_ frame.scan_id` (a pointer to `_diffrn_scan.id`) identifies the scan of which the frame is a part. Together `_diffrn_scan_frame.frame_id` and `*.scan_id` form the category key. The value of `_diffrn_scan_frame.date` gives the date and time of the start of the data collection for the frame. The value of `_diffrn_scan_frame.frame_number` gives the number of the frame (starting with 1). The value of `_diffrn_scan_frame.integration_time` gives the precise time in seconds to integrate this step of the scan.

Data items in the DIFFRN_SCAN_FRAME_AXIS category describe the settings of axes for particular frames. Unspecified axes are assumed to be at their zero points. If for any given frame non-zero values apply for any of the data items in this category, those values should be given explicitly in this category and not simply inferred from values in DIFFRN_SCAN_AXIS. Since the collection for a given frame may involve multiple axes, the frame involved is identified by the value of `_diffrn_scan_frame_axis.frame_id` (a pointer to `_diffrn_data_frame.id`) and each axis is identified by the value of `_diffrn_scan_frame_axis.axis_id` (a pointer to `_axis.id`). Together `_diffrn_scan_frame_axis.frame_id` and `*.axis_id` form the category key. If the axis is an axis of rotation, the axis settings for the frame are given by the values of `_diffrn_scan_frame_axis.angle`, `*.angle_increment` and `*.angle_rstrt_incr`. If the axis is a translation axis, the axis settings for the frame are given by the values of `_diffrn_scan_frame_axis.displacement`, `*.displacement_increment` and `*.displacement_rstrt_incr`. The integration begins at the setting given by the value of `_diffrn_scan_frame_axis.angle` or of `*.displacement`. The `*_increment` value gives the change of axis setting during the scan. At the end of the integration, the axis may need to be repositioned by an additional amount. That amount is given by `*_rstrt_incr`.

2.10.4.7. Map data

Data items in these categories are as follows:

(*a*) MAP
```
• _map.id
• _map.diffrn_id
        → _diffrn.id
• _map.entry_id
        → _entry.id
  _map.details
• _map.variant
        → _variant.variant
```

(*b*) MAP_SEGMENT
```
• _map_segment.id
• _map_segment.map_id
        → _map.id
  _map_segment.array_id
        → _array_structure.id
  _map_segment.array_section_id
        → _array_structure_list_section.id
  _map_segment.binary_id
        → _array_data.binary_id
  _map_segment.mask_array_id
        → _array_structure.id
  _map_segment.mask_binary_id
        → _array_data.binary_id
  _map_segment.mask_array_section_id
        → _array_structure_list_section.id
  _map_segment.details
• _map_segment.variant
        → _variant.variant
```

The bullet (•) indicates a category key. The arrow (→) is a reference to a parent data item.

Data items in the MAP and MAP_SEGMENT categories describe the details of maps. Maps record values of parameters, such as density, that are functions of position within a cell or are functions of orthogonal coordinates in three space. A map is composed of one or more map segments (sections or bricks) specified in the MAP_SEGMENT category.

The value of `_map_segment.array_id` identifies the array structure into which the map is organized. The value of `_map_segment.binary_id` distinguishes the particular set of data organized according to `_map_segment.array_id` in which the data values of the map are stored. The value of `_map_segment.mask_array_id`, if given, is the array structure into which the mask for the map is organized. If no value is given, then all elements of the map are valid. If a value is given, then only elements of the map for which the corresponding element of the mask is non-zero are valid. The value of `_map_segment.mask_array_id` differs from the value of `_map_segment.array_id` in order to permit the mask to be given as, say, unsigned 8-bit integers, while the map is given as a data type with more range. However, the two array structures must be aligned, using the same axes in the same order with the same displacements and increments.

2.10.4.8. Variants of data values

Data items in this category are as follows:

(*a*) VARIANT
```
• _variant.variant
• _variant.diffrn_id
        → _diffrn.id
• _variant.entry_id
        → _entry.id
  _variant.details
  _variant.role
  _variant.timestamp
  _variant.variant_of
        → _variant.variant
```

Data items in the VARIANT category describe the details about sets of variants of data items. There is sometimes a need to allow for multiple versions of the same data items in order to allow for refinements and corrections to earlier assumptions, observations and calculations. In order to allow data sets to contain more than one variant of the same information, an optional ...variant data item as a pointer to `_variant.variant` has been added to the key of every category, as an implicit data item with a null (empty) default value.

All rows in a category with the same variant value are considered to be related to one another and to all rows in other categories with the same variant value. For a given variant, all such rows are also considered to be related to all rows with a null variant value, except that a row with a null variant value is for which all other components of its key are identical to those entries in another row with a non-null variant value is not related the the rows with that non-null variant value. This behavior is similar to the convention for identifying alternate conformers in an atom list.

An optional role may be specified for a variant as the value of `_variant.role`. Possible roles are null, ”preferred”, ”raw data”, ”unsuccessful trial”.

Variants may carry an optional timestamp as the value of `_variant.timestamp`.

Variants may be related to other variants from which they were derived by the value of `_variant.variant_of`

Further details about the variant may be specified as the value of `_variant.details`.

In order to allow variant information from multiple datasets to be combined, `_variant.diffrn_id` and/or `_variant.entry_id` may be used.

We are grateful to Frances C. Bernstein, Paula Fitzgerald and Bob Sweet for their helpful comments and suggestions on the original version of this chapter in 2003 and to Frances C Bernstein for helpful comments on the revised version in 2017.

References

Aho, A. V., Hopcroft, J. E. & Ullman, J. D. (1987). *Data structures and algorithms.* Reading, MA: Addison-Wesley.

Berry H. H., Gabrielse, G. & Livingston, A. E. (1977), ”*Measurement of the Stokes parameters of Light.* Applied Optics, 16:12, 3200 – 3205.

Drenth, J. (2001). *Introduction to basic crystallography.* chapter 2.1 in Rossmann, M. G. and Arnold, E. *Crystallography of biological macromolecules*, Volume F of the IUCr’s ”International tables for crystallography”, Kluwer, Dordrecht 2001, pp 44 – 63

Leslie, A. G. W. & Powell, H. (2004). *MOSFLM v6.11.* MRC Laboratory of Molecular Biology, Hills Road, Cambridge, England. http://www.CCP4.ac.uk/dist/X-windows/Mosflm/.

Stout, G. H. & Jensen, L. H. (1989). *X-ray structure determination.* 2nd ed., Wiley, New York, 1989, 453 pp.

PROTEIN DATA BANK ATOMIC COORDINATE AND BIBLIOGRAPHIC ENTRY FORMAT DESCRIPTION. Brookhaven National Laboratory, February 1992.

