International Tables for Crystallography (2006). Vol. G, Chapter 2.6, pp. 61–70.

2.6. Specification of a relational dictionary definition language (DDL2)

BY J. D. WESTBROOK, H. M. BERMAN AND S. R. HALL

2.6.1. Introduction

The dictionary definition language version 2 (DDL2) presented here extends the DDL1 version (Hall & Cook, 1995) currently used by the IUCr for the description of data items common to all crystallographic studies (*i.e.* core items). The DDL2 extensions were introduced primarily to address two issues arising during the development of a CIF dictionary for the terminology of macromolecular crystallography: the need to accurately describe the hierarchical nature of macromolecular structure and associated structural features, and the desire to encode dictionary definitions in a manner that would permit more detailed software-driven validation.

The decision not to use DDL1 for the macromolecular CIF dictionary (mmCIF) was not made lightly. The Working Group responsible for the development of the mmCIF dictionary spent three years building a dictionary version within a DDL1 framework. When presented at the first mmCIF workshop in York in 1993 (Chapter 1.1), it was criticized as lacking the rigour to be usefully interpreted by software. In particular, the draft dictionary lacked machine-interpretable relationships between the components of macromolecular structure, the components of structure and structural properties, and the components of structure and the experimental description. These relationships are important to a complete description of macromolecular structural data, and need to be present in a dictionary in a form that permits software to navigate and validate them.

Following the York workshop, the Working Group set about redesigning the framework of the data model used to organize the dictionary definitions. Initially there was significant interest in adopting a more object-oriented data model so as to match closely the object-oriented characteristics of macromolecular structure. However, an object-oriented model would depart significantly from the organization of the core CIF dictionary based on DDL1, and interoperability between the two approaches would be likely to be problematic.

The new DDL concepts were presented at the Brussels mmCIF workshop in 1994. This approach, which later became known as DDL2 (Westbrook & Hall, 1995), employs a largely relational data model that adhered more closely to the data model implicit in the core DDL1. DDL2 added new elements to expand the DDL1 concept of a data category. Categories in DDL2 are fully realized definitional elements that have their own set of attributes (*e.g.* definitions and examples). DDL2 also added data elements to explicitly define parent–child relationships between data items within a hierarchy of categories. It was demonstrated that using this conservative relational data model it was possible to accurately describe the content in the mmCIF dictionary.

Affiliations: JOHN D. WESTBROOK and HELEN M. BERMAN, Protein Data Bank, Research Collaboratory for Structural Bioinformatics, Rutgers, The State University of New Jersey, Department of Chemistry and Chemical Biology, 610 Taylor Road, Piscataway, NJ 08854-8087, USA; SYDNEY R. HALL, School of Biomedical and Chemical Sciences, University of Western Australia, Crawley, Perth, WA 6009, Australia.

Example 2.6.2.1. *Typical mmCIF definition save frame.*

```
save__exptl.details
  _item_description.description
; Any special information about the experimental work
  prior to the intensity measurement. See also
  _exptl_crystal.preparation.
;
  _item.name                  '_exptl.details'
  _item.category_id           exptl
  _item.mandatory_code        no
  _item_aliases.alias_name    '_exptl_special_details'
  _item_aliases.dictionary    cif_core.dic
  _item_aliases.version       2.0.1
  _item_type.code             text
save_
```

In view of the importance of maintaining continuity between small- and large-molecule crystallography, the extensions in DDL2 have been introduced in a manner that provides the greatest degree of backward compatibility with applications and dictionaries developed on the core DDL1. Like DDL1, DDL2 dictionaries and data files are fully compliant with the underlying syntax rules of the Self-defining Text Archive and Retrieval (STAR) File (Hall, 1991; Hall & Spadaccini, 1994). Although DDL2 uses a different convention to name data items, an alias feature in DDL2 is used to maintain a correspondence with the published data-item names in the core CIF dictionary (Hall *et al.*, 1991; Chapter 4.1).

Since its introduction in 1994, DDL2 has been used to develop the mmCIF dictionary and this has been adopted as the data exchange standard of the Protein Data Bank, the international repository of three-dimensional macromolecular structure data (see Chapter 5.5).

2.6.2. The DDL2 presentation

The syntax used in mmCIF data files and DDL2 dictionaries is a subset of the STAR syntax. Definitions in DDL2 dictionaries are encapsulated in named save frames (see Chapter 2.1 for a description of the STAR File syntax). A save frame is a syntactical element that begins with the `save_` directive and is terminated by another `save_` directive. Save frames are named by appending a text string to the `save_` directive. In the mmCIF dictionary, save frames are used to encapsulate item and category definitions. The mmCIF dictionary is composed of a data block containing thousands of save frames, where each save frame contains a different definition. Save frames appear in DDL2 dictionaries but they are not used in data files. Save frames may not be nested.

DDL2 dictionary definitions typically contain a small number of items that specify the essential features of the item. Example 2.6.2.1 shows a save frame containing the mmCIF definition of the data item `_exptl.details`.

The example definition includes: a description or text definition, the name and category of the item, a code indicating that the item is optional (not mandatory), the name of a related definition in the core CIF dictionary, and a code specifying that the data type is text. A more detailed description of the elements of the dictionary definitions is presented in the following sections.

2.6.3. Overview of the elements of DDL2

The elements of DDL2 provide the organizational framework for building data dictionaries like mmCIF. The role of the DDL is to define which data items may be used to construct the definitions in the data dictionary, and also to define the relationships between these defining data items. The DDL2 attributes are defined in the dictionary presented in Chapter 4.10.

A dictionary language contains no specific information about a discipline, such as macromolecular crystallography; rather, it defines the data items that can be used to describe a discipline. The contents of the mmCIF dictionary are metadata, or data about data. The contents of the DDL are meta-metadata, the data defining the metadata. By design DDL2, like its predecessor DDL1, is quite generic. It defines data items that describe the general features of a data item like a textual description, a data type, a set of examples, a range of permissible values, or perhaps a discrete set of permitted values. Consequently, data modelling using a DDL can be applied in many fields.

The lowest level of organization provided by DDL2 is the description of an individual data item. Collections of related data items are organized in categories. Categories are essentially tables in which each repetition of the group of related items adds a row. The terms category and data item are used here in order to conform with the previous use of these terms by STAR and CIF applications; these terms could be replaced by relation and attribute (or table and column) commonly used to describe the relational model which underlies DDL2.

