cell, but it should be emphasized from a computer-programming viewpoint that this is coincidental; the attributes that constrain the value of this data item (and its relationship to others such as `_atom_site_fract_y` and `_atom_site_fract_z`) must be obtained from the dictionary and not otherwise inferred.

(8) *Comment:* In practice data names described in a DDL2 dictionary are constructed with a period character separating their specific function from the name of the category to which they have been assigned. In the absence of a dictionary file, this convention permits the inference that the data item with name `_atom_site.fract_x` will appear in the same looped list as other items with names beginning `_atom_site.`, and that all such items belong to the same category.

2.2.7.4.5. *Name space*

(9) The intention of the maintainers of public CIF dictionaries is to formulate a single authoritative set of data names for each CIF dialect (*i.e.* DDL1 and DDL2), thus facilitating the reliable archive and interchange of crystallographic data. However, it is also permissible for users to introduce local data names into a CIF. Two mechanisms exist to reduce the danger of collision of data names that are not incorporated into public dictionaries.

(10) The character string `[local]` (including the literal bracket characters) is *reserved* for local use. That is, no public dictionary will define a data name that includes this string. This allows experimentation with data items in a strictly local context, *i.e.* in cases where the CIF is not intended for interchange with any other user.

(11) Where CIFs including local data items are expected to enjoy a public circulation, authors may register a *reserved prefix* for their sole use. The registry is available on the web at http://www.iucr.org/iucr-top/cif/spec/reserved.html.

A reserved prefix, *e.g. foo*, must be used in the following ways:

(i) If the data file contains items defined in a DDL1 dictionary, the local data names assigned under the reserved prefix must contain it as their first component, *e.g.* `_foo_atom_site_my_item`.

(ii) If the data file contains items defined in a DDL2 dictionary, then the reserved prefix must be:

(*a*) the first component of data names in a category defined for local use, *e.g.* `_foo_my_category.my_item`.

(*b*) the first component following the period character in a data name describing a new item in a category already defined in a public dictionary, *e.g.* `_atom_site.foo_my_item`.

(12) There is no syntactic property identifying such a reserved prefix, so that software validating or otherwise handling such local data names must scan the entire registry and match registered prefixes against the indicated components of data names. Note that reserved prefixes may not themselves contain underscore characters.

2.2.7.4.6. *Note on handling of units*

(13) The published specification for CIF version 1.0 permitted data values expressed in different units to be tagged by variant data names (Hall *et al.*, 1991, p. 657):

> ... Many numeric fields contain data for which the units must be known. Each CIF data item has a default units code which is stated in the CIF Dictionary. If a data item is not stored in the default units, the units code is appended to the data name. For example, the default units for a crystal cell dimension are ångströms. If it is necessary to include this data item in a CIF with the units of picometres, the data name of `_cell_length_a` is replaced by `_cell_length_a_pm`. Only those units defined in the CIF Dictionary are acceptable. The default units, except for the ångström, conform to the SI Standard adopted by the IUCr.

This approach is deprecated and has not been supported by any official CIF dictionary published subsequent to version 1.0 of the core. All data values must be expressed in the single unit assigned in the associated dictionary.

A small number of archived CIFs exist with variant data names as permitted by the above clause. If it is necessary to validate them against versions of the core dictionary subsequent to version 1.0, the formal compatibility dictionary cif_compat.dic (ftp://ftp.iucr.org/cifdics/cif_compat.dic) may be used for the purpose. *No other use should be made of this dictionary.*

2.2.7.4.7. *Data-value semantics*

(14) The STAR syntax permits retrieval of data by simply requesting a specific data name within a specific data block. Prior knowledge about data type (*e.g.* text or numbers), whether the item is looped or whether the item exists in the file at all is unnecessary. However, applications in general need to know data type, valid ranges of values and relationships between data items, and a program designer needs to know the purpose of the data item (*i.e.* what physical quantity or internal book-keeping function it represents). While such semantic information may be defined informally for local data items (ones not intended for exchange between different users or software applications), formal descriptions of the semantics associated with data values are catalogued in data dictionary files. Currently two formalisms (dictionary definition languages) for describing data-value attributes are supported; full specifications of these formalisms (known as DDL1 and DDL2) are provided in Chapters 2.5 and 2.6.

2.2.7.4.7.1. *Data typing*

(15) Four base data types are supported in CIF. These are:

(i) **numb**: a value interpretable as a decimal base number and supplied as an integer, a floating-point number or in scientific notation;

(ii) **char**: a value to be interpreted as character or text data (where the value contains white-space characters, it must be quoted);

(iii) **uchar**: a value to be interpreted as character or text data but in a case-insensitive manner (*i.e.* the values `FOO` and `foo` are to be taken as identical);

(iv) **null**: a special data type associated with items for which no definite value may be stored in computer memory. It is the type associated with the special character literal values `?` (query mark) and `.` (full point), which may appear as values for any data item within a data file (see Section 2.2.7.4.8 below). It is also the type assigned to items defined in dictionary files that may not occur in data files.

(16) *Comment:* Many applications distinguish between multi-line text fields and character-string values that fit within a single line of text. While this is a convenient practical distinction for coding purposes, formally both manifestations should be regarded as having the same base type, which might be 'char' or 'uchar'. Applications are at liberty to choose whether to define specific multi-line text subtypes, and whether to permit casting between subtypes of a base type. The examples of character-string delimiters in Section 2.2.7.1.4(20) are predicated on an approach that handles all subtypes of character or text data equivalently.

