```
             CYCLOPS Check List
             ------------------
   Dictionary data names  = 2244
   New data names in text =    4
    [1]  Dictionary cif_core.dic 2.0.1 data names =  624
    [2]  Dictionary cif_mm.dic 0.9.0 data names = 1620

Data names NOT in Dictionary           Line Numbers

_blat1 . . . . . . . . . . . . . .      9   11   94  96
                                       181  183  290  296
_blat2 . . . . . . . . . . . . . .     13   15   98  100
                                       185  187  287  293
_dummy_test  . . . . . . . . . . .      5    7   90  92
                                       177  179  201
_rubbish_here. . . . . . . . . . .    431

[1]  Dictionary cif_core_2.0.1.dic
[2]  Dictionary cif_mm.dic
                                       Line Numbers

[2] _atom_site.calc_attached_atom      413
[1] = _atom_site_calc_attached_atom    412
[2] _atom_site.calc_flag  . . . . .    410
[1] = _atom_site_calc_flag             409
[2] _atom_site.fract_x  . . . . . .     38   44   50  390
[1] = _atom_site_fract_x               389
[2] _atom_site.fract_y  . . . . . .     39   45   51  394
[1] = _atom_site_fract_y               393
[2] _atom_site.fract_z  . . . . . .     40   46   52  398
[1] = _atom_site_fract_z               397
[2] _atom_site.id   . . . . . . . .     37   43   49  386
[1] = _atom_site_label                 385
[2] _atom_site.thermal_displace_type   406
[1] = _atom_site_thermal_displace_type 405
[2] _atom_site.type_symbol  . . . .    416  420  424  428
                                       434  438  442  450
[1] = _atom_site_type_symbol           415  419  423  427
                                       433  437  441  449
```

*[later in the validation output file, showing the transition to unreferenced data names . . . ]*

```
[1] _symmetry_cell_setting  . . . .    319
[2] = _symmetry.cell_setting           320
[1] _symmetry_space_group_name_H-M     323
[2] = _symmetry.space_group_name_H-M   324
[1] _symmetry_space_group_name_Hall    327   445
[2] = _symmetry.space_group_name_Hall  328   446

[1]  Dictionary cif_core_2.0.1.dic
[2]  Dictionary cif_mm.dic
                                  Names Not Referenced

[2] _atom_site.aniso_B[1][1]
[2] _atom_site.aniso_B[1][1]_esd
[2] _atom_site.aniso_B[1][2]
```

*[. . . portion of output omitted . . .]*

```
[2] _atom_site.aniso_U[3][3]_esd
[2] _atom_site.attached_hydrogens
[1] = _atom_site_attached_hydrogens
[2] _atom_site.auth_asym_id
[2] _atom_site.auth_atom_id
[2] _atom_site.auth_comp_id
[2] _atom_site.auth_seq_id
[2] _atom_site.B_equiv_geom_mean
[1] = _atom_site_B_equiv_geom_mean
[2] _atom_site.B_equiv_geom_mean_esd
[2] _atom_site.B_iso_or_equiv
[1] = _atom_site_B_iso_or_equiv
[2] _atom_site.B_iso_or_equiv_esd
```

*[. . . remainder of output omitted . . .]*

Fig. 5.3.4.1. Sample output from *CYCLOPS*. The output has been edited and reformatted slightly to fit into the present column width.

respectively. The special character hyphen (‘-’) may also be supplied as an argument to ‘*-i*’ or ‘*-o*’ to indicate standard input or standard output.

Finally, if the operating system supports the passing of environment variables to a program, the names of the input file, output file and dictionary file may be passed through the values of `$CYCLOPS_INPUT_TEXT`, `$CYCLOPS_VALIDATION_OUT` or `$CYCLOPS_CHECK_DICTIONARY`, respectively.

## 5.3.5. File transformation software

This section describes a number of applications that transform an input CIF either to another CIF that contains a subset of the original contents or to other formats suitable for use with general processing tools. (Conversion to other crystallographic data formats is not discussed here.)

### 5.3.5.1. *QUASAR*: a data extractor

The oldest CIF manipulation program is *QUASAR* (Hall & Sievers, 1993), which was described as the prototype CIF application in the original standard specification paper (Hall *et al.*, 1991). Much of the functionality of *QUASAR* has now been included in the *cif2cif* program (Section 5.3.5.2). However, it remains useful as an application in its own right, and so is briefly described here.

#### 5.3.5.1.1. *Purpose*

The program was designed to read a *request list* of data names, to locate the associated data in an input CIF and to output the data in the order of the request list. The output retains local conformance to CIF syntax rules, but the output file may not be strictly CIF conformant. For example, the same data can be requested multiple times and will be reproduced as often as requested in the output stream, a feature forbidden within a legal CIF.

