

## 5. APPLICATIONS

## 5.4.12.1. Programming style

A traditional Fortran style of programming is used in *CIFtbx*. Common blocks are declared to report and control the state of the processing. This allows argument lists to be kept short and avoids the need to create complex data structure types, but introduces extensive ‘programming by side-effects’. In order to reduce the impact of this approach on users, two different views of the common blocks are provided. The declarations in *ciftbx.cmn* are needed by all users. The more extensive declarations in *ciftbx.sys*, which include the same common declarations as are found in *ciftbx.cmn* and additional declarations used internally within *CIFtbx*, are provided for use in maintaining the library. Caution is needed in making internal modifications to the library to maintain the desired relationships among the actions of various routines and the states of variables declared in the common blocks.

Statements are written in the first 72 columns of a line, reserving columns one through five for statement labels and using column six for continuation. Approaches that would require the use of C libraries or non-portable Fortran extensions are avoided. For this reason, all the internal service routines are written in Fortran, all memory needed is preallocated with *DIMENSION* statements and a direct-access file is used to hold the working copy of a CIF.

## 5.4.12.2. Memory management

Since *CIFtbx* does static memory allocation with *DIMENSION* statements, it is sometimes necessary to adjust the array dimensions chosen to suit a particular application. It may also be necessary to increase the storage allocated for individual tags to allow for unusually long ones.

The sizes of most arrays and strings used in *CIFtbx* that might require adjustment are controlled by *PARAMETER* statements in the files *ciftbx.sys* and *ciftbx.cmv* (the variable-declaration portion of *ciftbx.cmn*). The parameters are shown in Table 5.4.12.1.

These values can result in *CIFtbx* requiring more than a megabyte of memory. On smaller machines working with a small dictionary and simple CIFs, considerable space can be saved by reducing the values of *NUMDICT* and *NUMBLOCK*.

On the other hand, an application working with several layered dictionaries and large and complex CIFs with many data items and many loops in a data block might require a version of *CIFtbx* with larger values of *NUMDICT*, *NUMBLOCK* and, perhaps, of *NUMLOOP*.

The variables *NUMPAGE* and *NUMCPP* control the amount of memory to be used to buffer the direct-access file and the size of the data transfers to and from that file. Smaller values will reduce the demand for memory at the expense of slower execution.

## 5.4.12.3. Use of INCLUDE

The *INCLUDE* statement allows the statements in the specified file to be treated as if they were being included in a program in place of the *INCLUDE* statement itself. This simplifies the maintenance of common-block declarations and is an important tool in keeping code well organized. In *CIFtbx*, the *INCLUDE* statement is used to bring the statements in the files *ciftbx.cmn* and *ciftbx.sys* into programs where they are needed, and to simplify *ciftbx.cmn* and *ciftbx.sys* by using *INCLUDES* of the files *ciftbx.cmv* and *ciftbx.cmf*. The file *ciftbx.cmv* contains the definitions of the essential *CIFtbx* data structures as common blocks, for inclusion in both *ciftbx.cmn* for user applications and in *ciftbx.sys* for the *CIFtbx* library routines themselves. Most compilers handle the *INCLUDE* statement, but, if necessary, a user may replace any or all of the *INCLUDE*

Table 5.4.12.1. Parameter statements in *CIFtbx*

NUMCHAR	Maximum number of characters in data names (default 48)
MAXBUF	Maximum number of characters in a line (default 200, increased to 4096 in <i>CIFtbx</i> 2.7 and later)
NUMPAGE	Number of memory resident pages (default 10)
NUMCPP	Number of characters per page (default 16 384)
NUMDICT	Number of entries in dictionary tables (default 3200)
NUMHASH	Number of hash table entries (a modest prime, default 53)
NUMBLOCK	Number of entries in data-block tables (default 500)
NUMLOOP	Number of loops in a data block (default 50)
NUMITEM	Number of items in a loop (default 50)
MAXTAB	Maximum number of tabs in output CIF line (default 10)
MAXBOOK	Maximum number of simultaneous bookmarks (default 1000)

statements with the contents of the indicated file. For example, the only non-comments in *ciftbx.cmn* are

```
include 'ciftbx.cmv'
include 'ciftbx.cmf'
```

This means that the file *ciftbx.cmn* could be replaced by a concatenation of the two files *ciftbx.cmv* and *ciftbx.cmf*.

## 5.4.12.4. Use of ENDDO

*CIFtbx* makes some use of the *ENDDO* statement (as well as nested *IF*, *THEN*, *ELSE*, *ENDIF* constructs) to improve readability of the source code. Most compilers accept the *ENDDO* statement, but if conversion is needed then constructs of the form

```
do index = istart, iend, incr
...
enddo
```

should be changed to

```
do nnn index = istart, iend, incr
...
nnn continue
```

where *nnn* is a unique statement number, not used elsewhere in the same routine.

## 5.4.12.5. Names of internal routines

The following routines are used internally by later versions of *CIFtbx*. If these names are needed for other routines, then changes in the library will be needed to avoid conflicts.

