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

to allow for recognition of the fact that no terminal line termination is intended – see below.)

In order to understand this scheme, suppose the CIF fragment (1) below were considered to have long lines. They could be transformed into (2) as follows:

(1) *Initial CIF*

```
#########################################################
### CIF submission form for Rietveld refinements
###
###                          Version 14 December 1998
###
#########################################################
data_znvodata
_chemical_name_systematic
; zinc dihydroxide divanadate dihydrate
;

_chemical_formula_moiety      'H2 O9 V2 Zn3, 2(H2 O)'
_chemical_formula_sum         'H6 O11 V2 Zn3'
_chemical_formula_weight      480.05
```

(2) *Transformed CIF*

```
#\
############################\
############################
### CIF submission form for Rietveld refinements
###
###                          Version 14 December 1998
###
#########################################################
data_znvodata
_chemical_name_systematic
;\
 zinc dihydroxide divan\
adate dihydrate
;

_chemical_formula_moiety
;\
H2 O9 V2 Zn3, 2(H2 O)\
;
_chemical_formula_sum         'H6 O11 V2 Zn3'
_chemical_formula_weight      480.05
```

In making the transformation from the backslash-folded form to long lines, it is very important to strip trailing blanks before attempting to recognize a backslash as the last character. When reassembling text-field lines, no reassembly should be done except in text fields that begin with the special sequence described above, line termination–semicolon–backslash–line termination, (`<eol>;\<eol>`), so that text fields that happen to contain backslashes but which were not created by folding long lines are not changed. It is also important to remove the trailing backslashes when reassembling long lines. The final line termination–semicolon sequence of a text field takes priority over the reassembly process and ends it, but a trailing backslash on the last line of a text field very nicely conveys the information that no trailing line termination is intended to be included within the character string.

Similarly, when reassembling long-line comments, the reassembly begins with a comment of the form hash–backslash–line termination. The initial hash mark is retained and then a forward scan is made through line terminations and blanks for the next comment, from which the initial hash mark is stripped and then the contents of the comment are appended. If that comment ends with a backslash, the trailing backslash is stripped and the process repeats. Note that the process will be ended by intervening tags, values, data blocks or other non-white-space information, and that the process will not start at all without the special hash–backslash–line termination comment.

Since there are very few, if any, CIFs that contain text fields and comments beginning this way, in most cases it is reasonable to adopt the policy of doing this processing unless it is disabled.

Here is another example of folding. The following three text fields would be equivalent:

```
;C:\foldername\filename
;

;\
C:\foldername\filename
;
```

and

```
;\
C:\foldername\file\
name
;
```

but the following example would be a two-line value where the first line had the value `C:\foldername\file\` and the second had the value `name`:

```
;
C:\foldername\file\
name
;
```

Note that backslashes should not be used to fold lines outside of comments and text fields. That would introduce extraneous characters into the CIF and violate the basic syntax rules. In any case, such action is not necessary.

2.2.7.4.12. *Dictionary compliance*

(27) Dictionary files containing the definitions and attribute sets for the data items contained in a CIF should be identified within the CIF by some or all of the data items

```
_audit_conform_dict_name
_audit_conform_dict_version
_audit_conform_dict_location
```

corresponding to DDL1 dictionaries or

```
_audit_conform.dict_name
_audit_conform.dict_version
_audit_conform.dict_location
```

for DDL2 dictionaries. Where no such information is provided, it may be assumed that the file should conform against the core CIF dictionary.

(28) The `_audit_conform` data items may be looped in cases where more than one dictionary is used to define the items in a CIF and they may include dictionaries of local data items provided such dictionary files have been prepared in accordance with the rules of the appropriate DDL.

(29) A detailed protocol exists for locating, merging and overlaying multiple dictionary files (McMahon *et al.*, 2000) (see Section 3.1.9).

2.2.7.4.13. *CIF markup conventions*

(30) If permitted by the relevant dictionary and if no other indication is present, the contents of a text or character field are assumed to be interpretable as text in English or some other human language. Certain special codes are used to indicate special characters or accented letters not available in the ASCII character set, as listed below.