International Tables for Crystallography (2006). Vol. G, Chapter 2.3, pp. 37–43.

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
```

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----
;
```

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 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.)

(5) The very start of the file has an identification item (magic number). This item also describes the CBF version or level (see Section 2.3.3.1 below).

(6) In the CBF binary format, the descriptive tags and values may be viewed as a 'header' section for the binary data section. The start of a header section is delimited by the usual CIF `data_` token (see Section 2.3.3.2 below).

(7) The header information must contain sufficient data names to fully describe the binary data sections.

(8) The binary data are presented as 'binary strings', which in CBF are specially formatted binary data within a semicolon-delimited text string. In imgCIF files, the binary data are MIME-encoded within true ASCII text fields.

(9) White space may be used within the pseudo-ASCII sections prior to the 'start of binary section' identifier to align the start-binary data sections to word or block boundaries. Similar use may be made of unused bytes within binary sections. However, no blank lines should be introduced among the MIME headers, since that would terminate processing of those headers and start the scan for binary data. In general, no guarantee is made of block or word alignment in a CBF of unknown origin.

(10) The end of the file need not be explicitly indicated, but including a comment of the form `###_END_OF_CBF` (including the carriage return, line feed pair) can help in debugging.

(11) For the most efficient processing of a CBF file, all binary data described in a single data block should appear as the last information in that data block. However, since binary strings can be parsed anywhere within the context of a CBF or imgCIF file, it is *recommended* that processing software for CBF accept such strings in any order, and it is *mandatory* that processing software for imgCIF accept such strings in any order. The binary identifier values used within a given data block section, and hence the binary data, must be unique for any given `*.array_id`, and it would be best to make them globally unique. However, a different data block may reuse binary identifier values. (This allows concatenation of files without renumbering the binary identifiers, and provides a certain level of localization of data within the file to avoid programs having to search potentially huge files for missing binary sections.)

(12) The recommended file extension for a CBF is 'cbf'. This allows users to recognize file types easily and gives programs a chance to 'know' the file type without having to prompt the user. However, a program should check for at least the file identifier to ensure that the file type is indeed CBF.

(13) The recommended file extensions for imgCIF are 'icf' or 'cif'.

(14) CBF format files are binary files, so when ftp is used to transfer files between different computer systems 'binary' or 'image' mode transfer should be selected.

(15) imgCIF files are ASCII files, so when ftp is used to transfer files between different computer systems 'ascii' transfer should be selected.

2.3.3.1. Details of the magic number

The magic number identifier is `###CBF: VERSION`. This must always be present so that a program can easily identify a CBF by simply inputting the first 15 characters. [The space is a blank (ASCII 32) and not a tab. All identifier characters must be upper case.] The first hash means that this line within the CIF is a comment line. Three hashes mean that this is a line describing the binary file layout for CBF. (All CBF internal identifiers start with three hashes, and all others must immediately follow a 'line separator'.) No white space may precede the first hash sign. Following the file identifier is the version number of the file; *e.g.* the full line might appear as `###CBF: VERSION 1.0`. The version number must be separated from the file identifier characters by white space, *e.g.* a blank (ASCII 32). The version number is defined as a major version number and minor version number separated by a decimal point. A change in the major version may mean that a program for the previous version cannot input the new version as some major change has occurred to CBF. A change in the minor version may also mean incompatibility if the CBF has been written using some new feature. For example, a new form of linearity scaling may be specified; this would be considered a minor version change. A file with the new feature would not be readable by a program supporting only an older version of the format.

2.3.3.2. Details of the header section

The start of a header section is delimited by the usual CIF `data_` token. Optionally, a header identifier, `###_START_OF_HEADER`, may be used before the `data_` token, followed by the carriage return, line feed pair, as an aid in debugging, but it is not required. (Naturally, another carriage return, line feed pair should immediately precede this and all other CBF identifiers, with the exception of the CBF file identifier at the very start of the file.) A header section, including the identification items which delimit it, uses only ASCII characters and is divided into 'lines'. The 'line separator' symbols `\r\n` (carriage return, line feed) are the same regardless of the operating system on which the file is written. This is an important difference from CIF, but must be so, as the file contains binary data and cannot be translated from one operating system to another, which is the case for ASCII text files. While a properly functioning CBF API should write the full `\r\n` line separator, it should recognize any of three sequences `\r`, `\n`, `\r\n` as valid line separators so that hand-edited headers will not be rejected.