(17) Where the attributes of a data value are not available in a dictionary listing, it may be assumed that a character string inter-

Copyright © 2006 International Union of Crystallography

pretable as a number should be taken to represent an item of type 'numb'. However, an explicit dictionary declaration of type will override such an assumption.

2.2.7.4.7.2. *Subtyping*

(18) The base data types detailed in the previous section are very general and need to be refined for practical application. Refinement of types is to some extent application-dependent, and different subtypes are supported for data items defined by DDL1 and DDL2 dictionary files. The following notes indicate some considerations, but the relevant dictionary files and documentation should be consulted in each case.

(19) *DDL1 dictionaries*. Values of type 'numb' may include a standard uncertainty in the final digit(s) of the number where the associated item definition includes the attribute

```
_type_conditions    esd
```

(or `_type_conditions su`, a synonym introduced to DDL1 in 2005). For example, a value of 34.5(12) means 34.5 with a standard uncertainty of 1.2; it may also be expressed in scientific notation as 3.45E1(12).

(20) *DDL2 dictionaries*. DDL2 provides a number of tags that may be used in a dictionary file to specify subtypes for data items defined by that dictionary alone. Examples of the subtypes specified for the macromolecular CIF dictionary are:

code	identifying code strings or single words
ucode	identifying code strings or single words (case-insensitive)
uchar1	single-character codes (case-insensitive)
uchar3	three-character codes (case-insensitive)
line	character strings forming a single line of text
uline	character strings forming a single line of text (case-insensitive)
text	multi-line text
int	integers
float	floating-point real numbers
yyyy-mm-dd	dates
symop	symmetry operations
any	any type permitted

2.2.7.4.8. *Special generic values*

(21) The unquoted character literals `?` (query mark) and `.` (full point) are special and are valid expressions for any data type.

(22) The value `?` means that the actual value of a requested data item is *unknown*.

(23) The value `.` means that the actual value of a requested data item is *inapplicable*. This is most commonly used in a looped list where a data value is required for syntactic integrity.

2.2.7.4.9. *Embedded data semantics*

(24) The attributes of data items defined in CIF dictionaries serve to direct crystallographic applications in the retrieval, storage and validation of relevant data. In principle, a CIF might include as data items suitably encoded fields representing data suitable for manipulation by text processing, image, spreadsheet, database or other applications. It would be useful to have a formal mechanism allowing a CIF to invoke appropriate content handlers for such data fields; this is under investigation for the next CIF version specification.

2.2.7.4.10. *CIF conventions for special characters in text*

(25) The one existing example of embedded semantics is the text character markup introduced in the CIF version 1.0 specification and summarized in paragraphs (30)–(37) below. The specification is silent on which fields should be interpreted according to these markup conventions, but the published examples suggest that they may be used in any character field in a CIF data file except as prohibited by a dictionary directive. It is intended that the next CIF version specification shall formally declare where such markup may be used.

2.2.7.4.11. *Handling of long lines*

(26) The restriction in line length within CIF requires techniques to handle without semantic loss the content of lines of text exceeding the limit (2048 characters in this revision, 80 characters in the initial CIF specification). The line-folding protocol defined here provides a general mechanism for wrapping lines of text within CIFs to any extent within the overall line-length limit. A specific application where this would be useful is the conversion of lines longer than 80 characters to the CIF version 1.0 limit. This 80-character limit is used in the examples below for illustrative purposes.

These techniques are applied only to the contents of text fields and to comments.

In order to permit such folding, a special semantics is defined for use of the backslash. It is important to understand that this does not change the syntax of CIF version 1.0. All existing CIFs conforming to the CIF version 1.0 specification can be viewed as having exactly the same semantics as they now have. Use of these transformational semantics is optional, but recommended.

In order to avoid confusion between CIFs that have undergone these transformations and those that have not, the special comment beginning with a hash mark immediately followed by a backslash (`#\`) as the last non-blank characters on a line is reserved to mark the beginning of comments created by folding long-line comments, and the special text field beginning with the sequence line termination, semicolon, backslash (`<eol>;\`) as the only non-blank characters on a line is reserved to mark the beginning of text fields created by folding long-line text fields.

The backslash character is used to fold long lines in character strings and comments. Consider a comment which extends beyond column 80. In order to provide a comment with the same meaning which can be fitted into 80-character lines, prefix the comment with the special comment consisting of a hash mark followed by a backslash (`#\`) and the line terminator. Then on new lines take appropriate fragments of the original comment, beginning each fragment with a hash mark and ending all but the last fragment with a backslash. In doing this conversion, check for an original line that ends with a backslash followed only by blanks or tabs. To preserve that backslash in the conversion, add another backslash after it. If the next lexical token (not counting blanks or tabs) is another comment, to avoid fusing this comment with the next comment, be sure to insert a line with just a hash mark.

Similarly, for a character string that extends beyond column 80,

(i) first convert it to be a text field delimited by line termination–semicolon (`<eol>;`) sequences,

(ii) then change the initial line termination–semicolon (`<eol>;`) sequence to line termination–semicolon–backslash–line termination (`<eol>;\<eol>`),

(iii) and break all subsequent lines that do not fit within 80 columns with a trailing backslash. In the course of doing the translation,

(*a*) check for any original text lines that end with a backslash followed only by blanks or tabs;

(*b*) to preserve that backslash in the conversion, add another backslash after it, and then an empty line.

(More formally, the line folding should be done separately and directly on single-line non-semicolon-delimited character strings

references