*International Tables for Crystallography* (2006). Vol. G, Chapter 3.1, pp. 73–91.

# 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 information contained in a CIF is expressed in terms of these data items. A data item consists of a value associated with a data name, or tag. The tag may appear immediately before a single data value or in the heading of a looped list where the values form a column. In either construction, the data value is identified by the tag and this unique character string is the key to the definition of the data value in the dictionary.

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

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. Of course, the same applies to data in XML (Bray *et al.*, 1998; W3C, 2004) or other data languages. In the adoption and application of CIF as a specific exchange mechanism, the crystallographic community has imposed on itself a particular discipline: the strict definition of its data with carefully maintained dictionaries. This is not to be seen as a restriction but as a means to unambiguous and effective communication.

As mentioned above, data with local definitions are easily accommodated in a CIF. However, for a CIF to be an effective exchange medium, data definitions need to be accessible to the community of users. This is most efficient when commonly used data items are collected into a dictionary or dictionaries that are readily obtainable and centrally coordinated. This is why the CIF dictionaries, containing the definitions of standard data names and their attributes, are published and maintained by a technical committee of the International Union of Crystallography (IUCr): the Committee for the Maintenance of the CIF Standard (COMCIFS). The dictionaries employ a dictionary definition language or DDL (see Chapters 2.5 and 2.6) to describe relevant attributes of CIF data items.

Affiliation: BRIAN MCMAHON, International Union of Crystallography, 5 Abbey Square, Chester CH1 2HU, England.

This chapter will discuss the general concepts behind defining data items in CIF dictionaries. It will describe 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. Some necessary details about the administration of standard dictionaries are also provided.

### 3.1.1.1. Authorship of data dictionaries

A difficulty in developing a standard for information exchange across the field of crystallography is the breadth of the subject area and the many subdisciplines it includes. One feature of the construction of data dictionaries for CIF is the delegation of responsibility for identifying and defining the data items important within a research area to experts in that field. In consequence, a richer compilation of definitions results than would be possible from a single author or small group of authors. However, each subdiscipline will have its own emphases and requirements, and it becomes a challenge to accommodate the needs of each individual subdiscipline within the framework of the general body of definitions covering the entire subject area. COMCIFS deals with this challenge by initiating and ratifying dictionaries written by IUCr Commissions or other specialist groups.

### 3.1.1.2. Certification for community use

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

### 3.1.1.3. DDL versions

Ideally, compatibility between the data dictionaries originating from specific subdisciplines would be ensured by the adoption of the same attribute sets for data items. However, at this point in the evolution of the CIF standard, two slightly different attribute sets have become established. These are expressed in two versions of the dictionary definition language, DDL1 and DDL2 (detailed in Chapters 2.5 and 2.6, respectively). The differences arise because some subdisciplines benefit from a strict data model that is not appropriate in other areas. The core data items in crystallography must of course be accessible across the field, and so there are two formulations of the dictionary of core items, one in each DDL version. The existence of two formulations can make full information interchange across all areas of crystallography difficult, so work is under way to bring about a convergence of the two current representations (Hall *et al.*, 2002). It is particularly important for future interchange between crystallography and other related disciplines that a full understanding be reached of the best way to include different data structure models within a common interchange format.

In this chapter, there will be some discussion of the differences in practice between the DDL versions DDL1 and DDL2, as these will strongly influence the choice of formalism for a dictionary relevant to a subdiscipline not yet represented.

### 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. With DDL1, a private data name will always have the form `_[local]_private_data_name`, while with DDL2 the forms `_[local]_new_category_name.private_data_name` and `_existing_category_name.[local]_private_data_name` may be used. The first DDL2 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 DDL2 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). For DDL1 applications it must be the first component of the data name; for DDL2 applications it forms the first component of the data name if describing data names in a category not defined in the official dictionaries; or the first component after the full stop (category delimiter) if the local data name is an extension to an existing category.

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

| String | Reserved for the use of |
|---|---|
| anbf | Australian National Beamline Facility |
| asd | Active Site Database |
| B+S | Software developers Bernstein + Sons |
| ccdc | Cambridge Crystallographic Data Centre |
| CCP4 | *CCP*4 program system |
| cgraph | Oxford Cryosystems *Crystallographica* package |
| cifdic | Register of CIF dictionaries |
| crystmol | *CrystMol* package |
| csd | Cambridge Structural Database |
| ebi | European Bioinformatics Institute |
| edchem | Edinburgh University Chemistry Department |
| gsas | *GSAS* powder refinement system |
| gsk | Glaxo Smith Kline |
| iims | EBI project on integration of information about macromolecular structure |
| iucr | IUCr journal use |
| mdb | Model Database (Glaxo) |
| msd | EBI Molecular Structure Database Group |
| ndb | Nucleic Acids Database Project, Rutgers University |
| oxford | *CRYSTALS* package, University of Oxford |
| parvati | Validation and statistical summaries from *PARVATI* validation server |
| pdb | Protein Data Bank |
| pdbx | Protein Data Bank exchange dictionary |
| pdb2cif | Additions to mmCIF used by program *pdb2cif* |
| rcsb | Research Collaboratory for Structural Bioinformatics |
| shelx | *SHELXL* solution and refinement programs |
| vrf | Validation reply form (IUCr/*Acta Crystallographica* use) |
| wdc | Entries in the World Directory of Crystallographers |
| xtal | *Xtal* program system |

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 March 2005; 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`.

#### 3.1.2.3. Name spaces

The allocation of special prefixes as in Sections 3.1.2.1 and 3.1.2.2 above is a basic form of name-space allocation, because it gives authors the freedom to reproduce portions of otherwise standard data names within their own private constructions. This raises the wider question of whether a complete formalism for name-space allocation is needed. That is, the same data name might appear with different meanings in different files, provided it was clear which of the alternative definitions must be used in each case. For now, the decision has been taken not to permit the use of the same data names with different meanings in different contexts. This is to enforce uniformity of definition across the whole field of crystallography as far as is possible. This policy might be reviewed in the future if similar formalisms to CIF are created in related disciplines.

### 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://www.iucr.org/iucr-top/cif) 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 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.

#### 3.1.3.2. mmCIF review cycle

The macromolecular CIF dictionary covers a very broad and active field, and a more formal procedure exists for the submission and review of proposed extensions. Possible new definitions are submitted using *pro forma* dictionary templates to a member of an editorial board appointed by the mmCIF dictionary maintenance group. Accepted proposals are approved by the DMG and released for general community review in provisional extension dictionaries as circumstances require. The extension dictionary is revised as necessary and is finally incorporated within the parent mmCIF dictionary after COMCIFS approval has been granted.

#### 3.1.3.3. 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.

### 3.1.4. Choice of data model

The following sections of this chapter describe the technical considerations in defining data items within a dictionary. Fundamental to this is the *data model* on which the dictionary is based. The STAR File upon which CIF is based is a very versatile data format and can accommodate a variety of data models. However, the use within CIF of a single level of looping enforces a rather flat data structure and a typical CIF maps most easily onto a relational database model. This is implicit in DDL1, which assigns different attributes to data items depending on whether they appear in data loops or not. Generally speaking, one may consider a list header and its associated data values as the head and body of a table of data values. The list header (or equivalently the table head) identifies the data items ranged by column within the table. For the dictionary entries relating to the data names in the list header, the `_category` attribute collects together data items which may be looped together in the same table, and the `_list_reference`, `_list_mandatory` and `_list_uniqueness` attributes work together to indicate the data items that must be present and collectively have a unique value to identify a specific row in a table of values.