The header section in a CBF obeys all CIF rules (Chapter 2.2) with the exception of the line separators, *i.e.*:

(i) 'Lines' are a maximum of 80 characters long. (This will be increased to 2048 in line with the CIF version 1.1 specification.)

(ii) All data names (tags) start with an underscore character '_'.

(iii) The hash symbol '`#`' (outside a quoted character string) means that all text up to the line separator is a comment.

(iv) White space outside character strings is not significant.

(v) Data names are case-insensitive.

(vi) The data item follows the data-name separator and may be of one of two types: character string (char) or number (numb). The type is specified for each data name.

(vii) Character strings may be delimited with single or double quotes, or blocks of text may be delimited by semicolons occurring as the first character on a line.

(viii) The `loop_` mechanism allows a data name to have multiple values. Immediately following the `loop_`, one or more data names are listed without their values, as column headings. Then one or more rows of values are given.

(ix) Any CIF data name may occur within the header section.

The tokens `data_` and `loop_` have special meaning in CIF and should not be used except in their indicated places. The tokens `save_`, `stop_` and `global_` also have special meaning in CIF's parent language, STAR, and should also not be used.

A single header section may contain one or more `data_` blocks.

2.3.3.3. Details of binary sections

All binary sections are contained within semicolon-delimited text fields, which may be thought of as 'binary strings'. Each such binary string must be given as a value of an appropriate CIF tag (usually `_array_data.data`), either directly or within a loop.

Before getting to the actual binary data, there are some preliminaries to allow a smooth transition from the conventions of CIF to those of raw streams of 'octets' (8-bit bytes). Within the binary-data text string, the conventions developed for transmitting e-mail messages including binary attachments are followed. There is secondary ASCII header information, formatted as MIME headers [see RFCs 2045–2049 by Freed & Borenstein (1996*a*,*b*,*c*), Freed *et al.* (1996) and Moore (1996)]. The boundary marker for the beginning of all this is the special string `--CIF-BINARY-FORMAT-SECTION--` at the beginning of a line. The initial '`--`' says that this is a MIME boundary. We cannot put '`###`' in front of it and conform to MIME conventions.

Immediately after the boundary marker are MIME headers, describing some useful information we will need to process the binary section. Only a few headers with a narrow range of values have to be understood to process a CBF (as opposed to an imgCIF, for which the headers can be more varied).

The 'Content-Type' header may be any of the discrete types permitted in RFC 2045 (Freed & Borenstein, 1996*b*); 'application/octet-stream' is recommended. If an octet stream was compressed, the compression should be specified by the parameter `conversions="x-CBF_PACKED"` or by the parameter `conversions="x-CBF_CANONICAL"`. Other compression schemes may be added at a future date. Details of the compression schemes are given in Chapter 5.6.

The 'Content-Transfer-Encoding' header should be `BINARY` for a CBF. For an ASCII imgCIF file, the 'Content-Transfer-Encoding' header may be `BASE64`, `QUOTED-PRINTABLE`, `X-BASE8`, `X-BASE10` or `X-BASE16`. The `BASE64` encoding is a reasonably efficient encoding of octets (approximately eight bits for every six bits of data) and is recommended for imgCIF files. The octal, decimal and hexadecimal transfer encodings are for convenience in debugging and are not recommended for archiving and data interchange.

The 'X-Binary-Size' header specifies the size of the binary data in octets. If compression was used, this size is the size after compression, including any book-keeping fields. In general, no portion of the binary header is included in the calculation of the size.

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 full list of valid types is:

```
unsigned 8-bit integer
signed 8-bit integer
unsigned 16-bit integer
signed 16-bit integer
unsigned 32-bit integer
signed 32-bit integer
signed 32-bit real IEEE
signed 64-bit real IEEE
signed 32-bit complex IEEE
```

The default value is `unsigned 32-bit integer`. See IEEE (1985) for details on the IEEE formats.