#### 5.3.5.1.2. *Mode of operation*

Written as a pure Fortran77 application, *QUASAR* requires three data streams: a file containing the request list, an input CIF and an output file. In an operating system such as Unix, it is convenient to attach the request list to the standard input channel; the first two lines of the input stream then take the form `star_arc_`*`infile`* and `star_out_`*`outfile`*, where *infile* and *outfile* are the file names of the input and output files, respectively.

The assignment of an output file may be replaced by a line containing `star_log`. When this is done, the program will test the syntactic validity of the input CIF and write any error messages to the standard output channel. In this mode the program may be used as a syntactic validator, although it is more tolerant of certain syntactic errors than *vcif* (Section 5.3.2.1).

#### 5.3.5.1.3. *The request list*

Fig. 5.3.5.1 is an example request list, intended to highlight some of the special features of the way the program operates. Fig. 5.3.5.2 shows an example CIF against which this request list will be tested; Fig. 5.3.5.3 shows the output. Both figures have been modified slightly to fit on the printed page; they are derived from the sample files distributed with the program.

The request list begins with directives specifying the input and output file names (qtest.cif and qtest.out, respectively). The file may contain comments prefaced by a hash character `#`; this is a useful feature for annotating a request list. Another use for such comments is seen in the standard request list distributed to authors for papers published in *Acta Crystallographica*. Here, data names that are *not* normally published are hidden within the request list as comments and may be activated if they occur in a **`publ_manuscript_incl_extra_item`** loop within a CIF (see Section 5.7.2.3).

```
star_arc_qtest.cif
star_out_qtest.out

data_    #<< wild-card block name - accepts first

# request all fractional coord items
   _atom_site_fract_
   _atom_site_label
# capitals to test case insensitivity
   _atom_site_aniso_LABEL
# request something that is not in the CIF
   _dummy
   _atom_site_aniso_U_11

data_P6122
_        #<< this requests all data in this block
```

Fig. 5.3.5.1. An example request list for *QUASAR*.

```
data_P6122

loop_
_atom_type_symbol
_atom_type_oxidation_number
_atom_type_number_in_cell
   # capitals to test case insensitivity
_atom_type_scat_dispersion_REAL
_atom_type_scat_dispersion_imag
_atom_type_scat_source
  S   0  6   .319  .557
        Int_Tab_Vol_III_p202_Tab._3.3.1a
  O   0  6   .047  .032
        Cromer,D.T._&_Mann,J.B._1968_AC_A24,321.
  C   0  20  .017  .009
        Cromer,D.T._&_Mann,J.B._1968_AC_A24,321.
  RU  0  1  -.105   3.296
        Cromer,D.T._&_Mann,J.B._1968_AC_A24,321.

loop_
_atom_site_label
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_U_iso_or_equiv
_atom_site_thermal_displace_type
_atom_site_calc_flag
_atom_site_calc_attached_atom
_atom_site_type_symbol
  s   .20200  .79800  .91667  .030(3)  Uij   ?  ?  s
  o   .49800  .49800  .66667  .02520  Uiso   ?  ?  o
  c1  .48800  .09600  .03800  .03170  Uiso   ?  ?  c

loop_
_atom_site_aniso_label
_atom_site_aniso_U_11
_atom_site_aniso_U_22
_atom_site_aniso_U_33
_atom_site_aniso_U_12
_atom_site_aniso_U_13
_atom_site_aniso_U_23
_atom_site_aniso_type_symbol
  s .035(4) .025(3) .025(3) .013(1) .000 .000 s
```

Fig. 5.3.5.2. Example CIF for demonstrating the use of *QUASAR*.

```
data_P6122

loop_
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_label
  .20200 .79800 .91667 s
  .49800 .49800 .66667 o
  .48800 .09600 .03800 c1

loop_
_atom_site_aniso_label
_dummy           # requested item not present
_atom_site_aniso_U_11
  s ? .035(4)

#                  -----end-of-data-block-----

data_P6122

loop_
_atom_type_symbol
_atom_type_oxidation_number
_atom_type_number_in_cell
_atom_type_scat_dispersion_REAL
_atom_type_scat_dispersion_imag
_atom_type_scat_source
  S  0  6  .319  .557
       Int_Tab_Vol_III_p202_Tab._3.3.1a
  O  0  6  .047  .032
       Cromer,D.T._&_Mann,J.B._1968_AC_A24,321.
  C  0 20  .017  .009
       Cromer,D.T._&_Mann,J.B._1968_AC_A24,321.
  RU 0  1 -.105 3.296
       Cromer,D.T._&_Mann,J.B._1968_AC_A24,321.

loop_
_atom_site_label
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_U_iso_or_equiv
_atom_site_thermal_displace_type
_atom_site_calc_flag
_atom_site_calc_attached_atom
_atom_site_type_symbol
  s  .20200 .79800 .91667 .030(3) Uij  ? ? s
  o  .49800 .49800 .66667  .02520 Uiso ? ? o
  c1 .48800 .09600 .03800  .03170 Uiso ? ? c

loop_
_atom_site_aniso_label
_atom_site_aniso_U_11
_atom_site_aniso_U_22
_atom_site_aniso_U_33
_atom_site_aniso_U_12
_atom_site_aniso_U_13
_atom_site_aniso_U_23
_atom_site_aniso_type_symbol
  s .035(4) .025(3) .025(3) .013(1) .000 .000 s

#                  -----end-of-data-block-----
```