Within a category, the set of data items determining the uniqueness of their group are designated as key items in the category. No data-item group in a category is allowed to have a set of duplicate values of its key items. Each data item is assigned membership in one or more categories. Parent–child relationships may be specified for items belonging to multiple categories. A parent–child relationship identifies cases in which the same data item, often an important identifier, occurs in different categories. These relationships permit the expression of the very complicated hierarchical data structures required to describe macromolecular structure.

Other levels of organization in addition to category are also supported. Related categories may be collected together in category groups and parent relationships may be specified for these groups. This higher level of association provides a vehicle for organizing a large complicated collection of categories into smaller more relevant and potentially interrelated groups. This effectively provides a chaptering mechanism for large and complicated dictionaries, like mmCIF. Within the level of a category, subcategories of data items may be defined among groups of related data items. The subcategory provides a mechanism for identifying, for example, that the data items month, day, and year collectively define a date.

For categories, subcategories and items, methods may be specified. Methods are computational procedures that are defined and expressed in a programming language (*e.g.* C/C++, Perl or Java) and stored within a dictionary. Among other things, these dictionary methods may be used to calculate a missing value or to check the validity of a particular value.

The highest levels of data organization provided by DDL2 are the data block and the dictionary. The dictionary level collects a set of related definitions into a single unit and provides the attributes for a detailed revision history of the collection.

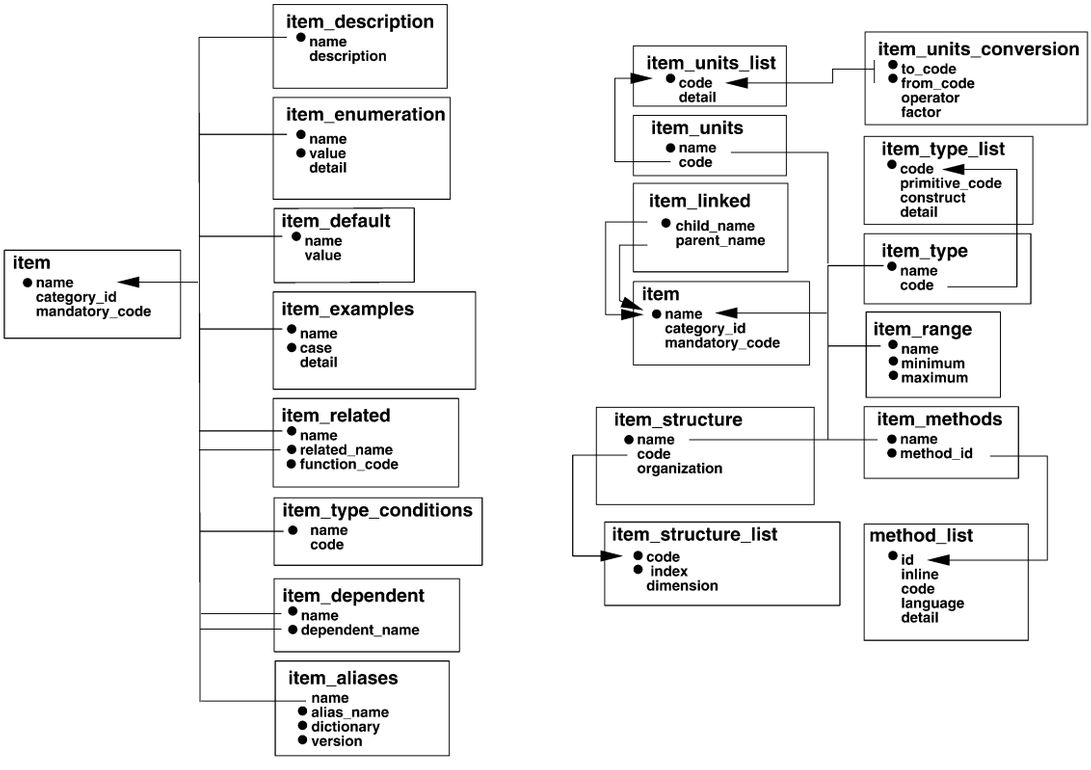

Fig. 2.6.4.1. DDL2 attributes used to specify item information. In this figure, boxes surround data items belonging to the same category. Category identifiers are given in a large typesize and item names are given in a smaller typesize. Parent–child relationships are specified by lines connecting data items with the arrow pointing at the parent item. Key items within a category are marked with a bullet.

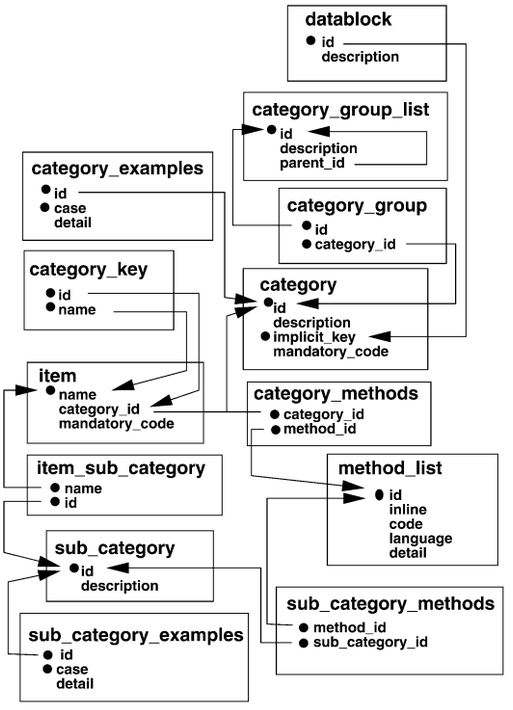

Fig. 2.6.4.2. DDL2 attributes used to specify category, subcategory and category group information. Category identifiers are given in a large typesize and item names are given in a smaller typesize. Parent–child relationships are specified by lines connecting data items with the arrow pointing at the parent item. Key items within a category are marked with a bullet.

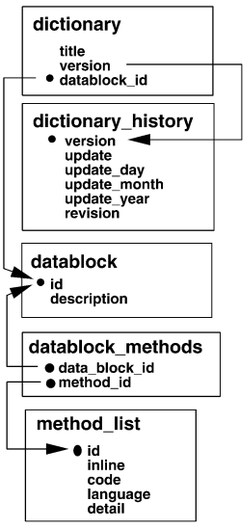

Fig. 2.6.4.3. DDL2 attributes used to specify dictionary and data-block information. Category identifiers are given in a large typesize and item names are given in a smaller typesize. Parent–child relationships are specified by lines connecting data items with the arrow pointing at the parent item. Key items within a category are marked with a bullet.

