CIF APPLICATIONS

Authors of any software that reads, writes or validates CIF data are invited to contribute to this series. Authors should state clearly when submitting a manuscript to a Co-editor that the paper should be included as part of the CIF Applications series. An appropriate series number will be assigned by the Editorial Office.

J. Appl. Cryst. (1996). **29**, 598–603

CIF Applications. V. *CIFtbx*2: extended tool box for manipulating CIFs*

SYDNEY R. HALL[a] AND HERBERT J. BERNSTEIN[b] *at* [a]*Crystallographic Centre, University of Western Australia, Nedlands 6009, Australia, and* [b]*Bernstein + Sons, 5 Brewster Lane, Bellport, NY 11713, USA. E-mail: syd@crystal.uwa.edu*

(Received 18 January 1996; accepted 14 May 1996)

Abstract

*CIFtbx*2 is a new version of a Fortran subroutine library for programmers developing CIF applications. The functions for reading and writing CIF data in *CIFtbx* [Hall (1993). *J. Appl. Cryst.* **26**, 482–494] have been expanded and facilities for handling macromolecular CIF data and dictionaries have been added. The *CIFtbx*2 library facilitates applications involving all current versions of the dictionary definition language.

1. Introduction

This paper is part of the continuing series on CIF applications. The first papers in the series appeared in this journal in 1993, and included a description of a Fortran subroutine library for programmers developing CIF applications, *CIFtbx* (Hall, 1993). Since that time, the CIF approach has been applied to new areas such as nuclear magnetic resonance (NMR) data and macromolecular structural data. The dictionary that defines the macromolecular structural data (Fitzgerald, Berman, Bourne, McMahon, Watenpaugh & Westbrook, 1995) uses an extended dictionary definition language (DDL2) with stronger relational attributes than that of the DDL (Hall & Cook, 1995) used for the core crystallographic dictionaries. The extended DDL2 (Westbrook & Hall, 1995) is not supported by the dictionary functions in *CIFtbx* and this was the primary motivation for creating the new *CIFtbx*2 library. This library, which is equally applicable to the macromolecular CIF (mmCIF) dictionary in DDL2 and to other dictionaries in DDL, is the subject of this paper.

The initial impetus for this work was to support one of us (HJB) in the use of CIF data derived from Protein Data Bank (Bernstein *et al.*, 1977) files. *CIFtbx*2 was used to map hundreds of CIF data names embedded in existing software into the mmCIF name format and to check the existence and application of these items. As a result, code for conversion of Protein Data Bank entries to mmCIF format was developed in the form of an awk script, *pdb2cif* (Bourne, Bernstein & Bernstein, 1996). *CIFtbx*2 is also used in the recent release of the *XTAL*3.4 System (Hall, King & Stewart, 1995).

With minor exceptions, *CIFtbx*2 is a fully upward-compatible extension of *CIFtbx*. The presentation of this paper will, therefore, focus on the differences at the user level and the changes to those algorithms that facilitate greater generality.

2. *CIFtbx*2 overview

The user interface to *CIFtbx*2 consists of Fortran functions, subroutines and common variables. The user-accessible functions, subroutines and variables are identified by a trailing underscore ('_') character. The functions and subroutines, often referred to as 'commands', may be divided into three basic groups: set-up commands that initialize data handling, read commands that input data from a CIF and write commands that output CIF data. The read and write commands are logically independent and may be applied simultaneously to copy and update CIFs.

Here is a summary of the functions, subroutines and variables.

