```
   /* Read and modify the template */

cbf_failnez (cbf_make_handle (\&cbf))
in = fopen (template_name, "rb");

if (!in)
  exit (4);
cbf_failnez (cbf_read_template (cbf, in))

  /* Wavelength */

wavelength =
  img_get_number (img, "WAVELENGTH");
if (wavelength)
  cbf_failnez (
    cbf_set_wavelength (cbf, wavelength))

  /* Distance */

distance = img_get_number (img, "DISTANCE");
cbf_failnez (
  cbf_set_axis_setting (
    cbf, 0, "DETECTOR_Z", distance, 0))

  /* Oscillation start and range */

axis = img_get_field (img, "OSCILLATION AXIS");

if (!axis)

  axis = "PHI";

osc_start = img_get_number (img, axis);
osc_range = img_get_number (img,
   "OSCILLATION RANGE");
cbf_failnez (
  cbf_set_axis_setting (
    cbf, 0, "GONIOMETER_PHI",
      osc_start, osc_range))
```

Fig. 5.6.5.1. An extract of program code for the conversion of image data to CBF format.

5.6.5. Example programs

The *CBFlib* API comes with example programs and sample templates. The example program *convert_image* reads either of the templates described in Section 5.6.4, template_adscquantum4_2304x2304.cbf and template_mar345_2300x2300.cbf, and uses the higher-level *CBFlib* routines to convert an image file to a CBF.

The example programs *makecbf* and *img2cif* read an image file from a MAR300, MAR345 or ADSC CCD detector and then use *CBFlib* to convert that image to CBF format (*makecbf*) or to either imgCIF or CBF format (*img2cif*). *makecbf* writes the CBF-format image to disk, reads it in again, and then compares it with the original. *img2cif* just writes the desired file without a verification pass. *makecbf* works only from files on disk, so that random input/output can be used. *img2cif* includes code to process files from stdin and to stdout. The example program *cif2cbf* copies an ASCII CIF to a binary CBF/imgCIF file.

5.6.5.1. *convert_image*

The program *convert_image* takes two arguments, `imagefile` and `cbffile`. These are the primary input and output. The detector type is extracted from the image file, converted to lower case and used to construct the name of a template CBF to use for the copy. The template file name is of the form *template_name_columnsxrows*. The program comes with img.c and img.h to read image files. It uses the `img_...` routines to extract data and metadata from the image and uses the higher-level *CBFlib* routines to populate the template. Fig. 5.6.5.1 is a portion of the code that reads in the template and inserts the wavelength, the distance to the detector, and the oscillation start and range.

```
/* Make the _diffrn_frame_data category */
cbf_failnez(
  cbf_new_category         /* create the category */
    (cbf, "diffrn_frame_data"))
cbf_failnez(
  cbf_new_column           /* create the column   */
    (cbf, "id"))
cbf_failnez(
  cbf_set_value            /* add data            */
    (cbf, "frame_1"))
cbf_failnez(
  cbf_new_column           /* create next column  */
    (cbf, "detector_element_id"))
cbf_failnez(
  cbf_set_integervalue     /* add data            */
    (cbf, 1))
cbf_failnez(
  cbf_new_column           /* create the column   */
    (cbf, "detector_id"))
cbf_failnez(
  cbf_set_value            /* add data            */
    (cbf, detector_id))
cbf_failnez(
  cbf_new_column           /* create next column  */
   (cbf, "array_id"))
cbf_failnez(
  cbf_set_value            /* add data            */
    (cbf, "image_1"))
cbf_failnez(
  cbf_new_column           /* create next column  */
    (cbf, "binary_id"))
cbf_failnez(
  cbf_set_integervalue     /* add data            */
    (cbf, 1))
```

Fig. 5.6.5.2. Code fragment to illustrate the creation of the DIFFRN_FRAME_DATA category.

5.6.5.2. *makecbf*

makecbf is a good example of how to use many of the lower-level *CBFlib* functions. An example MAR345 image can be found at http://smb.slac.stanford.edu/~ellis/CBF_examples/mar345/mb_example_070.cbf. To run *makecbf* with the example image, type:

```
./bin/makecbf example.mar2300 test.cbf
```

The typical code fragment from makecbf.c shown in Fig. 5.6.5.2 creates the DIFFRN_FRAME_DATA category.

The program *img2cif* is an extended version of *makecbf* that allows the user to choose the details of the format of the output CBF. *img2cif* has the following command-line interface:

```
img2cif [-i input_image] [-o output_cif]
        [-c  {p[acked]|c[anonical]|[n[one]}]
        [-m  {h[eaders]|n[oheaders]}]
        [-d  {d[igest]|n[odigest]}]
        [-e {b[ase64]|q[uoted-printable]|
             d[ecimal]|h[exadecimal]|o[ctal]|n[one]}]
        [-b {f[orward]|b[ackwards]}]
        [input_image] [output_cif]
```

