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
```

references