## (a) Variable initialization:

```
block data
```

Critical *CIFtbx* variables are initialized with data statements in a block data routine.

## (b) Control of floating-point exceptions:

```
subroutine clearfp
```

If a system requires special handling of floating-point exceptions, the necessary calls should be added to this subroutine.

## (c) Message processing:

```
subroutine err (mess)
character mess*(*)
subroutine warn (mess)
character mess*(*)
subroutine cifmsg (flag, mess)
character mess*(*), flag*(*)
```

Error and warning messages are processed through these three routines.

(d) Internal service routines:

```
subroutine dcheck
  (name, type, flag, tflag)
  logical flag, tflag
  character name*(*), type*4
subroutine eotext
subroutine eoloop
subroutine excat
  (sfname, bcname, lbcname)
  character*(*) sfname, bcname
  integer lbcname
subroutine getitm (name)
  character name*(*)
subroutine getstr
subroutine getlin (flag)
  character flag*4
subroutine putstr (string)
  character string*(*)
```

These routines are used internally by the library. The subroutine `dcheck` validates names against dictionaries. The subroutines `eotext` and `eoloop` are used to ensure termination of loops and text strings. The subroutines `getitm`, `getstr` and `getlin` extract items, strings and lines from the input CIF. The subroutine `putstr` writes strings to the output CIF.

(e) Numeric routines:

```
subroutine ctonum
subroutine putnum (numb, sdev, prec)
  double precision numb, sdev, prec
```

The routine `ctonum` converts a string to a number and its standard uncertainty. The subroutine `putnum` converts a number and standard uncertainty to an output string.

(f) String manipulation:

```
subroutine detab
integer function lastnb (str)
  character str*(*)
character*(MAXBUF) function locase (name)
  character name*(*)
```

The subroutine `detab` converts tabs to blanks. The function `lastnb` finds the column position of the last non-blank character in a string. The function `locase` converts a string to lower case.

(g) Hash-table processing:

```
subroutine hash_find (name, name_list,
  chain_list, list_length, num_list,
  hash_table, hash_length, ifind)
  character name*(*),
  name_list(list_length)
  integer hash_length,
  chain_list(list_length),
  hash_table(hash_length), ifind
subroutine hash_store (name, name_list,
  chain_list, list_length, num_list,
  hash_table, hash_length, ifind)
  character name*(*),
  name_list(list_length)
  integer hash_length,
  chain_list(list_length),
  hash_table(hash_length), ifind
integer function hash_value (name, hash_length)
  character name*(*)
  integer hash_length
```

These routines are used to manipulate the internal hash tables used by the library.

#### 5.4.12.6. Use of the underscore character

All the externally accessible *CIFtbx* commands and variables terminate with the underscore character. This works well on most systems, but can cause occasional problems, because traditional Fortran does not include the underscore in the character set and some operating systems reserve the underscore as a system flag, for example to distinguish C-language library routines from those written in Fortran. If conversion is needed, and the local compiler allows long variable and subroutine names, then the simplest approach would be to make a local variant of *CIFtbx* in which every occurrence of underscore in a function, subroutine or variable name is changed to a distinctive character pattern (e.g. 'CIF' or 'qq'), but caution is needed, since there are many *character strings* used in the library that include the underscore. For example, in changing the variable `loop_` to `loopCIF`, it would be a mistake to change the statement

```
if (strg_(1:5).eq.'loop_')
  type_='loop'
```

to

```
if (strg_(1:5).eq.'loopCIF')
  type_='loop'
```

#### 5.4.12.7. Names longer than six characters

*CIFtbx* uses some function, subroutine and variable names longer than six characters to improve readability, but, in most cases, consistent truncation of all uses of a name to six characters will not cause any problems.

#### 5.4.12.8. File management

*CIFtbx* allows the user to read from one CIF while writing to another. The input CIF is first copied to a direct-access file to allow random access to desired portions of the input CIF. Since CIF allows data items to be presented in any order, the alternatives to the use of a direct-access file would have been to create memory-resident data structures for the entire CIF or to track position and make multiple search passes through the file as data items are requested. When programming for personal and laboratory computers with limited memory and which may lack virtual memory capabilities, assuming the availability of enough memory for large CIFs would greatly restrict the applications within which *CIFtbx* could be used. However, the disk accesses involved in using a direct-access file slow execution. When working on larger computers, execution speed can be increased at the expense of memory by increasing the number of memory-resident pages (see the parameter `NUMPAGE` above). If the number of pages times the number of characters per page (`NUMCPE`) is large enough to hold the entire CIF, the application will run much faster.

Direct reading of the input CIF, making multiple passes when data items are requested in a different order to that in which they are presented in the CIF, is only practical when the number of out-of-order requests is small and the applications will not need to be used as a filter, perhaps reading the output of another program 'on-the-fly'. Since we cannot predict the range of applications and CIFs for which *CIFtbx* will be used, and direct reading could become impossibly slow, *CIFtbx* uses a direct-access file.

The processing of an output CIF is simpler than reading a CIF. The application determines the order in which the writing is to be done. No sorting is normally needed. Therefore *CIFtbx* writes an output CIF directly.