Example 3.1.4.1. *Core dictionary definitions for the atom-site labels and bond distances in a CIF table of molecular geometry.*

```
data_geom_bond_atom_site_label_
    loop_ _name
                        '_geom_bond_atom_site_label_1'
                        '_geom_bond_atom_site_label_2'
    _category                       geom_bond
    _type                           char
    _list                           yes
    _list_mandatory                 yes
    _list_link_parent             '_atom_site_label'
    _definition
;     The labels of two atom sites that form a bond.
      These must match labels specified as
      _atom_site_label in the atom list.
;

data_geom_bond_distance
    _name                         '_geom_bond_distance'
    _category                       geom_bond
    _type                           numb
    _type_conditions                esd
    _list                           yes
    _list_reference    '_geom_bond_atom_site_label_'
    _enumeration_range              0.0:
    _units                          A
    _units_detail                  'angstroms'
    _definition
;     The intramolecular bond distance in angstroms.
;
```

For example, the following example from the core CIF dictionary (Chapter 4.1) shows a table of bond distances. The dictionary definitions are given in Example 3.1.4.1.

```
loop_
_geom_bond_atom_site_label_1
_geom_bond_atom_site_label_2
_geom_bond_distance
  O1  C2   1.342(4)
  O1  C5   1.439(3)
  C2  C3   1.512(4)
  C2  O21  1.199(4)
  C3  N4   1.465(3)
  C3  C31  1.537(4)
  C3  H3   1.00(3)
  N4  C5   1.472(3)
```

Within the dictionary, entries for all of `_geom_bond_distance`, `_geom_bond_atom_site_label_1` and `_geom_bond_atom_site_label_2` share the same `_category` attribute, namely 'geom_bond'. (In the rest of this chapter, as elsewhere in the volume, we refer to categories by the upper-case form of their category attribute values; here, therefore, we are referring to the GEOM_BOND category.) The entry for `_geom_bond_distance` has a `_list_reference` value of `'_geom_bond_atom_site_label_'` indicating the data names that may be used to identify this particular table. The trailing underscore in this example indicates that all matching data names must be considered as components of a compound identifier; for this case the matching data names are `'_geom_bond_atom_site_label_1'` and `'_geom_bond_atom_site_label_2'`. The dictionary entry for `_geom_bond_atom_site_label_` has a `_list_mandatory` value of `yes`, indicating that these data items *must* be present within the table. In this way, the attributes specify the unique key within a database table (in this case, the key has multiple components: the labels of both contributing atom sites).

However, the mapping onto a relational database is not exact. In some cases CIFs may present data from a single category across

several tables, or the implied key may not have a unique value unless concatenated with other fields in the table row. For many applications this is only of academic interest; but in some subdisciplines it is important that the data model is constrained strictly to a relational one, and for those applications dictionaries built on the DDL2 formalism are more appropriate.

Of the dictionaries presented in this volume, the core, powder, modulated structures and electron density dictionaries use the DDL1 formalism and the symmetry, macromolecular and image dictionaries use the DDL2 formalism. The core dictionary uses DDL1 so that it can be used alongside other less rigorous dictionaries. The powder dictionary is one case of this, where the need to tabulate and merge extensive lists of raw or processed data is not well served by a relational model. Modulated structures are also best served by a data model that is not rigorously relational. The macromolecular dictionary uses DDL2 because many of the major database applications in macromolecular crystallography are relational in nature, but in consequence it contains a copy of the core data items re-expressed in DDL2 formalism. The image dictionary is in DDL2 because it was designed to operate closely alongside the macromolecular dictionary. The symmetry dictionary is an interesting case. It was constructed in DDL2 format as an exercise in supplying an extension dictionary immediately suitable for direct incorporation into other DDL2-based dictionaries and also suitable for transformation to the simpler DDL1 formalism as necessary to complement existing DDL1 dictionaries.

While the main difference between DDL1 and DDL2 lies in the rigour with which relational data structures are enforced, DDL2 also offers a larger set of attributes for specifying hierarchical relationships between data names and for typing data values, and in consequence a complete DDL2-based dictionary is richer (and correspondingly more complex to construct) than an equivalent DDL1 description.

There may be no obvious reason for selecting one formalism over the other when planning a new data dictionary, and prospective authors must give considerable thought to the merits of both formalisms. However, once the choice has been made, the structure of the dictionary and its component definitions is profoundly affected. The constructions of the two types of dictionary are discussed separately in Sections 3.1.5 and 3.1.6 below.

```
data_on_this_dictionary
    _dictionary_name              cif_core.dic
    _dictionary_version           2.3.1
    _dictionary_update            2005-06-27
    _dictionary_history
;
   1991-05-27  Created from CIF Dictionary text. SRH
   . . .
;
```

(*a*)

```
data_atom_site_[]
    _name                        '_atom_site_[]'
    _category                     category_overview
    _type                         null
    loop_ _example
    _example_detail   .   .

data_atom_site_adp_type
    _name                        '_atom_site_adp_type'
    _category                     atom_site
    _type                         char
    _definition                  'A standard code ...'

data_atom_site_aniso_B_
    loop_ _name                  '_atom_site_aniso_B_11'
                                  . . .
```

(*b*)

Fig. 3.1.5.1. Schematic structure of core CIF dictionary. (*a*) Dictionary identifiers. (*b*) Definitions of categories and data items.

### 3.1.5. Constructing a DDL1 dictionary

Dictionaries constructed according to DDL1 have quite a simple structure. The structure is summarized in this section; Sections 3.1.5.1–3.1.5.4 provide more detail. Each definition is encapsulated within its own data block. Fig. 3.1.5.1 outlines the contents of the core CIF dictionary. The order of the data blocks has no significance, but it is common practice to start the file with the data block that describes the name, version and revision history of the dictionary itself and then to arrange data blocks in alphabetical order, sorted first on category then on names within a category. This practice is not always followed – for example, the powder dictionary is ordered by theme. The choice of order in a dictionary is only used for presentation and dictionary parsers should not assume or rely on any order of data blocks.

The name of a data block is usually constructed from the name of the data item it describes, *e.g.* `data_refln_phase_meas`. Where the data block describes an entire category instead of a single data item, the category name is followed by matching square brackets, which may contain an alphabetic code representing the dictionary name if it is an extension to the core dictionary (*e.g.* `data_refln_[]`, `data_audit_link_[ms]`). Where the data block defines several data names, the initial common portion of the names is used with a trailing underscore (*e.g.* `data_refln_`).

A preliminary data block, by convention labelled with the header string `data_on_this_dictionary`, contains the dictionary identification information and revision history. The name of the dictionary itself (given by the data name `_dictionary_name`) is conventionally of the form `cif_`*`identifier`*`.dic`, where the *identifier* is a short code for the topic area of the dictionary (*e.g.* 'core' for the core dictionary, 'pd' for the powder dictionary, 'ms' for the modulated structures dictionary, 'rho' for the electron density dictionary).

Data names are classified by category. The `_category` attribute is a character string intended to indicate the 'natural grouping' of data items. If a data item occurs in a looped list, it must be grouped only with items from the same category. It is, however, permissible for a file to contain more than one looped list of the same category, provided that each loop has its own specific reference item identified by the `_list_reference` attribute of the data names included. Examples of this will be given below.

For each category, a data block is usually provided that contains information about the purpose of the category, generally illustrated with examples.

All other data blocks represent self-contained definitions of a single data item or a small set of closely related data items. The definition includes the physical units of and constraints on the values of the data labelled by the defined data name, and also information about relationships with other data items.

