International Tables for Crystallography (2006). Vol. G, Section 2.3.2, pp. 37–39.

2.3. Specification of the Crystallographic Binary File (CBF/imgCIF)

BY H. J. BERNSTEIN AND A. P. HAMMERSLEY

2.3.1. Introduction

The Crystallographic Binary File (CBF) format is a complementary format to the Crystallographic Information File (CIF) (Hall *et al.*, 1991) supporting efficient storage of large quantities of experimental data in a self-describing binary format. The image-supporting Crystallographic Information File (imgCIF) is an extension to CIF to assist in ASCII debugging and archiving of CBF files and to allow for convenient and standardized inclusion of images, such as maps, diagrams and molecular drawings, into CIFs for publication. The binary CBF format is useful for handling large images within laboratories and for interchange among collaborating groups. For smaller blocks of binary data, either format should be suitable. The ASCII imgCIF format is appropriate for interchange of smaller images and for long-term archiving.

CBF is designed to support efficient storage of raw experimental data (images) from area detectors with no loss of information, unlike some existing formats intended for this purpose. The format enables very efficient reading and writing of raw data, and encourages economical use of disk space. It may be coded easily and is portable across platforms. It is also flexible and extensible so that new data structures can be added without affecting the present definitions.

These goals are achieved by a simple file format, combining a CIF-like file header with compressed binary information. The file header consists of ASCII text giving information about the binary data as CIF tag–value pairs and tables. Each binary image is presented as a text-field value, either as raw octets of binary data in a CBF data set, or as an ASCII-based encoding of the same binary information in a true ASCII imgCIF data set. The ASCII-based encoded format uses e-mail MIME (Multipurpose Internet Mail Extensions) conventions to encode the binary data (Freed & Borenstein, 1996*a*,*b*,*c*; Freed *et al.*, 1996; Moore, 1996). The present version of the format tries to deal only with simple Cartesian data. These are essentially the 'raw' diffraction data that typically are stored in commercial formats or individual formats internal to particular institutes. Other forms of binary image data could be accommodated. It is hoped that CBF will replace individual laboratory or institute formats for 'home-built' detector systems, will be used as an inter-program data-exchange format, and will be offered as an output choice by commercial detector manufacturers specializing in X-ray and other detector systems. In this chapter we discuss the basic framework within which binary data and images are stored. The categories and data items that are used to describe beam and equipment axes, rastering methodologies, and image compression techniques are described in Chapter 3.7. The CBF/imgCIF dictionary is given in Chapter 4.6. An application programming interface (API) for the manipulation of image data is described in Chapter 5.6.

Affiliations: HERBERT J. BERNSTEIN, Department of Mathematics and Computer Science, Kramer Science Center, Dowling College, Idle Hour Blvd, Oakdale, NY 11769, USA; ANDREW P. HAMMERSLEY, ESRF/EMBL Grenoble, 6 rue Jules Horowitz, France.

2.3.2. CBF and imgCIF

CBF and imgCIF are two aspects of the same format. Since CIFs are pure ASCII text files, it was necessary to define a separate binary format to allow the combination of pseudo-ASCII sections and binary data sections. In the binary-file CBF format, the ASCII sections conform closely to the CIF standard but must use operating-system-independent 'line separators'. In order to facilitate interchange of files, an API that writes CBF files should use `\r\n` (carriage return, line feed) for the line separator. Use of this line separator allows the ASCII sections to be viewed with standard system utilities (*e.g.* 'more', 'pg') on a very wide range of operating systems (*e.g.* Unix, MacOS and Windows). However, an API that reads CBF format must accept any of the following three alternative line terminators as the end of an ASCII line: `\r`, `\n` or `\r\n`. As for all CIF data sets, an imgCIF file conforms to the normal text file-writing conventions of the system on which it is written. imgCIF is also one of the two names of the CIF dictionary (see Chapter 4.6) that contains the terms specific to describing image data in both CBF and imgCIF data sets. Thus a CBF or imgCIF data set uses data names from the CBF/imgCIF dictionary and other CIF dictionaries.