Fig. 2.6.4.2 illustrates the organization of attributes describing categories, subcategories and category groups. Similarly, Fig. 2.6.4.3 shows the organization of DDL2 attributes at the data-block and dictionary level. All of these attributes are discussed in terms of their application in building data dictionaries in the next section.

2.6.4. DDL2 organization

Figs 2.6.4.1–2.6.4.3 provide schematic illustrations of the definitional features provided by DDL2. These figures represent the elements of the DDL in terms of its own language constructs (*i.e.* categories and the relationships between attributes within those categories). This self-defining presentation has the important consequence of validating the internal consistency of the DDL data model.

Fig. 2.6.4.1 shows the organization of the attributes available to define each data item. These include: a description, examples, data type, allowed values and ranges, default values, internal structural features (*e.g.* vector and matrix properties), units, and other dependency relationships. These DDL attributes are shown as a collection of DDL categories enclosed in boxes in the figure. For instance, the description or textual definition for a data item is specified in a category named ITEM_DESCRIPTION. This DDL category contains the attributes 'name' and 'description'. The attribute 'name' corresponds to the DDL data item `_item_description.name`. This item is the key item in the category named ITEM_DESCRIPTION. In Fig. 2.6.4.1 this is denoted by a bullet. The name attribute in the ITEM_DESCRIPTION category is related to the parent definition of this data item in the category named ITEM. This is reflected in Fig. 2.6.4.1 by the line pointing to the parent data item.

The data-block level ties the contents of a dictionary to the `data_` section in which it is contained. The identifier for the data block and hence the dictionary is added implicitly to the key of each category. This builds into the data model a convenient means for distinguishing similar information recorded in separate data blocks. This feature is important in organizing the results from different crystallographic experiments, each being reported as a separate block of data.

2.6.5. DDL2 dictionary applications

In this section, several examples are presented which illustrate how the elements of the DDL are used to build dictionary definitions. Example 2.6.5.1 shows the definition of the `_citation.journal_abbrev` data item from the mmCIF dictionary.

The category ITEM_DESCRIPTION holds a text description of each data item. The category ITEM holds the item name, category name and a code indicating whether this item is mandatory in any row of this category. The value of the mandatory code is either `yes`, `no` or `implicit`. The `implicit` value is used to indicate that a value is required for the item but it can be derived from the context of the definition and need not be specified. This feature is most often used in DDL2 dictionaries to avoid re-specifying data-item names in each category since these values can be derived from the name of the save frame enclosing the definition. The value of the `_item.name` in the above example is enclosed in quotation marks. This is a requirement of the STAR syntax so that a value containing a data name is not mistaken for a dictionary attribute.

The mmCIF dictionary contains a superset of the definitions that were originally defined in the core CIF dictionary. In order to maintain backward compatibility with original definitions, the ITEM_ALIASES category was introduced to hold the item name, dictionary name and version in which the original definition of an item was published. In this example, the data name used in the core dictionary differs from the example definition only in the period that distinguishes the category and attribute portions of the item name.

The category ITEM_TYPE holds a reference to a data type specified in the ITEM_TYPE_LIST category. A reference to the data type is used here rather than a detailed data-type description in order to avoid repeating the description for other data items. A single list

Example 2.6.5.1. *A rather simple data definition.*

```
save_citation.journal_abbrev
   _item_description.description
; Abbreviated name of the cited journal given in
   the Chemical Abstracts Service Source Index.
;
   _item.name                '_citation.journal_abbrev'
   _item.category_id           citation
   _item.mandatory_code        no
   _item_aliases.alias_name
                             '_citation_journal_abbrev'
   _item_aliases.dictionary  cif_core.dic
   _item_aliases.version     2.0.1
   _item_type.code           line
   _item_examples.case       'J. Mol. Biol.'
save_
```

of data types and associated regular expressions is stored in the ITEM_TYPE_LIST category and this may be referenced by all of the definitions in the dictionary. In the mmCIF dictionary, the codes that are used to describe the data types are generally easy to interpret. In this example, the type code 'line' indicates that a single line of text will be accepted for this data item.

Descriptive examples of data items can be included in the ITEM_EXAMPLES category. In Example 2.6.5.1, one value, `'J. Mol. Biol.'`, is specified, but multiple examples can be provided using a **loop_** directive.

Other DDL item attributes are illustrated in the mmCIF definitions for the items **_cell.length_a** and **_cell.length_a_esd** in Example 2.6.5.2. Some data items are only meaningful as part of a complete set. The ITEM_DEPENDENT category is used to store this type of information. Those additional data items within the irreducible set are listed in this category. In Example 2.6.5.2, the cell lengths in the *b* and *c* directions are defined as dependent items of the cell length in the *a* direction.

The permissible ranges of values for a numerical data item are stored in the ITEM_RANGE category. Each boundary condition is defined as the non-inclusive range between a pair of minimum and maximum values. If multiple boundary conditions are specified using the **loop_** directive, then each condition must be satisfied. A discrete boundary value may be set by assigning the desired boundary value as both the maximum and minimum value. In the above example, the permissible cell-length range is defined as greater than or equal to zero, where the latter boundary condition is specified by setting both extrema as zero.

A number of special relationships may be defined between data items. For some relationships which occur frequently, the source or function of the relationship has been standardized. In the example above, this feature is used to identify that the **_cell.length_a_esd** is the standard uncertainty (estimated standard deviation) of **_cell.length_a**. The recognized relationships are fully described in the DDL definition of the data item **_item_related.function_code** in category ITEM_RELATED. The current list includes the kinds of relationships in Table 2.6.5.1.

Sets of data items within a category may be collected into named subcategories. ITEM_SUB_CATEGORY is used to store the subcategory membership of a data item. In the above example, item **_cell.length_a** is added to the subcategory CELL_LENGTH. The items **_cell.length_b** and **_cell.length_c** are similarly added to this subcategory in their definitions.

The ITEM_UNITS category holds the name of the system of units in which an item is expressed. The name assigned to **_item_units.code** refers to a single list of all of the unit types used in the dictionary. This list is stored in the category ITEM_UNITS_LIST. Conversion factors between different systems of units are provided in the data table stored in the ITEM_UNITS_CONVERSION category.