2.1. *Set-up*

`init_(devcif,devout,devdir,deverr)`
Set the device numbers of files

`dict_(fname,checks)`
Select a dictionary for data checks

2.2. *Read CIF data*

`ocif_(fname)`	Open an input CIF
`data_(name)`	Select data block containing requested data
`test_(name)`	Get data type and other information
`name_(name)`	Get data name of next data item in the block
`numb_(name,numb,sdev)`	Get numerical data item and its estimated standard deviation (e.s.d.)
`char_(name,strg)`	Get character string or text data item
`purge_` [new]	Subroutine to reset all input data and dictionary pointers

2.3. *Write CIF data*

`pfile_(fname)`	Open output CIF
`pdata_(bname)`	Output data block header line
`ploop_(name)`	Output data name into a loop structure
`pchar_(name,strg)`	Output data name and character string

* This paper is one of a series on CIF applications. Offprints are available from The Managing Editor, 5 Abbey Square, Chester CH1 2HU, England. See text of paper for availability of program(s) by e-mail.

`pnumb_(name,numb,sdev)`	Output data name and number with its e.s.d.
`ptext_(name,strg)`	Output data name and text line
`close_`	Subroutine to close output CIF

2.4. *Common variables*

*CIFtbx*2 supplies variables in the common block file `ciftbx.sys` so that the programmer can test and control how the functions are applied to specific data-handling tasks. Variables that are new to *CIFtbx*2 are flagged below with '[new]'.

`text_`		Logical variable returned true if another text line is present
`loop_`		Logical variable returned true if another loop packet is present
`bloc_`		Character variable containing the current block (`data_` or `save_`) name
`strg_`		Character variable containing the current data item
`type_`		Character variable containing the data type code
`list_`		Integer variable containing the loop block number
`long_`		Integer variable containing the length of the data string in `strg_`
`file_`		Character variable containing the filename of the current file
`longf_`		Integer variable containing the length of the filename in `file_`
`align_`		Logical variable controlling column alignment of data lists during writing of a CIF. The preset default is true.
`save_`	[new]	Logical variable returned true if the current data block is a save frame
`alias_`	[new]	Logical variable set true if data-name aliases are searched by the read functions. The preset default is true.
`aliaso_`	[new]	Logical variable set true if the alias data name `dicname_` is output by the write functions. The preset default is false.
`tagname_`	[new]	Character variable containing the tag name found in the input CIF
`dicname_`	[new]	Character variable containing the preferred alias specified in the first-entered dictionary
`diccat_`	[new]	Character variable containing the current category code
`dictype_`	[new]	Character variable containing the explicit data type code
`line_`	[new]	Integer variable specifying the expected CIF input line length. Nonblank lines exceeding this limit will cause a warning message to be issued. The default value of `line_` is 80. This variable may be set by the program up to a maximum value not exceeding `MAXBUF`, which has a default value of 200.

3. Other changes to *CIFtbx*

Because the dictionary definition languages DDL and DDL2 both serve the same function, it is relatively straightforward to have one piece of software handle the manipulation of associated dictionaries in a way that is transparent to the user. The amount of recoding needed to convert the *CIFtbx* subroutines to *CIFtbx*2 subroutines has been minimal and, as a consequence, the user interface remains virtually identical.

The major differences in *CIFtbx*2 cater to much larger dictionaries (*e.g.* the mmCIF dictionary), the more rigorous use of data categories in DDL2 and the addition of aliases to provide compatibility of the data names between DDL and DDL2 dictionaries. Other matters addressed are the use of save frames in DDL2 dictionaries (in place of data blocks) and the acceptance of data names that exceed 32 characters (a concession to recent CIF data naming). The following details summarize how *CIFtbx*2 has been modified to accommodate these requirements.

3.1. *Improved hash-table look-up*

The efficient handling of larger dictionary and data files has been achieved by improved algorithms for identifying and processing data names. New parameters have been introduced for the size-dependent changes and these will enable future changes to be made smoothly. Improvements were achieved by extensive use of hash-table-controlled lists, which are handled by routines in the `hash_funcs.f`. Ordinarily, the user will be unaware of hash-table operations, but these can be adjusted *via* the parameter `NUMHASH` in `ciftbx.sys`. The default value is 53, which means that, for lists of up to 2500 names, searches for name matches would look at sublists of fewer than 50 names. Faster look-ups are possible if the value of `NUMHASH` is increased. A prime number is recommended for the highest efficiency. For a discussion of the choices and implications of hash-table-controlled lists, see Knuth (1973).