Fig. 5.3.5.3. Result of running *QUASAR* with the example request list of Fig. 5.3.5.1 on the CIF listed in Fig. 5.3.5.2.

The request list must specify the data block from which the requested data are to be extracted. Multiple data blocks may be requested in the same file. An entry '`data_`' operates as a wild card and indicates that requests should be served from the next data block encountered. In the example above, the first group of requests will be met from the first data block in the CIF; the second set from the data block named 'P6122' (if present).

5.3.5.1.4. *Output from QUASAR*

The body of the request list is a series of data names. Where a data name appears in the CIF, it will be extracted with its associated data value or values. The user need not have prior knowledge of whether a data item occurs in a looped list or not: *QUASAR* will automatically retrieve the matching values and construct a loop header if necessary. However, because the requests are served in the exact order in which they occur in the file, data items in the same list in the input CIF may be extracted into different lists upon output. Although this breaks the semantic association between items grouped in the same list (especially for CIFs described by the DDL2 relational scheme), it is a syntactically valid construction and may be a valuable feature for some processes.

5.3.5.1.4.1. *Treatment of missing data*

When a requested data item is absent from the CIF, *QUASAR* will nevertheless emit a data name with a corresponding value of '`?`', the conventional CIF value of null type for 'unknown quantity'. A CIF comment is also generated by *QUASAR* to indicate that the entry was missing from the input CIF. If the missing data name is found between data names that have multiple values and that occur in the same looped list, it is assumed that the missing data name should be associated with the same looped list, and it will be emitted in the loop header; the integrity of the list is then satisfied by emitting a column of unknown values. Note how this behaviour differs from that of the generic STAR File extraction utility *Star_Base* (Spadaccini & Hall, 1994), which silently ignores missing data items. However, it is a useful behaviour for applications that depend on finding a specific data item in their processing stream, even where its value is unknown.

5.3.5.1.4.2. *Matching data names*

As with the specification of data-block names, the data names in the request list may have a trailing underscore. Where this is the case, *QUASAR* will retrieve all data items where the data name starts with the specified string. For example, a request for '`_atom_site_`' will extract *all* data names starting with '`_atom_site_`'. The special case of an isolated underscore character '_' matches *all* data names present in the current data block.

5.3.5.1.4.3. *Case sensitivity*

The example demonstrates the way in which the application handles the case insensitivity of a requested data item. Data names are converted internally to a lower-case representation, both from the request list and the input CIF. Matches are therefore determined in a case-insensitive manner. However, if a data name is present in the CIF, its original case is retained on output. This permits the computationally irrelevant but cosmetically useful retention of capitalization as used in canonical CIF dictionary definitions. Where the requested data name is absent, the output is all lower-case.

### 5.3.5.2. *cif2cif*

*cif2cif* (Bernstein, 1998) is a program built with the *CIFtbx* toolkit (Chapter 5.4) to copy a CIF while checking data names against dictionaries, optionally reformatting numbers to maintain standard uncertainties within a specified range. The output CIF may contain a subset of the data in the original CIF according to a request list, in the manner of *QUASAR* (Hall & Sievers, 1993).

The program was built as a sample application using *CIFtbx* routines and grew out of requirements from several sources.

5.3.5.2.1. *Operation*

5.3.5.2.1.1. *Copying*

In its simplest application, the program copies a CIF from the standard input channel to standard output. The copy is not verbatim (standard utilities of the computer operating system should be used for that purpose), but the output CIF differs from the input only in the following respects: some comments are deleted; lines in the input longer than 80 characters are wrapped to 80 characters or less; white space between tokens may be altered, especially in an attempt to align entries in looped lists in a cosmetically pleasing manner. While none of these changes should affect robust CIF-parsing applications, they are nevertheless useful in imposing a uniform style of presentation for browsing in a text editor or other human-readable framework.