Example 2.6.5.2. *Definition of a data item that has dependencies and associated items.*

```
save_cell.length_a
   _item_description.description
; Unit-cell length a corresponding to the structure
   reported in angstroms.
;
   _item.name                    '_cell.length_a'
   _item.category_id               cell
   _item.mandatory_code            no
   _item_aliases.alias_name      '_cell_length_a'
   _item_aliases.dictionary        cif_core.dic
   _item_aliases.version           2.0.1
    loop_
   _item_dependent.dependent_name
                                 '_cell.length_b'
                                 '_cell.length_c'
    loop_
   _item_range.maximum
   _item_range.minimum               .    0.0
                                   0.0    0.0
   _item_related.related_name    '_cell.length_a_esd'
   _item_related.function_code     associated_esd
   _item_sub_category.id           cell_length
   _item_type.code                 float
   _item_type_conditions.code      esd
   _item_units.code                angstroms
save_

save_cell.length_a_esd
   _item_description.description
; The standard uncertainty (estimated standard
   deviation) of _cell.length_a.
;
   _item.name                    '_cell.length_a_esd'
   _item.category_id               cell
   _item.mandatory_code            no
   _item_default.value             0.0
    loop_
   _item_dependent.dependent_name
                                 '_cell.length_b_esd'
                                 '_cell.length_c_esd'
   _item_related.related_name    '_cell.length_a'
   _item_related.function_code     associated_value
   _item_sub_category.id           cell_length_esd
   _item_type.code                 float
   _item_units.code                angstroms
save_
```

Example 2.6.5.3 shows the definition of the CELL category from the mmCIF dictionary. The name and textual description of a category are stored in the category named CATEGORY. The item named **_category.mandatory_code** indicates whether the category must appear in any data block based on this dictionary.

The list of data items that uniquely identify each row of a category are stored in the CATEGORY_KEY category. In the example above, the item **_cell.entry_id** is defined as the category key. This item is a reference to the top-level identifier in the mmCIF dictionary, **_entry.id**. Because only a single entry may exist within an mmCIF data block, this choice of category key specifies that only a single row may exist in the CELL category.

Membership in category groups is stored in the category named CATEGORY_GROUP. Each category group must have a corresponding definition in the category CATEGORY_GROUP_LIST. In the above example, the CELL category is assigned to category groups **cell_group** and **inclusive_group**. The former contains categories that describe properties of the crystallographic cell and the latter includes all the categories in the mmCIF dictionary. Organizing categories in category groups is a convenient means of providing a high-level organizational structure for a complex dictionary.

Complete and annotated examples of a category are stored in the CATEGORY_EXAMPLES category. The text of the category example is stored in the item **_category_examples.case** and any associated annotation is stored in the item **_category_examples.detail**.

Table 2.6.5.1. *Relationships defined by* **_item_related.function_code**

Code	Meaning
alternate	The item identified in **_item_related.related_name** is an alternative expression in terms of its application and attributes to the item in this definition
alternate_exclusive	The item identified in **_item_related.related_name** is an alternative expression in terms of its application and attributes to the item in this definition; only one of the alternative forms may be specified
convention	The item identified in **_item_related.related_name** differs from the defined item only in terms of a convention in its expression
conversion_constant	The item identified in **_item_related.related_name** differs from the defined item only by a known constant
conversion_arbitrary	The item identified in **_item_related.related_name** differs from the defined item only by an arbitrary constant
replaces	The defined item replaces the item identified in **_item_related.related_name**
replacedby	The defined item is replaced by the item identified in **_item_related.related_name**
associated_value	The item identified in **_item_related.related_name** is meaningful when associated with the defined item
associated_esd	The item identified in **_item_related.related_name** is the standard uncertainty (estimated standard deviation) of the defined item

Example 2.6.5.3. *Definition of an mmCIF category.*

```
save_CELL
   _category.description
; Data items in the CELL category record details
  about the crystallographic cell parameters.
;
  _category.id                    cell
  _category.mandatory_code        no
  _category_key.name            '_cell.entry_id'
   loop_
  _category_group.id             'inclusive_group'
                                 'cell_group'
  _category_examples.detail
# - - - - - - - - - - - - - - - - - - - - - - - - -
;
Example 1 - based on PDB entry 5HVP and laboratory
            records for the structure corresponding
            to PDB entry 5HVP
;
  _category_examples.case
;
_cell.entry_id                            '5HVP'
_cell.length_a                            58.39
_cell.length_a_esd                         0.05
_cell.length_b                            86.70
_cell.length_b_esd                         0.12
_cell.length_c                            46.27
_cell.length_c_esd                         0.06
_cell.angle_alpha                         90.00
_cell.angle_beta                          90.00
_cell.angle_gamma                         90.00
_cell.volume                             234237
_cell.details
    ; The cell parameters were refined every twenty
      frames during data integration. The cell
      lengths given are the mean of 55 such
      refinements;  the esds given are the root mean
      square deviations of these 55 observations
      from that mean.
    ;
;
save_
```

Example 2.6.5.4 illustrates the definition of a pair of mmCIF categories, CITATION and CITATION_AUTHOR, which share a common data item, **_citation.id**. This example illustrates how an item that occurs in multiple categories may be defined. In the case of the citation identifier, **_citation.id**, the ITEM category is preceded by a **loop_** directive and within this loop all of the definitions of the citation identifier are listed. For instance, the citation identifier is also an item in category CITATION_AUTHOR, where it has the item name **_citation_author.citation_id**. For conformity with the manner in which the core CIF dictionary has been organized, a skeleton definition of the child data item **_citation_author.citation_id** has been included in the dictionary. In fact, this skeleton definition is formally unnecessary.

As a matter of style, the mmCIF dictionary generally defines all of the instances of a data item within the parent definition. Items that are related to the parent definition are also listed in the ITEM_LINKED category. The repetition of a data item in multiple categories gives rise to parent–child relationships between such definitions. These relationships are stored in the ITEM_LINKED category. In Example 2.6.5.4, this category stores the list of data items that are children of the citation identifier **_citation.id**. These include **_citation_author.citation_id**, **_citation_editor.citation_id** and **_software.citation_id**.