APPENDIX 4: General considerations. This is an edited and updated extract of Chapter 3.1 of *International Tables for Crystallography Volume G: Definition and exchange of crystallographic data.* Dordrecht: Springer. Reproduced by permission. ©2005 International Union of Crystallography

3.1. General considerations when defining a CIF data item

BY B. MCMAHON

3.1.1. Introduction

Much of the power and usefulness of the Crystallographic Information File (CIF) arises from the existence of a comprehensive set of 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.

The existence of global dictionaries does not in any way restrict the expressive power of CIF. A CIF may contain items not in the standard dictionaries, as well as items in local dictionaries with quite idiosyncratic definitions. The choice of which items to include in a CIF depends on the capabilities of the applications that are intended to 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.

This chapter discusses the general concepts behind defining data items in CIF dictionaries. It describes how standard dictionaries may be constructed and disseminated, and also how local extensions may be built and used in ways that do not conflict with the need for community standards.

3.1.2 Informal definition procedures

Before considering the techniques for defining data items in standard globally adopted dictionaries, it is important to discuss the techniques for including information that is only of local interest in a way that does not conflict with public data names.

An author of a CIF is free to include data names for local use (*i.e.* names not intended for common use across the community). However, such local data names *must not* conflict with those defined in public dictionaries, since the data name alone identifies the meaning that one must attach to an associated data value. Some protocols and conventions exist to prevent conflict in data names when the local data name is invented or subsequently, when later public dictionaries are released.

An author may also 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. In the following, it is assumed that the author of a new data name wishes to define fully its attributes in an appropriate standard dictionary formalism.

3.1.2.1. The _[local]_ prefix

The string `_[local]_` is *reserved* as a prefix to identify data names that do 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. The forms `_[local]`*`_new_category_name.private_data_name`* and *`_existing_category_name.`*`[local]`*`_private_data_name`* should 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. Note that the initial underscore character is dropped in the second form.

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.

3.1.2.2. 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 3.1.2.1, the author's reserved prefix is simply an underscore-bounded string within the data name (*i.e.* it may not itself include an underscore character). 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 http://www.iucr.org/iucr-top/cif/spec/reserved.html. Table 3.1.2.1 gives a list of prefixes registered as of July 2017; 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 listing of a protein amino-acid sequence recorded temporarily by the Protein Data Bank before a protein structure is released, `_pdbx_prerelease_seq.seq_one_letter_code`.

3.1.3. Formal definition process

This section describes the formal system for creating public dictionaries or appending to them. It includes information on the review and approval cycles currently required by COMCIFS, which could change if these procedures are modified. The IUCr web page (http://cif.iucr.org) should be consulted for current practice. However, a short overview of the existing procedures is helpful in describing how the community can participate in extending the standard.

3.1.3.1. Dictionary maintenance groups

Each published dictionary authorized by COMCIFS has a group of specialists appointed or invited to extend and maintain the dictionary to serve the changing needs of the subdiscipline that sponsors the dictionary. Members of these dictionary

Table 3.1.2.1. *Reserved prefixes for private CIF data names*

String	Reserved for the use of
acihd	Local names used by ACI Heidelberg
aflow	*AFLOW* high-throughput density functional calculations
amcsd	Identifier for file created as entry in the *American Mineralogist* Structure Database
anbf	Australian National Beamline Facility
asd	Active Site Database
B+S	Software developers Bernstein + Sons
BplusS	Alternative to B+S for use with DDLm
bruker	Bruker *AXS* software
ccdc	Cambridge Crystallographic Data Centre
CCP4	*CCP*4 program system
cgraph	Oxford Cryosystems *Crystallographica* package
cifdic	Register of CIF dictionaries
cod	Data items added by maintainers and depositors of the quad Crystallography Open Database
crystmol	*CrystMol* package
csd	Cambridge Structural Database
dft	Data pertaining to density functional theory calculations, such as convergence criteria, precise method used etc.
ebi	European Bioinformatics Institute
edchem	Edinburgh University Chemistry Department
gsas	*GSAS* powder refinement system
gsk	Glaxo Smith Kline
H5	HDF5 related tags for CIF support of HDF5 and NeXus
iims	EBI project on integration of information about macromolecular structure
itqb	Instituto de Tecnologia Quimica e Biologica da Universidade de Lisboa, especially for molecular modelling extensions
iucr	IUCr journal use
mdb	Model Database (Glaxo)
montpellier	University of Montpellier
mpod	Material Properties Open Database
msd	EBI Molecular Structure Database Group
ndb	Nucleic Acids Database Project, Rutgers University
NIEHS	general-use local prefix for 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 prefix
pdb2cif	Additions to mmCIF used by program pdb2cif
phenix	*Phenix* software suite for the automated determination of macromolecular structures using X-ray crystallography and other methods
prop	Properties as used in the Material Properties Open Database
publcif	Local items used by the *publCIF* editor
raman	Data items for recording spectra obtained by the Raman spectroscopy technique
rayonix	Information specific to Rayonix (Mar USA) instruments
rcsb	Research Collaboratory for Structural Bioinformatics
shelx	*SHELXL* solution and refinement programs
solsa	*SOLSA* H2020 project: sonic drilling coupled with automated mineralogy and chemistry on-line-on-mine-real-time
SSAD	Prefix for Sulfur SAD Database CIFs
tcod	Theoretical Crystallography Open Database
vrf	Validation reply form (IUCr/*Acta Crystallographica* use)
wdc	Entries in the World Directory of Crystallographers
xtal	*Xtal* program system

maintenance groups (DMGs) may suggest extensions or corrigenda on their own initiative or may pass on requests for extensions from individual crystallographers. A DMG will typically debate and review any suggested amendments and produce a draft revised dictionary for approval by COMCIFS.

3.1.3.4. New dictionaries

A completely new dictionary to cover a subdiscipline not otherwise catered for may be commissioned by COMCIFS or may arise from community action, occasionally sponsored by an IUCr Commission. A working group is appointed to create the dictionary and relevant example files or software. The working group is expected to test the new dictionary extensively within its own community before submitting it to COMCIFS for initial approval. It is the responsibility of COMCIFS to check the dictionary for technical consistency and for compatibility with related dictionaries. COMCIFS may refer the dictionary back to the working group for further revisions. When the dictionary finally receives formal COMCIFS approval and is published, a dictionary maintenance group is formed to promote its further development (Section 3.1.3.1). The DMG usually includes one or more members of the initial working group and at least one voting member of COMCIFS.