-i takes the name of the input image file in MAR300, MAR345 or ADSC CCD detector format. If no `input_image` file is specified or is given as '-', an image is copied from stdin to a temporary file. *-o* takes the name of the output CIF (if BASE64 or quoted-printable encoding is used) or CBF (if no encoding is used). If no `output_cif` is specified or is given as '-', the output is written to stdout. *-c* specifies the compression scheme (packed, canonical or none; the default is packed). *-m* selects MIME (Freed &

```
cbf_failnez (cbf_count_datablocks(cif, &blocks))
for (blocknum = 0; blocknum < blocks;  blocknum++)
{ /* start of copy loop */
  cbf_failnez (
    cbf_select_datablock(cif, blocknum))
  cbf_failnez (
    cbf_datablock_name(cif, &datablock_name))
  cbf_failnez (
    cbf_force_new_datablock(cbf, datablock_name))
  if ( !cbf_rewind_category(cif) ) {
    cbf_failnez (
      cbf_count_categories(cif, &categories))
    for (catnum = 0; catnum < categories; catnum++) {
      cbf_select_category(cif, catnum);
      cbf_category_name(cif,&category_name);
      cbf_force_new_category(cbf, category_name);
      cbf_count_rows(cif,&rows);
      cbf_count_columns(cif,&columns);
      /*  Transfer the column names from cif to cbf */
      if ( ! cbf_rewind_column(cif) ) {
      do {
        cbf_failnez(
          cbf_column_name(cif, &column_name))
        cbf_failnez(
          cbf_new_column(cbf, column_name))
      } while ( ! cbf_next_column(cif) );
      cbf_rewind_column(cif);
      cbf_rewind_row(cif);
      }
      /* Transfer the rows from cif to cbf */
      for (rownum = 0; rownum < rows; rownum++) {
        cbf_failnez (
          cbf_select_row(cif, rownum))
        cbf_failnez ( cbf_new_row(cbf))
        cbf_rewind_column(cif);
        for (colnum = 0; colnum < columns; colnum++) {
          cbf_failnez (cbf_select_column(cif, colnum))
          if ( ! cbf_get_value(cif, &value) ) {
            cbf_failnez (
              cbf_select_column(cbf, colnum))
            cbf_failnez ( cbf_set_value(cbf, value))
          } else {
            void * array;
            int binary_id, elsigned, elunsigned;
            size_t elements,elements_read, elsize;
            int minelement, maxelement;
            unsigned int cifcompression;

            cbf_failnez (
              cbf_get_integerarrayparameters(cif,
              &cifcompression, &binary_id, &elsize,
              &elsigned, &elunsigned, &elements,
              &minelement, &maxelement))
            if (array=malloc(elsize*elements)) {
              cbf_failnez (
                cbf_select_column(cbf,colnum))
              cbf_failnez (
                cbf_get_integerarray(
                cif, &binary_id, array, elsize,
                elsigned, elements, &elements_read))
              cbf_failnez(
                cbf_set_integerarray(
                cbf, compression, binary_id, array,
                elsize, elsigned, elements))
              free(array);
            } else {
              fprintf(stderr,
              "\nFailed to allocate memory %d bytes",
                elsize*elements);
               exit(1);
            }
          }
        }
      }
    }
  }
```

Fig. 5.6.5.3. Listing of the main code fragment in the program *cif2cbf* for conversion of an ASCII CIF to a CBF file.

Borenstein, 1996) headers within binary data value text fields. The default is headers for CIFs, no headers for CBFs. *-d* specifies the use of digests. The default is the MD5 digest (Rivest, 1992) when MIME headers are selected. *-e* specifies one of the standard MIME encodings: BASE64 (the default) or quoted-printable; or a non-standard decimal, hexadecimal or octal encoding for an ASCII CIF; or 'none' for a binary CBF. *-b* specifies the direction of mapping of bytes into words for decimal, hexadecimal or octal output, marked by '>' for forwards or '<' for backwards as the second character of each line of output, and in # comment lines. The default is backwards.

5.6.5.3. *cif2cbf*

The test program *cif2cbf* (Fig. 5.6.5.3) uses the same command-line options as *img2cif*, but accepts either a CIF or a CBF as input instead of an image file. The heart of the code is a series of nested loops. The outermost loop scans through all the data blocks. The next innermost loop scans through all the categories within each data block. The first of the two next innermost loops scans through the columns to copy the column headings. Then the second of these two loops scans through the rows. Finally, the innermost loop scans through the columns for each row.

We are grateful to Frances C. Bernstein for her helpful comments and suggestions.

References

ADSC (1997). *Quantum 4R specifications.* Poway, CA, USA: Area Detector Systems Corporation. http://www.adsc-xray.com/Q4techspecs.html.

Freed, N. & Borenstein, N. (1996). *Multipurpose Internet Mail Extensions (MIME) part one: format of internet message bodies.* RFC 2045. Internet Engineering Task Force. http://www.ietf.org/rfc/rfc2045.txt.

MAR Research (1997). *The mar345 image plate detector.* Norderstedt: MAR Research GmbH. http://www.marresearch.com/products.mar345.html.

Moffat, A., Bell, T. C. & Witten, I. H. (1997). *Lossless compression for text and images. Int. J. High Speed Electron. Syst.* **8**, 179–231.

Rivest, R. (1992). *The MD5 message-digest algorithm.* RFC 1321. Internet Engineering Task Force. http://www.ietf.org/rfc/rfc1321.txt.

Tosic, O. & Westbrook, J. D. (2000). *CIFParse. A library of access tools for mmCIF*. Reference guide. http://sw-tools.pdb.org/apps/CIFPARSE-OBJ/cifparse/index.html.