```
int cbf_get_detector_normal (cbf_detector detector,
     double *normal1, double *normal2,
     double *normal3);
int cbf_get_pixel_coordinates (cbf_detector detector,
     double index1, double index2,
     double *coordinate1, double *coordinate2,
     double *coordinate3);
int cbf_get_pixel_normal (cbf_detector detector,
     double index1, double index2,
     double *normal1, double *normal2,
     double *normal3);
int cbf_get_pixel_area (cbf_detector detector,
     double index1, double index2,
     double *area, double *projected_area);
```

`cbf_construct_detector` uses data from the categories DIFFRN, DIFFRN_DETECTOR, DIFFRN_DETECTOR_ELEMENT, DIFFRN_DETECTOR_AXIS, AXIS, ARRAY_STRUCTURE_LIST and ARRAY_STRUCTURE_LIST_AXIS to construct a geometric representation of detector element *element_number* and initializes the `cbf_detector` handle, *detector*. `cbf_free_detector` frees the detector structure; `cbf_get_beam_center` calculates the location at which the beam intersects the detector surface, either in terms of the pixel indices (*index1, index2*) along the slow and fast detector axes, respectively, or the displacement in millimetres along the slow and fast axes (*center1, center2*); `cbf_get_detector_distance` and `cbf_get_detector_normal` calculate the distance of the sample from the plane of the detector surface and the normal vector of the detector at pixel $(0, 0)$, respectively; `cbf_get_pixel_coordinates`, `cbf_get_pixel_normal` and `cbf_get_pixel_area` calculate the coordinates, normal vector, and area and apparent area as viewed from the sample position of the pixel with the given indices, respectively.

5.6.3. Compression schemes

Two schemes for lossless compression of integer arrays (such as images) have been implemented in this version of *CBFlib*:

(i) an entropy-encoding scheme using canonical coding;

(ii) a *CCP4*-style packing scheme.

Both encode the difference (or error) between the current element in the array and the prior element. Parameters required for more sophisticated predictors have been included in the compression functions and will be used in a future version of the library.

5.6.3.1. Canonical-code compression

The canonical-code compression scheme encodes errors in two ways: directly or indirectly. Errors are coded directly using a symbol corresponding to the error value. Errors are coded indirectly using a symbol for the number of bits in the (signed) error, followed by the error itself.

At the start of the compression, *CBFlib* constructs a table containing a set of symbols, one for each of the 2^n direct codes from -2^{n-1} to $2^{n-1} - 1$, one for a stop code and one for each of the $maxbits - n$ indirect codes, where n is chosen at compression time and *maxbits* is the maximum number of bits in an error. *CBFlib* then assigns to each symbol a bit code, using a shorter bit code for the more common symbols and a longer bit code for the less common symbols. The bit-code lengths are calculated using a Huffman-type algorithm and the actual bit codes are constructed using the canonical-code algorithm described by Moffat *et al.* (1997).

Table 5.6.3.1. *Structure of compressed data using the canonical-code scheme*

Byte	Value
1 to 8	Number of elements (64-bit little-endian number)
9 to 16	Minimum element
17 to 24	Maximum element
25 to 32	(Reserved for future use)
33	Number of bits directly coded, n
34	Maximum number of bits encoded, *maxbits*
35 to $35 + 2^n - 1$	Number of bits in each direct code
$35 + 2^n$	Number of bits in the stop code
$35 + 2^n + 1$ to $35 + 2^n + maxbits - n$	Number of bits in each indirect code
$35 + 2^n + maxbits - n + 1 \ldots$	Coded data

Table 5.6.3.2. *Structure of compressed data using the CCP4-style scheme*

Value in bits 3 to 5	Number of bits in each error
0	0
1	4
2	5
3	6
4	7
5	8
6	16
7	65

Byte	Value
1 to 8	Number of elements (64-bit little-endian number)
9 to 16	Minimum element (currently unused)
17 to 24	Maximum element (currently unused)
25 to 32	(Reserved for future use)
33 . . .	Coded data

The structure of the compressed data is described in Table 5.6.3.1.

5.6.3.2. *CCP4*-style compression

The *CCP4*-style compression writes the errors in blocks. Each block begins with a 6-bit code. The number of errors in the block is 2^n, where n is the value in bits 0 to 2. Bits 3 to 5 encode the number of bits in each error. The data structure is summarized in Table 5.6.3.2.

5.6.4. Sample templates

The construction of CBF/imgCIF files can be simplified using templates. A template is itself an imgCIF file populated with data entries but without any binary sections. This file is normally associated with a CBF handle using the `cbf_read_template` call and provides a framework into which images and other experiment-specific data may be entered.

Fig. 5.6.4.1 is a sample template for an ADSC Quantum 4 detector (ADSC, 1997) with a κ-geometry diffractometer at Stanford Synchrotron Radiation Laboratory (SSRL) beamline 1-5.

The template for a MAR345 image plate detector (MAR Research, 1997) is almost identical. The major differences are in the size of the array (2300 × 2300 *versus* 2304 × 2304), the parameters for the CCD elements and the geometry of the elements. Therefore a few of the values in the AXIS, ARRAY_STRUCTURE_LIST, ARRAY_STRUCTURE_LIST_AXIS and ARRAY_INTENSITIES categories are different, as listed in Fig. 5.6.4.2.