3.1.6. Constructing a DDL2 dictionary

The DDL2 dictionary definition language was designed to specify a relational data model and has provision for including within a dictionary tables of relationships between data entries. Like a relational database which contains tables describing the data tables in the database, DDL2-based dictionaries contain definition blocks describing CIF categories, units and relationships as well as data items.

A DDL2 dictionary is presented as a single data block. Within this data block a number of looped lists describe properties of the dictionary as a whole, or properties and relationships shared across the items defined in the dictionary. Typically these are: the dictionary name, version identifiers and revision history; the category groupings that give structure to the items defined by the dictionary; the labels that identify closely related data items; and the physical units employed in the dictionary, their definitions in terms of base units and their interconversion factors.

Definitions of individual data items and categories are contained within save frames. While the save frames are not referenced by name in any dictionary application, they permit multiple occurrences of data definition tags within the scope of a single data block and are therefore suitable for structuring a data dictionary. It is a convention that the name of a save frame defining a category is given in capitals, and the name of a save frame for a definition of a data item is given as lower-case. For example, `save_ATOM_SITE` is the name of the save frame defining the category with the `atom_site` identifier, while `save__atom_site.details` is the name of the save frame holding the definition of the individual data name `_atom_site.details` (note how the initial underscore character of the data name is preserved following the initial `save_` string of the save-frame name).

The name of the dictionary itself (given by the data name `_dictionary.title`) is usually of the form `cif_`***`identifier`***`.dic`, where the *identifier* is a short code for the topic area of the dictionary (*e.g.* 'img' for the image dictionary, 'sym' for the symmetry dictionary).

The names themselves are formed from the category name separated by a full stop from the specific descriptor of the item.

Fig. 3.1.6.1 shows the structure of the macromolecular CIF dictionary. The ordering of the various looped lists and save frames is of no significance for machine parsing but has been selected to make it easier for someone to browse through the dictionary file. The name of the sole data block is chosen to be the same as the dictionary title string and the data block is introduced by the dictionary identification data items. The dictionary revision history has been placed at the end of the file so that someone reading through the file in sequence reaches the dictionary entries early on. Units tables are also near the end of the dictionary. Near the beginning are the lists of closely related items (called ‘subcategories’) and lists of category groupings. The body of the dictionary contains category and item definitions. Each category definition is followed by the definitions of its component data items. The ordering is alphabetic by category and then alphabetic by item name within categories.

```
data_mmcif_std.dic

    _dictionary.title           mmcif_std.dic
    _dictionary.version         2.0.07
    _dictionary.datablock_id    mmcif_std.dic
                          (a)

     loop_
    _sub_category.id
    _sub_category.description     .  .

     loop_
    _category_group_list.id
    _category_group_list.parent_id
    _category_group_list.description   .   .   .

                          (b)
  save_CATEGORY_A   .   .   .   save_
    save__category_a.item_1   .   .   .   save_
    save__category_a.item_2   .   .   .   save_
    save__category_a.item_3   .   .   .   save_

  save_CATEGORY_B   .   .   .   save_
    save__category_b.item_1   .   .   .   save_
    save__category_b.item_2   .   .   .   save_
                          (c)
     loop_
    _item_units_list.code
    _item_units_list.detail   .   .

     loop_
    _item_units_conversion.from_code
    _item_units_conversion.to_code
    _item_units_conversion.operator
    _item_units_conversion.factor   .   .   .   .
                          (d)
     loop_
    _dictionary_history.version
    _dictionary_history.update
    _dictionary_history.revision   .   .   .
                          (e)
```

Fig. 3.1.6.1. Schematic structure of the macromolecular CIF dictionary. (*a*) Dictionary identifiers. (*b*) Subcategory and category group listings. (*c*) Multiple category and item definition blocks. (*d*) Units descriptions and conversion tables. (*e*) Dictionary history.

Example 3.1.6.1. *DDL2 dictionary identification entries.*

```
data_mmcif_std.dic

    _dictionary.title           mmcif_std.dic
    _dictionary.version         2.0.07
    _dictionary.datablock_id    mmcif_std.dic

     loop_
    _dictionary_history.version
    _dictionary_history.update
    _dictionary_history.revision
   0.1.1  1993-02-11
;  Highlighted all notes with # %%%%% surrounds.
;
   .  .  .
```

Example 3.1.6.2. *DDL2 subcategories defined in the mmCIF dictionary.*

```
loop_
  _sub_category.id
  _sub_category.description
    'fractional_coordinate'
; The collection of x, y, and z components of a
  position specified with reference to unit cell
  directions.
;
    'matrix'
; The collection of elements of a matrix.
;
    'miller_index'
; The collection of h, k, and l components of the
  Miller index of a reflection.
;
    'cell_length'
; The collection of a, b, and c axis lengths of a
  unit cell.
;
    'mm_atom_site_label'
; The collection of alt id, asym id, atom id, comp
  id and seq id components of the label for a
  macromolecular atom site.
;
```

3.1.6.1. Dictionary identification

Dictionary files must contain information that unambiguously states their identity and version. The name of the data block that includes the whole content of a DDL2 dictionary is chosen by convention to be the same as the dictionary identification string given as `_dictionary.title`. This value is repeated as the value of `_dictionary.datablock_id` (see Example 3.1.6.1) for use in checking the consistency of the dictionary.

The dictionary history is also an important audit record of changes to the dictionary content. DDL2 provides a looped list of version labels, dates and annotations. For convenience, the history records in large dictionaries are placed at the end of the dictionary file.

3.1.6.2. Subcategory definitions

Mechanisms exist for formal and machine-parsable statements of relationships. The `_sub_category.id` attribute is a label shared by several data items within a category that are related in a specific way described by the associated `_sub_category.description` attribute. The relationships may be rather general, such as elements of a matrix; or they may

Example 3.1.6.3. *Category groups in a DDL2 dictionary.*

```
loop_
  _category_group_list.id
  _category_group_list.parent_id
  _category_group_list.description
    'inclusive_group'   .
; Categories that belong to the macromolecular
  dictionary.
;
    'atom_group'
    'inclusive_group'
; Categories that describe the properties of
atoms.
;
    'audit_group'
    'inclusive_group'
; Categories that describe dictionary maintenance
  and identification.
;
    'cell_group'
    'inclusive_group'
; Categories that describe the unit cell.
;
```

be specific physical properties or attributes, such as the collection of axis lengths of a unit cell. The dictionary should list all such labels that occur within its included data definition blocks. Example 3.1.6.2 is an extract from the macromolecular dictionary.

3.1.6.3. Category groupings

In the DDL2 data model, a *category* of data corresponds to a set of related data items that may be stored in a single relational database table. A number of such tables may collectively describe the complete properties of some physical object. This is expressed formally by assigning the same label (`_category_group.id`) to the relevant categories. Relationships between categories are formally stated.

