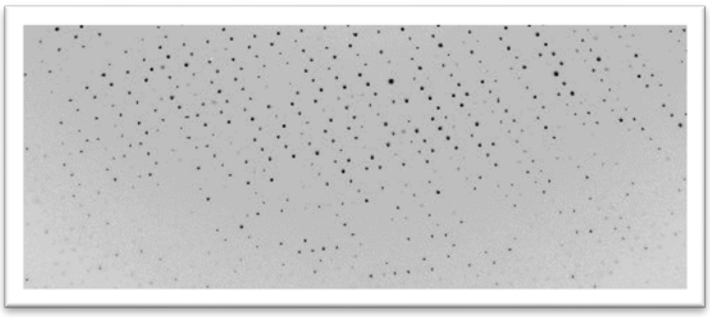
DECTRIS®

PILATUS CBF
Header Specification



Version 1.4

Table of Contents

1 Document History

Current document

Version	Date	status	prepared	checked	released
1.4	20.02.2013	released	MM	SB	MM

1.1 Changes

Version	Date	Changes	released
1.0	23.08.2011	Initial document	CS
1.1	02.09.2011	“Detector” keyword updated	MM
1.2	26.01.2012	“Detector” keyword updated; Appendix A with Python code updated	MM
1.3	19.10.2012	"Flux" keyword updated	MM
1.4	20.02.2013	Optional keywords "Omega" and "Omega_increment" added. This suggestion by Global Phasing Ltd. now allows to fully describe settings of an Eulerian cradle goniostat. Table matching header version and specification version in 1.2 added. Recommendations for reporting parameters of multi-axis goniostats added.	MM

1.2 CBF header version and specification version

This table lists which version of the Pilatus CBF header, as reported by the '_array_data.header_convention' *data item* (see 4), is described by which version(s) of the CBF header specification, this document.

Pilatus CBF header version	Pilatus CBF specification version(s)
1.2	1.0 - 1.4

2 Address and Support

DECTRIS Ltd.
Neuenhoferstrasse 107
5400 Baden
Switzerland
Phone: +41 56 500 21 00
Fax: + 41 56 500 21 01

Email: support@dectris.com

Should you have questions concerning the system or its use, please contact us via phone, mail or fax.

3 Preamble

This documentation describes the format of the Pilatus CBF header embedded in minimal CBF file written by the DECTRIS Detector control software “camserver”. The aim of this documentation is to provide users and software developers with specifications and guidelines for reading and parsing Pilatus CBF headers.
In addition to this documentation, DECTRIS provides a reference implementation in Python for parsing Pilatus CBF headers and retrieving experimental parameters from these. This reference implementation can be used as a template for your own code or directly in your own software. It can be obtained from dectris.com or email to support@dectris.com

4 Pilatus CBF header in a minimal CBF header

The header of a minimal CBF file follows the cif syntax described in the cif specification.

In the following, the terms *data name*, *data value*, *non-simple data value*, *data item*, and *tag* are used as described in “Definition of terms” of the cif syntax documentation.

The two *data items* with the *data names* ' _array_data.header_convention' and '_array_data.header_contents' appear in a minimal CBF header containing a Pilatus CBF header.

The '_array_data.header_convention' *data item* contains information on type and version of the Pilatus CBF header convention used, according to its description in the imgCIF dictionary:
“This item is an identifier for the convention followed in constructing the contents of _array_data.header_contents.
The permitted values are of the of an **image creator identifier followed by an underscore and a version string**. To avoid confusion about conventions, **all creator identifiers should be registered with the IUCr** and the conventions for all identifiers and versions should be posted on the MEDSBIO.org web site.”

“Image creator identifiers” (see above) used in the current and previous versions of camserver are “PILATUS” and “SLS”.

Example:

```
_array_data.header_convention "PILATUS_1.2"
```

Note 1:

The cif syntax allows to omit the double quotes or to substitute these with single quotes.
Note 2:
The following regular expression can be used to search for a '_array_data.header_convention' *data item* with a *data value* identifying a Pilatus CBF header:

```
_array_data.header_convention +["']?(SLS|PILATUS)_\d+(\.?\d*)*["']?
```

See also the Python reference implementation for an example.