It is conventional, although not mandatory in DDL1 dictionaries, that the category name should appear as the leading component or components of a data name. For example, the data name `_exptl_crystal_colour` is a member of the core category EXPTL, while `_exptl_crystal_density_meas` is a member of the category EXPTL_CRYSTAL and `_exptl_crystal_face_perp_dist` is a member of the category EXPTL_CRYSTAL_FACE. However, it will

Example 3.1.5.1. *A DDL1 dictionary identification block.*

```
data_on_this_dictionary
    _dictionary_name            cif_core.dic
    _dictionary_version         2.3.1
    _dictionary_update          2005-06-27
    _dictionary_history
;   1991-05-27  Created from CIF Dictionary text. SRH
    1991-05-30  Validated with CYCLOPS & CIF ms. SRH
    ...
;
```

be seen that there is no sure way of working out the category from the complete data name except by referring to its **_category** attribute in the associated dictionary. This differs from the DDL2 convention of including an explicit separator (a full stop) between the category name and the remainder of a data name.

While it is not mandatory that a data name should incorporate its category name as a leading component, authors are strongly encouraged to adopt this convention. A small number of core data items that did not conform to this convention have been deprecated in later releases of the core dictionary. However, in the powder dictionary the convention has been broken so that one can present data sets separately or merge them together. In this dictionary, some data names beginning with the strings **_pd_calc**, **_pd_meas** and **_pd_proc** all belong formally to the category PD_DATA. This allows calculated data values to be tabulated with raw and processed measurements if this is useful.

One other case where a data name does not begin with its associated category name is that of the pseudo data names such as **_exptl_[]** that appear in the dictionary to describe the purpose of a category (Section 3.1.5.3). Such data names are always assigned the category CATEGORY_OVERVIEW and are further differentiated from other data names by having a data type of 'null'.

### 3.1.5.1. The dictionary identification block

As mentioned above, the dictionary file must contain information that unambiguously states its identity and version. In DDL1-based CIF dictionaries, this is achieved by itemizing the full set of dictionary attributes (see Section 2.5.6.5) within a data block named **data_on_this_dictionary**, as in Example 3.1.5.1 from the core dictionary.

### 3.1.5.2. Irreducible sets of data items

In general, a dictionary data block defines a single data item. However, there are instances where several related data names are defined in the same data block. Sometimes this has been done for convenience, to produce a compact listing of similar data names that have common attributes and whose small differences in meaning can best be expressed by a single definition. Such groupings are discouraged, except where they represent components of a larger entity that has no sensible meaning in the absence of any of the components. For example, the data block **data_refln_index_** defines the three data items **_refln_index_h**, **_refln_index_k** and **_refln_index_l** that represent the Miller indices of a reflection. All three indices must have a value in order to specify a reflection and so each has no meaning in isolation.

Note that there is no formal method of expressing this close relationship within DDL1 except by grouping the definitions in the same data block in this way. In DDL2 dictionaries, it is common to assign the components of an irreducible set to a specific subcategory.

Example 3.1.5.2. *A category description in a DDL1 dictionary.*

```
data_exptl_[]
    _name                       '_exptl_[]'
    _category                     category_overview
    _type                         null
    loop_ _example
    _example_detail
# - - - - - - - - - - - - - - - - - - - - - - - - - -
;   _exptl_absorpt_coefficient_mu     0.962
    _exptl_absorpt_correction_type    psi-scan
    _exptl_absorpt_process_details
                'North, Phillips & Mathews (1968)'
    _exptl_absorpt_correction_T_min   0.929
    _exptl_absorpt_correction_T_max   0.997
;
;   Example 1 - based on a paper by Steiner [Acta
                Cryst. (1996), C52, 2554-2556].
;
# - - - - - - - - - - - - - - - - - - - - - - - - - -
    _definition
;       Data items in the EXPTL category record
        details about the experimental work prior
        to the intensity measurements and details
        about the absorption-correction technique
        employed.
;
```

### 3.1.5.3. Category descriptions

As discussed above, categories in DDL1 are intended as 'natural groupings' of data items. To document the purpose of a category within a dictionary, 'pseudo' data names are used. All pseudo data names are assigned a **_category** attribute of category_overview and have an associated **_type** value of 'null'. They are also named by convention as ***_category_name_[dictionarycode]***, for example **_pd_data_[pd]** for the description of the PD_DATA category in the powder dictionary (indicated by the code 'pd' in square brackets). For the core dictionary, *dictionarycode* is not given, resulting in names like **_exptl_[]** to describe the EXPTL category.

Example 3.1.5.2 is a slightly edited extract from the core dictionary showing how a data block for a category description is composed, including the presence of an example.

Note that the *dictionarycode* extension allows a dictionary to include comments on items that it defines in a category already established in the core dictionary. For example, the modulated structures dictionary includes the category overview item **_audit_link_[ms]**. This describes the convention adopted to express the relationship between data blocks in a modulated structures data file using the **_audit_link_** data names already defined in the core dictionary.

### 3.1.5.4. Data-item definitions

The data blocks described in Sections 3.1.5.1 and 3.1.5.3 are used to identify the dictionary and to describe the nature and purpose of a category. The remaining data blocks in a dictionary provide the attributes of data values in a form suitable for machine extraction and validation. The following examples show how this is done for various types of data.

#### 3.1.5.4.1. *Definitions of single quantities*

Example 3.1.5.3 is the core dictionary definition of the data name for the ambient temperature during the experiment. Because this is a single (non-looped) value, the relevant data name is one among several discrete items in the DIFFRN category. No further description of its relationship to other data items is required.

The type of the associated data value (*numb* for numerical) is specified, together with any constraint on its legal value. The range

Example 3.1.5.3. *A simple definition of a data item describing a physical quantity.*

```
data_diffrn_ambient_temperature
    _name                 '_diffrn_ambient_temperature'
    _category                    diffrn
    _type                        numb
    _type_conditions             esd
    _enumeration_range           0.0:
    _units                       K
    _units_detail                kelvin
    _definition
;       The mean temperature in kelvins at which the
        intensities were measured.
;
```

specified (`0.0:`) indicates that it may be any non-negative real number. The physical units of the quantity are also indicated.

The `_definition` attribute is a concise human-readable documentation of the meaning associated with the data name.

Example 3.1.5.4 is taken from the powder dictionary and illustrates a data item that can have only one of a limited set of values. This data item indicates the geometry of the experiment. The associated data value is of type *char* and may legally take only one of the two possible values listed.

Example 3.1.5.4. *A data item that can take only one of a discrete set of allowed values.*

```
data_pd_spec_mount_mode
    _name                        '_pd_spec_mount_mode'
    _category                     pd_spec
    _type                         char
    loop_ _enumeration            reflection
                                  transmission
    _definition
;       A code describing the beam path through
        the specimen.
;
```

Example 3.1.5.5. *Definition relating a looped data item to the item used to identify a 'loop packet', or row of entries in a table.*

```
data_atom_site_U_iso_or_equiv
    _name                '_atom_site_U_iso_or_equiv'
    _category                    atom_site
    _type                        numb
    _type_conditions             esd
    _list                        yes
    _list_reference             '_atom_site_label'
```

Example 3.1.5.6. *Definition of a mandatory item within a loop.*

```
data_atom_site_label
    _name                        '_atom_site_label'
    _category                     atom_site
    _type                         char
    _list                         yes
    _list_mandatory               yes
    loop_ _list_link_child
          '_atom_site_aniso_label'
          '_geom_bond_atom_site_label_1'
          '_geom_bond_atom_site_label_2'
```

#### 3.1.5.4.2. *Looped data*

Many of the attributes of looped data items, such as their physical units or valid numerical values, may be defined in exactly the same way as for non-looped data. However, more care needs to be taken to describe the relationships between different looped data items.