For subcategories, the category-group relationships present in the dictionary are listed in a separate looped list. Example 3.1.6.3 is an extract from the macromolecular dictionary. The `inclusive_group` entry shows the common parentage of all categories (and ultimately all data items) in the dictionary.

3.1.6.4. Category definitions

The dictionary entry for a category includes the name of the category (an identifying label which is referenced by the `_item.category_id` attribute of each component data item) and a list of the category groups of which it may be considered a member. The category *key* is explicitly specified – that is, the data item (or group of items) that uniquely identifies an individual row in a table of data of that category.

Where a category encompasses a set of data items that are not normally specified in a looped list, the category may nevertheless be taken to represent a degenerate table with a single row, and therefore there is still a category key. For degenerate categories the key value is often set equal to the name of the parent data block.

Example 3.1.6.4 shows a category of non-looped core data items.

For categories of looped items (those normally presented in a table of values) it is sometimes appropriate to have as the category key a data item that has the sole function of indexing unique table rows. However, it is also often the case that a composite key is formed from existing data items, and in these cases the category definition must loop the components of the key, as in Example 3.1.6.5 from the macromolecular dictionary definition of the GEOM_BOND category.

It must be remembered that, in practice, data files may lack some of the items required to determine the category key formally. For example, in the data set given in the GEOM_BOND example here, it is possible that the `_geom_bond.site_symmetry_` items may be absent because the listing is for a single connected molecule within an asymmetric unit. Robust parsing software must construct data keys by assigning NULL or other suitable default values to the missing key components.

3.1.6.5. Data-item definitions

The bulk of a DDL2 data dictionary comprises the save frames that include descriptions of the meaning and properties of individual data names.

Note that each save frame contains the definition of a *single* addressable concept. For example, the three Miller index components of a diffraction reflection are described

Example 3.1.6.4. *A category description in a DDL2 dictionary.*

```
save_EXPTL
     _category.description
;    Data items in the EXPTL category record
     details about the growth of the crystal,
     and about experimental measurements on
     the crystal, such as shape, size, density,
     and so on.
;
     _category.id                  exptl
     _category.mandatory_code      no
     _category_key.name           '_exptl.entry_id'
      loop_
     _category_group.id            'inclusive_group'
                                   'exptl_group'
      loop_
     _category_examples.detail
     _category_examples.case
# - - - - - - - - - - - - - - - - - - - - - - - - -
;  Example 1 - based on laboratory records for
   Yb(S-C5H4N)2 (THF)4
;
; _exptl.entry_id                   datablock1
  _exptl.absorpt_coefficient_mu     1.22
  _exptl.absorpt_correction_T_max   0.896
  _exptl.absorpt_correction_T_min   0.802
  _exptl.absorpt_correction_type    integration
  _exptl.absorpt_process_details
  ; Gaussian grid method from SHELX76
    Sheldrick, G. M., "SHELX-76: structure
    determination and refinement program",
    Cambridge University, UK, 1976
  ;
  _exptl.crystals_number            1
  _exptl.details
  ; Enraf-Nonius LT2 liquid nitrogen
    variable-temperature device used
  ;
  _exptl.method 'single-crystal x-ray diffraction'
  _exptl.method_details
  ; graphite monochromatized Cu K(alpha) fixed
    tube and Enraf-Nonius CAD4 diffractometer used
  ;
;
# - - - - - - - - - - - - - - - - - - - - - - - - -
save_
```

Example 3.1.6.5. *A DDL2 category with a composite key.*

```
save_GEOM_BOND
    _category.description
;   Data items in the GEOM_BOND category record
    details about molecular and crystal bonds, as
    calculated from the contents of the ATOM,
    CELL, and SYMMETRY data.
;
    _category.id                  geom_bond
    _category.mandatory_code      no
     loop_
    _category_key.name '_geom_bond.atom_site_id_1'
                       '_geom_bond.atom_site_id_2'
                       '_geom_bond.site_symmetry_1'
                       '_geom_bond.site_symmetry_2'
     loop_
    _category_group.id           'inclusive_group'
                                 'geom_group'
     loop_
    _category_examples.detail
    _category_examples.case
# - - - - - - - - - - - - - - - - - - - - - - - - -
- -
;  Example 1 - based on data set TOZ of Willis,
   Beckwith & Tozer [(1991). Acta Cryst. C47,
   2276-2277].
;
;
     loop_
    _geom_bond.atom_site_id_1
    _geom_bond.atom_site_id_2
    _geom_bond.dist
    _geom_bond.site_symmetry_1
    _geom_bond.site_symmetry_2
    _geom_bond.publ_flag
      O1  C2   1.342(4)  1_555  1_555  yes
      O1  C5   1.439(3)  1_555  1_555  yes
      C2  C3   1.512(4)  1_555  1_555  yes
      C2  O21  1.199(4)  1_555  1_555  yes
      C3  N4   1.465(3)  1_555  1_555  yes
      C3  C31  1.537(4)  1_555  1_555  yes
      C3  H3   1.00(3)   1_555  1_555  ?
      N4  C5   1.472(3)  1_555  1_555  yes
    # - - - - data truncated for brevity - - - -
;
# - - - - - - - - - - - - - - - - - - - - - - - - -
- -
save_
```

in a DDL2 dictionary in three separate save frames, `save__diffrn_refln.index_h`, `save__diffrn_refln.index_k` and `save__diffrn_refln.index_l`. The intimate relationship between these three components is expressed through the common `_item_sub_category.id` value of `miller_index` and the mutual reference of the other Miller-index components by the `_item_dependent.dependent_name` entries in each separate save frame.

An apparent exception to this general rule is the case of save frames defining an item, often a category key, that is an identifier common to several categories. In this case, the save frame defining the 'parent' identifier implicitly defines the complete property set of each child identifier. For completeness, the respective child identifiers are each declared in their own save frames, but these act only as back references to the parent definition. This is explained more completely in Section 3.1.6.5.1 below.

3.1.6.5.1. Inheritance of identifiers

Example 3.1.6.6 is from an mmCIF of two related categories that describe characteristics of an active site in a macromolecular complex. The sites are described in general terms with a label and textual description in the STRUCT_SITE category (the first looped list in the example). Details of how each site is generated from a list of structural features form the STRUCT_SITE_GEN category (second loop or table).

It is clear that each instance of the data item `_struct_site_gen.site_id` in the second table must have one of the values listed as `_struct_site.id` in the first loop, because it is the purpose of these identifiers to relate the two sets of data: they are the glue between the two separate tables and must have the same values to ensure the referential integrity of the data set (that is, the consistency and completeness of cross-references between tables). Within a group of related categories like this, it is normal to consider one as the 'parent' and the others as 'children'.