The MIME header items are followed by an empty line and then by a special sequence marked here as `START_OF_BIN`, consisting of 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 control (Ctrl-. . .) characters are inserted as a convenience to inhibit accidental listing or printing of large amounts of binary data on character-oriented output devices such as terminals or printers.

In general, if the value given for 'Content-Transfer-Encoding' is one of the real encodings `BASE64`, `QUOTED-PRINTABLE`, `X-BASE8`, `X-BASE10` or `X-BASE16`, this file is an imgCIF. More details on the imgCIF encodings are given in Section 2.3.5 below.

For either a CBF or an imgCIF, the optional 'Content-MD5' header provides a sophisticated check [the 'RSA Data Security, Inc. MD5 Message-Digest Algorithm' (Rivest, 1992)] on the integrity of the binary data.

In a CBF, the raw binary data begin after an empty line terminating the MIME headers and after the `START_OF_BIN` identifier. `START_OF_BIN` contains bytes to separate the ASCII lines from the binary data, bytes to try to stop the listing of the header, bytes which define the binary identifier, which should match the `*.binary_id` defined in the header, and bytes which define the length of the binary section.

Octet	Hexadecimal	Decimal	Purpose
1	0C	12 (Ctrl-L)	End the current page
2	1A	26 (Ctrl-Z)	Stop listings in MS-DOS
3	04	04 (Ctrl-D)	Stop listings in Unix
4	D5	213	Binary section begins
$5 \ldots 5+n-1$			Binary data (n octets)

Only bytes $5 \ldots 5+n-1$ are encoded for an imgCIF file using the indicated Content-Transfer-Encoding.

The binary characters serve specific purposes:

(i) The Ctrl-L will terminate the current page in listings in most operating systems.

(ii) The Ctrl-Z will stop the listing of the file in MS-DOS-type operating systems.

(iii) The Ctrl-D will stop the listing of the file in Unix-type operating systems.

(iv) The unsigned byte value 213 (decimal) is binary 11010101 (octal 325, hexadecimal D5). This has the eighth bit set, so it can be used to detect the error of seven-bit rather than eight-bit transmission. It is also asymmetric, providing one last check for reversal of bit order within bytes.

The carriage return, line feed pair before the `START_OF_BIN` and other lines can also be used to check that the file has not been corrupted (*e.g.* by being sent by ftp in ASCII mode).

The 'line separator' immediately precedes the 'start of binary identifier', but blank spaces may be added prior to the preceding 'line separator' if desired (*e.g.* to force word or block alignment). The binary data do not have to completely fill the bytes defined by the byte-length value, but clearly cannot be greater than this value (except when the value zero has been stored, which means that the size is unknown, and no other headers follow). The values of any unused bytes are undefined. The end of binary section identifier is placed exactly at the byte following the full binary section as defined by the length value. The end of binary section identifier consists of the carriage return/line feed pair followed by

```
--CIF-BINARY-FORMAT-SECTION----
;
```

with each of these lines followed by the carriage return/line feed pair. This brings us back into a normal CIF environment.

The first 'line separator' separates the binary data from the pseudo-ASCII line. The end-of-binary-section identifier is in a sense redundant, since the binary-data-length value tells a program how many bytes to jump over to the end of the binary data. However, this redundancy has been deliberately added for error checking, and for possible file recovery in the case of a corrupted file. This identifier must be present at the end of every block of binary data.

Example 2.3.4.1. *A multiple-block CBF with several images.*

```
###CBF: VERSION 1.0
# CBF file written by cbflib v0.6

# A comment cannot appear before the file identifier,
# but can appear anywhere else, except within the
# binary sections.

# Here the first data block starts
data_xxx

### ... various CIF tags and values here
###     but none that define array data items

# The "data_" identifier finishes the first data
# block and starts the second
data_yyy

### ... various CIF tags and values here including
###     ones that define array data items

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: 3745758
X-Binary-ID: 1
X-Binary-Element-Type: "signed 32-bit integer"
Content-MD5: 1zsJjWPfol2GYl2V+QSXrw==

START_OF_BIN
<D5>^P<B8>P^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ...
[This is where the raw binary data would be – we can't print them here]

--CIF-BINARY-FORMAT-SECTION----
;
```