3.2. *PARAMETER defined array sizes*

Additional `PARAMETER` variables may be adjusted in the *CIFtbx*2 source code if optimal efficiency is needed in particularly large calculations. They are:

`NUMCHAR`	The maximum number of characters in a name (default 48)
`NUMDICT`	The maximum number of names in all dictionaries (default 2500)
`NUMBLOCK`	The maximum number of names in a data block (default 500)
`NUMLOOP`	The maximum number of loops in a data block (default 50)
`NUMITEM`	The maximum number of items in a loop (default 50)
`MAXBUF`	The maximum number of characters in a line (default 200)

The maximum number of categories is also controlled by `NUMDICT`, but these do not compete for space with ordinary names.

3.3. *New dictionary checks*

The most extensive differences between *CIFtbx* and *CIFtbx*2 are in the dictionary processing function `dict_`. The category

codes of data items, if present, are now used to check consistency within CIF save frames, data blocks and loop structures. In order to accommodate both DDL and DDL2 dictionaries, the checking routines accept all the current approaches to categories. For example, the original core dictionary `cifdic.c91` does not define categories, the more recent core dictionary `cifdic.c94` lists them explicitly, and the mmCIF dictionary contains category codes as explicit definitions and embedded into the data name as a character string preceding a period character ('.'). These variations in usage means that `dict_` will only proceed with category checking if these are specified in the input dictionary.

In addition to accepting the flags `dtype` and `valid`, which specifiy checking level when given as the second argument of `dict_`, *CIFtbx*2 accepts the new flags `reset` and `close`, which specify dictionary conditions. When either of these new flags is used, the first argument of `dict_` must be blank. The `reset` flag resets the checking levels and disables the dictionary checks without affecting the stored definitions and the aliased name facilities. The dictionary checking functions are restarted by the input of either `dtype` or `valid` as argument 2, with argument 1 blank. The `close` flag switches off the dictionary functions completely and these may only be restarted with a new `dict_` command to specify a dictionary file.

3.4. *Chained applications*

Additional controls in *CIFtbx*2 allow for a sequence of applications in a single program involving different dictionaries and different CIFs. Two such controls, `reset` and `close`, are described above for the `dict_` function. The `purge_` command enables a program to switch off all file access and table pointers. This can be important when different CIFs need to be entered into the same calculation and must be validated against different dictionaries. Each invocation of the `purge_` subroutine resets all tables to their *ab initio* status.

3.5. *Warning messages*

The improved checking facilities in *CIFtbx*2 have necessitated more extensive run-time error messages. These are detailed in the next section.

3.6. *Name aliases*

The dictionary definition language DDL2 permits data names to be aliased or made equivalent to other data names. DDL2 enables aliases for two purposes: to make data names equivalent to DDL dictionaries and to link equivalent data names within the same dictionary. It allows for the use of synonyms appropriate to the application. *CIFtbx*2 is capable of handling aliased data names transparently. Input CIFs may use any of the equivalent aliases, as may the application software processing the file. In addition, output CIFs may be written with the data names specified in the putting functions, or with names that are automatically converted to preferred dictionary names. If more than one dictionary is used, the first one loaded is assumed to carry the preferred names, even if the alias declarations are in dictionaries loaded later. The default behaviour of *CIFtbx*2 is to accept all combinations of aliases and to produce output CIFs with the exact names specified in the user calls.

The interpretation of aliased data names is modified by setting of the logical variables `alias_` and `aliaso_`. When `alias_` is set false, the automatic recognition and translation of aliases stops. When `aliaso_` is set true, the automatic conversion of user-supplied names to dictionary-preferred alias names in the writing of data to output CIFs is enabled. The preferred alias name is stored in the variable `dicname_` following any invocation of a getting function, such as `numb_` or `test_`. If `alias_` is set false, `dicname_` will agree with the called name. The variable `tagname_` is always set to the actual name used in an input CIF.