Because all such linking data items must have compatible attributes, it is conventional in DDL2 dictionaries to define all the attributes in a single location, namely the save frame which hosts the definition of the 'parent' data item. In early drafts of DDL2 dictionaries, the 'children' were not referenced at all in separate save frames; software validating a data file against a dictionary was required to obtain all information about a child identifier from the contents of the save frame defining the parent. However, subsequent drafts introduced a minimal save frame for the children to accommodate dictionary browsers that depended on the existence of a separate definition block for each individual data item.

Consequently, the definition blocks in current DDL2 dictionaries conform to the structure in Example 3.1.6.7, which refers to the simple STRUCT_SITE example used above.

Example 3.1.6.6. *Illustration of parent/child relationships between identifiers in related categories.*

```
loop_
  _struct_site.id
  _struct_site.details
    'P2 site C'
; residues with a contact < 3.7 Angstrom to an
atom
  in the P2 moiety of the inhibitor in the
  conformation with _struct_asym.id = C
;
    'P2 site D'
; residues with a contact < 3.7 Angstrom to an
atom
  in the P1 moiety of the inhibitor in the
  conformation with _struct_asym.id = D
;

loop_
  _struct_site_gen.id
  _struct_site_gen.site_id
  _struct_site_gen.label_comp_id
  _struct_site_gen.label_asym_id
  _struct_site_gen.label_seq_id
  _struct_site_gen.symmetry
  _struct_site_gen.details
    1  'P2 site C'  VAL  A   32  1_555  .
    2  'P2 site C'  ILE  A   47  1_555  .
    3  'P2 site C'  VAL  A   82  1_555  .
    4  'P2 site C'  ILE  A   84  1_555  .
    5  'P2 site D'  VAL  B  232  1_555  .
    6  'P2 site D'  ILE  B  247  1_555  .
    7  'P2 site D'  VAL  B  282  1_555  .
    8  'P2 site D'  ILE  B  284  1_555  .
```

Example 3.1.6.7. *A definition of an identifier which is parent to identifiers in other categories.*

```
save__struct_site.id
   _item_description.description
;     The value of _struct_site.id must uniquely
      identify a record in the STRUCT_SITE list.

      Note that this item need not be a number;
      it can be any unique identifier.
;
  loop_
    _item.name
    _item.category_id
    _item.mandatory_code
'_struct_site.id'          struct_site
yes
'_struct_site_gen.site_id'  struct_site_gen
yes
'_struct_site_keywords.site_id'
                           struct_site_keywords
yes
'_struct_site_view.site_id' struct_site_view
yes
  loop_
    _item_linked.child_name
    _item_linked.parent_name
'_struct_site_gen.site_id'       '_struct_site.id'
'_struct_site_keywords.site_id'  '_struct_site.id'
'_struct_site_view.site_id'      '_struct_site.id'

   _item_type.code               line
save_
```

Example 3.1.6.8. *Definition of a child identifier.*

```
save__struct_site_gen.id
   _item_description.description
;     The value of _struct_site_gen.id must
uniquely
      identify a record in the STRUCT_SITE_GEN
list.

      Note that this item need not be a number;
      it can be any unique identifier.
;
  _item.name                  '_struct_site_gen.id'
  _item.category_id             struct_site_gen
  _item.mandatory_code          yes
  _item_type.code               line

save_
```

Note that the dependent data names are listed twice: once in the loop that declares their `_item.name` values and the categories with which they are associated; and again in a loop that makes the direction of the relationship explicit. A parent data item may have several children, but each child can have only a single parent (*i.e.* related data name whose value may be checked for referential integrity). Note also that each listed item has an `_item.mandatory_code` value of `yes`: because they are identifiers which link categories, they must be present in a table to allow the relationships between data items in different tables to be traced.

Other than the specific description text field, any declared attributes (in this example only the data type) have a common value across the set of related identifiers.

As mentioned above, it is not formally necessary to have a separate save frame for the individual children; but it is conventional to have such individual save frames containing minimal definitions that serve as back-references to the primary information in the parent frame. These also provide somewhere for the specific text definitions for the children to be stored. The definition frame for `_struct_site_gen.id` is shown in Example 3.1.6.8.

3.1.6.5.2. Definitions of single quantities

While it is important to ensure the referential integrity of the data in a CIF through proper book-keeping of links between tables, the crystallographer who wishes to create or extend a CIF dictionary will be more interested in the definitions of data items that refer to real physical quantities, the properties of a crystal or the details of the experiment. The DDL2 formalism makes it easy to create a detailed machine-readable listing of the attributes of such data.

Example 3.1.6.9 shows a definition of the ambient temperature during the experiment. In the definition save frame, the category is specifically listed (although it is deducible from the convention of separating the category name from the rest of the name by a full stop in the data name). The data type is specified as a floating-point number. The range of values is also specified with separate maximum and minimum values The assignment of the same value to a maximum and a minimum means that the absolute value is permitted; without the repeated '0.0' line the range in this example would be constrained to be positive definite; the equal value of 0.0 for maximum and minimum means that it may be identically zero.

The `_item_units.code` value must be one of the entries in the units table for the dictionary and can thus be converted into other units as specified in the units conversion table.

The aliases entries identify equivalent names for the corresponding quantity defined in other dictionaries.

3.1.6.6. Units

The physical unit associated with a quantitative value in a DDL2-based file is specified in the relevant dictionary. There is no option to express the quantity in other units. However, DDL2 permits a dictionary file to store not only a table of the units referred to in the dictionary (listed under

Example 3.1.6.9. *DDL2 definition of a physical quantity.*

```
save__diffrn.ambient_temp
    _item_description.description
;       The mean temperature in kelvins at which
        the intensities were measured.
;
    _item.name               '_diffrn.ambient_temp'
    _item.category_id             diffrn
    _item.mandatory_code          no
    _item_aliases.alias_name
                  '_diffrn_ambient_temperature'
    _item_aliases.dictionary      cif_core.dic
    _item_aliases.version         2.0.1
     loop_
    _item_range.maximum
    _item_range.minimum            .     0.0
                                  0.0    0.0
    _item_related.related_name
                       '_diffrn.ambient_temp_esd'
    _item_related.function_code   associated_esd
    _item_type.code               float
    _item_type_conditions.code    esd
    _item_units.code              kelvins
save_
```

_item_units_list.code and the accompanying descriptive item _item_units_list.detail), but also a table specifying the conversion factors between individual codes in the _item_units_list.code list. In principle this allows a program to combine or otherwise manipulate different physical quantities while handling the units properly.

