ADVACAM
Imaging the Unseen

File types

Online version: https://wiki.advacam.cz/wiki/File_types

Contents

Summary 3
File type and extensions constants 3
File saving flags summary 4
File extensions and flags: TXT/PBF/PMF details 5
The files formats 5
Multi-files names generation 6
Files with flags=0 6
Flags influence to files 7
Timepix3 specific data files 8
T3PA files details 9
T3P files details 11
T3R files 12
DSC/INFO metadata files 12
DSC files details 13
INFO files details 16
IDX files details 18
HDF5 files 19
Saving a HDF5 files 19
Pixet structures in HDF5 21
TIFF images 21
Pixel matrix configuration files 22
Obsolete files 22
CLOG and PLOG files 22
CLOG and CLOG.IDX files details 22
PLOG files details 24
Advapix specific data files 24
BMF details 24
AMF details 25
Other files 27
Related 27

Summary

	General image/data files
txt	ASCII frame. Text files with img lines converted to text lines with numbers separated by spaces.
pbf	Pixet binary frame. Simple binary files, numbers only.
pmf	Pixet multi frame. Default is same as the txt, but multiple frames on top of each other. Can use BINARY flag.
t3pa	Tpx3 pixels ASCII. Text format, tab-separated columns with the header in the first row. Biggest to saving.
t3p	Tpx3 pixels. Binary format. Lower saved size than T3PA, contains simple repeats of 1 structure.
t3r	Tpx3 raw. Complete data stream of Tpx3 chips. Lahge to saving, difficult to understand, slow to processing and can cause processing errors.
png	Lossless compressed image. Easy to view, but not good for data processing.
tiff	TIFF, TIF, high bit-depth file usable in common graphic softwares or data processing.
h5	HDF5, hierarchical data format 5. Used as one of multi-frame formats.
	Files auxilliary for image/data
dsc	Text metadata list saved beside a standard multiframe files (PMF for example).
info	Text metadata list saved beside other than standard multiframe files (T3PA for example).
idx	Binary index for multi-frame files. Two formats existing. Usesfull for fast access to n-th frame of large text files, necessary for frame seeking in binary sparse files.
	Special data files
clog, plog	Clusters/pixels logs. Text files contains clusters separated to frames with pixels lists. Historic formats for saving a data with few hited pixels in a frames. (obsolette)
bstg	Binary settings file. Measured or processed data with all configuration. See Binary Spectral Imaging API: BSTG files or see the "Spectraimg and data files" chapter in the Python API manual.
vtxt	ASCII vertical CSV-like file used in PIXet Basic and Clustering plugin for saving histograms
	Configuration files
xml	See #Configuration XML files 1. Device settings. Device configuration and calibration files. Name like as MiniPIX-A06-W0050.xml. 2. Pixet Pro devcontrol settings Name like as devcontrol_MiniPIX-A06-W0050.xml. 3. Pixet Basic devcontrol settings Name like as eduview_MiniPIX-A06-W0050.xml. 4. User configuration files from ISettings object
ini	See pixet.ini file 1. pixet.ini Main configuration file of the Pixet core 2. hwlibname.ini Configuration of single hwlib. Name like as minipix.ini, zem.ini, zest.ini... 3. pyscripting.ini Configuration of the Python scripting plugin in the Pixet program.
txt	1. (ASCII frames) 2. Calibation files Set of 4 txt files with ASCII frames containing abct constants for each pixel 3. Pixel matrix configuration files ASCII frame containing complete pixel matrix configuration, mask bits, test bits, or THL adjustments.

File type and extensions constants

There are constants for file types and extensions. It can be used with Python API for filenames testing or with acquisition functions. But mostly PX_FTYPE_AUTODETECT will be enough.

Python example:

```
# measure and save one 0.25 second frame to png file named "testFile.png"
dev.doSimpleAcquisition(1, 0.25, pixet.PX_FTYPE_PNG, "testFile")
dev.doSimpleAcquisition(1, 0.25, pixet.PX_FTYPE_AUTODETECT, "testFile.png")
```

