Table 5.6.2.2. *Values for headers in* `cbf_read_file`

`MSG_DIGEST`	Check that the digest of the binary section matches any header value. If the digests do not match, the call will return `CBF_FORMAT`. The evaluation and comparison is delayed (a 'lazy' evaluation) to ensure maximal processing efficiency. If an immediate evaluation is desired, see `MSG_DIGESTNOW` below.
`MSG_DIGESTNOW`	Check that the digest of the binary section matches any header value. If the digests do not match, the call will return `CBF_FORMAT`. This evaluation and comparison is performed during initial parsing of the section to ensure timely error reporting at the expense of processing efficiency. If a more efficient delayed ('lazy') evaluation is desired, see `MSG_DIGEST` above.
`MSG_NODIGEST`	Do not check the digest (default).

Once a handle has been created, the data structures can be loaded with all the information held in a CBF file:

```
int cbf_read_file (cbf_handle handle, FILE *file,
     int headers);
```

Conceptually, all data values are associated with the handle at the `cbf_read_file` call. In practice, however, only the non-binary data are actually stored in memory. To work with potentially large binary sections most efficiently, these are skipped until explicitly referenced. For this reason, *`file`* must be a random-access file opened in binary mode [`fopen (..., "rb")`] and must not be closed by the calling program. *CBFlib* will call `fclose` when the file is no longer required.

The *`headers`* parameter controls the handling of any message digests embedded in the binary sections (Table 5.6.2.2). A *`headers`* value of `MSG_DIGEST` will cause the code to compare the digest of the binary section with any header message digest value. To maximize processing efficiency, this comparison will be delayed until the binary section is actually read into memory or copied (a 'lazy' evaluation). If immediate evaluation is required, use `MSG_DIGESTNOW`. In either case, if the digests do not match, the function in which the evaluation is taking place will return the error `CBF_FORMAT`. To ignore any digests, use the *`headers`* value `MSG_NODIGEST`.

The `cbf_write_file` call writes out the data associated with a CBF handle:

```
int cbf_write_file (cbf_handle handle, FILE *file,
     int readable, int ciforcbf, int headers,
     int encoding);
```

This call has several options controlling whether binary sections are written unencoded (CBF) or encoded in ASCII to conform to the CIF syntax (imgCIF), the type of headers in the binary sections, and the type of ASCII encoding and line termination used. The acceptable values for *`ciforcbf`* are `CIF` for ASCII-encoded binary sections or `CBF` for unencoded binary sections. The *`headers`* parameter (Table 5.6.2.3) can take the value `MIME_HEADERS` to select MIME-type binary section headers or `MIME_NOHEADERS` for simple ASCII headers. The value `MSG_DIGEST` will generate digests for validation of the binary data and the value `MSG_NODIGEST` will skip digest evaluation. The header and digest flags may be combined using the logical OR operator.

Similarly, there are several combinable flags for the parameter *`encoding`* (Table 5.6.2.4). `ENC_BASE64` selects BASE64 encoding, `ENC_QP` selects quoted-printable encoding, and `ENC_BASE8`, `ENC_BASE10` and `ENC_BASE16` select octal, decimal and hexadecimal, respectively. `ENC_FORWARD` maps bytes to words forward (1234) for BASE8, BASE10 or BASE16 encoding and `ENC_BACKWARD` maps bytes to words backward (4321). Finally, `ENC_CRTERM` terminates lines with carriage return (CR) and `ENC_LFTERM` terminates lines with line feed (LF) (thus `ENC_CRTERM|ENC_LFTERM` will use CR LF).

Table 5.6.2.3. *Values for headers in* `cbf_write_file`

Values may be combined bit-wise.

`MIME_HEADERS`	Use MIME-type headers (default)
`MIME_NOHEADERS`	Use simple ASCII headers
`MSG_DIGEST`	Generate message digests for binary data validation
`MSG_NODIGEST`	Do not generate message digests (default)

CBFlib maintains temporary storage on disk as necessary for files to be written, so that *`file`* does not have to be random-access. However, if it is random-access and readable, resources can be conserved by setting *`readable`* nonzero.

The remaining low-level functions are involved in navigating the tree structure, creating and deleting data blocks, categories and table columns and rows, and retrieving or modifying data values.

The navigation functions are:

```
int cbf_find_datablock (cbf_handle handle,
     const char *datablockname);
int cbf_find_category (cbf_handle handle,
     const char *categoryname);
int cbf_find_column (cbf_handle handle,
     const char *columnname);
int cbf_find_row (cbf_handle handle,
     const char *value);
int cbf_find_nextrow (cbf_handle handle,
     const char *value);
int cbf_select_datablock (cbf_handle handle,
     unsigned int datablock);
int cbf_select_category (cbf_handle handle,
     unsigned int category);
int cbf_select_column (cbf_handle handle,
     unsigned int column);
int cbf_select_row (cbf_handle handle,
     unsigned int row);
int cbf_rewind_datablock (cbf_handle handle);
int cbf_rewind_category (cbf_handle handle);
int cbf_rewind_column (cbf_handle handle);
int cbf_rewind_row (cbf_handle handle);
int cbf_next_datablock (cbf_handle handle);
int cbf_next_category (cbf_handle handle);
int cbf_next_column (cbf_handle handle);
int cbf_next_row (cbf_handle handle);
```

The function `cbf_find_datablock` selects the first data block with name *`datablockname`* as the current data block. Similarly, `cbf_find_category` selects the category within the current data block with name *`categoryname`* and `cbf_find_column` selects the corresponding column within the current category. The function `cbf_find_row` differs slightly in that it selects the first row in the current column with the corresponding *`value`* and `cbf_find_nextrow` selects the row with the corresponding value following the current row. Note that selecting a new data block makes the current category, column and row undefined and that selecting a new category similarly makes the column and row undefined. In contrast, repositioning by column does not change the current row and repositioning by row does not change the current column.

The remaining functions navigate on the basis of the order of the data blocks, categories, columns and rows. Thus, `cbf_select_datablock` selects data-block number *`datablock`*, counting from 0, `cbf_rewind_datablock` selects the first data