3.1.7. Composing new data definitions

Preceding sections have described the framework within which CIF dictionaries exist and are used, and their individual formal structures. While this is important for presenting the definition of new data items, it does not address what is often the most difficult question: what quantities, concepts or relationships merit separate data items? On the one hand, the extensibility of CIF provides great freedom of choice: anything that can be characterized as a separate idea may be assigned a new data name and set of attributes. On the other hand, there are practical constraints on designing software to write and read a format that is boundless in principle, and some care must be taken to organize new definitions economically and in an ordered way.

3.1.7. Granularity

Perhaps the most obvious decision that needs to be made is the level of detail or granularity chosen to describe the topic of interest. CIF data items may be very specific (the deadtime in microseconds of the detector used to measure diffraction intensities in an experiment) or very general (the text of a scientific paper). In general, a data name should correspond to a single well defined quantity or concept within the area of interest of a particular application. It can be seen that the level of granularity is determined by the requirements of the end application.

A practical example of determining an appropriate level of granularity is given by the core dictionary definitions for bibliographic references cited in a CIF. The dictionary originally contained a single character field, _publ.section_references, which was intended to contain the complete reference list for an article as undifferentiated text. *Notes for Authors* in journals accepting articles in CIF format advised authors to separate the references within the field with blank lines, but otherwise no structure was imposed upon the field. In a subsequent revision to the core dictionary, the much richer CITATION category was introduced to allow the structured presentation of references to journal articles and chapters of books. This was intended to aid queries to bibliographic databases. However, a full structured markup of references with multiple authors or editors in CIF requires additional categories, so that the details of the reference may be spread across three tables corresponding to the CITATION, CITATION_AUTHOR and CITATION_EDITOR categories. Populating several disjoint tables greatly complicates the author's task of writing a reference list. Moreover, the CITATION category does not yet cover all the many different types of bibliographic reference that it is possible to specify, and is therefore suitable only for references to journal articles and chapters of books. However, it is possible to write a program that can deduce the structure of a standard reference within an undifferentiated reference list (provided the journal guidelines have been followed by the author) to the extent that enough information can be extracted to add hyperlinks to references using a cross-publisher reference linking service such as CrossRef (CrossRef, 2004). Therefore, in practice, IUCr journals still ask the author of an article to supply their reference list in the _publ.section_references field, rather than using the apparently more useful _citation_ fields. It remains to be seen whether this is the best strategy in the long term.

In more technical topic areas, the details of an experimental instrument could be described by a huge number of possible data names, ranging from the manufacturer's serial number to the colour of the instrument casing. However, many of these details are irrelevant to the analysis of the data generated by the instrument, so the characteristics of an instrument that are assigned individual data names are typically just those parameters that need to be entered in equations describing the calibration or interpretation of the data it generates.

3.1.7.2. Category 'special details' fields

When the specific items in a particular topic area that need to be recorded under their own data names have been decided, there is likely to be other information that could be recorded, but is felt to be irrelevant to the immediate purposes of the data collection and analysis. It is good practice to provide a place in the CIF for such additional information; it encourages an author to record the infomation and permits data mining at a later stage. Each category typically contains a data name with the suffix _details (or _special_details) which identifies a text field in which additional information relating to the category may be stored. This field often contains explanatory text qualifying the information recorded elsewhere in the same category, but it might contain additional specific items of information for which no data name is given and for which no obvious application is envisaged. This helps to guard against the loss of information that might be put to good use in the future. Of course, if a *_details field is regularly used to store some specific item of information *and* this information is seen to be valuable in the analysis or interpretation of data elsewhere in the file, there is a case for defining a new, separate tag for this information.

3.1.7.3. Construction of data names

Since a dictionary definition contains all the machine-readable attributes necessary for validating the contents of a data field, the data name itself may be an arbitrary tag, devoid of semantic content. However, while dictionary-driven access to a CIF is useful in many cases, there are circumstances where it is useful to browse the file. It is therefore helpful to construct a data name in a way that gives a good indication of the quantity described. From the beginning, CIF data names have been constructed from self-descriptive components in an order that reflects the hierarchical relationship of the component ideas, from highest (most general) level to lowest (most specific) level when read from left to right.

In a typical example from the core CIF dictionary, the data name _atom_site.type_symbol defines a code (symbol) indicating the chemical nature (type) of the occupant of a location in the crystal lattice (atom_site). The full stop (.) separates the *category* to which the data name belongs from its more specific qualifiers.

However, it may not always be easy to establish the best order of components when constructing a new data

```
_database.code_CSD                    'VOBYUG'
                        (a)

_database_2.database_id               'PDB'
_database_2.database_code             '5HVP'
                        (b)
```

Fig. 3.1.7.1. Alternative quantities described (*a*) by data-name extension (core dictionary) or (*b*) by paired data names (mmCIF dictionary).

name. In the JOURNAL category, there was initially some uncertainty about whether to associate the telephone numbers of different contact persons by appending codes such as `_coeditor` and `_techeditor` to a common base name. In the end, the order of components was reversed to give names like `_journal.coeditor_phone` and `_journal.techeditor_phone`. Examining the JOURNAL category in the core CIF dictionary will show why this was done. Similarly, the extension of geometry categories to include details of hydrogen bonding went through a stage of discussing adding new data names to the existing categories, but with suffixes indicating that the components were participating in hydrogen bonding, before it was decided that a completely new category for describing all elements of a hydrogen bond was justified. These examples show that the correct ordering of components within a data name is closely related to the perceived classification of data names by category and subcategory.

Sometimes it is useful to differentiate alternative data items by appending a suffix to a root data name. For example, the core dictionary defines several data names for recording the reference codes associated with a data block by different databases: `_database.code_CAS`, `_database.code_CSD` *etc.* This is convenient where there are two or three alternatives, but becomes unwieldy when the number of possibilities increases, because new data names need to be defined for each new alternative case. A better solution is to have a single base name and a companion data item that defines which of the available alternatives the base item refers to. The mmCIF dictionary follows this principle: the category DATABASE_2 contains two data names, `_database_2.database_code` (the value of which is an assigned database code) and `_database_2.database_id` (the value of which identifies which of the possible databases assigned the code) (Fig. 3.1.7.1).

Note the distinction between a data name constructed with a suffix indicating a particular database, and a data name which incorporates a prefix registered for the private use of a database. The data name `_database.code_PDB` is a *public* data name specifying an entry in the Protein Data Bank, while `_pdb_database.code` is a *private* data name used for some internal purpose by the Protein Data Bank.

