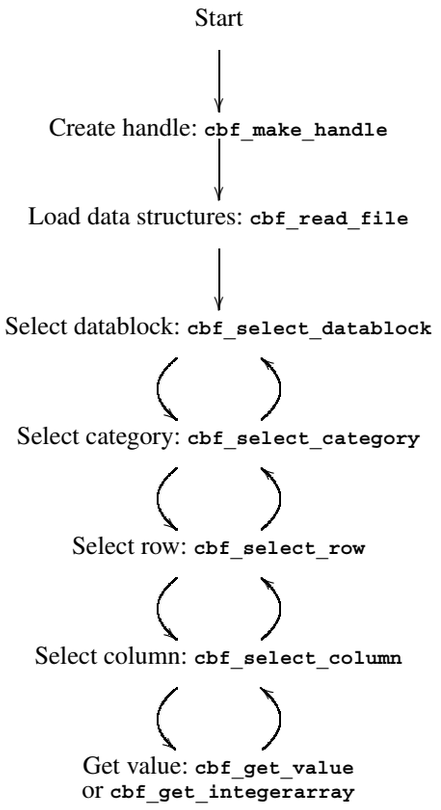

Fig. 5.6.1.2. Flow chart for a typical application reading CBF/imgCIF data.

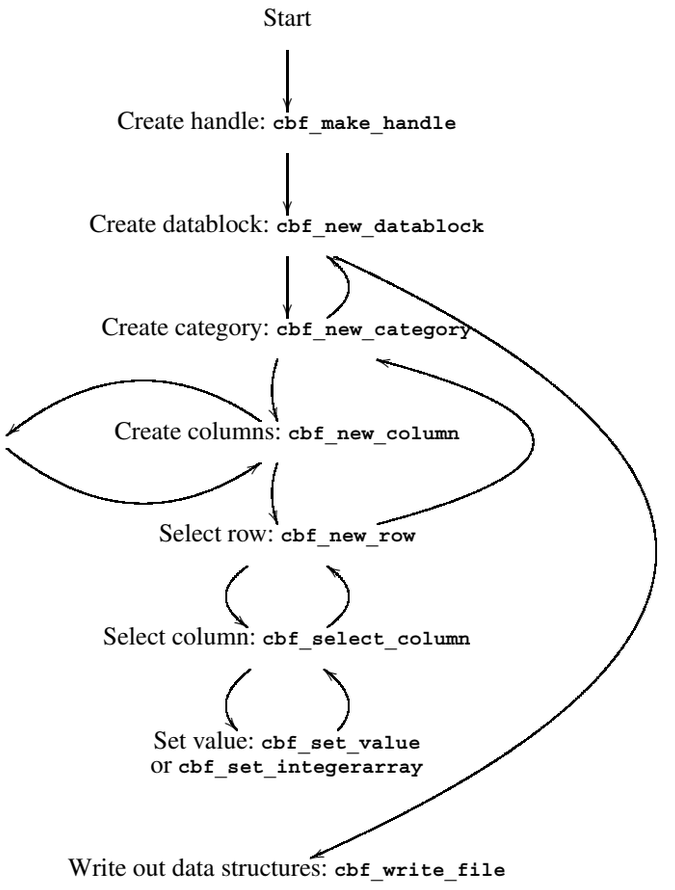

Fig. 5.6.1.3. Flow chart for an application writing CBF/imgCIF data.

The general approach to reading CBF/imgCIF data with *CBFlib* is to create an empty data structure with `cbf_make_handle`, load the data structures with `cbf_read_file` and then use nested loops to work through data blocks, categories, rows and columns in turn to extract values. Conceptually, all data values are held in the memory-resident data structures. In practice, however, only pointers to text fields with image data are held in memory. The data themselves remain on disk until explicitly referenced.

The basic flow of an application writing CBF/imgCIF data with the low-level *CBFlib* functions is shown in Fig. 5.6.1.3.

The general approach to writing CBF/imgCIF data with *CBFlib* is to create empty data structures with `cbf_make_handle` and load the data structures with nested loops, working through data blocks, categories, rows and columns in turn, to store values. The major difference from the nested loops used for reading is that empty columns are created before data are stored into the data structures row by row. Alternatively, the data could be stored column by column. Finally, the fully loaded memory data structures are written out with `cbf_write_file`. As with reading, text fields with image data are actually held on disk.

5.6.2. *CBFlib* function descriptions

All *CBFlib* functions have two common characteristics: (i) they return an integer equal to 0 for success or an error code for failure; (ii) any pointer argument for the result of an operation can be safely set to NULL. The error codes are given in Table 5.6.1.1.

CBFlib provides two low-level functions to create or destroy the structure used to hold a data set:

```
cbf_make_handle
cbf_free_handle
```

There are two functions to copy a data set from or into a file:

```
cbf_read_file
cbf_write_file
```

The data structures 'behind' the handles retain pointers to current locations. This facilitates scanning through a CIF or CBF by data blocks, categories, rows and columns. The term '`rewind`' refers to setting the internal pointer for the type of item specified so that the first such item is pointed to.

In general, CIF does not permit duplication of the names of data blocks or category names. In practice, however, duplications do occur. *CBFlib* provides '`force`' variants of some functions to allow creation of duplicate names.

In *CBFlib*, the term '`set`' refers to changing the name of the currently specified item. The term '`reset`' refers to emptying a data block or category without deleting it. The term '`remove`' refers to deleting a data block, category, column or row. The terms '`select`' and '`next`' refer to finding the designated item by number, while the term '`find`' refers to finding the designated item by name.

CBFlib provides the following functions to manage data blocks and categories:

```
cbf_set_datablockname
cbf_{new, force_new, reset, remove, rewind, select, next, find}{_datablock, _category}
cbf_reset_datablocks
cbf_count{_datablocks, _categories}
cbf_{datablock, category}_name
```


references