The '_array_data.header_contents' *data item* as defined by imgCIF contains the Pilatus CBF header in a semi-colon delimited text field (as a *non-simple data value*):

“This item is an **text field for use in minimal CBF files** to carry essential header information to be kept with image data in _array_data.data when the tags that normally carry the structured metadata for the image have not been populated.
Normally this data item should not appear when the full set of tags have been populated and _diffrn_data_frame.details appears.”

Example:

```
_array_data.header_contents
;
# Detector: PILATUS 300K, 3-0101
# 2011-07-22T17:33:22.529
# Pixel_size 172e-6 m x 172e-6 m
# Silicon sensor, thickness 0.000320 m
# Exposure_time 0.0970000 s
# Exposure_period 0.1000000 s
# Tau = 383.8e-09 s
# Count_cutoff 126367 counts
# Threshold_setting: 4024 eV
# Gain_setting: high gain (vrf = -0.150)
# N_excluded_pixels = 19
# Excluded_pixels: badpix_mask.tif
# Flat_field: (nil)
# Trim_file: p300k0101_E8048_T4024_vrf_m0p15.bin
# Image_path: /ramdisk/
;
```

5 Properties of a Pilatus CBF header

- Pilatus header information is given in a text field tagged with the '_array_data.header_contents' *data name*.
- The text field is delimited by semi-colons according to cif syntax.
- Each line in the Pilatus header starts with a hash character ('#').
- Each line in the Pilatus header may contain one header keyword with its associated value. The keyword is separated by one or more spaces from the hash character at the start of the line.
- The keyword is followed by one or more values, separated by one or more spaces or characters that are in a Pilatus header equivalent to spaces and may be added for better human readability of the header.
- The space equivalent characters of a Pilatus header are '#=:,()'
- Data values and potentially their units appear at specified positions with respect to the space delimitation described above.
- The character positions of keywords and values within a line of the header are not part of the specification and might change in future versions of the Pilatus CBF header.

Example:

The header line
```
# Beam_xy (243.12, 309.12) pixels
```
is equivalent to all following lines:
```
# Beam_xy  243.12  309.12  pixels
# Beam_xy 243.12 309.12 pixels
#   Beam_xy   (243.12,     309.12)   pixels
# Beam_xy (243.12 309.12) pixels
# Beam_xy: ((243.12, 309.12))   pixels
# Beam_xy = 243.12, 309.12 pixels
```

- Non-optional keywords always appear in a Pilatus CBF header unless they were introduced with a specific camserver release. Obviously, non-optional keywords will not appear in headers written by previous releases of camserver.
- Non-optional keywords report information on detector and camserver settings used for image acquisition.
- Optional keywords report experimental information as provided by the user or beamline controls to camserver. They only appear when the corresponding information was provided to camserver.
- For all optional keywords the **correctness of the reported header information depends entirely on correct input to camserver/tvx** by the user or beamline controls, in particular with respect to dimensions! No sanity checks are performed on the input.

- The Pilatus CBF header reports the value 'NaN' in cases a numeric value was expected but a non-numeric value was input to camserver.
- Date and time of image acquisition given in the header are not indicated by a keyword but can easily be recognized by the formatting of the header line (e.g. with the regular expression: `\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}.\d+`)
- The order of the keywords in the header and the line number on which a keyword appears are not part of the specification and may change.
- New keywords may be introduced in the future.