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

```
loop_
     _atom_site_label
     _atom_site_fract_x
     _atom_site_fract_y
     _atom_site_fract_z
     _atom_site_U_iso_or_equiv
     _atom_site_thermal_displace_type
    O1 .4154(4) .56990(10) .3026000   .0600(10) Uani
    C2 .5630(5) .5087(2)   .32460(10) .060(2)   Uani
    C3 .5350(5) .4920(2)   .39970(10) .0480(10) Uani
    N4 .3570(3) .55580(10) .4167000   .0390(10) Uani
    C5 .3000(5) .6122(2)   .35810(10) .0450(10) Uani

loop_
     _atom_site_aniso_label
     _atom_site_aniso_U_11
     _atom_site_aniso_U_22
     _atom_site_aniso_U_33
     _atom_site_aniso_U_12
     _atom_site_aniso_U_13
     _atom_site_aniso_U_23
O1 .071(1) .076(1) .0342(9) .008(1)  .0051(9) -.0030(9)
C2 .060(2) .072(2) .047(1)  .002(2)  .013(1)  -.009(1)
C3 .038(1) .060(2) .044(1)  .007(1)  .001(1)  -.005(1)
N4 .037(1) .048(1) .0325(9) .0025(9) .0011(9) -.0011(9)
C5 .043(1) .060(1) .032(1)  .001(1) -.001(1)   .001(1)

loop_
     _geom_bond_atom_site_label_1
     _geom_bond_atom_site_label_2
     _geom_bond_distance
      O1  C2   1.342(4)
      O1  C5   1.439(3)
      C2  C3   1.512(4)
      C2  O21  1.199(4)
```

These loops, or tables of values, are properties of atom sites, each identified by a label such as O1. The definition of a data name such as `_atom_site_U_iso_or_equiv` expresses this by using the DDL1 `_list_reference` attribute (Example 3.1.5.5).

For an entry in the table to make sense, the site identifier must be present, so the definition for `_atom_site_label` declares it a mandatory item within its list (Example 3.1.5.6).

It is common for an atom-site identifier to be used in several related tabulations in a particular crystal structure description, and in a CIF description this means that it may occur in several different looped lists. The dictionary definition gives a formal account of this by listing the data names in other looped lists which are just different manifestations of this same item. This is done using the `_list_link_child` attribute, which identifies the data names to which the one being currently defined is 'parent'. In Example 3.1.5.6 (which is a subset of the full list in the core dictionary), `_atom_site_aniso_label`, `_geom_bond_atom_site_label_1` and `_geom_bond_atom_site_label_2` are identified as children of `_atom_site_label`.

It can been seen immediately that `_atom_site_aniso_label` is the atom-site identification label appearing in the second table in the example listing above, and the `_geom_bond_` items are clearly atom-site labels in a table of bonding properties between specified sites. There is, however, a difference between the two secondary tables: the bond-properties table is described by data items in the GEOM_BOND category, but the table of anisotropic displacement parameters includes data names that have the same `_category` attribute as the coordinate data items, namely ATOM_SITE. The latter is an example of multiple lists or tables belonging to the same category, a feature permitted only in DDL1-based data files.

#### 3.1.5.4.3. *Units*

The physical units in which a quantitative data item must be expressed are identified by the DDL1 attributes `_units` and

**_units_detail**. The latter is a character field describing the units; the **_units** attribute is a code that may be interpreted by machine. In DDL1-based dictionaries, type codes are purely conventional, and there is no mechanism for converting units or relating quantities in different units. Table 3.1.5.1 lists the units codes used in the DDL1-based dictionaries described in this volume. There can be some inconsistencies: two codes ('s' and 'sec') are already in use to indicate the time unit of seconds.

The original CIF paper (Hall *et al.*, 1991) described a convention allowing physical quantities to be listed in a CIF in units other than those specified in the dictionary. Under this convention, a data name representing a value expressed in different units could be constructed by appending one of a series of known 'units extension codes' to the standard data name. Thus **_cell_length_a_pm** would represent a cell length expressed in picometres instead of the default ångströms. This approach is now deprecated, and all quantities must be expressed in the single unit permitted in their definition block. However, to allow the formal validation of old CIFs, a 'compatibility dictionary' is available which defines all data names that could have been constructed under this convention in a properly DDL1.4-compliant form. *This dictionary should only be used for validating old CIFs, and must not be used to construct new data files.* The dictionary is called cif_compat.dic in the IUCr CIF dictionary register (see Section 3.1.8.2).

Table 3.1.5.1. *Units codes and their interpretation in DDL1-based dictionaries*

| Unit code (**_units**) | Meaning (**_units_detail**) |
|---|---|
| **A** | Ångströms |
| **A^-1^** | Reciprocal ångströms |
| **A^2^** | Ångströms squared |
| **A^3^** | Ångströms cubed |
| **Da** | Daltons |
| **K** | Kelvins |
| **Kmin^-1^** | Kelvins/minute |
| **Mgm^-3^** | Megagrams per cubic metre |
| **\ms** | Microseconds |
| **deg** | Degrees |
| **deg/min** | Degrees per minute |
| **eV** | Electronvolts |
| **e_A^-3^** | Electrons per cubic ångström |
| **fm** | Femtometres |
| **kPa** | Kilopascals |
| **kV** | Kilovolts |
| **kW** | Kilowatts |
| **mA** | Milliamperes |
| **min** | Minutes |
| **mm** | Millimetres |
| **mm^-1^** | Reciprocal millimetres |
| **s** | Seconds |
| **sec** | Seconds |

### 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.

Unlike DDL1 dictionaries, 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).

As with DDL1 dictionaries, 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).

As is invariable with DDL2 data names, 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. The sole data block has the same name as the dictionary title string and the data block is introduced by the dictionary identification data items. The dictionary revision history introduces the file, followed by information about the extended data types and physical units used within the current dictionary. These are followed by the lists of closely related items (corresponding to 'irreducible sets' in DDL1 dictionaries and called 'subcategories' in the terminology of DDL2) 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.

#### 3.1.6.1. Dictionary identification

Dictionary files must contain information that unambiguously states their identity and version. In DDL2-based dictionaries this is done using the dictionary attributes described in Section 2.6.6.4. The name of the data block comprising the whole content of a DDL2 dictionary is by convention 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. Unlike in DDL1-based dictionaries where the history is contained in a single field, DDL2 provides a looped list of version labels, dates and annotations. For convenience, the history records in large DDL2-based dictionaries are sometimes placed at the end of the dictionary file.

#### 3.1.6.2. Subcategory definitions

In the DDL1 formalism, particular relationships between data items may sometimes be stated within a text description or may be implied by the organization of the dictionary (where several data

```
data_mmcif_std.dic

    _dictionary.title           mmcif_std.dic
    _dictionary.version         2.0.09
    _dictionary.datablock_id    mmcif_std.dic
                          (a)

     loop_
         _dictionary_history.version
         _dictionary_history.update
         _dictionary_history.revision   .   .   .
                          (b)

     loop_
    _sub_category.id
    _sub_category.description    .   .

     loop_
    _category_group_list.id
    _category_group_list.parent_id
    _category_group_list.description   .   .   .
                          (c)

    loop_
    _item_type_list.code
    _item_type_list.primitive_code
    _item_type_list.construct
    _item_type_list.detail

    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)


  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_
                          (e)
```

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

items are defined in the same data block and are understood to share the common attributes itemized in that data block).

