International Tables for Crystallography (2006). Vol. G, Section 2.3.3, pp. 39–41.

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 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').

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----
;
```

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.