The general structure of a CBF or imgCIF data set is shown in Example 2.3.2.1. After a special comment to identify the file type (a so-called 'magic number') and any other initial comments, the data set begins with a '**`data_`*`blockname`***' which gives the name of the data block. Tags and values that describe the image data and how they were collected come next. For efficiency in processing, it is recommended that all the descriptive tags come before the actual image data. This recommendation is a *requirement* for the binary CBF format. It is *optional* for the ASCII imgCIF format. The image data are given as the value of the tag **`_array_data.data`**. The image data are given in a text field, using MIME conventions to describe the encoding.

2.3.2.1. A simple example

Before describing the format in full, we start by showing a simple but important and complete use of the format: that of storing a single detector image in a file together with a small amount of auxiliary information. This is intended to be a useful example that can be understood without reference to the full definitions. It also serves as an introduction or overview of the format definition. This example uses CIF DDL2-based dictionary items (see Chapter 2.6).

Example 2.3.2.2 relates to an image of 768 × 512 pixels stored as 16-bit unsigned integers, in little-endian byte order (this is the native byte ordering on a PC). The pixel sizes are 100.5 × 99.5 μm.

The example will be presented and discussed in three sections. The circled numerals (*e.g.* ①) are included to allow us to comment on portions of the example. They are not part of the CBF/imgCIF format.

The line marked by ①, starting with a hash character (**`#`**), is a CIF and CBF comment line. As a first line, the pattern of three hashes followed by 'CBF' helps to identify the data set as a CBF. It is a so-called 'magic number'. The text **`###CBF: VERSION`** must be present as the very first line of every CBF file. Following

Example 2.3.2.1. *General structure of a CBF or imgCIF data set. The critical values that define an image are marked with* ▌.

```
###CBF: VERSION 1.0
  data_blockname

  # cif tags and values describing the image, e.g.
  loop_
   _array_intensities.array_id
   _array_intensities.binary_id
   _array_intensities.linearity
   _array_intensities.undefined_value
   _array_intensities.overload
     image_1     1   linear     0      65535

# the image data are given as the value of
# the tag _array_data.data, usually in a loop_
# at the end of the data block.  The first two
# values identify the structure of the image
# and assure a unique identifier if there are
# multiple images of the same structure.

  loop_
   _array_data.array_id
   _array_data.binary_id
   _array_data.data

▌    image_1
▌    1

# The image itself begins and ends as a CIF
# text field, within which MIME conventions
# are used to describe the encoding of the image

▌;
▌--CIF-BINARY-FORMAT-SECTION--
▌Content-Type: application/octet-stream;
▌     conversions="x-CBF_PACKED"
▌Content-Transfer-Encoding: BINARY
▌X-Binary-Size: 374578
▌X-Binary-ID: 1
▌X-Binary-Element-Type:  "unsigned 16-bit integer"
▌Content-MD5: jGmkxkrpnizOetd9T/Np4NufAmA==

▌START_OF_BIN
▌*************<D5>9*****<D4>********* ...
▌[This is where the raw binary data would be – we can't print them here]
▌--CIF-BINARY-FORMAT-SECTION----
▌;
```

'VERSION' is the number of the corresponding version of the CBF/imgCIF extension dictionary and supporting documentation. Comment lines and white space (blanks and newlines) may appear anywhere outside the binary sections. In an imgCIF data set, the descriptive tags and values may be presented in any convenient order, *e.g.* the data could come first and the parameters necessary to interpret the data could come later. This order-independent convention holds for an imgCIF file, but for a CBF all the tags and values describing binary data (*i.e.* all the tags other than those in the ARRAY_DATA category) should be presented before the binary data, in the form of a header. This does not mean that there cannot be more useful information after the binary data. There could be another full header and more blocks of binary data. In the interest of efficiency in processing a CBF, the parameters that relate to a particular block of binary data must appear earlier in the CBF than the block itself.

The header begins at the line marked with ②. The `data_` token is the CIF token for identifying a data block. The name of the data block, `image_1`, follows immediately without any intervening white space. The name of the data block is arbitrary. Within a data block any given tag may be presented only once, either directly with a value following immediately, or as one of the column headings for the rows of a table. To reuse the same tag one must start a new data block.

