Table 5.6.2.5. *Values for the parameter compression in cbf_get_integerarrayparameters and cbf_set_integerarray*

CBF_CANONICAL	Canonical-code compression (Section 5.6.3.1)
CBF_PACKED	*CCP4*-style packing (Section 5.6.3.2)
CBF_NONE	No compression

current entry as a string. The functions cbf_get_integervalue and cbf_get_doublevalue interpret the retrieved string as an integer or real value and the functions cbf_set_integer and cbf_set_doublevalue convert the *number* argument into a string before setting the entry.

The functions for working with binary sections are more complicated as they must take into account compression, array size and the variety of different integer types available on different systems: signed/unsigned and various sizes.

The function cbf_get_integerarrayparameters retrieves the parameters of the current, binary, entry. The *compression* argument is set to the compression type used (Table 5.6.2.5). At present, this may take one of three values: CBF_CANONICAL, for canonical-code compression (see Section 5.6.3.1 below); CBF_PACKED, for *CCP4*-style packing (see Section 5.6.3.2 below); or CBF_NONE, for no compression. [*Note*: CBF_NONE is by far the slowest scheme of the three and uses much more disk space. It is intended for routine use with small arrays only. With large arrays (like images) it should be used only for debugging.] The *binary_id* value is a unique integer identifier for each binary section, *elsize* is the size in bytes of the array entries, *elsigned* and *elunsigned* are nonzero if the array can be read as unsigned or signed, respectively, *elements* is the number of entries in the array, and *minelement* and *maxelement* are the lowest and highest elements. If a destination argument is too small to hold a value, it will be set to the nearest value and the function will return CBF_OVERFLOW. If the current entry is not binary, cbf_get_integerarrayparameters will return CBF_ASCII.

cbf_get_integerarray reads the current binary entry into an integer array. The parameter *array* points to an array of elements interpreted as integers. Each element in the array is signed if *elsigned* is nonzero and unsigned otherwise, and each element occupies *elsize* bytes. The argument *elements_read* is set to the number of elements actually obtained. If the binary section does not contain sufficient entries to fill the array, the function returns CBF_ENDOFDATA. As before, the function will return CBF_OVERFLOW on overflow and CBF_ASCII if the entry is not binary.

cbf_set_integerarray sets the current binary or ASCII entry to the binary value of an integer array. As before, the acceptable values for compression are CBF_PACKED, CBF_CANONICAL and CBF_NONE. Each binary section should be given a unique integer identifier *binary_id*.

Two macros are provided to facilitate processing and propagation of error returns: one to return from the current function immediately and one to execute a given command first:

```
#define cbf_failnez(f) \
     {int err; err = (f); if (err) return err; }
#define cbf_onfailnez(f,c) \
     {int err; err = (f); if (err) {{c; }return err; }}
```

If the symbol CBFDEBUG is defined, alternative definitions that print out the error number as given in Table 5.6.1.1 are used:

```
#define cbf_failnez(x) \
{int err; err = (x); \
  if (err) { fprintf (stderr, \
     "\nCBFlib error %d in \"x\"\n", \
     err); return err; }}

#define cbf_onfailnez(x,c) \
{int err; err = (x); \
  if (err) { fprintf (stderr, \
    "\nCBFlib error %d in \"x\"\n", \
     err); \
     { c; } return err; }}
```

5.6.2.2. High-level *CBFlib* functions

The high-level *CBFlib* functions provide a level of abstraction above the CIF file structure and their prototypes are defined in the header file cbf_simple.h. Most of these functions simply use the low-level routines to navigate the CBF/imgCIF structure and read and modify data entries, and consequently expect a cbf_handle argument. There are also, however, additional sets of functions used to analyse the geometry of the goniometer and detector. These functions use additional handles of type cbf_goniometer and cbf_detector, respectively. All functions return the same error codes as the low-level functions do. The function return values are given in Table 5.6.1.1. The formal parameters for the high-level *CBFlib* functions are given in Table 5.6.2.6.

5.6.2.3. General high-level functions

The general high-level functions use the low-level routines to accomplish common tasks with a single call.