2.6.6. Detailed DDL2 specifications

DDL2 is presented here (Chapter 4.10) in the form of a dictionary that is defined in terms of its own definitional elements. This self-consistent description not only provides a prototype for other application dictionaries, but also provides a mechanism by which the consistency and relational integrity of the DDL data model can be independently verified. DDL2 defines a relatively simple set of organizational elements including data blocks, categories, category groups, subcategories and items. Data dictionaries (*e.g.* mmCIF) apply these elements provided by the DDL to describe the knowledge base of an application domain. The following sections provide detailed specifications of each definitional element of DDL2.

2.6.6.1. DDL2 definitions describing data items

In this section, the DDL2 categories that describe the properties of data items are presented. Figs 2.6.4.1 and 2.6.4.2 illustrate the organization of definitional elements in these categories.

2.6.6.1.1. *ITEM*

The category named ITEM is used to assign membership of data items to categories. This category forms the bridge between the category and data-item levels of abstraction. The key data item in this category is the full data-item name, **_item.name**. This name contains both the category and data-item identifiers, and is thus a unique identifier for the data item. The category identifier, **_item.category_id**, is included in this category as a separate mandatory data item. This has been done to provide an explicit reference to those categories that use the category identifier as a unique identifier.

One could alternatively use the category and item identifiers as the basis for this category rather than the concatenated form of the item name, and thus eliminate the redundant specification of the

Example 2.6.5.4. *Related categories linked by parent–child relationships.*

```
save_CITATION
_category.description
; Data items in the CITATION category record details
  about the literature cited as being relevant to
  the contents of the data block.
;
_category.id                  citation
_category.mandatory_code      no
_category_key.name          '_citation.id'
 loop_ _category_group.id    'inclusive_group'
                             'citation_group'
#      ---------  abbreviated definition  ----------
save_

save__citation.id
  _item_description.description
; The value of _citation.id must uniquely identify a
  record in the CITATION list. The _citation.id
  'primary' should be used to indicate the citation
  that the author(s) consider to be the most
  pertinent to the contents of the data block.
;
 loop_       _item.name
             _item.category_id
             _item.mandatory_code
'_citation.id'                   citation         yes
'_citation_author.citation_id'   citation_author  yes
'_citation_editor.citation_id'   citation_editor  yes
'_software.citation_id'          software         yes
_item_aliases.alias_name    '_citation_id'
_item_aliases.dictionary     cif_core.dic
_item_aliases.version        2.0.1
 loop_       _item_linked.child_name
             _item_linked.parent_name
'_citation_author.citation_id'   '_citation.id'
'_citation_editor.citation_id'   '_citation.id'
'_software.citation_id'          '_citation.id'
_item_type.code              code
 loop_ _item_examples.case       'primary'   '1'   '2'
save_

save_CITATION_AUTHOR
_category.description
; Data items in the CITATION_AUTHOR category record
  details about the authors associated with the
  citations in the CITATION list.
;
_category.id                  citation_author
_category.mandatory_code      no
 loop_
_category_key.name    '_citation_author.citation_id'
                      '_citation_author.name'
 loop_ _category_group.id     'inclusive_group'
                              'citation_group'
#      --------- abbreviated definition  ----------
save_

save__citation_author.citation_id
_item_description.description
; This data item is a pointer to _citation.id in the
  CITATION category.
;
_item.name              '_citation_author.citation_id'
_item.mandatory_code         yes
_item_aliases.alias_name
                       '_citation_author_citation_id'
_item_aliases.dictionary     cif_core.dic
_item_aliases.version        2.0.1
save_
```

category identifier. The full name has been used here in order to provide compatibility with existing applications.

The item category also includes a code to indicate whether a data item is mandatory in a category and therefore must be included in any tuple of items in the category. This code, **_item.mandatory_code**, may have three values: `yes`, `no` and `implicit`. This last named value indicates that the item is mandatory, but that the value of this item may be derived from the context. In the case of an item name or a category identifier, these values can be obtained from the current save-frame name. Implicit specification dramatically simplifies the appearance of each dictionary definition because it avoids the repeated declaration of item names and category identifiers that are basis components or the unique identifiers for most categories.

Although the data item **_item.name** is the basis for all of the item-level categories, its definition and properties need only be specified at a single point. Here, the data items that occur in multiple categories are defined only in the parent category. In certain situations, a child data item may be used in a manner which requires a description distinct from the parent data item. For instance, **_item_linked.parent_name** and **_item_linked.child_name** are both data-item names as well as children of **_item.name**, but clearly the manner in which these items are used in the ITEM_LINKED category requires additional description. It is important to note that although the design of this DDL supports the definition of data items in multiple categories within the parent category, it is also possible to provide separate complete definitions within each category.

2.6.6.1.2. *ITEM_ALIASES*

The DDL category ITEM_ALIASES defines the alias names that can be substituted for a data-item name. The alias mechanism also provides a means of identifying items by names other than those that follow the naming conventions used in this DDL. This feature should be used primarily to guarantee the stability of names defined in previously published dictionaries. The items **_item_aliases.name**, **_item_aliases.dictionary** and **_item_aliases.version** form the key for this category. The items **_item_aliases.dictionary** and **_item_aliases.version** are provided to distinguish between dictionaries and different versions of the same dictionary. Any number of unique alias names can be defined for a data item.

2.6.6.1.3. *ITEM_DEFAULT*

The DDL category ITEM_DEFAULT holds default values assigned to data items. Default data values are specified in item **_item_default.value**. Default values are assigned to data items that are not declared within a category. The key item for this category, **_item_default.name**, is a child of **_item.name**. A single default value may be specified for a data item.

2.6.6.1.4. *ITEM_DEPENDENT*

The ITEM_DEPENDENT category defines dependency relationships among data items within a category. Each data item on which a particular data item depends is specified as an item **_item_dependent.dependent_name**. For a data item to be considered completely defined, each of its dependent data items must also be specified.

2.6.6.1.5. *ITEM_DESCRIPTION*

The DDL category ITEM_DESCRIPTION holds a description for each data item. The key item for this category is **_item_description.name**, which is defined in the parent category ITEM. The text of the item description is held by data item **_item_description.description**. A single description may be provided for each data item.

2.6.6.1.6. *ITEM_ENUMERATION*

The DDL category ITEM_ENUMERATION holds lists of permissible values for a data item. Each enumerated value is specified in item **_item_enumeration.value**, each of which may have an associated description item **_item_enumeration.detail**. The combination of items **_item_enumeration.name** and **_item_enumeration.value** form the key for this category. The parent definition of the former item is defined in the category ITEM. Multiple unique enumeration values may be specified for each data item.