Example 2.3.2.2. *A single image.*

```
###CBF: VERSION 1.0                                  ①
data_image_1                                         ②

_entry.id                                 'image_1'  ③
_chemical.entry_id                        'image_1'
_chemical.name_common                     'Protein X'

# Experimental details
_exptl_crystal.id                         'CX-1A'    ④
_exptl_crystal.colour                     'pale yellow'

_diffrn.id                                 DS1
_diffrn.crystal_id                        'CX-1A'

_diffrn_measurement.diffrn_id              DS1
_diffrn_measurement.method                 Oscillation
_diffrn_radiation_wavelength.id            L1
_diffrn_radiation_wavelength.wavelength
                                           0.7653
_diffrn_radiation_wavelength.wt            1.0

_diffrn_radiation.diffrn_id                DS1
_diffrn_radiation.wavelength_id            L1

_diffrn_source.diffrn_id                   DS1
_diffrn_source.source                      synchrotron
_diffrn_source.type                       'ESRF BM-14'

_diffrn_detector.diffrn_id                 DS1
_diffrn_detector.id                        ESRFCCD1
_diffrn_detector.detector                  CCD
_diffrn_detector.type              'ESRF Be XRII/CCD'

_diffrn_detector_element.id                1
_diffrn_detector_element.detector_id       ESRFCCD1

_diffrn_frame_data.id                      F1
_diffrn_frame_data.detector_element_id     1
_diffrn_frame_data.array_id                'image_1'
_diffrn_frame_data.binary_id               1
```

Information about the image begins at the line marked with ③. In the following lines, the apostrophes enclose strings that contain a space. Values that contain white space, or that could be confused with a CIF token, must always be quoted. A double quote (`"`) could have been used. There is a third way to quote a string, with the string `\r\n;`, *i.e.* with a semicolon at the beginning of a line, which allows multi-line strings to be presented. We shall use this form of text quotation for the binary data.

The experimental details begin at the line marked with ④. Many more data items could be defined, but here we are giving an example of one useful minimal (but not mandatory) set. See the imgCIF dictionary in Chapter 4.6 and the classification of image data in Chapter 3.7 for a discussion of which items are mandatory.

After describing the parameters of the experiment, we describe the organization of the image data (Example 2.3.2.3).

Note that we have changed from listing a value directly with each tag to a tabular format, using the CIF `loop_` token.

The `*.array_id` tags identify data items belonging to the same array. Here we have chosen the name `image_1`, but another name could have been used, as long as it is used consistently. The `*.index` tags refer to the dimension being defined, and the `*.dimension` column defines the number of elements in that dimension. The `*.precedence` tag defines the precedence of rastering of the data. In this case, the first dimension is the faster changing dimension. The `*.direction` column tells us the direction in which the data raster runs within a dimension. Here the data raster runs from the minimum element towards the maximum element ('increasing') in the first dimension, and from the maximum element towards the minimum element in the second dimension. This is the default rastering order.

Example 2.3.2.3. *Organization of image data in a CBF/imgCIF.*

```
# Define image storage mechanism
loop_
_array_structure.id
_array_structure.encoding_type
_array_structure.compression_type
_array_structure.byte_order
image_1  "unsigned 16-bit integer"  none
little_endian

loop_
_array_intensities.array_id
_array_intensities.binary_id
_array_intensities.linearity
_array_intensities.undefined_value
_array_intensities.overload
image_1     1   linear     0      65535

loop_
_array_structure_list.array_id
_array_structure_list.index
_array_structure_list.dimension
_array_structure_list.precedence
_array_structure_list.direction
image_1  1  768  1  increasing
image_1  2  512  2  decreasing

loop_
_array_element_size.array_id
_array_element_size.index
_array_element_size.size
image_1  1  100.5e-6
image_1  2  99.5e-6
```

We have given the abstract ordering of the data. The physical view of data is described in detail in the CBF/imgCIF dictionary (Chapters 3.7 and 4.6).

In general, the physical sense of the image is from the sample to the detector.

The storage of the binary data is now fully defined. Further data items could be defined, but we are ready to present the image data (Example 2.3.2.4). This is done with the ARRAY_DATA category. The actual binary data will come just a little further down, as the essential part of the value of **_array_data.data**, which begins as semicolon-quoted text.