5.3.5.2.1.2. *Constraining standard uncertainties to specified ranges*

Some journals require that standard uncertainties in experimental values should be quoted within a specified range. Typically the standard uncertainty (s.u.) should be quoted as an integer in parentheses, modifying the last place or two of decimals in the experimental data, and with a value between 2 and 19. *cif2cif* permits s.u. values in the ranges 1–9, 2–19 or 3–29, selectable by a command-line switch. The effect of applying the 'rule of 19' would be to change a value of 1.458(1) in the input CIF to 1.4580(10) in the output.

5.3.5.2.1.3. *Dictionary validation*

*cif2cif* will open one or more CIF dictionary files as it copies the input CIF and identify certain classes of error against the dictionary definitions. The conditions that will raise an error are an unrecognized data name or a wrong data type. The program will also optionally indicate a warning if a data name has been assigned a category different from the leading portion of the data name – this may indicate an inconsistency within the dictionary itself.

5.3.5.2.1.4. *Serving a request list*

*cif2cif* will extract a subset of the data items contained in a CIF as specified by a request list, in the manner of *QUASAR*. The handling of data names specified in the request list is as described in Section 5.3.5.1.3 above, with the following additional feature. The special string `data_which_contains:` will extract the specified data items from the first data block in which at least one occurs; the block code need not be known in advance.

Some care must be exercised in attempting to extract data from data blocks by context without prior knowledge of the file contents. Consider the following simple example file:

```
data_A
   loop_    _A1
            _A2
             a1 a2 aa1 aa2


data_B
   loop_    _A1
            _B1
             a b aa bb
```

The loop containing **`_A1`** and **`_B1`** *cannot* be extracted with a request list of the form

```
data_which_contains:
_A1
_B1
```

because `_A1` occurs in the first data block encountered; the output from *cif2cif* in this example will be

```
data_A
    loop_     _A1
              a1   aa1
#            ---end-of-data-block---
```

The behaviour of the program differs from *QUASAR* in two other small ways. When the request list forces the output data stream to contain the same data-block header more than once, an error message is posted to the standard error channel and the data-block headers in the output stream are annotated with a comment of the form '`#<---- duplicate data block`'. In this case the output file does *not* conform to the CIF syntax rules.

When a data name is requested but no matching data item appears in the output file, *cif2cif* writes an error message to the standard error channel. However, unlike *QUASAR*, which inserts the requested data name in the output stream with an associated value of '`?`' (for *unknown*), *cif2cif* produces *no* output for the requested data item.

##### 5.3.5.2.1.5. *Other features*

Some additional features are of use in special circumstances.

The user may preserve the layout of the contents of looped lists exactly as in the input file, or may ask the program to adjust the layout to a more visually pleasing tabular form.

The user may enable recognition of data-name aliases in the dictionaries used for validation. When the relevant command-line argument is set to *true*, user-supplied data names will be transformed to the canonical forms in the validating dictionary. This would permit, for example, a small-molecule CIF using the core dictionary definitions to be converted to mmCIF format.

The user may prefix each line of output with an identical character string. A typical reason for so doing would be to include a fragment of CIF listing within the body of an email message or some other document. Such an output would not conform to the syntax rules for CIF.

#### 5.3.5.2.2. *Invocation of the program*

*cif2cif* is another application of the *CIFtbx* library by the same author, and so has a similar user interface to that of *CYCLOPS* (Section 5.3.4.1.2). Under a Unix-like operating system, the program is typically called with a command such as

```
cif2cif -i infile -o outfile [-q reqfile]
```

where *infile* is the name of the input file, *outfile* is the output file and *reqfile* is an optional file containing a request list for a subset of the original contents.

A more complete set of options available in a Unix-like operating environment is

```
cif2cif [-i infile] [-o outfile] [-d dictfile] [-q reqfile]
        [-f cmndfile] [-c catck] [-a alias] [-t tab] [-e sulim]
        [-p prefix]
```

where the options are as follows:

*-i* specifies the name of the input file, *infile*.

*-o* specifies the name of the output file, *outfile*.

*-d* specifies the name of a dictionary file, *dictfile*, against which the existence, type and category of data names are checked. The dictionary file may be *either* a CIF dictionary or a list of file names. That is, it may contain dictionary definitions in DDL format or (if the file begins with the characters `#DICT`) it may contain a list of dictionary file names to be entered. Thus, multiple dictionaries may be specified to the program.

*-q* specifies the name of the request file, *reqfile*, containing a list of data names (with associated data-block directives) that should be extracted as a subset of the contents of the original file.

*-f* specifies the name of a command file *cmndfile* that contains additional directives to the program.

*-c* is a flag indicating whether an error message should be raised if a data name has been assigned a category different from the leading portion of the data name itself. The Boolean variable *catck* may take the values 't', '1' or 'y' for *true*, 'f', '0' or 'n' for *false*.

*-a* is a flag indicating whether data-name aliases in the validating dictionary should be used to replace user-supplied names by their canonical forms. The Boolean variable *alias* may take the same values for *true* or *false* as above.