2.6.6.1.7. *ITEM_EXAMPLES*

The DDL category ITEM_EXAMPLES is provided to hold examples associated with individual data items. An example specification consists of the text of the example, **_item_examples.case**, and an optional comment item, **_item_examples.detail**, which can be used to qualify the example. Multiple examples may be provided for each item.

2.6.6.1.8. *ITEM_LINKED*

The ITEM_LINKED category defines parent–child relationships between data items. This provides the mechanism for specifying the relationships between data items that may exist in multiple categories. Link relationships are most commonly defined between key items, which form the keys for many different categories.

In the DDL definition, all child relationships are expressed within the parent category.

Because the item **_item_linked.parent_name** has been defined as an implicit item, the child relationships can be specified most economically in the parent category where the parent item name can be automatically inferred. If link relationships are specified in a child category, then both parent and child item names must be specified.

Both parent and child item names in this category are children of **_item.name**, which ensures that all link relationships can be properly resolved. However, it is possible to define cyclical link relationships within this category. Any implementation of this DDL category should include a method to check for the existence of such pathological cases.

2.6.6.1.9. *ITEM_METHODS*

The ITEM_METHODS category is used to associate method identifiers with data items. Any number of unique method identifiers may be associated with a data item. The method identifiers reference the full method definitions in the parent METHOD_LIST category.

2.6.6.1.10. *ITEM_RANGE*

The ITEM_RANGE category defines a restricted range of permissible values for a data item. The restrictions are specified as one or more sets of the items **_item_range.minimum** and **_item_range.maximum**. These items give the lower and upper bounds for a permissible range. To specify that an item value may be equal to the upper or lower bound or a range, the minimum and maximum values of the range are equated. The special STAR value indicating that a data value is not appropriate (denoted by a period, '.') can be used to avoid expressing an upper or lower bound value. When limits are applied to character data, comparisons are made following the collating sequence of the character set. When limits are applied to abstract data types, methods must be provided to define any comparison operations that must be performed to check the boundary conditions.

2.6.6.1.11. *ITEM_RELATED*

The ITEM_RELATED category describes specific relationships that exist between data items. These relationships are distinct from the parent–child relationships that are expressed in the category. The related item is identified as the item **_item_related.related_name** that is a child of **_item.name**.

Item relationships defined by **_item_related.function_code** in this category include some of the following (Table 2.6.5.1): an item is related to another item by a conversion factor; an item is a replacement for another item; an item is replaced by another item; an item is an alternative expression of an item; items which differ only in some convention of their expression; and items which express a set of related characteristics. One can also identify whether the declaration of an item is mutually exclusive with its alternative item. Multiple related items can be associated with each data item and multiple relationship codes can be specified for each related item.

2.6.6.1.12. *ITEM_STRUCTURE*

The ITEM_STRUCTURE category holds a code which identifies a structure definition that is associated with a data item. A structure in this context is a reusable matrix or vector definition declared in category ITEM_STRUCTURE_LIST. The data item **_item_structure.code** is a child of the item **_item_structure_list.code**. The item **_item_structure.code** provides an indirect reference into the list of structure-type definitions in category ITEM_STRUCTURE_LIST. The **_item_structure.organization** item describes the row/column precedence of the matrix organization.

2.6.6.1.13. *ITEM_STRUCTURE_LIST*

The ITEM_STRUCTURE_LIST category holds definitions of matrices and vectors that can be associated with data items. A component of the key for this category is **_item_type_list.code**, which is referenced by **_item_structure.code** to assign a structure type to a data item. The definition of a structure involves the specification of a length for each dimension of the matrix structure. The combination of items **_item_structure_list.code** and **_item_structure_list.index** forms the key for this category. The latter index item is the identifier for the dimension, hence multiple unique dimensions can be specified for each structure code. The length of each dimension is assigned to **_item_structure_list.dimension**.

2.6.6.1.14. *ITEM_SUB_CATEGORY*

The ITEM_SUB_CATEGORY category is used to assign subcategory membership for data items. A data item may belong to any number of subcategories. Each subcategory must be defined in a category named SUB_CATEGORY.

2.6.6.1.15. *ITEM_TYPE*

The ITEM_TYPE category holds a code that identifies the data type of each data item. The data item **_item_type.code** is a child of the item **_item_type_list.code**. Data-type definitions are actually made in the ITEM_TYPE_LIST parent category. The item **_item_type.code** provides an indirect reference into the list of data-type definitions in category ITEM_TYPE_LIST. This indirect

reference is provided as a convenience to avoid the redeclaration of the full data-type specification for each data item. The key item for this category is `_item_type.name`, which is defined in the parent category ITEM. Only one data type may be specified for a data item.

2.6.6.1.16. *ITEM_TYPE_CONDITIONS*

The category ITEM_TYPE_CONDITIONS defines special conditions applied to a data-item type. This category has been included in order to comply with previous applications of STAR and CIF. Since the constructions that are embodied in this category are antithetical to the data model that underlies DDL2, it is recommended that this category only be used for the purpose of parsing existing data files and dictionaries.

2.6.6.1.17. *ITEM_TYPE_LIST*

The ITEM_TYPE_LIST category holds the list of item data-type definitions. The key item in this category is `_item_type_list.code`. Data types are associated with data items by references to this key from the ITEM_TYPE category. One of the data-type codes defined in this category must be assigned to each data item.

The definition of a data type consists of the specification of the item's primitive type and a regular expression that defines the pattern that must be matched by any occurrence of the item. The primitive type code, `_item_type_list.primitive_code`, can assume values of `char`, `uchar`, `numb` and `null`. This code is provided for backward compatibility with STAR and CIF applications that employ loose data typing. The data item `_item_type_list.construct` holds the regular expression that must be matched by the data type. Simple regular expressions can be used to define character fields of restricted width, floating-point and integer formats.

Molecular Information File (MIF) applications (Allen *et al.*, 1995) have extended the notion of the regular expression to include data-item components. This permits the construction of complex data items from one or more component data items using regular expression algebra. These extended regular expressions are defined in the category ITEM_TYPE_CONDITIONS.