The line immediately after the line with the semicolon is a MIME boundary marker. As with all MIME boundary markers, it begins with '--' (two hyphens). The next few lines are MIME headers, describing some useful information we will need in order to process the binary section. MIME headers can appear in different orders and can be very confusing (look at the raw contents of an e-mail message with attachments), but there are only a few headers that have to be understood to process a CBF.

The 'Content-Type' header serves to describe the nature of the following data sufficiently to allow an association with an appropriate agent or mechanism for presenting the data to a user. It may be any of the discrete types permitted in RFC 2045 (Freed & Borenstein, 1996*b*), but, unless the binary data conform to an existing standard format (*e.g.* TIFF or JPEG), the description 'application/octet-stream' is recommended. If the octet stream has been compressed, the compression should be specified by the parameter **conversions="x-CBF_PACKED"** or by specifying one of the other compression types allowed as described in Chapter 5.6.

The 'Content-Transfer-Encoding' header describes any encoding scheme applied to the data, most commonly to transform it to an ASCII-only representation. For a CBF the value should be 'BINARY'. We consider the other values used for imgCIF below.

The 'X-Binary-Size' header specifies the size of the binary data in octets. Calculation of the size where compression is used

Example 2.3.2.4. *Representation of the binary data.*

```
loop_
_array_data.array_id
_array_data.binary_id
_array_data.data

image_1 1
;
--CIF-BINARY-FORMAT-SECTION--
Content-Type: application/octet-stream;
     conversions="x-CBF_PACKED"
Content-Transfer-Encoding: BINARY
X-Binary-Size: 374578
X-Binary-ID: 1
X-Binary-Element-Type: "unsigned 16-bit integer"
Content-MD5: jGmkxkrpnizOetd9T/Np4NufAmA==

START_OF_BIN
************<D5>9******<D4>********* ...
[This is where the raw binary data would be – we can't print them here]
--CIF-BINARY-FORMAT-SECTION----
;
```

is described in Section 2.3.3.3. The 'X-Binary-Element-Type' header specifies the type of binary data in the octets, using the same descriptive phrases as in **_array_structure.encoding_type** (the default value is 'unsigned 32-bit integer').

The other MIME headers in the example provide an identifier and a content checksum. The MIME header items are followed by an empty line and then by a special sequence (marked here as **START_OF_BIN**), consisting of the single characters Ctrl-L, Ctrl-Z, Ctrl-D and a single binary flag character of hexadecimal value D5 (213 decimal). The binary data follow immediately after this flag character. The reasons for choosing this sequence are discussed in Section 2.3.3.3.

After the last octet (*i.e.* byte) of the binary data, there is a special trailer `\r\n--CIF-BINARY-FORMAT-SECTION----\r\n;`. This repeats the initial boundary marker with an extra `--` at the end (a MIME convention for the last boundary marker), followed by a closing semicolon quote for a text section. This is essential in an imgCIF, and we include it in a CBF for consistency.

2.3.3. Overview of the format

This section describes the major elements of the CBF format.

(1) CBF is a binary file, containing self-describing array data, *e.g.* one or more images, and auxiliary data, *e.g.* describing the experiment.

(2) Apart from the handling of line terminators, the way binary data are presented and more liberal rules for ordering information, an ASCII imgCIF file is the same as a CBF binary file.

(3) A CBF consists of pseudo-ASCII text header sections consisting of 'lines' of no more than 80 ASCII characters separated by 'line separators', which are the pair of ASCII characters carriage return (`\r`, ASCII 13) and line feed (`\n`, ASCII 10), followed by zero, one or more binary sections presented as 'binary strings'. The file returns to the pseudo-ASCII format after each string, allowing additional binary strings to appear later in the file after additional headers.

(4) An imgCIF consists of ASCII lines of no more than 80 characters using the the normal line-termination conventions of the current system (*e.g.* `\n` in UNIX) with MIME-encoded binary strings at any appropriate point in the file. (For both CBF and imgCIF, the limitation of 80 characters per line will be increased to 2048 as CIF 1.1 is adopted.)