3.1.7.4. Parsable data values *versus* separate data names

An advantage of defining multiple data names for the individual components of a complicated quantity is that there is no ambiguity in resolving the separate components. Hence the Miller indices of a reflection in the list of diffraction measurements are specified in the core dictionary by the group of three data names `_diffrn_refln.index_h`, `_diffrn_refln.index_k` and `_diffrn_refln.index_l`. In principle, a single data name associated with the group of three values in some well defined format (*e.g.* comma separated, as h, k, l) could have been defined instead. However, this would require a parser to understand the internal structure of the value so that it could parse out the separate values for h, k and l.

On the other hand, there are many examples of data values that are stored as string values parsable into distinct components. An extreme example is the reference list mentioned in Section 3.1.7.1. More common are dates (`_audit.creation_date`), chemical formulae (*e.g.* `_chemical_formula.moiety`), symmetry operations (`_symmetry_equiv.pos_as_xyz`) or symmetry transformation codes (`_geom_bond.site_symmetry_1`). There is no definitive answer as to which approach is preferred in a specific case. In general, the separation of the components of a compound value is preferred when a known application will make use of the separate components individually. For instance, applications may list structure factors according to a number of ordering conventions on individual Miller indices. As an extreme example of separating the components of a compound value, the mmCIF dictionary defines data names for the standard uncertainty values of most of the measurable quantities it describes, while the core dictionary just uses the convention that a standard uncertainty is specified by appending an integer in parentheses to a numeric value.

A related problem is how to handle data names that describe an indeterminate number of parameters. For example, in the modulated structures dictionary an extra eight Miller indices are defined to span a reciprocal space of dimension up to 11. In principle, the dimensionality could be extended without limit. According to the practice of defining a unique data name for each modulation dimension, new data names would need to be defined as required to describe higher-dimensional systems. Beyond a certain point this will become unwieldy, as will the set of data names required to describe the n^2 components of the W matrix for a modulated structure of dimensionality n (`_cell_subsystem.matrix_W_1_1` *etc.*).

The modulated structures dictionary was constrained to define extended Miller indices in this way for compatibility with the core dictionary. Data names describing new quantities that are subject to similar unbounded extensibility should perhaps refer to values that are parsable into vector or matrix components of arbitrary dimension.

3.1.7.5. Consistency of abbreviations

One further consideration when constructing a data name is the use of consistent abbreviations within the components of the data name. This is of course a matter of style, since if a data name is fully defined in a dictionary with a machine-readable attribute set, the data name itself can be anything. Nonetheless, to help to find and group similar data names it is best to avoid too many different abbreviations.

Table 3.1.7.1 lists the abbreviations used in the current public dictionaries. Note that there are already cases where different abbreviations are used for the same term.

3.1.8. Management of multiple dictionaries

So far this chapter has discussed the mechanics of writing dictionary definitions and of assembling a collection of definitions

Table 3.1.7.1. *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	org	organism
absorpt	absorption		standard deviation) (*see* su)	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	I	intensity	prof	profile
	parameter (a.d.p.)	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	Rmerge	agreement index of merging
clust	cluster		Applied Chemistry	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 θ
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	suppl	supplementary
diff	difference*		National Institute of Standards and	sys	systematic
diffr	diffractometer		Technology)	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	dead time	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.

in a single global or local dictionary file. In practice, the set of data names in a CIF data file may include names defined in several dictionary files. A mechanism is required to identify and locate the dictionaries relevant to an individual data file. In addition, because dictionaries are suitable for automated validation of the contents of a data file, it is convenient to be able to overlay the attributes listed in a dictionary with an alternative set that permit validation against modified local criteria. This section describes protocols for identifying, locating and overlaying dictionary files and fragments of dictionary files.

3.1.8.1. Identification of dictionaries relevant to a data file

A CIF data file should declare within each of its data blocks the names, version numbers and, where appropriate, locations of the global and local dictionaries that contain definitions of the data names used in that block. The relevant identifiers are the items **`_audit_conform.dict_name`**, **`_audit_conform.dict_version`** and **`_audit_conform.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 registry, as described in Section 3.1.8.2. 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   .
    cif_pd.dic        1.0   .
    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.

3.1.8.2. The dictionary register

COMCIFS maintains a register of dictionaries known to it, including the identifying name and version strings within those dictionaries. The register also includes the location of each dictionary, expressed at present as a URL designed to allow retrieval by file transfer protocol (ftp) from the IUCr server. Changes in the location of a particular dictionary file can be made by modifying the entry in the register, avoiding the problem of specifying a URL in a data file that would then become outdated if the dictionary was moved. Dictionary applications can consult the register (according to a protocol outlined below) to locate and retrieve the dictionaries needed for validating data files. It is of course essential that the validation software knows how to locate the register. The location is at present given by the URL ftp://ftp.iucr.org/pub/cifdics/cifdic.register. The problem of changing URLs has therefore not disappeared completely, but is at least confined to the need to maintain one single address.

Table 3.1.8.1 is an extract from the current register (the complete version includes contact details for the maintainer of each dictionary). The latest version of the register will always be available from the URL given above.

The entries for each dictionary include one with the version string set to '`.`', representing the current version; this is the version that should be retrieved unless a data file specifies otherwise.

Note that the register may also contain locators for local dictionaries constructed by owners of reserved prefixes (Section 3.1.2.2) when the owner has requested that a dictionary of local names be made publicly available. An appropriate name for a local dictionary in the register (`_dictionary.title`) would be `cif_local_`***`myprefix`***`.dic`, where the string indicated by *myprefix* is one of the prefixes reserved for private use by the author of the dictionary (see Section 3.1.2.2). This scheme complements the naming convention for public dictionaries.

3.1.8.3. 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*). `*_version` (*V*) and `*_location` (*L*) are optional. In the event that no dictionaries are specified, the default validation dictionary should be that identified as having *N* = `cif_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