File types and extensions constants table

File type constants	File extensions constants	Ext value
PX_FTYPE_NONE	(No direct file saving – data stored only in memory)	
PX_FTYPE_AUTODETECT	(FTYPE detected by extension in a filename)	
PX_FTYPE_ASCII_FRAME	PX_EXT_ASCII_FRAME	"txt"
PX_FTYPE_BINARY_FRAME	PX_EXT_BINARY_FRAME	"pbf"
PX_FTYPE_MULTI_FRAME	PX_EXT_MULTI_FRAME	"pmf"
PX_FTYPE_BINARY_MULTIFRAME	PX_EXT_BINARY_MULTI_FRAME	"bmf"
PX_FTYPE_TPX3_PIXELS	PX_EXT_TPX3_PIXELS	"t3p"
PX_FTYPE_TPX3_PIXELS_ASCII	PX_EXT_TPX3_PIXELS_ASCII	"t3pa"
PX_FTYPE_CLUSTER_LOG	PX_EXT_CLUSTER_LOG	"clog"
PX_FTYPE_PIXEL_LOG	PX_EXT_PIXEL_LOG	"plog"
PX_FTYPE_PNG	PX_EXT_PNG	"png"
PX_FTYPE_TPX3_RAW_DATA	PX_EXT_TPX3_RAW_DATA	"t3r"
PX_FTYPE_PIXET_RAW_DATA	PX_EXT_PIXET_RAW_DATA	"prd"
PX_FTYPE_EXTERNAL	(reserved)	
(description file saved automatically with pmf/txt)	PX_EXT_FRAME_DESC	"dsc"
(index file saved automatically with pmf/txt)	PX_EXT_INDEX	"idx"

File saving flags summary

File saving flags can do additional settings for file(s) saving.

- Can be used in saving files or in doAdvancedAcquisition python methods, for example.
- Flags can be combined.
- Default frame file settings is set of separate subframes text files, with all pixels include zeros, each subframe with idx+dsc files:

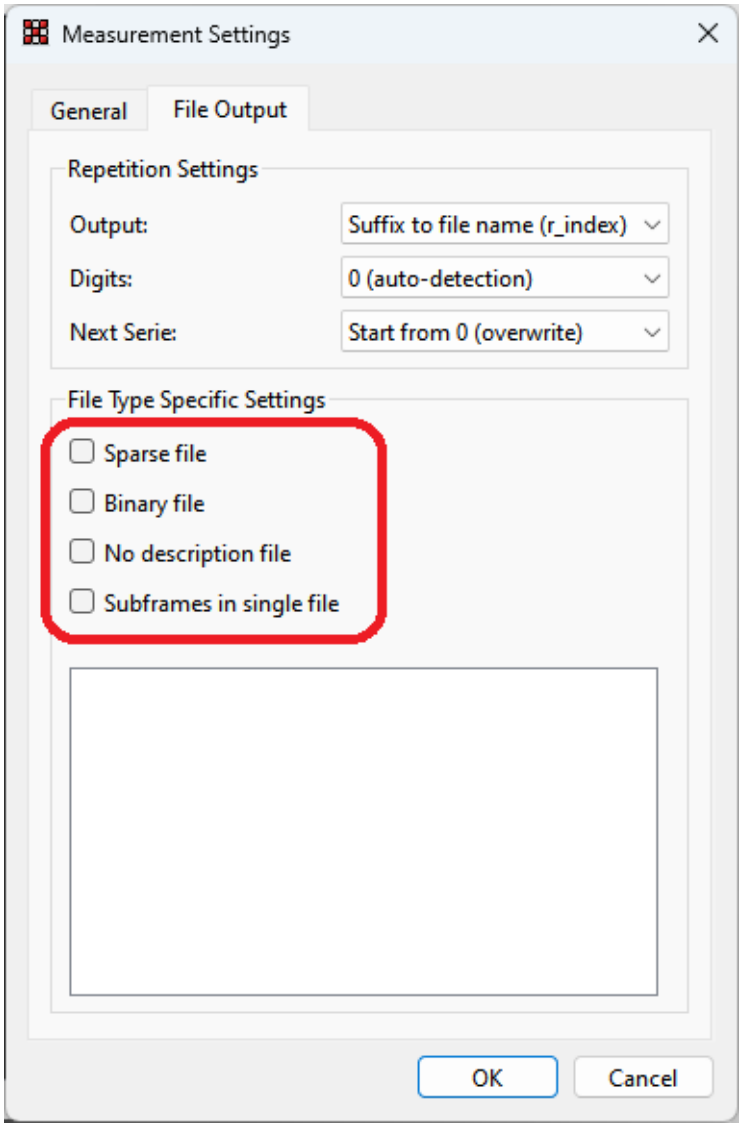
file_ToT.pmf, file_ToT.pmf.dsc, file_ToT.pmf.idx, file_ToA.pmf, file_ToA.pmf.dsc, file_ToA.pmf.idx

File saving flags

Flag constant base name	Description
PX_FRAMESAVE_BINARY	Use binary format in pmf.
PX_FRAMESAVE_SPARSEX	Index + non-zero pixels in file. # separates (sub)frames.
PX_FRAMESAVE_SPARSEXY	X, Y + non-zero pixel in file. # separates (sub)frames.
PX_FRAMESAVE_NODSC	Do not add dsc file.
PX_FRAMESAVE_NOSUBFRAMES	Do not use subframes, save main frame only.
PX_FRAMESAVE_SUBFRAMES_ONEFILE	Save all subframes to a single file.
PX_FRAMESAVE_SUBFRAMES_SAVEMAINFRAME	Save separate all subframes and main frame extra.

