International Tables for Crystallography (2006). Vol. G, Section 3.1.9, pp. 88–89.

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 $n.m.l\ldots$, 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

references