*-t* is a flag indicating whether the output should be reformatted with tabs to produce a regular table layout within looped lists. The Boolean variable *tab* takes the same values as above. If *true*, text is reformatted; if *false*, the original formatting is retained.

For the flags expecting Boolean values, the default is 'f' (*false*).

*-e* specifies the precision to retain in rounding standard uncertainty values. The permitted integer values are 9, 19 (the default) and 29.

*-p* takes a string value which is prefixed to every line of output. Every occurrence of the underscore character '_' in the prefix is changed to a space on output.

If no input or output file names are specified, the program will read from the standard input channel or write to standard output, respectively. The special character hyphen ('`-`') may also be supplied as an argument in place of a file name to indicate standard input or standard output as appropriate.

Finally, if the operating system supports the passing of environment variables to a program, the name of the input file may be passed as the value of `$cif2cif_INPUT_CIF`, and likewise the output file, `$cif2cif_OUTPUT_CIF`, dictionary file, `$cif2cif_CHECK_DICTIONARY`, and request file, `$cif2cif_REQUEST_LIST`, may be specified.

### 5.3.5.3. *ciftex*: translating to a typesetting language

The program *ciftex* (McMahon, 1993) was developed to create files for typesetting the journal *Acta Crystallographica* using the text-formatting language TEX (Knuth, 1986). Details of its use in the journal production process are given in Chapter 5.7. It is discussed here as an example of translating a CIF to some output format where data values are annotated with different text depending on their accompanying data names.

#### 5.3.5.3.1. *Basic operation of ciftex*

The program is designed to act as a filter, typically in a Unix-style environment, reading a CIF on the standard input channel and outputting a modified data stream to standard output. The output is a file of TEX code that is processed by the TEX program to produce a device-independent file describing the content of a formatted typeset document. Further post-processing allows the formatted document to be viewed on the screen or printed.

Each input token (number, character or text string; data name; `loop_` or `data_` keywords) is transformed as it is identified; there is no lookahead and minimal retention of context. The data stream is treated purely syntactically; no transformations are applied on the basis of the supposed meaning of any of the file contents.

```
_cell_formula_units_Z        2
_cell_length_a               8.79(2)
_refine_ls_extinction_coef    .347e4(5)
_chemical_name_common        'copper sulphate'
```

Fig. 5.3.5.4. Sample CIF data input to *ciftex*.

```
\cellz{2}
\nobreak\cella{8.79 (2)}
\extcoeffLarson{0.347 (5) $\times$ $10^{4}$}
\chemcom{copper sulfate}
```

Fig. 5.3.5.5. Output from *ciftex* run on the data of Fig. 5.3.5.4.

5.3.5.3.1.1. *Non-looped data*

For portions of the CIF that are not contained in looped lists, the transformations are trivial. A (*data name*, *data value*) pair is transformed to a TEX macro and its argument. The macro name is determined from an external 'map' file which the program reads at run time; this file associates CIF data names and the corresponding TEX macros through a simple lookup table.

A CIF data value is in most cases passed as the argument to the corresponding TEX macro with few modifications. If the data value is a character string beginning with an integer, full point, hyphen or plus character, it is assumed to be of type 'numb'. A space is introduced ahead of an embedded open parenthesis (to separate a standard uncertainty from its parent value). A leading zero is printed before any bare decimal point. An embedded `E` is taken to indicate exponential notation and the format of the number is accordingly modified.

If the input data value is of type 'char' (*i.e.* is a single token beginning with characters other than those recognized as the leading characters for numerical data; or contains multiple tokens delimited by quote marks or semicolons), the program will search the map file for key values exactly matching each token, and if found will substitute the token by its replacement word or text. If no replacement is specified in the map file, the token is passed unchanged to the standard output channel. This facility was found to be useful in making global substitutions of individual words during file processing, but must be used with care since the substitutions are unconditional, without any reference to context.

Some small examples of typical non-looped data items are shown in Fig. 5.3.5.4 and the corresponding *ciftex* translation based on a map file used for typesetting *Acta Crystallographica Section C* is shown in Fig. 5.3.5.5.

Note the transformations of the numerical arguments and the translation of 'sulphate' to 'sulfate'.