Within DDL2, there are mechanisms for more 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 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. While relationships between categories are implied in DDL1 dictionaries by the hierarchical structure of the names of data items, in DDL2 dictionaries the relationships 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.

Example 3.1.6.1. *DDL2 dictionary identification entries.*

```
data_mmcif_std.dic

    _dictionary.title           mmcif_std.dic
    _dictionary.version         2.0.09
    _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.
;
```

### 3.1.6.4. Category definitions

In the DDL2 formalism, a category of data items may be mapped to a relational table. 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 3.1.6.3. *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.
;
```

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 experimental work prior to the
    intensity measurements and details about the
    absorption-correction technique employed.
;
    _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 3.1.6.4 shows a category of non-looped core data items. It may be compared with the DDL1 version in Example 3.1.5.2.

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.

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 the bond lengths 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 [Acta Cryst. (1991), C47,
   2276-2277].
;
;    loop_
     _geom_bond.atom_site_id_1
     _geom_bond.atom_site_id_2
     _geom_bond.dist
     _geom_bond.dist_esd
     _geom_bond.site_symmetry_1
     _geom_bond.site_symmetry_2
     _geom_bond.publ_flag
      O1  C2   1.342  0.004  1_555  1_555  yes
      O1  C5   1.439  0.003  1_555  1_555  yes
      C2  C3   1.512  0.004  1_555  1_555  yes
      C2  O21  1.199  0.004  1_555  1_555  yes
      C3  N4   1.465  0.003  1_555  1_555  yes
      C3  C31  1.537  0.004  1_555  1_555  yes
      C3  H3   1.00   0.03   1_555  1_555  ?
      N4  C5   1.472  0.003  1_555  1_555  yes
    # - - - - data truncated for brevity - - - -
;
# - - - - - - - - - - - - - - - - - - - - - - - - - -
save_
```

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.

Careful inspection of corresponding definitions in the DDL1 and DDL2 versions of core data items will demonstrate that the explicit category key specification in DDL2 dictionaries may be deduced within DDL1 dictionaries from the appropriate `_list_reference`, `_list_mandatory` and `_list_uniqueness` attributes of data-item definitions within a category (see also Section 2.5.6.4).

### 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.

Unlike DDL1 dictionaries, where the definitions of several data names may be contained in a single data block (most commonly for a set of items that form a logical irreducible set), save frames in

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_
```

DDL2 dictionaries each contain the definition for a single addressable concept.

For example, the three Miller index components of a diffraction reflection (`_diffrn_refln_index_h`, `_diffrn_refln_index_k`, `_diffrn_refln_index_l` that are described in the DDL1 core CIF dictionary in the data block `data_diffrn_refln_`) are described 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`. In the DDL2 formalism, 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.

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.

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_
```

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_
```

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 parallels the example chosen for DDL1 dictionaries of the ambient temperature during the experiment.

In the definition save frame, the category is specifically listed (although it is deducible from the DDL2 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. (In the core dictionary there are fewer data types and the fact that the value may be a real rather than integer number must be inferred from the declared range.) The range of values is also specified with separate maximum and minimum values (unlike in DDL1 dictionaries, which give a single character string that must be parsed into its component minimum and maximum 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 the corresponding quantity defined in the DDL1 core dictionary.

#### 3.1.6.6. Units

As with data files described by DDL1 dictionaries, 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 `_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.

#### 3.1.7.1. 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 pos-

sible 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 equivalent data name from the mmCIF dictionary, **_atom_site.type_symbol**, explicitly separates the category to which the data name belongs from its more specific qualifiers by using a full stop (**.**) instead of an underscore (_). While this use of a full stop is mandated in DDL2 dictionaries, it should nevertheless be considered a convenience, since the category membership is explicitly listed in the dictionary definition frame for every data name.

```
_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).

However, it may not always be easy to establish the best order of components when constructing a new data 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 (see Section 3.1.8.2).

### 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.

When compound values are left as parsable strings, the parsing rules for individual data items need to be made known to applications. The DDL1 attribute `_type_construct` was envisaged as a mechanism for representing the components of a data value with a combination of regular expressions and reference to primitive data items, but this has not been implemented in existing CIF dictionaries (or in dictionary utility software). An alternative approach used in DDL2-based dictionaries defines within the dictionaries a number of extended data types (expressed in regular-expression notation through the attribute `_item_type_list.code`).

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 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. For DDL1 dictionaries, the relevant identifiers are the items `_audit_conform_dict_name`, `_audit_conform_dict_version` and `_audit_conform_dict_location`, defined in the core dictionary. DDL2 dictionaries are identified by the equivalent items `_audit_conform.dict_name`, `*.dict_version` and `*.dict_location`. For convenience, the DDL1 versions will be used in the following discussion.

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

The data item `_audit_conform_dict_location` may be used to specify a file name or uniform resource locator (URL). However, a file name on a single computer or network will be of use only to an application with the same view of the local file system, and so is not portable. A URL may be a better indicator of the location of a dictionary file on the Internet, but can go out of date as server names, addresses and file-system organization change over time. The preferred method for locating a dictionary file is to make use of a dynamic register, as described in Section 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      2.3.1   .
    cif_pd.dic        1.0.1   .
    cif_local_my.dic  1.0
       /usr/local/dics/my_local_dictionary
```

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 standard deviation) (*see* su) | org | organism |
| absorpt | absorption | | | orient | orientation |
| alt | alternative | expt | experiment | origx | orthogonal coordinate matrix (PDB files) |
| amp | amplitude | exptl | experimental | os | operating system |
| AN | accession number | fom | figure of merit | param | parameter |
| anal | analyser | fract | fractional | pd | powder diffraction |
| aniso | anisotropic* | Fsqd | *F* squared | PDB | Protein Data Bank |
| anisotrop | anisotropic* | gen | generation | PDF | Powder Diffraction File |
| anom | anomalous | gen | generator | perp | perpendicular |
| ASTM | American Society for Testing and Materials | gen | genetic | phos | phosphate |
| asym | asymmetric | geom | geometric | pk | peak |
| atten | attenuation | H-M | Hermann–Mauguin | polarisn | polarization |
| au | arbitrary units | ha | heavy atom | poly | polymer |
| auth | author | hbond | hydrogen bond | pos | position |
| av | average | hist | history | prep | preparation |
| ax | axial | horiz | horizontal | proc | processed |
| B | *B* form of atomic displacement parameter (a.d.p.) | I | intensity | prof | profile |
| | | ICSD | Inorganic Crystal Structure Database | prot | protein |
| backgd | background* | id | identifier | ptnr | partner |
| beg | begin | illum | illumination | publ | publication |
| bg | background* | imag | imaginary | R | agreement index |
| biol | biology | inc | increment | rad | radius |
| bkg | background* | incl | include | recd | received |
| bond | bonding | info | information | recip | reciprocal |
| Bsol | *B* form of a.d.p. for solvent | instr | instrument | ref | reference |
| calc | calculated | Int | international | refine | refinement |
| calib | calibration (pd) | ISBN | International Standard Book Number | refln | reflection |
| cartn | Cartesian | iso | isotropic | reflns | reflections |
| CAS | *Chemical Abstracts* Service | iso | isomorphous | res | resolution |
| char | characterization (pd) | ISSN | International Standard Serial Number | restr | restraints |
| chem | chemical | IUCr | International Union of Crystallography | rev | revision |
| chir | chirality | IUPAC | International Union of Pure and Applied Chemistry | Rmerge | agreement index of merging |
| clust | cluster | | | rms | root mean square |
| coef | coefficient | len | length | rot | rotation |
| com | common | lim | limit | S | goodness of fit |
| comp | component | loc | lack of closure | samp | sample |
| conc | concentration | ls | least squares | scat | scattering factor |
| conf | conformation | max | maximum | seq | sequence |
| config | configuration | MDF | Metals Data File | sigI | $\sigma(I)$* |
| conform | conformant | meanI | mean intensity | sigmaI | $\sigma(I)$* |
| conn | connectivity | meas | measured | sint | $\sin\theta$ |
| cons | constant | mid | middle (between max and min) | sint/lambda | $\sin(\theta)/\lambda$* |
| CSD | Cambridge Structural Database | min | minimum | sol | solvent |
| db | database | mod | modification | spec | specimen |
| defn | definition | mods | modifications | src | source |
| detc | detector | mon | monomer | std | standard |
| der | derivative | monochr | monochromator (pd)* | stol | $\sin(\theta)/\lambda$* |
| dev | standard deviation | mono | monochromator (pd)* | struct | structure |
| dict | dictionary | nat | natural | su | standard uncertainty |
| dif | difference* | NBS | National Bureau of Standards (now National Institute of Standards and Technology) | suppl | supplementary |
| diff | difference* | | | sys | systematic |
| diffr | diffractometer | | | tbar | mean path length |
| diffrn | diffraction | NCA | number of connected atoms | temp | temperature |
| displace | displacement | ncs | noncrystallographic symmetry | tor | torsion angle |
| dist | distance | netI | net intensity | tran | transformation* |
| divg | divergence | NH | number of connected hydrogen atoms | transf | transformation* |
| dom | domain | nha | non-hydrogen atoms | transform | transformation* |
| dtime | deadtime | norm | normal | tvect | translation vector (PDB files) |
| ens | ensemble | nst | nonstandard | vert | vertical |
| eq | equatorial* | nucl | nucleic acid | wR | weighted agreement index |
| equat | equatorial* | num | number | wt | weight |
| equiv | equivalent | obs | observed | | |

* Terms with multiple definitions.

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. *CIF dictionary register (maintained as a STAR File)*