For example, the data name `_atom_site_aniso_U_11` from the core dictionaries `cifdic.c91` and `cifdic.c94` is the alias of `_atom_site_anisotrop.u[1][1]` in the mmCIF dictionary `cifdic.m95`. In the following application of *CIFtbx*2 function `test_`, the input DDL name `atom_site_aniso_U_11` is used to inquire as to the names used in an input CIF:

```
read(8,'(a)',end=400) name
f1 = test_(name)
write(6,'(2(3x,a32)') name,dicname_
name=dicname_
f1 = test_(name)
write(6,'(2(3x,a32)') name,tagname_
```

This gives the following printout.

```
_atom_site_aniso_U_11
                _atom_site_anisotrop.u[1][1]

_atom_site_anisotrop.u[1][1]
                      _atom_site_aniso_U_11
```

3.7. *Save frames*

*CIFtbx*2 handles save frames, whereas *CIFtbx* does not. The logical variable `save_` is set to true if the data currently being accessed exist in a save frame rather than a data block. The *CIFtbx* function `data_` is responsible for setting `save_` to true at the start of a save frame and to false at the end of a save frame. A warning is issued if save-frame statements are not matched.

3.8. *Input line lengths*

The way in which input lines are tested has been changed. *CIFtbx* clips all lines at 80 characters.

*CIFtbx*2 issues a warning message if the length of a line (*i.e.* the column position of the last nonblank character) exceeds the integer variable `line_`. Characters exceeding `line_` are processed provided they do not exceed `MAXBUF` (the `PARAMETER` specification of the input buffer array length). The default setting of `line_` is 80.

3.9. *Data typing*

*CIFtbx*2 supports a much wider range of data types than *CIFtbx*. This is because DDL2 provides for two tiers of data type: *primitive* and *precise*. Primitive-type codes are `char`, `numb`, `text` and `null` and these are returned as the character variable `type_`. Precise-type codes such as `int`, `float`, `yyyy-mm-dd`, `uchar` *etc.* are, if available, stored in the character variable `dictype_`

4. Error-message glossary

The *CIFtbx*2 functions are designed so that a program can, *via* the returned true or false values, detect and respond to run-time abnormalities without interference from the checking algo-

rithms. However, some types of errors, such as the dimensions of critical arrays being exceeded, require processing to cease. In addition, the processing of dictionaries of the size and complexity of mmCIF is greatly simplified if there are independent warning messages. This is because functions such as `dict_` or `data_`, while trivial to invoke, actually initiate a complex sequence of processes and checks that make the notification of run-time errors difficult in terms of single true or false values. For these reasons, the repertoire of system error messages, fatal and nonfatal, has been increased in *CIFtbx2*.

System error messages have a common format. Each begins with either a 'warning' or 'error' header line with the name of the file being processed, the current data block or save frame and the line number. The next line contains the text of the message. For example, the following warning message is issued by the test program `tbx_ex.f` when it finds that the file `test.cif` contains the unknown data item name `_dummy_test`, in the data block `data_mumbo_jumbo`:

```
ciftbx warning: test.cif
                data_mumbo_jumbo line: 12
Data name _dummy_test not in dictionary!
```

4.1. *Fatal errors: array bounds*

The following messages are issued if the *CIFtbx2* array bounds are exceeded. Array bounds can be adjusted by changing of the `PARAMETER` values in `ciftbx.sys`. If the value of `MAXBUF` needs to be changed, the file `ciftbx.cmn` must also be updated.

```
Input line_ value > MAXBUF
Number of categories > NUMBLOCK
Number of data names > NUMBLOCK
Cifdic names > NUMDICT
Dictionary category names > NUMDICT
Items per loop packet > NUMITEM
Number of loop_s > NUMLOOP
```

4.2. *Fatal errors: data sequence, syntax and file construction*

`Dict_ must precede ocif_`

Dictionary files must be loaded before an input CIF is opened because some checking occurs during the CIF loading process.