The file saving flags can be used in

- Python API: Use pixet.PX_FRAMESAVE_... constants in flags parameter of some measuring/saving methods.
- Binary API: Use PX_FRAMESAVE_... constants in flags parameter of some measuring functions.
- The Pixet program. Available in the More measurement settings, after compatible filename was selected



File saving flags in More measurement settings in Pixet

File extensions and flags: TXT/PBF/PMF details

The files formats

txt	Text	Single frame in the text file.
pbf	Pixet Binary Frame	Single frame in the binary file.
pmf	Pixet Multi Frame	Multiframe file with text or binary format, depends on flags used with saving.
idx	Index	Binary array of structs with 64b pointers to start of frames, frame metadata and subframes.

dsc Description	List of all metadata for each frame and subframe. Actual device and acquisition parameters, data types, etc. The "Frame name" item can be helpful to orientation in pmf structure if the ONEFILE flag used. The Type=item is helpful to understanding the structure of data if the BINARY flag used.

Multi-files names generation

Note

All the next examples are for Timepix3, single chip, opm = TPX3_OPM_TOATOT

flags 0 (default), input filename = "name", acqCount = 1

name_ToA.txt, name_ToA.txt.dsc, name_ToT.txt, name_ToT.txt.dsc

acqCount = 6

name_0_ToA.txt, name_0_ToA.txt.dsc, name_0_ToT.txt, ...

...

name_5_ToA.txt, name_5_ToA.txt.dsc, ...

PMF note

With each pmf generating .pmf.idx binary file, other is same as TXT with acqCount = 1.

Files with flags=0

Note

All the next examples are for Timepix3, single chip, opm = TPX3_OPM_TOATOT

TXT file data, default

0 0 0 5 0 0 0 ... 256 numbers (int for non-calibrated values or float if the calibration used) and enter

0 872 0 0 0 ... 256 numbers (int for non-calibrated values or float if the calibration used) and enter

(256 lines)

PBF file data, default

Simple pixels binary data without anything else

Data can be typically 16 or 32 bit raw integers with little-endian order and doubles for calibrated data.
For example, MiniPIX has single chip, this has 65536 pixels, it's binary file has 65536 words (size 128 kB binary), sometimes 65536 doubles (size 512 kB binary).
The data format can be read in the line starting with Type= line in the DSC file saved beside the data file.

PMF file data, default

0.00000 78.65742 0.00000 ... 256 numbers (int for non-calibrated values or float if the calibration used) and enter
0.00000 0.00000 999785.5 ... 256 numbers (int for non-calibrated values or float if the calibration used) and enter
(256 lines * acqCount)

Flags influence to files

TXT file data: FRAMESAVE_SPARSEX flag

_ToA.txt file		_ToT.txt file	
px index	ToA	px index	ToT
0	227212.500000	0	20
17	310685.937500	17	13
255	265487.500000	255	11
1274	105728.125000	1274	9

- Lists of all hited pixels
- ToT: int for non-calibrated data or float if the calibration used

TXT file data: FRAMESAVE_SPARSEXY flag

_ToA.txt file			_ToT.txt file		
X	Y	ToA	X	Y	ToT
247	3	189851.562500	247	3	16
250	4	140042.187500	250	4	12
5	9	317195.312500	5	9	5

- Lists of all hited pixels
- ToT: int for non-calibrated data or float if the calibration used

PMF file data, pixet.PX_FRAMESAVE_SPARSEX(Y) flag

Same as TXT, but containing single lines with only # to separate frames

X	Y	ToA	Line description
232	139	321620.312500	frame 1, px 1
4	252	340231.250000	frame 1, px 2
#			frames separator
39	0	258270.312500	frame 2, px 1
201	0	76593.750000	frame 2, px 2

92 1 268642.187500 frame 2, px 3

PX_FRAMESAVE_SUBFRAMES_ONEFILE

All the data is in one file, subframes are placed one behind the other. If the measurement result has 10 frames with 2 subframes A/B, each _n TXT file contains 2 subrfames and the PMF contains 20 frames in order:
sfr0A, sfr0B, sfr1A, sfr1B, ...
The exact order and names of type of (sub)frames is listed in the DSC file. The DSC have separate records [Fn] for all the items.

