Hence there is a real need for a utility to validate data *names* – effectively a CIF spelling checker.

5.3.4.1. *CYCLOPS*

The program *CYCLOPS* (Hall, 1993; Bernstein & Hall, 1998) was written specifically to address the problem of validating CIF data names. Its use extends beyond simply identifying data names in a CIF data file and checking that they are defined in a dictionary. Any ASCII file may be input, allowing for the checking of CIF data names in any text documents or program source.

The program was originally written in Fortran as an aid to ensuring that the original core CIF dictionary was free from data-name errors; subsequently it was extended to be able to read multiple dictionaries in DDL1 and DDL2 formats, and to resolve data-name aliases across multiple dictionaries. The extended version was written with the library routines of the *CIFtbx* toolkit (Hall & Bernstein, 1996) described in Chapter 5.4 and is distributed as an example application with *CIFtbx*. The description below refers to this extended version, also known as *CYCLOPS2*.

5.3.4.1.1. *Operation*

The program determines the dictionary (or list of dictionaries) against which to validate the input text file (see below for the method of passing such information to the program). It opens each dictionary in turn and stores all data names defined in the dictionaries. Where the same name is defined in multiple dictionaries, the behaviour is determined by a command-line switch.

The text file is then input and parsed for candidate data names. Because the program is designed to check potential data names embedded in ordinary text files, it is not sufficient to apply the CIF parsing rule of a white-space-delimited character string beginning with an underscore character. Instead, character strings are sought that begin with an underscore optionally preceded by white space or one of the characters `,.([{</\|'":*` and followed by white space, one of the characters `,.)]}>/\|'"-=?!;:` or by the end of a line.

For each candidate data name found in this way, matching data names in the stored list are identified in one of three ways:

(i) If the data name is not preceded by the asterisk character `*` and it does not end with the underscore character _, then search for an identical match.

(ii) If the data name ends with the underscore character _, then search for a match in the dictionary where the leading characters in the dictionary name are the same as all the characters in the data name found in the text. For example, the text `_atom_site.label_` would match the mmCIF dictionary entry **`_atom_site.label_alt_id`**.

(iii) If the data name is preceded by the asterisk character `*`, then search for a match in the dictionary where the trailing characters in the dictionary name are the same as all the characters in the data name found in the text. The first match found in the dictionary is accepted. For example, the text `*_alt_id` would match **`_atom_site.label_alt_id`**, or, if that name had not been in the dictionary, **`_struct_conn.ptnr1_label_alt_id`**. If one of the searches succeeds, add the line number of the data name to a list attached to the dictionary name. Up to 19 line numbers are retained for each dictionary name (the first ten matches and the last nine).

If no match is found, the unmatched data name is added to the list of unmatched names, along with the appropriate line number. If a data name has been misspelled it will be caught at this step.

When the text file has been processed, a validation report file is output containing the alphabetically sorted list of unmatched names and line numbers, followed by the sorted list of names from all dictionaries that are used within the text. If requested, this is followed by the sorted list of names from all dictionaries that are not used within the text in the file. If a data name has an alias defined in the dictionaries, a warning about the existence of the alias is given. If more than one dictionary has been used, the source dictionary is identified for each data name. An example of the output from *CYCLOPS* is shown in Fig. 5.3.4.1.

5.3.4.1.2. *Invocation of the program*

CYCLOPS is generally invoked from a command line that specifies the input and output file names and the dictionary files against which to validate the input. However, because the program is portable across a wide range of operating systems, there is substantial flexibility in the way in which it may be invoked. Under a Unix-like operating system, the program may typically be called with a command such as

`cyclops -i` *`infile`* `-o` *`outfile`* `-d` *`dictfile`*

where *infile* is the name of the input file for validation, *outfile* is the file to which the detailed output of the program is written and *dictfile* is a dictionary file.

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

`cyclops` [`-i` *`infile`*] [`-o` *`outfile`*] [`-d` *`dictfile`*] [`-p` *`priority`*] [`-f` *`cmndfile`*] [`-c` *`catck`*] [`-v` *`verbose`*] [`-s` *`short`*]

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 the dictionary file, *dictfile*. For compatibility with the original version of the software, 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. As implied by this last statement, multiple dictionaries may be specified to the program.

-p specifies the priority that should be assigned if multiple definitions for the same data name are encountered when multiple dictionaries are accessed. The permitted values are: *first* (the default), in which the first of duplicate definitions to be loaded takes priority; *final*, in which the last takes priority; and *nodup*, in which an instance of a duplicate definition should be treated as a fatal error.

-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*.

-v is a flag indicating whether a verbose listing of unreferenced data names should be generated. The Boolean variable *verbose* may take the same values for *true* or *false* as above.

-s is a flag indicating whether the output should be short (*i.e.* restricted to items not in dictionaries). The Boolean variable *short* takes the same values as above.

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

If no input or output file names are specified, the program will read from the standard input channel or write to standard output,

```
           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).