`Illegal tag/value construction`

Data name (*i.e.* a 'tag') and data values are not matched (outside a looped list). This usually means that a data name immediately follows another data name, or a data value was found without a preceding data name. The most likely cause of this error is the failure to provide a '`.`' or '`?`' for missing or unknown data values, or a failure to declare a loop when one was intended.

`Item miscount in loop`

Within a looped list, the total number of data values must be an exact multiple of the number of data names in the `loop_` header.

```
Prior save-frame not terminated
Save-frame terminator found out of context
```

Save frames must start with `save_<frame code>` and end with `save_`. These messages will be issued if this does not occur.

`Syntax construction error`

Within a data block or save frame, the number of data values does not match the number of data names (ignoring loop structures).

`Unexpected end of data`

When processing multiline text the end of the CIF is encountered before the terminal semicolon.

4.3. *Warnings: input errors*

`Category <cat-code> first implicitly defined in cif`

The category code in the DDL2 data name is not matched by an explicit definition in the dictionary. This may be intentional, but it more likely indicates a typographical error in the dictionary or the CIF.

`Data name <name> not in dictionary!`

The data item name `<name>` was used in the CIF but could not be found in the dictionary.

`Duplicate data item <name>`

There were two or more identical data names `<name>` in a data block or save frame.

`Heterogeneous categories in loop <new cat-code> vs <old cat-code>`

Looped lists should not contain data from different categories. This message is issued if the category of new data items fails to match the old category of prior data items. A special category (`none`) is used to denote item names for which no category has been declared. No warning is issued on this level for a loop for which all data items have no category declared.

`Input line length exceeds line_`

Nonblank characters were found beyond the value given by the variable `line_` The default value for `line_` is 80, which is the upper bound for lines in a valid CIF. The extra characters will be processed but the input file should be reformatted for input to other CIF-handling programs.

`Missing loop items set as DUMMY`

During writing of an output CIF, a looped list of items was truncated with an incomplete loop packet (*i.e.* the number of items did not match the number of loop data names). The missing values were set to the character string '`DUMMY`'.

`Numb type violated <name>`

The data item `<name>` has been processed with an explicit dictionary type `numb`, but with a nonnumeric value. Note that the values '`?`' and '`.`' will not generate this message.

`Quoted string not closed`

Character values may be enclosed by bounding quotes. The strict definition of a 'quoted string' value is that it must start with a `<wq>` digraph and end with a `<qw>` digraph, where w is a white-space character blank or tab and q is a single or double quote. This message is issued if these conditions are not met.

4.4. *Warnings: dictionary checks*

`Aliases and names in different loops; only using first alias`

When a DDL2 dictionary contains a loop of alias declarations, the corresponding data name declarations are expected to be in the same loop. This message is issued if separate loops are used. Only the first alias name is used, but processing continues.

`Attempt to redefine category for item`
`Attempt to redefine type for item`

If a DDL2 dictionary contains a category or type for a data item that conflicts with an earlier declaration, these warnings are issued. The redeclaration is ignored.

`Categories and names in different loops`

When a DDL2 dictionary contains a loop of category declarations, the corresponding data name declarations are expected to be in the same loop. This message is issued if separate loops are used. Only the first category name is used, but processing continues.

`Category id does not match block name`

In a DDL2 dictionary, the save-frame code is expected to start with the category name. If a category name within the frame is not within a loop, it is checked against that in the frame code and a warning is issued if these do not match.

`Conflicting definition of alias`

When a DDL2 dictionary contains a new declaration of an alias for a data name that is in conflict with a previous alias definition, this warning is issued. The second alias declaration is ignored.

`Duplicate definition of same alias`

When a DDL2 dictionary contains a new declaration of an alias for a data name that duplicates a previously defined alias pair, this warning is issued.

`Item type <type-code> not recognised`

In *CIFtbx*2, DDL2 dictionary precise-type codes are translated to the DDL primitive-type codes `numb`, `char` and `text`. If an unrecognised type code is found for which *CIFtbx* does not have a translation, this warning is issued.