Keyword	Description	Value position(s)	Unit positions(s)	Dimen-sion	Type
`Detector`	Information on detector type, model, etc.	1 to -1 (detector type: 1; model: 2)	n/a		string
`Pixel_size`	Pixel size in meters.	1, 4	2, 5	m	float (scientific)
`Silicon sensor, thickness`		3	4	m	float
`Exposure_time`		1	2	s	float
`Exposure_period`		1	2	s	float
`Tau`	Pixel dead time after photon absorption event	1	2	s	float (scientific)
`Count_cutoff`	Maximum number of counts possible due to count rate limitation	1	2		integer
`Threshold_setting`		1	2	eV	integer
`Gain_setting`		1 and 2	n/a		string
`N_excluded_pixels`	Number of pixels flagged as bad	1	n/a		integer
`Excluded_pixels`	Filename of bad pixel mask	1	n/a		string
`Flat_field`	Filename of flatfield image	1	n/a		string
`Trim_file`	Filename of the calibration file used to trim the	1	n/a		string

	detector				
`Image_path`	Path of the directory the image file was written to by camserver	1	n/a		string

Table 1: Non-optional keywords of a Pilatus CBF header reporting information on detector and camserver settings used for image acquisition. “-1” is the index for the last item in a line.

Keyword	Description	Value position(s)	Unit positions(s)	Dimen-sion	Type
`Wavelength`		1	2	Å	float
`Energy_range`		1, 2	3	eV	integer
`Detector_distance`		1	2	m	float
`Detector_Voffset`		1	2	m	float
`Beam_xy`		1, 2	3	pixels	float
`Flux`	The parameter value should report photon flux in ph/s. However, an arbitrary string may be provided to camserver as input and will subsequently be reported in the header.	1 to -2 or -1 (depends on user input)	-1 (optional, depends on user input)	ph/s (optional, depends on user input)	string
`Filter_transmission`		1	n/a		float
`Start_angle`		1	2	degree	float
`Angle_increment`		1	2	degree	float
`Detector_2theta`		1	2	degree	float
`Polarization`		1	n/a		float
`Alpha`		1	2	degree	float
`Kappa`		1	2	degree	float
`Phi`		1	2	degree	float
`Phi_increment`		1	2	degree	float
`Chi`		1	2	degree	float
`Chi_increment`		1	2	degree	float

Omega		1	2	degree	float
Omega_increment		1	2	degree	float
Oscillation_axis		1 to -1	n/a		string
N_oscillations	number of oscillations	1	n/a		integer
Start_position		1	2	mm	float
Position_increment		1	2	mm	float
Shutter_time		1	2	s	float

Table 2: Optional keywords of a Pilatus CBF header reporting experimental settings as provided by the user or beamline controls to camserver. "-1" is the index for the last item in a line.

6 Recommendations for reporting experimental parameters

6.1 Multi-axis goniostats

Camserver/tvx release 7.3.13-121212 or newer allow to fully describe the settings of kappa or Eulerian-cradle goniostats. To facilitate consistent header interpretation by data reduction software such as autoPROC[1], the following best practices are recommended:
Instruments with a permanently or optionally installed multi-axis goniostat should report the angles of all axes that allow orienting the sample in the image header. For all axes that can be rotated during diffraction data acquisition their corresponding increment for a particular scan should be reported.

Example set-up:

- Eulerian cradle goniostiat
- omega, chi, and phi can be used to orient the sample
- omega or phi can be rotated during diffraction data acquisition

Example 1; omega scan with all goniostat angles at datum position:

```
...
# Start_angle 60.4500 deg.
# Angle_increment 0.0500 deg.
...
# Phi 0.0000 deg.
# Phi_increment 0.0000 deg.
# Omega 60.4500 deg.
# Omega_increment 0.0500 deg.
# Chi 0.0000 deg.
# Oscillation_axis OMEGA
...
```

[1] Vonrhein, C., *et al.* (2011). *Acta Cryst.* D**67**, 293-302

Example 2; omega scan with goniostat angles Chi and Phi at non-zero values

```
...
# Start_angle 60.4500 deg.
# Angle_increment 0.0500 deg.
...
# Phi 8.2300 deg.
# Phi_increment 0.0000 deg.
# Omega 60.4500 deg.
# Omega_increment 0.0500 deg.
# Chi 20.0000 deg.
# Oscillation_axis OMEGA
...
```