```
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     .    1.4.1   .
      ftp://ftp.iucr.org/pub/cifdics/cif_core.dic
      'Core CIF Dictionary'
  cif_core.dic   1.0   .   .
      ftp://ftp.iucr.org/pub/cifdics/cifdic.C91
      'Original Core CIF Dictionary'
  cif_core.dic  2.3.1  1.4.1   .
      ftp://ftp.iucr.org/pub/cifdics/cif_core_2.3.1.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    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_pd_1.0.1.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    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_ms_1.0.1.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    1.4   .
      ftp://ftp.iucr.org/pub/cifdics/cif_rho_1.0.1.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    .   2.1.6  .
      ftp://ftp.iucr.org/pub/cifdics/mmcif_std.dic
      'Macromolecular CIF Dictionary'
  mmcif_std.dic    2.0.09   2.1.6  .
      ftp://ftp.iucr.org/pub/cifdics/cif_mm_2.0.09.dic
      'Macromolecular CIF Dictionary'
  cif_img.dic     .    2.1.3  .
      ftp://ftp.iucr.org/pub/cifdics/cif_img.dic
      'Image CIF Dictionary'
  cif_img.dic    1.0   2.1.3  .
      ftp://ftp.iucr.org/pub/cifdics/cif_img_1.0.dic
      'Image CIF Dictionary'
  cif_img.dic    1.3.2   2.1.3  .
      ftp://ftp.iucr.org/pub/cifdics/cif_img_1.3.2.dic
      'Image CIF Dictionary'
  cif_sym.dic    .    2.1.3  .
      ftp://ftp.iucr.org/pub/cifdics/cif_sym.dic
      'Symmetry CIF Dictionary'
  cif_sym.dic   1.0.1   2.1.3  .
      ftp://ftp.iucr.org/pub/cifdics/cif_sym_1.0.1.dic
      'Symmetry CIF Dictionary'
  cif_compat.dic   .   1.4  .
      ftp://ftp.iucr.org/pub/cifdics/cif_compat.dic
      'Legacy CIF Dictionary of deprecated terms'
  ddl_core.dic   .   1.4.1  .
      ftp://ftp.iucr.org/pub/cifdics/ddl_core.dic
      'Non-relational dictionary definition language'
  ddl_core_2.1.3.dic   .   2.1.3  .
      ftp://ftp.iucr.org/pub/cifdics/ddl_core_2.1.3.dic
      'Relational dictionary definition language'
  mmcif_ddl.dic   .   2.1.6  .
      ftp://ftp.iucr.org/pub/cifdics/mmcif_ddl.dic
      'Relational dictionary definition language'
```

Table 3.1.8.1 is an extract from the current register. 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_name` or `_dictionary.title` for DDL1 or DDL2 dictionaries, respectively) 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` (`_audit_conform.dict_name` for DDL2 files) (*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).

This default is appropriate for most well formed CIFs, but if it is important to provide formal validation of old CIFs conforming to the earliest printed specification, which used the now-deprecated units extension convention, the dictionary cif_compat.dic may also be added to the default list (Section 3.1.5.4.3).

There is a difficulty associated with assuming this default for CIFs containing DDL2 data names. At present, the DDL2 version of the core dictionary does not exist as a separate file. Most existing CIFs built on the DDL2 model conform to the macromolecular (mmCIF) dictionary, and so best current working practice is to assume a default validation dictionary for DDL2-style CIFs with *N* = `mmcif_std.dic` and *V* = '.' (*i.e.* the most recent version of the mmCIF dictionary), since this includes the core data names as a subset. However, to anticipate future developments, it is suggested that applications built to validate DDL2 files first search the register for a default entry with *N* = `cif_core.dic`, *V* = '.' and a value of 2 or higher for the relevant DDL version:

```
loop_
  _cifdic_dictionary.name
  _cifdic_dictionary.version
  _cifdic_dictionary.DDL_compliance
  cif_core.dic     .    2.1.2
```

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

If *N*, *V* and *L* are all given, try to load the file from the location *L*, or a locally cached copy of the referenced file. If this fails, raise a warning. Then search the dictionary register for entries matching the given *N* and *V*. (An appropriate strategy would be to search a locally cached copy of the register, and to refresh that local copy with the latest version from the network if the search fails.) If a successful match is made, try to retrieve the file from the location given by the matching entry in the register (or a locally cached copy with the same *N* and *V* previously fetched from the location specified in the register). If this fails, try to load files identified from the register with the same *N* but progressively older versions *V* (version numbering takes the form *n.m.l* . . ., where *n*, *m*, *l*, . . . are integers referring to progressively less significant revision levels). Version '.' (meaning the current version) should be accessed before any other numbered version. If this fails, raise a warning indicating that the specified dictionary could not be located.

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

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

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

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

For any dictionary file successfully loaded according to this protocol, the validation application must perform a consistency check by scanning the file for internal identifiers (`_dictionary_name`, `_dictionary_version` or the DDL2 equivalents) and ensuring that they match the values of *N* and *V* (where *V* is not '.'). Failure in matching should raise an error.

### 3.1.9. Composite dictionaries

The dictionaries referenced by a data file are those that contain the definitions of the data names used in the data file. Typically these include or consist entirely of public dictionaries that are necessarily permissive in the range of values allowed for data items. However, the power and flexibility of validating against machine-readable dictionaries could be harnessed by applications that need to impose stricter validation criteria. For example, the core dictionary permits an enumeration range of 0 to 8 for `_atom_site_attached_hydrogens`, but one might wish to validate a data set describing well behaved organic molecules where anything above 4 is almost certainly an error. It would be helpful to have a validation dictionary identical to the core dictionary except for this enumeration range; however it would be inefficient to create an alternative dictionary of the same size simply for this one change. In Section 3.1.9.1, we consider how to build a dictionary file that includes the bulk of the content of the public dictionaries cited in the CIF, together with modifications in local dictionary files to allow alternative specifications of what constitutes a 'valid' data item.

Proper applications of this approach include restricting the enumeration range specified for an item in a public dictionary; enforcing a more strict data typing than allowed by the parent dictionary; storing a list of all data names (including local ones) permitted in a CIF; or adding to existing dictionary entries references to local data items in an extension dictionary. An example of the latter application would be the addition of a `_list_link_child` entry to a public definition to accompany the introduction of a new child category in a local dictionary. The protocol to be described does not prohibit other applications, but care must be taken to generate dictionaries that retain internal consistency and are properly parsable by standard validation tools.

#### 3.1.9.1. A dictionary merging protocol

The following protocol describes the construction of a *composite*, or *virtual*, dictionary by merging and overlaying fragments of a local validation dictionary and the public dictionaries referenced from within a data file. The term 'dictionary fragment' refers here to a physical disk file which contains one or more data blocks or save frames (according to whether the relevant data model is DDL1 or DDL2) containing complete or partial sets of attributes associated with data names identified in the relevant dictionary data block or save frame through the item `_name` (DDL1) or `_item.name` (DDL2).

(i) Assemble and load all dictionary fragments against which the current data block will be validated. The order of presentation is important. Complete dictionaries referenced by a data file should be assembled in the order cited. A dictionary validation application may then accept a list of additional dictionary fragments to PREPEND to, REPLACE or APPEND to each file in the list of cited dictionaries. In most applications, it will be appropriate to append to or replace attributes defined in a public dictionary, and the PREPEND operation is presented only for completeness.

(ii) Define three modes in which conflicting data names in the aggregate dictionary file may be resolved, called STRICT, REPLACE and OVERLAY.

(iii) Scan the aggregate dictionary fragments in the order of loading. Assemble for each defined data name a composite definition frame (data block or save frame as appropriate) as follows, depending on the mode in which the validation application is operating:

STRICT: If a data name appears to be multiply defined, generate a fatal error. This mode permits the interpolation of local dictionaries that do not attempt to modify the attributes of public data items.

REPLACE: All attributes previously stored for the conflicting data name are deleted, and only the attributes in the later data block (or save frame) containing the definition are preserved. This mode permits the complete redefinition of public data names and is not appropriate for validation of CIFs to be archived. Its main use would be in testing modifications of individual definition frames outside the parent dictionary.

OVERLAY: New attributes are added to those already stored for the data name; conflicting attributes replace those already stored. This is the standard mechanism for modifying attributes for application-specific validation purposes.

This protocol allows the creation of a coherent virtual dictionary from several different dictionary files or fragments. Although it must be used with care, it permits different levels of validation based on dictionary-driven methods without modifying the original dictionary files themselves.

Example 3.1.9.1. *A standard CIF dictionary definition block.*

```
data_atom_site_attached_hydrogens
    _name              '_atom_site_attached_hydrogens'
    _category            atom_site
    _type                numb
    _list                yes
    _list_reference      '_atom_site_label'
    _enumeration_range   0:8
    _enumeration_default 0
    loop_ _example
          _example_detail  2    'water oxygen'
                           1    'hydroxyl oxygen'
                           4    'ammonium nitrogen'
    _definition
;       The number of hydrogen atoms attached
        to the atom at this site excluding any
        hydrogen atoms for which coordinates
        (measured or calculated) are given.
;
```

Example 3.1.9.2. *A modified data attribute for overlaying a public definition.*