Table 3.1.8.1. *CIF dictionary register (maintained as a STAR File), as current in March 2005*

```
data_validation_dictionaries
  loop_
    _cifdic_dictionary.name
    _cifdic_dictionary.version
    _cifdic_dictionary.DDL_compliance
    _cifdic_dictionary.reserved_prefix
    _cifdic_dictionary.URL
    _cifdic_dictionary.description
  cif_core.dic    .    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_core.dic
      'Core CIF Dictionary'
  cif_core.dic   1.0   1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cifdic.C91
      'Original Core CIF Dictionary'
  cif_core.dic  2.0.1  1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_core_2.0.1.dic
      'Core CIF Dictionary'
  cif_core.dic  2.1    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_core_2.1.dic
      'Core CIF Dictionary'
  cif_core.dic  2.2    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_core_2.2.dic
      'Core CIF Dictionary'
  cif_core.dic  2.3    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_core_2.3.dic
      'Core CIF Dictionary'
  cif_pd.dic      .    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_pd.dic
      'Powder CIF Dictionary'
  cif_pd.dic     1.0   1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_pd_1.0.dic
      'Powder CIF Dictionary'
  cif_ms.dic      .    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_ms.dic
      'Modulated structures CIF Dictionary'
  cif_ms.dic     1.0   1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_ms_1.0.dic
      'Modulated structures CIF Dictionary'
  cif_rho.dic     .     1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_rho.dic
      'Modulated structures CIF Dictionary'
  cif_rho.dic    1.0    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_rho_1.0.dic
      'Electron density CIF Dictionary'
  cif_mm.dic      .    2.1.2  .
      ftp://ftp.iucr.org/pub/cifdics/cif_mm.dic
      'Macromolecular CIF Dictionary'
  cif_mm.dic     1.0   2.1.2  .
      ftp://ftp.iucr.org/pub/cifdics/cif_mm_1.0.dic
      'Macromolecular CIF Dictionary'
  mmcif_std.dic    2.0.7   2.1.2  .
      ftp://ftp.iucr.org/pub/cifdics/mmcif_std.dic
      'Macromolecular CIF Dictionary'
  cif_img.dic     .    2.1.2  .
      ftp://ftp.iucr.org/pub/cifdics/cif_img.dic
      'Image CIF Dictionary'
  cif_img.dic    1.0   2.1.2  .
      ftp://ftp.iucr.org/pub/cifdics/cif_img_1.0.dic
      'Image CIF Dictionary'
  cif_img.dic    1.3.1   2.1.2  .
      ftp://ftp.iucr.org/pub/cifdics/cif_img_1.3.1.dic
      'Image CIF Dictionary'
  cif_sym.dic     .    2.1.2  .
      ftp://ftp.iucr.org/pub/cifdics/cif_sym.dic
      'Symmetry CIF Dictionary'
  cif_sym.dic    1.0   2.1.2  .
      ftp://ftp.iucr.org/pub/cifdics/cif_sym_1.0.dic
      'Symmetry CIF Dictionary'
  cif_compat.dic   1.0   1.4  .
      ftp://ftp.iucr.org/pub/cifdics/cif_compat_1.0.dic
      'Legacy CIF Dictionary of deprecated terms'
```

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.

Reference

CrossRef (2004). *Query spec.* http://www.crossref.org/03libraries/25query_spec.html.

About our sponsors

We acknowledge the generosity of corporate sponsors who have made possible this Workshop and have contributed to other activities of the IUCr related to data standardisation. The Workshop focusses on the creation of data dictionaries and domain ontologies in the fields of crystallography and related structural sciences (though the approach could be extended to any discipline). Particular attention is addressed to CIF dictionaries, which are based on dictionary definition languages that powerfully express data relationships and can support computational methods.

The **International Union of Crystallography** is an International Scientific Union. Its objectives are to promote international cooperation in crystallography and to contribute to all aspects of crystallography, to promote international publication of crystallographic research, to facilitate standardization of methods, units, nomenclatures and symbols, and to form a focus for the relations of crystallography to other sciences.

Australian Government

The **Australian Nuclear Science and Technology Organisation (ANSTO)** is an Australian public research organisation responsible for delivering specialised scientific services and products. As operator of both the OPAL multipurpose reactor and the Australian Synchrotron, ANSTO brings two of Australia's most significant pieces of landmark infrastructure together. ANSTO is committed to delivering best-practice research data management across the entire breadth of its instrument portfolio.

The **International Centre for Diffraction Data** is a non-profit scientific organization dedicated to collecting, editing, publishing, and distributing powder diffraction data for the identification of materials. The membership of the ICDD consists of worldwide representation from academe, government, and industry. Our Vision: The International Centre for Diffraction Data will continue to develop tools and support the education required for materials analyses of tomorrow. Our Mission: The International Centre for Diffraction Data will continue to be the world center for quality diffraction and related data to meet the needs of the technical community. ICDD promotes the application of materials characterization methods in science and technology by providing forums for the exchange of ideas and information.

Wiley's Scientific, Technical, Medical, and Scholarly (STMS) business, also known as Wiley-Blackwell, serves the world's research and scholarly communities, and is the largest publisher for professional and scholarly societies. Wiley-Blackwell's programs encompass journals, books, major reference works, databases, and laboratory manuals, offered in print and electronically. Through Wiley Online Library, we provide online access to a broad range of STMS content: over 4 million articles from 1,500 journals, 9,000+ books, and many reference works and databases. Access to abstracts and searching is free, full content is accessible through licensing agreements, and large portions of the content are provided free or at nominal cost to nations in the developing world through partnerships with organizations such as HINARI, AGORA, and OARE.

Bruker Corporation has been driven by the idea to always provide the best technological solution for each analytical task for more than 50 years.

Today, worldwide more than 6,000 employees are working on this permanent challenge at over 90 locations on all continents. Bruker systems cover a broad spectrum of applications in all fields of research and development and are used in all industrial production processes for the purpose of ensuring quality and process reliability.

Bruker continues to build upon its extensive range of products and solutions, its broad base of installed systems and a strong reputation among its customers. Being one of the world's leading analytical instrumentation companies, Bruker is strongly committed to further fully meet its customers' needs as well as to continue to develop state-of-the-art technologies and innovative solutions for today's analytical questions.

The **Worldwide Protein Data Bank** (wwPDB) consists of organizations that act as deposition, data processing and distribution centers for Protein Data Bank (PDB) data. Members are: RCSB PDB (USA), PDBe (Europe) and PDBj (Japan), and BMRB (USA). The wwPDB's mission is to maintain a single PDB archive of macromolecular structural data that is freely and publicly available to the global community.

CODATA, the Committee on Data for Science and Technology, is an interdisciplinary Scientific Committee of the International Council for Science (ICSU), established in 1966 to promote and encourage, on a world-wide basis, the compilation, evaluation and dissemination of reliable numerical data of importance to science and technology.

The mission of CODATA is to strengthen international science for the benefit of society by promoting improved scientific and technical data management and use.

It works to improve the quality, reliability, management and accessibility of data of importance to all fields of science and technology. CODATA provides scientists and engineers with access to international data activities for increased awareness, direct cooperation and new knowledge. It is concerned with all types of data resulting from experimental measurements, observations and calculations in every field of science and technology, including the physical sciences, biology, geology, astronomy, engineering, environmental science, ecology and others. Particular emphasis is given to data management problems common to different disciplines and to data used outside the field in which they were generated.