Example 2.6.6.1 illustrates the data types that are defined within this DDL. The DDL uses a number of character data types which have subtly different definitions. For instance, the data type identified as `code` defines a single-word character string; `char` extends the code type with the addition of a white-space character; and `text` extends the `char` type with the addition of a newline character. Two special character data types `name` and `idname` are used to define the full STAR data name and the STAR name components, respectively. The data type `any` is used to match any potential data type. This type is used for data items that may hold a variety of data types. The data type `int` is defined as one or more decimal digits and the `yyyy-mm-dd` type defines a date string.

2.6.6.1.18. *ITEM_UNITS*

The ITEM_UNITS category holds a code that identifies the system of units in which a data item is expressed. The data item `_item_units.code` is a child of the item `_item_units_list.code`. Unit definitions are actually made in the ITEM_UNITS_LIST parent category. The item `_item_units.code` provides an indirect reference into the list of data-type definitions in category ITEM_UNITS_LIST. This indirect reference is provided as a convenience to avoid the redeclaration of the full data-type specification for each data item. The key item for this category is `_item_units.name`, which is defined in the parent category ITEM. Only one type of unit may be specified for a data item.

Example 2.6.6.1. *The description of permitted data types in the DDL2 dictionary.*

```
#               DATA TYPE CONVERSION TABLE
#               --------------------------
#
     loop_
     _item_type_list.code
     _item_type_list.primitive_code
     _item_type_list.detail
     _item_type_list.construct


code  char  'A single word'               '[^\t\n "]*'
char  char  'A single line of text'       '[^\n]*'
text  char  'Text which may span lines'   '.*'
int   numb  'Unsigned integer data'       '[0-9]+'
name  uchar 'A data item name'
                '_[_A-Za-z0-9]+[.][][_A-Za-z0-9%/-]+'

idname uchar
     'A data item name component or identifier'
                                          '[_A-Za-z0-9]+'
any   char  'Any data type'                 '.*'
yyyy-mm-dd
      char  'A date format'
      '[0-9][0-9][0-9][0-9]-[0-9]?[0-9]-[0-9][0-9]'
```

2.6.6.1.19. *ITEM_UNITS_CONVERSION*

The ITEM_UNITS_CONVERSION category holds a table of conversion factors between the systems of units described in the ITEM_UNITS_LIST category. The systems of units are identified by a `*.from_code` and a `*.to_code`, which are both children of the item `_item_units_list.code`. The conversion is defined in terms of an arithmetic operator and a conversion factor, `_item_units_conversion.operator` and `_item_units_conversion.factor`, respectively.

2.6.6.1.20. *ITEM_UNITS_LIST*

The ITEM_UNITS_LIST category holds the descriptions of systems of physical units. The key item in this category is `_item_units_list.code`. Units are assigned to data items by references to this key from the ITEM_UNITS category.

2.6.6.2. DDL2 definitions describing categories

In this section, the DDL definitions that describe the properties of categories, category groups and subcategories are presented. Fig. 2.6.4.2 illustrates the organization of these categories.

2.6.6.2.1. *CATEGORY*

The category named CATEGORY contains the data items that describe the properties of collections of related data items. A DDL category is essentially a table. In this category the characteristics of the table as a whole are defined. This category includes the data items `_category.id` to identify a category name; `_category.description` to describe a category; `_category.mandatory_code` to indicate whether the category must appear in a data block; and `_category.implicit_key`, which can be used to merge like categories between data blocks. The category identifier `_category.id` is a component of the key in most of the DDL categories in this section. The parent definition of the category identifier and all its child relationships are defined in this category.

Because special rules exist in the STAR grammar for the specification of data items that belong to a common category, the organization of data items within categories has a significant influence on how these items may be expressed in a data file. For example, a data category may be specified only once within a STAR data block or save frame, and at any level of a STAR loop structure only data items of a common category may appear.

2.6.6.2.2. *CATEGORY_EXAMPLES*

The category named CATEGORY_EXAMPLES holds examples that apply to an entire category. This typically includes a complete specification of the category with annotations. An example specification consists of the text of the example, **_category_examples.case**, and an optional comment item, **_category_examples.detail**, which can be used to qualify the example. The key for this category includes the items **_category_examples.id** and **_category_examples.case**. The former is completely defined in the parent category named CATEGORY.

2.6.6.2.3. *CATEGORY_GROUP*

The category CATEGORY_GROUP names the category groups to which a category belongs. The assignment of a category to a category group is made when the category is defined. Each category group that is specified in this category must also be defined in the parent category, CATEGORY_GROUP_LIST. The basis for this category also includes the category identifier **_category_group.category_id**, which is completely defined in the parent category named CATEGORY.

2.6.6.2.4. *CATEGORY_GROUP_LIST*

The DDL category CATEGORY_GROUP_LIST holds data items that define category groups. Category groups are collections of related categories. Parent–child relationships may be defined for these groups. The specification of category groups and the relationships between these groups allow a complicated collection of categories to be organized into a hierarchy of more relevant groups. This higher level of structure is essential for large application dictionaries that may contain hundreds of category definitions.

The category CATEGORY_GROUP_LIST holds the description of each category group, **_category_group_list.description**, and an optional identifier of the parent group, **_category_group_list.parent_id**. Category groups can be formed from collections of base categories, those categories that hold data. Category groups can also be formed from collections of base categories and category groups.

Example 2.6.6.2 illustrates the category groups that are defined in this DDL. These include the group of categories that define categories, the group of categories defining data items and the group of categories that define properties of the dictionary. An additional compliance group is also defined for categories that are included specifically for compliance with previous versions of DDL. Each of these category groups is defined as a child of the group named **ddl_group** to which all of the base DDL categories belong.

2.6.6.2.5. *CATEGORY_KEY*

The category CATEGORY_KEY identifies the data items within a category that form the basis for the category. The category basis uniquely identifies each group or tuple of items in the category. In the analogy of the category as a table, no row in a table may have duplicate values for its key data items.