Example 2.3.4.1. *(cont.)*

```
# Following the "end of binary" identifier the file
# is pseudo-ASCII again, so comments are valid
# up to the next "start of binary" identifier.  Note
# that we have increased the binary ID by one.

image_1 2
;
--CIF-BINARY-FORMAT-SECTION--
Content-Type: application/octet-stream;
     conversions="x-CBF_PACKED"
Content-Transfer-Encoding: BINARY
X-Binary-Size: 3745758
X-Binary-ID: 2
X-Binary-Element-Type: "signed 32-bit integer"
Content-MD5: xR5kxiOetd9T/Nr5vMfAmA==

START_OF_BIN
<D5>^P<B8>P^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ...
[This is where the raw binary data would be – we can't print them here]

--CIF-BINARY-FORMAT-SECTION----
;

# Third binary section; note that we have a new
# array ID.

image_2 3
;
--CIF-BINARY-FORMAT-SECTION--
Content-Type: application/octet-stream;
     conversions="x-CBF_PACKED"
Content-Transfer-Encoding: BINARY
X-Binary-ID: 3
Content-MD5: yS5kxiOetd9T/NrqTLfAmA==

START_OF_BIN
*************'9*****`********* ...
[This is where the raw binary data would be – we can't print them here]

--CIF-BINARY-FORMAT-SECTION----
;

# Second header section
data_zzz

### ... various CIF tags and values here
###     including ones that define array
###     data items

# Since we only have one block left, we won't
# use a loop

_array_data.array_id   image
_array_data.binary_id  1
_array_data.data

# Note that we can put a comment here
;
--CIF-BINARY-FORMAT-SECTION--
Content-Type: application/octet-stream;
     conversions="x-CBF_PACKED"
Content-Transfer-Encoding: BINARY
X-Binary-ID: 1
Content-MD5: fooxiOetd9T/serNufAmA==

START_OF_BIN
*************'9*****`********* ...
[This is where the raw binary data would be – we can't print them here]

--CIF-BINARY-FORMAT-SECTION----
;

###_END_OF_CBF
```

2.3.4. A complex example

In Example 2.3.4.1 only the ARRAY_DATA category CIF identifiers are shown. The other CIF data items are not shown. This shows a possible structuring of a more complicated example. There are two header sections; the first contains two data blocks and defines three binary sections. CIF comment lines, starting with a hash mark '#', are used to clarify the structure.

2.3.5. imgCIF encodings

For an imgCIF, there are several alternative encodings for binary image data as ASCII text. Each binary image may use a different encoding in the same imgCIF data set or even in the same data block. The choice of encoding is specified in the 'Content-Transfer-Encoding' MIME header.

If the Transfer Encoding is `X-BASE8`, `X-BASE10` or `X-BASE16`, the data are presented as octal, decimal or hexadecimal data, respectively, organized into lines or words. Each word is created by composing octets of data in fixed groups of 2, 3, 4, 6 or 8 octets, either in the order . . .4321 ('big-endian') or 1234. . . ('little-endian'). If there are fewer than the specified number of octets to fill the last word, then the missing octets are presented as '`==`' for each missing octet. Exactly two equal signs are used for each missing octet even for octal and decimal encoding. The format of lines is

rnd xxxxxx xxxxxx xxxxxx

where *r* is H, O or D for hexadecimal, octal or decimal, *n* is the number of octets per word, and *d* is '<' or '>' for the '. . .4321' and '1234. . .' octet orderings, respectively. The '`==`' padding for the last word should be on the appropriate side to correspond to the missing octets, *e.g.*

```
H4< FFFFFFFF FFFFFFFF 07FFFFFF ====0000
```

or

```
H3> FF0700 00====
```

For these hexadecimal, octal and decimal formats only, comments beginning with '`#`' are permitted to improve readability.