The first of these is used to facilitate the preparation of the complex CBF/imgCIF header structure:

```
int cbf_read_template (cbf_handle handle, FILE *file);
```

cbf_read_template simply reads the CBF/imgCIF file *file* into the data structure associated with the given handle and selects the first data block. It is typically used to read a template – an imgCIF file populated with data entries, but without any binary sections, into which experimental information can then be inserted. Template files are discussed further in Section 5.6.4 below.

The value of **_diffrn_radiation_wavelength.wavelength** can be retrieved or set. The functions

```
int cbf_get_wavelength (cbf_handle handle,
     double *wavelength);
int cbf_set_wavelength (cbf_handle handle,
     double wavelength);
```

operate on the categories DIFFRN_RADIATION and DIFFRN_RADIATION_WAVELENGTH. The wavelength is found indirectly. The value of **_diffrn_radiation.wavelength_id** is retrieved and used to find a matching row in the DIFFRN_RADIATION_WAVELENGTH category, from which the value of **_diffrn_radiation_wavelength.wavelength** is obtained.

The value of the ratio of the intensities of the polarization components **_diffrn_radiation.polarizn_source_ratio** and the value of the angle **_diffrn_radiation.polarizn_source_norm** between the normal to the polarization plane and the laboratory *Y* axis can be retrieved or set. The functions

```
int cbf_get_polarization (cbf_handle handle,
     double *polarizn_source_ratio,
     double *polarizn_source_norm);
int cbf_set_polarization (cbf_handle handle,
     double polarizn_source_ratio,
     double polarizn_source_norm);
```

operate on the DIFFRN_RADIATION category.

Table 5.6.2.6. *Formal parameters for high-level CBFlib functions*

array	Pointer to image array data
axis_id	Axis ID
center1	Displacement along the slow axis
center2	Displacement along the fast axis
compression	Compression type
coordinate1	*x* component
coordinate2	*y* component
coordinate3	*z* component
crystal_id	ASCII crystal ID
day	Timestamp day (1–31)
detector	Detector handle
diffrn_id	ASCII diffraction ID
distance	Distance
div_x_source	Value of **_diffrn_radiation.div_x_source**
div_x_y_source	Value of **_diffrn_radiation.div_x_y_source**
div_y_source	Value of **_diffrn_radiation.div_y_source**
element_id	ASCII element ID
element_number	Detector element counting from 0
elements	Count of elements
elsigned	Set to nonzero if the destination array elements are signed
elsize	Size in bytes of each destination array element
file	File descriptor
gain	Detector gain in counts per photon
gain_esd	Gain e.s.d. value
goniometer	Goniometer handle
handle	CBF handle
hour	Timestamp hour (0–23)
increment	Increment value
index1	Slow index
index2	Fast index
initial1	*x* component of the initial vector
initial2	*y* component of the initial vector
initial3	*z* component of the initial vector
minute	Timestamp minute (0–59)
month	Timestamp month (1–12)
ndim1	Slow array dimension
ndim2	Fast array dimension
normal1	*x* component of the normal vector
normal2	*y* component of the normal vector
normal3	*z* component of the normal vector
overload	Overload value
polarizn_source_norm	Polarization normal
polarizn_source_ratio	Polarization ratio
precision	Timestamp precision in seconds
projected_area	Apparent area in mm^2
ratio	Goniometer setting (0 = beginning of exposure, 1 = end)
real1	*x* component of the real-space vector
real2	*y* component of the real-space vector
real3	*z* component of the real-space vector
reciprocal1	*x* component of the reciprocal-space vector
reciprocal2	*y* component of the reciprocal-space vector
reciprocal3	*z* component of the reciprocal-space vector
reserved	Unused; any value other than 0 is invalid
second	Timestamp second (0–60.0)
start	Start value
time	Timestamp in seconds since 1 January 1970 or integration time in seconds
timezone	Time zone difference from universal time in minutes or CBF_NOTIMEZONE
vector1	*x* component of the rotation axis
vector2	*y* component of the rotation axis
vector3	*z* component of the rotation axis
wavelength	Wavelength in Å
year	Pointer to the destination timestamp year

The values of the divergence parameters, represented by the data names **_diffrn_radiation.div_x_source**, ***.div_y_source** and ***.div_x_y_source**, can be retrieved or set. The functions

```
int cbf_get_divergence (cbf_handle handle,
     double *div_x_source, double *div_y_source,
     double *div_x_y_source);
int cbf_set_divergence (cbf_handle handle,
     double div_x_source, double div_y_source,
     double div_x_y_source);
```

operate on the DIFFRN_RADIATION category.