Example 2.6.6.2. *Category groups defined in the DDL2 dictionary.*

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

     'ddl_group'              .
;
     Component categories of the macromolecular DDL.
;
     'datablock_group'        'ddl_group'
;
     Categories that describe the characteristics of
     data blocks.
;
     'category_group'         'ddl_group'
;
     Categories that describe the characteristics of
     categories.
;
     'sub_category_group'     'ddl_group'
;
     Categories that describe the characteristics of
     subcategories.
;
     'item_group'             'ddl_group'
;
     Categories that describe the characteristics of
     data items.
;
     'dictionary_group'       'ddl_group'
;
     Categories that describe the dictionary.
;
     'compliance_group'       'ddl_group'
;
     Categories that are retained specifically for
     compliance with older versions of the DDL.
;
```

The choice of basis has important consequences in the specification of a category. It is important to ensure that the key items that form the category basis can unambiguously identify any tuple of data items within the category. If this is not the case, then it may not be possible to reliably recover data items that are stored in the category. Because key items are required to address each tuple of items in a category, key items are considered mandatory items in the category.

It is interesting to note how the key data items have been selected for the categories that define the DDL, and how this choice of key items influences the structure of the DDL dictionary. In the DDL category CATEGORY_KEY, the basis includes both the identifier for the category, **_category_key.id**, and the name of the key data item, **_category_key.name**. This choice of basis allows for any unique groups of items in a category to be defined as key items. Duplicate key-item values within a category are forbidden by the data model. In the DDL category ITEM_TYPE, the basis includes only the identifier for the item name, **_item_type.name**. This choice of basis has the desired effect of limiting the specification of item data type, **_item_type.code**, to a single choice for each data item.

2.6.6.2.6. *CATEGORY_METHODS*

The CATEGORY_METHODS category is used to associate method identifiers with categories. Any number of unique method identifiers may be associated with a category. The method identifiers reference the full method definitions in the parent METHOD_LIST category.

2.6.6.2.7. *SUB_CATEGORY*

The category SUB_CATEGORY provides data items to describe a subcategory and to associate a procedure with the subcategory (see Section 2.6.6.2.9). A subcategory is a set of data items within a category that have a particular association. A typical example would be a triad of positional coordinates *x*, *y*, *z* that are collectively assigned to a 'cartesian' subcategory.

2.6.6.2.8. *SUB_CATEGORY_EXAMPLES*

The DDL category SUB_CATEGORY_EXAMPLES holds examples of a subcategory. A subcategory example might illustrate valid instances of the items comprising the subcategory. An example specification contains the text of the example, `_sub_category_examples.case`, and an optional comment item, `_sub_category_examples.detail`, that can be used to qualify the example. The key for this category includes the items `_sub_category_examples.id` and `_sub_category_examples.case`. This compound basis permits multiple unique examples to be provided for each subcategory.

2.6.6.2.9. *SUB_CATEGORY_METHODS*

The SUB_CATEGORY_METHODS category is used to associate method identifiers with subcategories. Any number of unique method identifiers may be associated with a subcategory. The method identifiers reference the full method definitions in the parent METHOD_LIST category.

The procedure that is identified as `_sub_category_methods.method_id` may be used to validate the subcategory identified as `_sub_category_methods.sub_category_id`. Subcategory validation may be required in instances where conditions are placed on the values of data items within the subcategory that are more restrictive than those associated with each component data item. A simple example of such a restriction would be a normalization restriction on the components of a subcategory. Any procedure referenced in this category must also be defined in the category METHOD_LIST.

2.6.6.3. DDL2 definitions describing methods

In this section, the DDL categories that define the methods associated with data blocks, categories, subcategories and items are presented. Figs. 2.6.4.1, 2.6.4.2 and 2.6.4.3 illustrate the relationships between the method categories and other DDL categories.

2.6.6.3.1. *METHOD_LIST*

The METHOD_LIST category defines methods that can be associated with data blocks, categories, subcategories and items. This category attempts to capture only the essential information required to define these methods, without defining any implementation details. The implementation details are appropriately left to application-dictionary developers. It is assumed here that, within a domain of dictionaries, a consistent method interface will be adopted that is tailored to the requirements of that domain. This of course complicates the sharing of methods between domains; however, it would be impossible at this time to define an implementation strategy inside the DDL that would even begin to satisfy the diverse requirements of potential DDL users. Consequently, the definition of each method is limited to: its unique identifier, `_method_list.id`; a textual description, `_method_list.detail`; the source text of the method, `_method_list.inline`; the name of the language in which the method is expressed, `_method_list.language`; and a code to identify the purpose of the method, `_method_list.code`.

2.6.6.4. DDL2 definitions describing dictionaries and data blocks

In this section, the DDL categories that describe the characteristics of dictionaries and data blocks are presented. In this context, a dictionary is defined as a group of related definitions within a STAR data block. Fig. 2.6.4.3 illustrates the organization for these categories.

2.6.6.4.1. *DATABLOCK*

The DATABLOCK category holds the essential identifying information for a data block: the name of the data block, `_datablock.id`; and a description of the block, `_datablock.description`. The `_datablock.id` is the parent identifier for both `_category.implicit_key` and `_dictionary.datablock_id`. The former guarantees that the identifier for the data block, and hence the dictionary, is added implicitly to the key of each category.

2.6.6.4.2. *DATABLOCK_METHODS*

The DATABLOCK_METHODS category may be associated with a data block. The method identifiers reference the full method definitions in the parent METHOD_LIST category.

2.6.6.4.3. *DICTIONARY*

The DICTIONARY category holds the essential identifying information for a data dictionary. The items recorded in this category include the title for the dictionary, `_dictionary.title`, the current version identifier, `_dictionary.version`, and the data-block identifier in which the dictionary is defined, `_dictionary.datablock_id`. The version identifier references the parent identifier in the DICTIONARY_HISTORY category in which each dictionary revision is described.

2.6.6.4.4. *DICTIONARY_HISTORY*

The DICTIONARY_HISTORY category holds the revision history for a dictionary. Each revision is assigned a version identifier that acts as the key item for the category. Along with the version information, a text description of the revision and date of revision must be specified.

References

Allen, F. H., Barnard, J. M., Cook, A. F. P. & Hall, S. R. (1995). *The Molecular Information File (MIF): core specifications of a new standard format for chemical data. J. Chem. Inf. Comput. Sci.* **35**, 412–427.

Hall, S. R. (1991). *The STAR File: a new format for electronic data transfer and archiving. J. Chem. Inf. Comput. Sci.* **31**, 326–333.

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. & Cook, A. P. F. (1995). *STAR dictionary definition language: initial specification. J. Chem. Inf. Comput. Sci.* **35**, 819–825.

Hall S. R. & Spadaccini, N. (1994). *The STAR File: detailed specifications. J. Chem. Inf. Comput. Sci.* **34**, 505–508.

Westbrook, J. D. & Hall, S. R. (1995). *A dictionary description language for macromolecular structure.* http://ndbserver.rutgers.edu/mmcif/ddl/.