`Multiple DDL 1 and 2 category definitions`
`Multiple DDL 1 and 2 name definitions`
`Multiple DDL 1 and 2 type definitions`

These messages are issued if both DDL and DDL2 style declarations for categories, data names or data types are used in the same data block or save frame.

`Multiple categories for one name`
`Multiple types for one name`

These messages are issued if a dictionary contains a loop of category or type definitions and an unlooped declaration of a single data name. The first category or type definition is used and processing continues.

`No category defined in block <name> and name <name> does not match`

This message is issued if a DDL2 dictionary contains no category for the defined data item and it was not possible to derive an implicit category from the block name. This message usually indicates a typographical error in the dictionary.

`No category specified for name <name>`

This warning is issued if a dictionary contains categories but none for the named data item.

`No name defined in block`
`No name in the block matches the block name`

These messages are issued if a dictionary save frame or data block contains no name definition or if all the names defined fail to match the block name.

`No type specified for name <name>`

This message is issued if a type code is missing from a dictionary and type checking was requested in the `dict_` invocation.

`One alias, looped names, linking to first`
`Types and names in different loops`

A DDL2 dictionary may contain a list of data names and a single alias outside of this loop. In this case, the correct name to which to link the alias must be derived implicitly. If the save-frame code matches the first name in the loop, no warning is issued, because the use of the block name was probably the intended result, but, if no such match is found, this warning is issued.

5. Distribution

The latest version of this software is available from the World Wide Web servers http://ndbserver.rutgers.edu/software and http://www.crystal.uwa.edu.au/Software/ciftbx/ or from the anonymous ftp site 130.95.232.12 in directory /cif.

*CIFtbx*2 is distributed as the file `ciftbx.cshar` containing the Fortran sources `ciftbx.f` and `hash_funcs.f`, two common files `ciftbx.sys` (used by `ciftbx.f` internally) and `ciftbx.cmn` (used by applications), test files and other useful files. The structure of this file permits automatic unpacking in UNIX systems having the C shell csh, but, unlike the more commonly used 'shar' format, also allows unpacking with a text editor. A `ciftbx.shar` version is also available. The mmCIF dictionary `cifdic.m95` and CIF core dictionary `cifdic.c91` or `cifdic.c94` may also be needed.

For further information, e-mail Syd Hall at syd@crystal.uwa.edu or Herbert Bernstein at yaya@aip.org.

We thank Frances C. Bernstein of the Protein Data Bank at Brookhaven National Laboratory for helpful comments and suggestions.

References

Bernstein, F. C., Koetzle, T. F., Williams, G. J. B., Meyer, E. F. Jr, Brice, M. D., Rodgers, J. R., Kennard, O., Shimanouchi, T. & Tasumi, M. (1977). *J. Mol. Biol.* **112**, 535–542.

Bourne, P., Bernstein, F. C. & Bernstein, H. J. (1996). *pdb2cif: Translating PDB Entries into mmCIF.* CIF Workshop, XVII Congress of the International Union of Crystallography, Seattle, USA.

Fitzgerald, P., Berman, H., Bourne, P., McMahon, B., Watenpaugh, K. D. & Westbrook, J. (1995). *Macromolecular CIF Dictionary*, Version 0.7.28. IUCr COMCIFS, International Union of Crystallography, Chester, England.

Hall, S. R. (1993). *J. Appl. Cryst.* **26**, 482–494.

Hall, S. R. & Cook, A. P. C. (1995). *J. Chem. Inform. Comp. Sci.* **35**, 819–825.

Hall, S. R., King, O. S. D. & Stewart, J. M. (1995). *XTAL*3.4 *Users Manual*. University of Western Australia, Australia.

Knuth, D. E.. (1973). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Reading, MA: Addison-Wesley.

Westbrook, J. & Hall, S. R. (1995). *A Dictionary Description Language for Macromolecular Structure, Draft DDL V 2.1.0*, IUCr COMCIFS, Chester, England.