The values of **_diffrn.id** and **_diffrn.crystal_id** can be retrieved or set:

```
int cbf_get_diffrn_id (cbf_handle handle,
     const char **diffrn_id);
int cbf_set_diffrn_id (cbf_handle handle,
     const char *diffrn_id);
int cbf_get_crystal_id (cbf_handle handle,
     const char **crystal_id);
int cbf_set_crystal_id (cbf_handle handle,
     const char *crystal_id);
```

Changing **_diffrn.id** also modifies the corresponding ***.diffrn_id** entries in the DIFFRN_SOURCE, DIFFRN_RADIATION, DIFFRN_DETECTOR and DIFFRN_MEASUREMENT categories.

The starting value and increment of an axis may be retrieved or set:

```
int cbf_get_axis_setting (cbf_handle handle,
     unsigned int reserved, const char *axis_id,
     double *start, double *increment);
int cbf_set_axis_setting (cbf_handle handle,
     unsigned int reserved, const char *axis_id,
     double start, double increment);
```

The cbf_set_axis_setting call is used during the creation of a CBF/imgCIF file to store the goniometer settings and rotation. The cbf_get_axis_setting is not generally useful when interpreting a file as there are no standard identifiers and the arrangement of the experimental axes is not consistent. Much more useful are the goniometer geometry functions described below.

The number of detector elements can be retrieved:

```
int cbf_count_elements (cbf_handle handle,
     unsigned int *elements);
```

This is the number of rows in the DIFFRN_DETECTOR_ELEMENT category. For each element, counting from 0, the detector identifier (the ***.detector_id** entry) can be retrieved and the gain and overload values in the ARRAY_INTENSITIES category retrieved or set:

```
int cbf_get_element_id (cbf_handle handle,
     unsigned int element_number,
     const char **element_id);
int cbf_get_gain (cbf_handle handle,
     unsigned int element_number,
     double *gain, double *gain_esd);
int cbf_set_gain (cbf_handle handle,
     unsigned int element_number, double gain,
     double gain_esd);
int cbf_get_overload (cbf_handle handle,
     unsigned int element_number, double *overload);
int cbf_set_overload (cbf_handle handle,
     unsigned int element_number, double overload);
```

For each element, counting from 0, the values of the parameters of the detector can be retrieved and some can be set. The value of **_diffrn_detector_element.id** is retrieved as *element_id*. The value of **_diffrn_data_frame.array_id** can be retrieved as *array_name*. The values of **_array_intensities.gain** and **_array_intensities.gain_esd** are retrieved as *gain*

and *gain_esd*. The value of **_array_intensities.overload** can be retrieved or set as *overload*. The value of **_diffrn_scan_frame.integration_time** can be retrieved or set as *integration_time*.

Timestamp calls operate on the DATE entry in the DIFFRN_SCAN_FRAME category:

```
int cbf_get_timestamp (cbf_handle handle,
     unsigned int reserved, double *time,
     int *timezone);
int cbf_set_timestamp (cbf_handle handle,
     unsigned int reserved, double time,
     int timezone, double precision);
int cbf_get_datestamp (cbf_handle handle,
     unsigned int reserved, int *year, int *month,
     int *day, int *hour, int *minute, double *second,
     int *timezone);
int cbf_set_datestamp (cbf_handle handle,
     unsigned int reserved, int year, int month,
     int day, int hour, int minute,  double second,
     int timezone, double precision);
int cbf_set_current_timestamp (cbf_handle handle,
     unsigned int reserved, int timezone)
```

cbf_get_timestamp and cbf_set_timestamp measure time in seconds since 1 January 1970. cbf_get_datestamp and cbf_set_datestamp work in terms of individual *year*, *month*, *day*, *hour*, *minute* and *second*. The optional collection time zone, *timezone*, is the difference from universal time in minutes; *precision* is the fraction, in seconds, to which the time will be recorded. cbf_set_current_timestamp sets the collection timestamp from the current time, to the nearest second.