Example 3; phi scan with goniostat angles Omega and Chi at non-zero values

```
...
# Start_angle 30.0000 deg.
# Angle_increment 0.0500 deg.
...
# Phi 30.0000 deg.
# Phi_increment 0.0500 deg.
# Omega 60.4500 deg.
# Omega_increment 0.0000 deg.
# Chi 20.0000 deg.
# Oscillation_axis PHI
...
```

7 Appendix A

Code of pilatus_header.py, the Python reference implementation for parsing of Pilatus CBF headers.

```
#
# ============================================================================
# == pilatus_header.py
# == Parsing detector and experiment parameters from Pilatus file headers
# == 19.09.2011
# == Marcus Mueller (marcus.mueller@dectris.com)
# == Dectris Ltd.
#
# ============================================================================

import re
import sys

SPACE_EQUIVALENT_CHARACTERS = '#:=,()'

NON_OPTIONAL_KEYWORDS = {
    #key: (pattern, [value_indeces], type)
    'Detector_identifier': ('Detector ', [slice(1, None)], str),
    'Pixel_size': ('Pixel_size', [1, 4], float),
    'Silicon': ('Silicon', [3], float),
    'Exposure_time': ('Exposure_time', [1], float),
    'Exposure_period': ('Exposure_period', [1], float),
    'Tau': ('Tau', [1], float),
    'Count_cutoff': ('Count_cutoff', [1], int),
    'Threshold_setting': ('Threshold_setting', [1], float),
    'Gain_setting': ('Gain_setting', [1, 2], str),
    'N_excluded_pixels': ('N_excluded_pixels', [1], int),
    'Excluded_pixels': ('Excluded_pixels', [1], str),
    'Flat_field': ('Flat_field', [1], str),
    'Trim_file': ('Trim_file', [1], str),
    'Image_path': ('Image_path', [1], str),
    }

OPTIONAL_KEYWORDS = {
    'Wavelength': ('Wavelength', [1], float),
    'Energy_range': ('Energy_range', [1, 2], float),
    'Detector_distance': ('Detector_distance', [1], float),
    'Detector_Voffset': ('Detector_Voffset', [1], float),
    'Beam_xy': ('Beam_xy', [1, 2], float),
    'Beam_x': ('Beam_xy', [1], float),
    'Beam_y': ('Beam_xy', [2], float),
    'Flux': ('Flux', [1], str),
    'Filter_transmission': ('Filter_transmission', [1], float),
    'Start_angle': ('Start_angle', [1], float),
    'Angle_increment': ('Angle_increment', [1], float),
    'Detector_2theta': ('Detector_2theta', [1], float),
    'Polarization': ('Polarization', [1], float),
    'Alpha': ('Alpha', [1], float),
    'Kappa': ('Kappa', [1], float),
    'Phi': ('Phi', [1], float),
    'Phi_increment': ('Phi_increment', [1], float),
    'Chi': ('Chi', [1], float),
```

```
    'Chi_increment': ('Chi_increment', [1], float),
    'Oscillation_axis': ('Oscillation_axis', [slice(1, None)], str),
    'N_oscillations': ('N_oscillations', [1], int),
    'Start_position': ('Start_position', [1], float),
    'Position_increment': ('Position_increment', [1], float),
    'Shutter_time': ('Shutter_time', [1], float),
    }

ALL_KEYWORDS = {}
ALL_KEYWORDS.update(NON_OPTIONAL_KEYWORDS)
ALL_KEYWORDS.update(OPTIONAL_KEYWORDS)

class PilatusHeader(object):
    """
    Class for parsing contents of a Pilatus cbf header from a minimal
    or full cbf file.

    Parsing the Pilatus cbf header populates the header_dict dictionary,
    using Pilatus cbf header keywords as keys.
    """
    def __init__(self, filepath, non_binary_length=4096):
        self.filepath = filepath
        self.non_binary_length = non_binary_length
        self.header_dict = {}
        self.header_lines = []

        self.read_header_lines()
        self.parse_header()

    def _rawheader(self):
        return open(self.filepath, 'rb').read(self.non_binary_length)

    def has_pilatus_cbf_convention(self):
        # Check for the _array_data.header_convention data item
        pilatus_header_pattern = re.compile(
            r'''_array_data.header_convention +["']?(SLS|PILATUS)'''
            r'''_\d+(\.?\d*)*["']?''')
        return bool(pilatus_header_pattern.search(self._rawheader()))

    def read_header_lines(self):
        """
