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 (NUMCPP) 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.