5.3.5.3.1.2. *Looped data*

If the input token is a **loop_** keyword, the program enters a different mode of operation. Looped data may be represented in print either as repetitive lists or in tabular format. There is no indication in a CIF dictionary of the appropriate representation (nor should there be, for what is essentially a matter of presentation) and the choice is made based on a flag associated with each data name in the map file. For non-tabular lists, the structure

```
loop_
    _dataname_1
    _dataname_2
       value_1    value_2
       value_3    value_4
```

is translated to a sequence of TEX codes of the form

```
\macro_one(value_1)
\macro_two(value_2)
\macro_one(value_3)
\macro_two(value_4)
```

In the case of tabulated data, the **loop_** header is translated into a set of table headings and typographic codes are introduced to lay out in columnar format the values in the body of the list. The number of different data names in the loop header is counted and the data values are identified by their position in the loop modulo the total number of data names in the header (in effect, by their 'phase' in the loop). In the simplest case, a TEX command is emitted that builds a table with *n* columns, where *n* is the number of different data names. Then the data values are counted as they are processed. After every *n*th data value, a TEX code is emitted indicating 'end of table row' and a further code is emitted before the next value (if there is one) that means 'beginning of new table row'. In all other cases, a code is emitted signifying 'move to next column'.

```
\settabs 5 \columns
\+\relax & $x$   &    $y$    &   $z$    &$U_{\rm eq}$$&\cr
\+Re &0.222 (1) &0.003 (1) &0.146 (1) &0.042 (1) &\cr
\+Co &0.234 (1) &0.139 (1) &0.299 (1) &0.046 (1) &\cr
\+P1 &0.358 (1) &0.222 (1) &0.197 (1) &0.044 (1) &\cr
\+P2 &0.106 (2) &0.051 (1) &0.289 (1) &0.046 (1) &\cr
\+C1 &0.308 (6) &0.029 (6) &0.034 (4) &0.057 (4) &\cr
\+O1 &0.356 (5) &0.044 (5) &0.030 (3) &0.079 (3) &\cr
\+C2 &0.066 (6) &0.039 (6) &0.111 (4) &0.056 (4) &\cr
```

Fig. 5.3.5.6. TEX markup for typesetting a table of atomic coordinates.

Fig. 5.3.5.6 is a simplified extract from a table of atomic coordinates derived from the **_atom_site_** loop in a CIF.

5.3.5.3.1.3. *The ancillary map file*

The translation between a CIF data name and its replacement text in the TEX output file is defined in the external map file. The format of the translation is very simple, as illustrated in Fig. 5.3.5.7.

Each line starts with a CIF data name, which is terminated by a space character. The next character is either '`T`' or '`N`' to indicate whether the output should be tabulated or not. The next character is an arbitrary character from the ASCII character set, and is chosen to collect together data that will appear in the same logical section of the output file. This locator character may be associated, in another ancillary file described below, with additional text for output. The remainder of the line is the replacement text.

In the example supplied, the cell-length parameters map to the TEX macros `\cella`, `\cellb` and `\cellc` (each preceded by a standard TEX macro forbidding a page break immediately before the contents are printed). The details of the publication authors are described by a set of TEX macros that will occur in two different locations in the output file (the authors' names and addresses may be looped together in the location labelled by the character `a`; any explanatory footnotes and email addresses will be printed elsewhere in the paper, at the location labelled `x`). The anisotropic displacement parameters $U^{ij}$ will be printed in a table and the replacement text consists of the TEX codes that will be printed at the head of each column in the table.

The initial text on the line need not be a CIF data name; it may be any other single word. In this case, every occurrence of that word in the input CIF will be replaced by the replacement text.

```
_cell_length_a Ng\nobreak\cella
_cell_length_b Ng\nobreak\cellb
_cell_length_c Ng\nobreak\cellc

_publ_author_name Na\author
_publ_author_address Na\address
_publ_author_footnote NX\aufootnote
_publ_contact_author_email NX\email

_atom_site_aniso_label TU\relax
_atom_site_aniso_U_11 TU{\hfill $U^{11}$ \hfill}
_atom_site_aniso_U_12 TU{\hfill $U^{12}$ \hfill}
_atom_site_aniso_U_13 TU{\hfill $U^{13}$ \hfill}
_atom_site_aniso_U_22 TU{\hfill $U^{22}$ \hfill}
_atom_site_aniso_U_23 TU{\hfill $U^{23}$ \hfill}
_atom_site_aniso_U_33 TU{\hfill $U^{33}$ \hfill}
```

Fig. 5.3.5.7. Example map file for use with *ciftex*.

```
#[:\newif\ifproof \prooftrue
#]:\iftwocol\vfill\enddoublecolumns\fi
#a:\pretolerance1000\parskip0pt\tolerance5000
#a:\vskip10pt
#g:
#g:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#g:\iftwocol\enddoublecolumns\twocolfalse\fi
#g:\tenbf Experimental
#g:\noindent\ninebf Compound \datablock\vskip2pt
#g:\noindent\nineit Crystal data\par
#g:\vskip2pt\begindoublecolumns\twocoltrue\defaultfont
#U:%%%%%%%%%%% Table of anisotropic U's %%%%%%
#U:\iftwocol\enddoublecolumns\twocolfalse\fi
#U:\rm Table \tableno. \it Anisotropic displacement
#U:parameters \rm (\AA$^2$) for \datablock
#U:\vskip 6pt
```

Fig. 5.3.5.8. Example format file for *ciftex*.