Also in the DIFFRN_SCAN_FRAME category is the integration time of the image:

```
int cbf_get_integration_time (cbf_handle handle,
     unsigned int reserved, double *time);
int cbf_set_integration_time (cbf_handle handle,
     unsigned int reserved, double time);
```

Finally, these functions include routines for working with binary images:

```
int cbf_get_image_size (cbf_handle handle,
     unsigned int reserved,
     unsigned int element_number,
     size_t *ndim1, size_t *ndim2);
int cbf_get_image (cbf_handle handle,
     unsigned int reserved,
     unsigned int element_number,
     void *array, size_t elsize, int elsign,
     size_t ndim1, size_t ndim2);
int cbf_set_image (cbf_handle handle,
     unsigned int reserved,
     unsigned int element_number,
     unsigned int compression, void *array,
     size_t elsize, int elsign, size_t ndim1,
     size_t ndim2);
```

cbf_get_image_size retrieves the dimensions of detector element *element_number* from the ARRAY_STRUCTURE_LIST category, setting *ndim1* and *ndim2* to the slow and fast array dimensions, respectively. These dimensions can be used to allocate memory before calling cbf_get_image. cbf_get_image reads the image data from detector element *element_number* into a signed or unsigned integer array of size *ndim1 * ndim2* and cbf_set_image associates image data with a detector element. As in the description of the integer array functions, the compression argument can currently take one of three values: CBF_CANONICAL, for canonical-code compression (see Section 5.6.3.1); CBF_PACKED, for *CCP4*-style packing (see Section 5.6.3.2); or CBF_NONE, for no compression.

5.6.2.4. Goniometer geometry functions

A CBF/imgCIF file includes a geometric description of the goniometer used to orient the sample during the experiment. Practical use of this information, however, is not trivial as it involves combining data from several categories and analysing in three dimensions the nested axes in which the description is framed (see Section 3.7.3 for a discussion of the axis system). *CBFlib* provides six functions to facilitate this task:

```
int cbf_construct_goniometer (cbf_handle handle,
     cbf_goniometer *goniometer);
int cbf_free_goniometer (cbf_goniometer goniometer);
int cbf_get_rotation_axis (cbf_goniometer goniometer,
     unsigned int reserved, double *vector1,
     double *vector2, double *vector3);
int cbf_get_rotation_range (cbf_goniometer goniometer,
     unsigned int reserved, double *start,
     double *increment);
int cbf_rotate_vector (cbf_goniometer goniometer,
     unsigned int reserved, double ratio,
     double initial1, double initial2, double initial3,
     double *final1, double *final2, double *final3);
int cbf_get_reciprocal (cbf_goniometer goniometer,
     unsigned int reserved, double ratio,
     double wavelength, double real1, double real2,
     double real3, double *reciprocal1,
     double *reciprocal2, double *reciprocal3);
```

cbf_construct_goniometer uses the data in the categories DIFFRN_MEASUREMENT, DIFFRN_MEASUREMENT_AXIS, AXIS, DIFFRN_SCAN_FRAME_AXIS and DIFFRN_SCAN_AXIS to construct a geometric representation of the goniometer and initializes the cbf_goniometer handle, *goniometer*. cbf_free_goniometer frees the goniometer structure. cbf_get_ rotation_axis and cbf_get_rotation_range get the normalized rotation vector, and the starting value and increment of the first rotating axis of the goniometer, respectively. The cbf_rotate_vector call applies the goniometer axis rotation to the given initial vector, with the *ratio* value specifying the goniometer setting from 0.0 at the beginning of the exposure to 1.0 at the end, irrespective of the actual rotation range. Finally, cbf_get_reciprocal transforms the given real-space vector (*real1, real2, real3*) to the corresponding reciprocal-space vector (*reciprocal1, reciprocal2, reciprocal3*). As before, the transform corresponds to the goniometer initial position with a *ratio* of 0.0 and the goniometer final position with a *ratio* of 1.0.

5.6.2.5. Detector geometry functions

In a similar manner, a CBF/imgCIF file includes a description of the surface of each detector and the arrangement of the pixels in space. *CBFlib* provides eight functions for analysing this description:

```
int cbf_construct_detector (cbf_handle handle,
     cbf_detector *detector,
     unsigned int element_number);
int cbf_free_detector (cbf_detector detector);
int cbf_get_beam_center (cbf_detector detector,
     double *index1, double *index2,
     double *center1, double *center2);
int cbf_get_detector_distance (cbf_detector detector,
     double *distance);
```