        Populate the self.header_lines list.
        """
        contents_pattern = re.compile(
            r'''_array_data.header_contents\s+;.*?;''',
            re.DOTALL)
        contents_match = contents_pattern.search(self._rawheader())
        assert contents_match is not None
        self.header_lines = contents_match.group().splitlines()

    def _spaced_header_lines(self):
        """
        Return header_lines with all space equivalent charecters converted
        to space.
        """
        spaced_header_lines = []
        for line in self.header_lines:
            for space_equivalent in SPACE_EQUIVALENT_CHARACTERS:
                line = line.replace(space_equivalent, ' ')
```

```
            spaced_header_lines.append(line)
        return spaced_header_lines

    def parse_header(self):
        """
        Populate self.header_dict with contents of Pilatus cbf header
        """
        assert self.has_pilatus_cbf_convention()
        if len(self.header_lines) == 0:
            self.read_header_lines()
        # parse the header lines
        for key, (pattern, valueindices, datatype) in ALL_KEYWORDS.items():
            for line in self._spaced_header_lines():
                if pattern in line:
                    values = []
                    for idx in valueindices:
                        try:
                            # handle multiple or single values
                            if isinstance(idx, slice):
                                values += line.split()[idx]
                            else:
                                values.append(line.split()[idx])
                        except IndexError:
                            print ('No value at index %d on header line:
\n'
                                   '%s' % (idx, line))
                    value = self._datatype_handling(values, key, datatype)
                    if value is not None:
                        self.header_dict[key] = value

    def _datatype_handling(self, values, key, datatype):
        # handle rare cases of value "not set"
        if datatype is float and values[0] == 'not':
            # NON_OPTIONAL_KEYWORDS should always have value, at least NaN
            if key in NON_OPTIONAL_KEYWORDS:
                return float('NaN')
            else:
                return None
        # do the conversion for standard cases
        if len(values) == 1:
            values = datatype(values[0])
        else:
            if datatype is str:
                values = ' '.join(values)
            else:
                values = tuple([datatype(v) for v in values])
        return values

    def get_beam_xy_mm(self, factor=1000):
        return tuple([n * size * factor for n, size in zip(
            self.header_dict['Beam_xy'], self.header_dict['Pixel_size'])])

    def get_date_time(self):
        """
        Return date and time of image acquistion.

        Works for format of current camserver versions
        2011-06-04T04:57:02.976
        or format of old camserver versions
```

```
        2011/Sep/12 09:21:27.252
        """
        date_time_pattern = re.compile(
            r'(\d{4}-\d{2}-\d{2}T|\d{4}/\D+/\d{2} )\d{2}:\d{2}:\d{2}.\d+')
        return date_time_pattern.search(self._rawheader()).group()

    def get_time(self):
        time_pattern = re.compile(r'\d{2}:\d{2}:\d{2}.\d+')
        return time_pattern.search(self.get_date_time()).group()

    date_time = property(get_date_time)
    time = property(get_time)

if __name__ == '__main__':
    if len(sys.argv) == 3:
        header = PilatusHeader(sys.argv[1], int(sys.argv[2]))
    elif len(sys.argv) == 2:
        header = PilatusHeader(sys.argv[1])
    else:
        print "Usage: %s cbf_file_path [bytes_to_read]" % sys.argv[0]
        sys.exit()
    #header.parse_header()
    print header.get_date_time()
    for k, v in header.header_dict.items():
        print k, v
```