If the initial *character* of the line is a hash mark #, the line is treated as a comment and discarded.

#### 5.3.5.3.1.4. *The ancillary format file*

Because a printed paper may be more verbose than its parent CIF data file, it is necessary to add text to the output from *ciftex* to represent section headings, line spaces or other formatting instructions. The program reads an ancillary file, known as the format file, for such additional text.

Each line in the format file begins with a hash mark #, a single ASCII character and a colon. The second character is chosen to match the corresponding locator character associated with data names in the map file. The rest of the line is text to be output. When the locator character associated with the data name currently being processed differs from the previous one, the output text from all lines in the format file with the new locator character are output.

The special strings `#[:` and `#]:` indicate text to be emitted at the beginning and end of the output stream, respectively.

Fig. 5.3.5.8 is an example of a simplified format file. The first line is printed at the start of the output TEX file; the second line at the end. The next line will be printed on the first occurrence of a data name flagged with the locator code `a` in the map file. In this example, that will be the name or address of an author of the paper; some typographic directives are emitted immediately before the authors' names and addresses, including the introduction of a blank line ('vertical skip', or 'vskip') of height 10 typographic points.

The lines beginning `#g:` are emitted immediately before the first data name in the group that is associated with locator code `g`. In this example, the effect is to output a heading and subheading before printing the cell-length parameters and to switch to double-column format. The line containing *only* the characters `#g:` provides for the introduction of a blank line into the TEX file, with the sole purpose of making the file more readable by human editors.

The lines beginning `#U:` are emitted at the beginning of the table of anisotropic $U$ values.

The mechanism looks complicated at first sight, but addresses the need to generate headings at standard locations in a printed paper when the exact content of the paper is not known in advance.

The different format for directives in the map and format files means that the same file can be used for both purposes, if required. In practice it is often easier to maintain different files: the same mapping between CIF data names and TEX macros might be common to different journals, while each journal uses its own format file.

#### 5.3.5.3.2. *Invocation of the program*

The program reads a CIF on the standard input channel and outputs TEX code on standard output. There is no provision to specify file names. It is therefore invoked within a Unix-style operating system by a command such as

```
ciftex < infile > outfile
```

where *infile* and *outfile* are the input and output files respectively; or it may be called as part of a pipeline of procedures:

```
program 1 < infile | ciftex | program 2 ...
```

A number of command-line options may be supplied to modify the operation of the program. Other than the specification of the map and format files, they are largely relevant to differing house styles for IUCr journals.

The options -map *mapfile* and -format *formatfile* specify the names of the ancillary map and format files. If not specified, they are sought in default locations on the user's file system (different values may be defined when the program is compiled) or as specified in the environment variables `$CIFTEX_MAP` and `$CIFTEX_FORMAT`, respectively.

The options *-H* and *-N* specify, respectively, whether or not hydrogen atoms in coordinate tables should be printed. The hydrogen-atom lines in the table are in fact always emitted on standard output, but in the case of the *-N* option are prefixed by a % (TEX comment) character and so ignored by TEX.

Options *-c* and *-F* specify the printing of centred decimal points or commas for decimal points, respectively. Finally, the option *-d* modifies certain assumptions that *ciftex* makes when typesetting CIF dictionaries. The details are of interest only to a specialist.

#### 5.3.5.3.3. *Some general comments*

Although *ciftex* is available for public use and redistribution within the academic community, it is clearly of most interest to users who need to generate typeset representations of the contents of CIFs. Nevertheless, some elements of its design are relevant to other applications that perform on-the-fly file transformations on a strictly syntactic basis.

First, the functionality is very simple, essentially tokenizing the input data stream and exchanging tokens for replacement text as directed. An immediate consequence of this is the need for additional utilities to manipulate the input file if, for example, the data need to be presented in a particular order. In the journals production process, *QUASAR* is used to reorder an input file before passing it to *ciftex*.

Second, the replacement text should be externalized as much as possible. The use of map and format files means that the same basic program can be used for formatting according to any set of typographic rules; only the ancillary files need to be modified. In the current version of *ciftex*, the program performs some replacements internally; an objective of further development is to remove this function from the program and to externalize it either in more sophisticated table lookup files or in separate methods modules.

Third, the concept of replacement should be abstracted as much as possible. The software was written initially with the objective of replacing data names with TEX macros. Experience suggests that a generic transformation program could be written with the philosophy of replacing data names and data values by directives implemented before and after the occurrence of the data value, and as events upon its first, last and intervening occurrences. Such 'directives' and 'events' could be mapped to arbitrary replacement strings in any markup scheme, such as SGML, XML, HTML, TEX, LATEX or commercial word-processing encodings.