BASE64 encoding follows MIME conventions. Octets are in groups of three, $c1$, $c2$, $c3$. The resulting 24 bits are reorganized in the following way (where we use the C operators $\gg$, $\ll$, & and | to denote, respectively, a right shift, a left shift, bit-wise intersection and bit-wise union). Four six-bit quantities are specified, starting with the high-order six bits $(c1 \gg 2)$ of the first octet, then the low-order two bits of the first octet followed by the high-order four bits of the second octet $((c1 \,\&\, 3) \ll 4 \,|\, (c2 \gg 4))$, then the bottom four bits of the second octet followed by the high-order two bits of the last octet $((c2 \,\&\, 15) \ll 2 \,|\, (c3 \gg 6))$, then the bottom six bits of the last octet $(c3 \,\&\, 63)$. Each of these four quantities is translated into an ASCII character using the mapping

```
          1         2         3         4
01234567890123456789012345678901234567890123456789
 | | | | | | |
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwx
5         6
01234567890123
 | | | | | | |
yz0123456789+/
```

Short groups of octets are padded on the right with one '`=`' if $c3$ is missing, and with '`==`' if both $c2$ and $c3$ are missing.

QUOTED-PRINTABLE encoding also follows MIME conventions, copying octets without translation if their ASCII values are 32...38, 42, 48...57, 59...60, 62, 64...126 and the octet is not a '`;`' in column 1. All other characters are translated to '`=nn`', where `nn` is the hexadecimal encoding of the octet. All lines are 'wrapped' with a terminating '`=`' (*i.e.* the MIME conventions for an implicit line terminator are never used).

Appendix 2.3.1
Deprecated CBF conventions

There was an earlier, now deprecated, CBF format in which the compression type was given as eight bytes of binary header. In this case, the eight bytes used for the compression type are subtracted from the size, so that the same size will be reported if the compression type is supplied in the MIME header. Use of the MIME header is the recommended way to supply the compression type.

These earlier versions of the specification also included three eight-byte words of information in binary that replicated information now available in the MIME header:

5...12	Binary section identifier (see `_array_data.binary_id`), 64-bit, little-endian
13...20	The size (n) of the binary section in octets (*i.e.* the offset from octet 29 to the first byte following the data)
21...28	Compression type:
	`CBF_NONE` 0x0040 (64)
	`CBF_CANONICAL` 0x0050 (80)
	`CBF_PACKED` 0x0060 (96)
	...

The three eight-byte words were followed by binary data. These words are not included when a MIME header is provided.

We are grateful to Frances C. Bernstein, Sydney R. Hall, Brian McMahon and Nick Spadaccini for their helpful comments and suggestions.

References

Freed, N. & Borenstein, N. (1996*a*). *Multipurpose Internet Mail Extensions (MIME) part five: Conformance criteria and examples.* RFC 2049. Network Working Group. http://www.ietf.org/rfc/rfc2049.txt.

Freed, N. & Borenstein, N. (1996*b*). *Multipurpose Internet Mail Extensions (MIME) part one: Format of Internet message bodies.* RFC 2045. Network Working Group. http://www.ietf.org/rfc/rfc2045.txt.

Freed, N. & Borenstein, N. (1996*c*). *Multipurpose Internet Mail Extensions (MIME) part two: Media types.* RFC 2046. Network Working Group. http://www.ietf.org/rfc/rfc2046.txt.

Freed, N., Klensin, J. & Postel, J. (1996). *Multipurpose Internet Mail Extensions (MIME) part four: Registration procedures.* RFC 2048. Network Working Group. http://www.ietf.org/rfc/rfc2048.txt.

Hall, S. R., Allen, F. H. & Brown, I. D. (1991). *The Crystallographic Information File (CIF): a new standard archive file for crystallography. Acta Cryst.* A**47**, 655–685.

IEEE (1985). *IEEE standard for binary floating-point arithmetic.* ANSI/IEEE Std 754–1985. New York: The Institute of Electrical and Electronics Engineers, Inc.

Moore, K. (1996). *MIME (Multipurpose Internet Mail Extensions) part three: Message header extensions for non-ASCII text.* RFC 2047. Network Working Group. http://www.ietf.org/rfc/rfc2047.txt.

Rivest, R. (1992). *The MD5 message-digest algorithm.* RFC 1321. Network Working Group. http://www.ietf.org/rfc/rfc1321.txt.