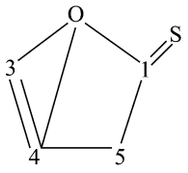


```
loop_
    _display_id
    _display_symbol
    _display_coord_x
    _display_coord_y
    loop_
        _display_conn_id
        _display_conn_symbol
            1  1    180  110    2 1b    stop_
            2  O    140  140    3 1b    4 1b    stop_
            3  3    100  110    4 2b    stop_
            4  4    120   60    5 1b    stop_
            5  5    160   60    1 1b    stop_
            6  S    205  140    1 2b    stop_
```

(*a*)

```
data_display_coord_
    loop_ _name               '_display_coord_x'
                              '_display_coord_y'
    _type                      numb
    _list                      yes
    _list_level                1
    _definition
;   The projected coordinates of the display
    object identified in _display_object and
    _display_symbol.
;

data_display_conn_symbol
    _name                    '_display_conn_symbol'
    _type                      char
    _list                      yes
    _list_level                2
    _definition
;   The symbol code for the bond connection to the
    displayed objects identified by _display_id and
    _display_conn_id. The symbol codes and
    descriptions are stored in the validation file
    'mif_core_bonds.val'.
;
```

(*b*)

Fig. 2.5.6.1. (*a*) Hypothetical example and (*b*) the typical definition of nested items.

by white space, quotation characters or, in the case of multi-line text, by semicolon characters in column 1 of a line.

The attribute **_type_conditions** is used to identify application-specific conditions that apply to the typing of data items. This attribute is used to alert parsing software of string constructions that may disrupt the normal identification and validation of data types. The permitted attribute states are `none`, `esd` (and its preferred synonym `su`) and `seq`. In the definition example in Fig. 2.5.5.3 the code `esd` signals that cell-length values are measurements and may have a standard uncertainty value appended in parentheses. Fig 2.5.5.8 indicates the use of the code `su` for the same purpose. A full description of **_type_conditions** states is given in the DDL1 dictionary in Chapter 4.9.

If a data item consists of a concatenation of values, the attribute **_type_construct** may be used to specify the encoding algorithm for the component values. The language used for this algorithm is POSIX regex (IEEE, 1991). The specification of **_type_construct** can include the data names of other defined items. Validation software interprets this attribute as format specifications and constraints, and expands the embedded data items according to their

```
data_publ_date
    _name                    '_publ_date'
    _type                     char
    _type_construct
        (_publ_year)/(_publ_month)/(_publ_day)
    _definition
;   Specifies the syntax for declaring the
    publication date.
;

data_publ_year
    _name                    '_publ_year'
    _type                    char
    _type_construct          19|20[0-9][0-9]
    _definition
;   Specifies how the publication year will
    appear. Syntax is valid only for publication
    in the 20th and 21st centuries!
;
```

Fig. 2.5.6.2. Definition example: type constructions.

separate definition. For example, the chronological date is a composite of day, month and year. These may be represented in many different ways and the **_type_construct** attribute enables a specific date representation to be defined. The two definitions shown in Fig. 2.5.6.2 illustrate how a date value of the construction '1995/03/25' may be embedded into the dictionary definitions (the month and day definitions have been omitted here for brevity).

Units attributes specify the measurement units permitted for a numerical data item. The attribute **_units** specifies a code that uniquely identifies the measurement units. A description of the units identified by the code is given by the attribute **_units_detail**. Typical uses of this attribute are shown in Fig. 2.5.5.8.

2.5.6.4. Relational attributes

This class of attributes is used to describe special relationships and dependencies between data items. These attributes are machine parsable and are formally defined in the DDL1 dictionary in Chapter 4.9. They are

```
_category
_list_link_child
_list_link_parent
_list_mandatory
_list_reference
_list_uniqueness
_related_item
_related_function
```

The attribute **_category** is used to specify to which group, or basis set, a data item belongs. The value of this attribute is a name string that identifies the group and it is usually the leading part of the tags for all items in this group. Data items in a list must have the same category value. Data items in the same category may, however, be divided into different lists, provided each list contains an appropriate key data item (see **_list_reference** below).

List-link attributes are used to specify dependencies between data items in different lists. The attribute **_list_link_parent** identifies an item in another list from which the defined item was derived. The parent data item, or items in the case of irreducible sets, must have a unique value within its own list. The **_list_link_child** attribute is declared in the definition of a parent item and identifies items in other lists that depend implicitly on the defined data item being present in the same data instantiation. The functionalities of these two attributes mirror each other. The parent item must be present in any data instantiation containing the child items, but not the converse. The definition examples in Figs 2.5.5.6 and 2.5.5.7 illustrate the use of these attributes.

The attributes **_list_mandatory** and **_list_reference** are also connected to each other. The former signals (with values of yes or no) whether the presence of a data item is essential to preserving the validity of a list of items. The latter attribute identifies the data item in the list that provides the unique key value to each packet or row of items in a list. A packet is made up of listed values, one for each item in the name list (*i.e.* the packet size matches the number of data names at the head of the list). The **_list_reference** attribute identifies items that are the keys to specific packets in the list. In Example 2 of Fig. 2.5.5.5 the key item is **_atom_site_label** and the listed labels S1, S2, N1 and C1 must be unique.

The attribute **_list_uniqueness** is used to identify items that must be unique for a list to be valid and accessible. This attribute is similar to **_list_reference** except that it appears only in a definition in which **_list_mandatory** is set to yes. This simplifies validation because it may be used as the placeholder for all items that jointly identify the uniqueness of a list packet. This is in contrast to the attribute **_list_reference**, which appears in the definition of every item dependent on this item.

Relational attributes are used to link equivalent data. The **_related_item** attribute identifies items related to the defined data item. The nature of this relationship is specified with the **_related_function** attribute according to the restricted value states of alternate, convention, conversion and replace. The definition of these states is detailed in Chapter 4.9. Relational attributes are used to provide equivalent data items, to replace definitions when definitions are superseded or to change access pathways. These facilities are for archives, because they enable old data to be accessed and the associated definitions to remain in a dictionary even when superseded by new definitions. The old and new definitions are linked by these attributes so that all related data items can be validated and accessed.

2.5.6.5. Dictionary registration attributes

This class of attributes is used to register the dictionary version and audit information. They are

_dictionary_history
_dictionary_name
_dictionary_update
_dictionary_version

Dictionary attributes are used to record the creation and update history of a dictionary. The attribute **_dictionary_history** specifies the entry and update information of the dictionary, and **_dictionary_name** specifies the generic name of the electronic file containing the dictionary (the actual name of the file can vary from site to site). The attributes **_dictionary_version** and **_dictionary_update** specify the version number and the date of the last change in the dictionary. Both items represent important external reference information. An example application of these attributes is shown in Fig. 2.5.5.2.

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.

Bourne, P. E., Berman, H. M., McMahon, B., Watenpaugh, K. D., Westbrook, J. D. & Fitzgerald, P. M. D. (1997). *Macromolecular Crystallographic Information File. Methods Enzymol.* **277**, 571–590.

Gray, P. M. D., Kulkarni, K. G. & Paton, N. W. (1992). *Object oriented databases*. New York: Prentice Hall.

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.

IEEE (1991). *IEEE Standard for Information Technology – Portable Operating System Interface (POSIX) – Part 2: Shell and utilities*, Vol. 1, IEEE Standard 1003.2-1992. New York: The Institute of Electrical and Electronic Engineers, Inc.

Kim, W. (1990). *Introduction to object oriented databases*. Boston: MIT Press.

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