```
data_atom_site_attached_hydrogens_restricted
    _name              '_atom_site_attached_hydrogens'
    _enumeration_range   0:4
```

As an example, consider the core CIF dictionary definition mentioned above of the number of hydrogen atoms that might be attached to an atom site (Example 3.1.9.1).

For a particular application, any structures reporting more than four attached hydrogen atoms might be considered as invalid. A validation program to satisfy this requirement might therefore build a composite dictionary from the public cif_core.dic, which contains the definition in Example 3.1.9.1, and the fragment of Example 3.1.9.2, processed in APPEND/OVERLAY modes.

### 3.1.9.2. Protocol implementation

At the time of publication (2005), there is no reference implementation for this protocol, and so the proper treatment of the fine details of merging and overlay operations is not available. The following guidelines outline the first steps in an implementation under DDL1.4.

The description assumes that a composite dictionary is to be assembled from two public dictionaries, a.dic and b.dic, and a local dictionary mod.dic that includes some modifications to the definitions in one or both of the public dictionaries (and is therefore processed in OVERLAY mode). It is assumed that the composite dictionary will be written to disk as a separate file, virtual.dic, although in practice applications may simply construct the image of the composite dictionary in memory.

(1) Each contributing dictionary fragment should have at most one data block containing the data names `_dictionary_name` and `_dictionary_version` (with, optionally, `_dictionary_update` and `_dictionary_history`). The `*_name` and `*_version` together identify the dictionary file uniquely and should match the corresponding entries in the IUCr register if this is a public dictionary. This information is conventionally stored in a data block named `data_on_this_dictionary`.

In DDL1.4, all four of the items `_dictionary_name`, `*_version`, `*_update` and `*_history` are scalars, *i.e.* may not be looped. Hence a new dictionary identifier section in virtual.dic may be constructed as follows.

(i) Create a data block `data_on_this_dictionary` at the beginning of virtual.dic.

(ii) If a name for the composite dictionary is supplied (*via* a command-line switch, for example), write this as the value of `_dictionary_name`; otherwise generate a pseudo-unique string (*e.g.* concatenate the computer identifier string, process number and current date string).

(iii) If a dictionary version number is supplied (*via* a command-line switch, for example), write this as the value of `_dictionary_version`; otherwise supply the value '1.0'.

(iv) Supply the current date in the format *yyyy-mm-dd* as the value of `_dictionary_update`.

(v) Create a composite `_dictionary_history` by concatenation of the individual `_dictionary_history` fragments. The application may add details of the current merge operation to the history field.

(2) There is no significance to the ordering of data blocks containing definitions in dictionaries, although they are conventionally sorted alphabetically. For convenience, data blocks should be written out in the order in which they are encountered in the input primitive dictionary files, except that definitions modified by subsequent entries remain in their initial location.

(3) In STRICT mode, if the same value of `_name` is present in two or more data blocks, the composite dictionary is invalid and the application should raise a fatal error. Otherwise the composite dictionary simply contains the aggregate definitions from multiple input dictionaries.

(4) In REPLACE mode, a stored definition block is discarded and replaced by a new definition of the item referenced by `_name`.

(5) For the OVERLAY mode (assumed in the present discussion), the following procedure is proposed. Load a data block from the first dictionary file. Locate the `_name` tag. (Because `_name` may be looped, a data block may contain definitions for more than one data name. For convenience, we consider only the case of a data block containing a single value of `_name`. In any event, it is possible to separate a set of looped definitions into individual data blocks, each defining only one of the data names in the initial `_name` loop.) Search the next dictionary file for a data block containing the same value of `_name`. Load the contents of that data block.

(i) If the new data block contains only data items that do not appear in the first data block, they are simply concatenated with those already present.

(ii) If the new data block contains a scalar data item already present in the first data block (*i.e.* with `_list no`), discard the stored attributes.

(iii) If the new data block contains data items that may be looped and that occur in the first data block, build a new composite table of values in the following way: (*a*) construct a valid loop header if necessary; (*b*) do not repeat identical sets of values (*i.e.* collapse identical table rows); (*c*) if it is possible to identify the category key, then raise a fatal error if there are identical instances of a key value [after the normalization of step (*b*) has occurred]; (*d*) else append new rows to the table.

When the new composite data block has been built according to these principles, search the next dictionary file specified and repeat.

## 3.1.10. Public CIF dictionaries

So far, seven CIF dictionaries have been published by the IUCr with COMCIFS approval. They are described in the remaining chapters in this part of the volume. This section provides an overview of the large-scale structure of these dictionaries and forms a general introduction to Chapters 3.2 to 3.8.

The public CIF dictionaries have been constructed by experts in a number of different crystallographic fields. They are intended to serve the individual fields in which they have been commissioned and therefore vary in character depending on the requirements

Table 3.1.10.1. *High-level grouping of categories by dictionary*

Category groups are organized into families by common function and purpose.

| cif_core.dic | cif_pd.dic | cif_ms.dic | cif_rho.dic | mmcif_std.dic | cif_img.dic | cif_sym.dic |
|---|---|---|---|---|---|---|
| (*a*) Experimental measurements | | | | | | |
| | | | | | ARRAY | |
| | | | | | AXIS | |
| CELL | | CELL | | CELL | | |
| DIFFRN | | DIFFRN | | DIFFRN | DIFFRN | |
| EXPTL | | EXPTL | | EXPTL | | |
| | PD_CALIB | | | | | |
| | PD_CHAR | | | | | |
| | PD_DATA | | | | | |
| | PD_INSTR | | | | | |
| | PD_MEAS | | | | | |
| | PD_PREP | | | | | |
| | PD_SPEC | | | | | |
| (*b*) Analysis | | | | | | |
| | PD_CALC | | | | | |
| | PD_PEAK | | | | | |
| | PD_PROC | | | | | |
| | | | | PHASING | | |
| REFINE | | REFINE | | REFINE | | |
| REFLN | REFLN | REFLN | | REFLN | | |
| (*c*) Structure | | | | | | |
| ATOM | | ATOM | ATOM | ATOM | | |
| CHEMICAL | | | | CHEMICAL | | |
| | | | | CHEM_COMP | | |
| | | | | CHEM_LINK | | |
| | | | | ENTITY | | |
| GEOM | | GEOM | | GEOM | | |
| | PD_PHASE | | | | | |
| | | | | STRUCT | | |
| SYMMETRY | | SYMMETRY | | SYMMETRY | | SPACE_GROUP |
| VALENCE | | | | VALENCE | | |
| (*d*) Publication | | | | | | |
| CITATION | | | | CITATION | | |
| COMPUTING | | | | COMPUTING | | |
| DATABASE | | | | DATABASE | | |
| JOURNAL | | | | JOURNAL | | |
| PUBL | | | | PUBL | | |
| | | | | SOFTWARE | | |
| (*e*) File metadata | | | | | | |
| AUDIT | | AUDIT | | AUDIT | | |
| | PD_BLOCK | | | | | |

and practices of each field. Here we provide a general framework within which the category groups of each separate dictionary may be described.

### 3.1.10.1. Categories and category groups

The only formal unit of classification common to all CIF dictionaries is the *category*. For example, in the core CIF dictionary information about the chemical and physical properties of the different atomic species in a crystal cell is collected in a few data names such as **_atom_type_oxidation_number** which belong to the same category, in this case the ATOM_TYPE category. As described in Section 3.1.5.3, it is conventional (although not mandatory) that CIF data names begin with components corresponding to the name of the category to which they belong.

The term *category* as used in CIF dictionaries has a technical meaning which constrains its normal use in grouping items that are understood to have a 'natural' relationship. In a CIF, only items belonging to the same category may appear together in the same looped list. This means, for example, that data items describing collective properties of the atom sites in the lattice (such as the number of atoms of each atomic species in the unit cell) must be assigned to a different category from the data items that describe the properties of the individual sites. Hence the properties of individual sites (such as the positional coordinates defined by **_atom_site_fract_x** *etc.*) belong to the ATOM_SITE category, while the transformation matrix between Cartesian and fractional components (expressed by a collection of data names such as **_atom_sites_fract_tran_matrix_11**) belong to the ATOM_SITES category. Clearly, the category names have been chosen to be similar to reflect their close relationship, while the EXPTL category containing data names such as **_exptl_crystal_colour** is named quite differently. It is natural to wish to describe related categories in a common higher level of classification, and indeed *category groups* exist as formal components of DDL2-structured dictionaries. We shall, however, refer informally to 'category groups' in discussions of DDL1 dictionaries as collections of categories with a close relationship that is usually implicit in their names.

### 3.1.10.2. Overview of category classification

Table 3.1.10.1 provides an informal classification at a high level of the category groups represented in each of the CIF dictionaries in this volume. Related category groups are clustered within the table in families sharing a common function. The five families (*a*) to (*e*) in Table 3.1.10.1 refer to: the crystallographic experiment itself; the processing and analysis of data from the experiment; the derived structural model; the reporting and publication of the results; and general auditing of the file itself, its purpose, authorship, history and links to other data sets, *i.e.* the file metadata. Detailed discussions of the individual categories (and formal category groups for DDL2 dictionaries) will be found in the relevant chapters in the rest of this part of the volume.

Table 3.1.10.1 shows the different characters of the seven dictionaries. The macromolecular dictionary (mmcif_std.dic; Chapter 4.5) contains an embedded version of the core dictionary

(cif_core.dic; Chapter 4.1) in DDL2 format and so includes all the categories defined in the core. However, it extends the description of the structural model extensively by introducing families of categories for the description of chemical components of a macromolecular structure (ENTITY) and for the detailed description of the structure itself (STRUCT). New categories are also introduced to describe the phasing of the structure and the SOFTWARE category allows the inclusion of more details of computational techniques than the core COMPUTING category does.

The other dictionaries are purely extensions which either introduce new data names (and occasionally new categories) into existing category groups or, where necessary, introduce completely new groups of categories.

The powder dictionary (cif_pd.dic; Chapter 4.2) contains several new category groups reflecting the need for substantially different methods of describing the experiment and analysing the data, as well as a need for the structural model to be able to handle multiple crystalline phases. The modulated structures dictionary (cif_ms.dic; Chapter 4.3) introduces no new category groups, but does introduce several new data names and categories within the existing framework. The electron density dictionary (cif_rho.dic; Chapter 4.4) introduces two new categories within an existing category group. The image CIF dictionary (cif_img.dic; Chapter 4.6) has several new categories that characterize arrays of data from two-dimensional X-ray detectors and the consequent detailed descriptions of the relevant axes within the experimental setup. The symmetry dictionary (cif_sym.dic; Chapter 4.7) was commissioned specifically to replace the symmetry categories in the core dictionary with a more detailed treatment.

## References


Bray, T., Paoli, J. & Sperberg-McQueen, C. M. (1998). *Extensible Markup Language (XML) 1.0*. W3C Recommendation 10-February-1998. http://www.w3.org/TR/1998/REC-xml-19980210.

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

Hall, S. R., Allen, F. H. & Brown, I. D. (1991). *The Crystallographic Information File (CIF): a new standard archive file for crystallography. Acta Cryst.* A**47**, 655–685.

Hall, S. R., Spadaccini, N., Castleden, I. R., du Boulay, D. & Westbrook, J. D. (2002). *StarDDL: towards the unification of Star dictionaries. Acta Cryst.* A**58** (Suppl.), C256.

W3C (2004). *Extensible Markup Language (XML)*. http://www.w3c.org/XML/.