## 5.3.6. Libraries for scripting languages

Recent years have seen a great increase in the popularity of scripting languages – broadly, computer languages where the source code is run-time interpreted and that have powerful facilities for interacting with other processes and operating-system utilities. In part this is because such languages lend themselves to rapid prototyping; but the powerful features of languages such as Perl (Wall *et al.*, 2000), Python (van Rossum, 1991) and Tcl/Tk (Ousterhout, 1994) make it entirely feasible to create and maintain complex programs that can run efficiently. Several of the applications discussed in this chapter make effective use of one or more of these languages, either solely or alongside more traditional compiled languages.

As the scripting languages have become more powerful, they have also acquired more structure, so that authors now frequently build libraries (or 'modules') of functions or subroutines that can be re-used in a range of applications. This is a welcome development, for the availability of public libraries not only reduces the effort required to develop new applications, but also goes a long way towards establishing common application programming interfaces that help to standardize the way in which software from different sources is developed.

In this section two available libraries of this type are reviewed: *STAR::Parser* and *PyCifRW*.

### 5.3.6.1. *STAR::Parser* and related Perl modules

A collection of Perl modules has been developed at the San Diego Supercomputer Center (Bluhm, 2000) to provide basic library routines for object-oriented manipulation of STAR files with restricted syntax appropriate to CIF applications. The module name suggests that a more complete STAR implementation may be considered in future developments, but at present the modules do not handle nested loops or the inheritance of data values from global blocks. Indeed, they are still rather limited in scope; nevertheless, for the programmer wishing to prototype CIF applications in Perl, they offer a very rapid entry to parsing CIFs and constructing useful data structures that can be manipulated with standard Perl tools.

The use of some of the modules is illustrated in Fig. 5.3.6.1, which is a simplified version of the main program loop in the application written to typeset the CIF dictionaries printed in Part 4 of this volume.

5.3.6.1.1. *STAR::Parser*

*STAR::Parser* is the basic parsing module and may parse either data files or dictionaries (which may include save frames). It contains a single class method, *parse*, which returns an array of DataBlock objects. Each DataBlock object contains all the data items within an individual data block of the file. Even if the file contains only a single data block, the resulting object is passed in an array.

The contents of the data blocks may be accessed and manipulated by the methods provided by the *STAR::DataBlock* and *STAR::Dictionary* modules. They are stored internally as a multidimensional hash (the Perl term for an associative array with keys and associated values, which may themselves be complex data objects). Keys are provided for data blocks, save blocks, categories and data items identified during the parse. The module provides no error checking of files or objects, against the dictionary or otherwise – limited checking functionality is available through other modules in this collection.

In the example of Fig. 5.3.6.1, the *parse* method is called at line 9 to read a DDL2 dictionary file (indicated by the `-dict=>1` parameter) and return an array of data blocks. In DDL2 dictionaries (such as the mmCIF dictionary of Chapter 4.5) an entire dictionary is contained within a data block; save frames partition the data block into definitions for separate items. Normally a DDL2 CIF dictionary has only a single data block; nevertheless, the example program can handle multiple data blocks in the array, and traverses the one or several data blocks in the array through a Perl *foreach* construct (line 15).

5.3.6.1.2. *STAR::Dictionary*

The *STAR::Dictionary* module contains class and object methods for Dictionary objects created by the *STAR::Parser* module, and is in fact a subclass of *STAR::DataBlock* (see next section). Since CIF dictionaries are fully compliant STAR files, they require little that is different from the methods developed for handling data files. The method *get_save_blocks* is provided to return an array of all save frames found in the Dictionary object.

In line 25 of the example, the method is called on each dictionary loaded from the input file (as described above, normally there will only be one). The method is combined with the Perl *sort* function to create an array of save frames from the dictionary, arranged in alphabetic order. All further manipulations of the contents of these save frames will use the methods of the generic *STAR::DataBlock* class.

5.3.6.1.3. *STAR::DataBlock*

This package provides several useful methods for handling the objects within a data block returned by the *STAR::Parser* module.

The class has a constructor method *new*, which can create a completely new DataBlock object if called with no argument. This is of course essential for applications that wish to write new CIFs. Alternatively, it may be called with a `$file` argument to retrieve an existing object that has previously been written to the file system using the *store* object method described below:

```
$data_obj = STAR::DataBlock->new{ -file=>$file };
```

Table 5.3.6.1 summarizes the object methods provided by the package. The *store* method allows a DataBlock object to be serialized and written to hard disk for long-term storage. The Perl public *Storable::* module is used.

The *get_item_data* method returns the data values for a named data item. It is used frequently in the example program of Fig. 5.3.6.1; for example, at lines 28–30 the array of categories

references