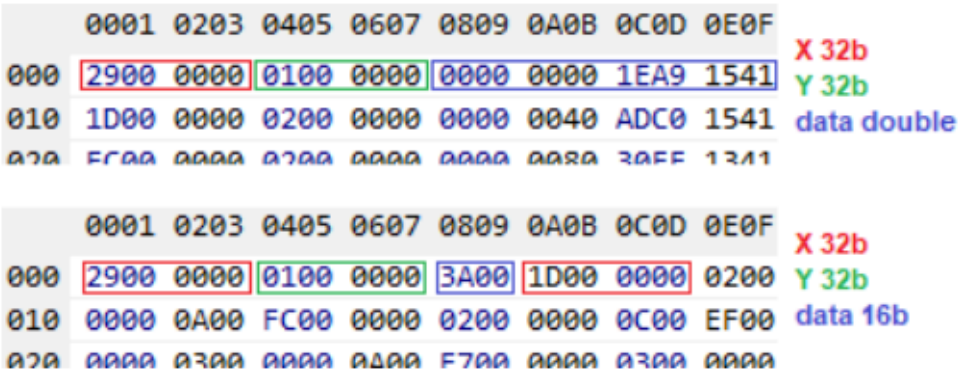
PX_FRAMESAVE_SUBFRAMES_SAVEMAINFRAME

The group of the saved files contains the main frame and all subframes. Subframe files end in _sfrName, the main frame does not. In DSC file accompanying the TXT with main frame is not the "Frame name" item.
Not applicable if combined with the ONEFILE flag.

PX_FRAMESAVE_BINARY

If the file type supports text and binary format, ex. PMF, save the binary.
Not applicable to TXT, must use PBF instead.
Data in the file are the simple array of non-calibrated 16 or 32b integers or calibrated doubles. See the DSC file for used data type.

BINARY + SPARSEXY examples:



Example of data saved if flags BINARY+SPARSEXY used

Timepix3 specific data files

The Timepix3 have the data-driven mode feature. It is "frameless" mode, where the device can continuously send the data of the pixels just hit indefinitely. Each sent pixel contains information:

1. Pixel position index
2. Event registration time (raw ToA count and FToA, conversion and corrections needed)
3. Energy deposited in a pixel (raw ToT count, need conversion using the chip-specific calibration table containing cal. constants for each pixel)

Note: Theese files can be very large. You can collecting a data from cosmic particles using Minipix continuously for more weeks and get a T3PA sized in tens of megabytes. But if some noising pixel occurs, a files can has many gygabytes per day. If the Advapix used with x-ray mashines or accelerators, the output data can has gigabytes in an instant.

The formats:

- T3PA files are text/csv files with basic data. User can simply see it in text editor and process it in Python etc.
- T3P files are binary files with basic data same as T3PA. Faster saving, shorter files.
- T3R files are binary files with complete raw communication data. For special purposes only.

Data order

- The order of the data roughly corresponds to the order of events, so data an hour later will definitely be further in the file.
- But the exact order corresponds to the order in which the data came from the device, so for example a later event that occurred at the edge where the chip is read can be recorded earlier than an earlier event that occurred further from the edge.
- The unevenness of the order occurs from tens of ns on a lightly loaded Advapix to several ms with a heavy load on the Minipix.

How to get the files:

- In the Pixet program set operation mode to ToA+ToT and use the "Pixels" measurement type and turn on file saving.
- In the binary API using programs set operation mode to PXC_TPX3_OPM_TOATOT and use the pxcMeasureTpx3DataDrivenMode function.
- In the Python API using programs set operation mode to pixet.PX_TPX3_OPM_TOATOT and use the dev.doAdvancedAcqquisition with acqType=pixet.PX_ACQTYPE_DATADRIVEN.

T3PA files details

The Timepix3 pixels ASCII file is timepix3 data file in text format with lines and tabs. Can be read as CSV, but its size is not limited to sizes readable by Office-like programs . Contains the header line and data lines with record index, pixel index in the matrix, Time of arrival, Time over threshold, Fine ToA and Overflow.

The T3PA example:

```
Index   Matrix Index    ToA     ToT     FToA    Overflow
0       1028    1918    14      22      0
```

```
1       1028    3126    8       28      0
2       1028    3778    5       23      0
...
156003  39793   98473646054     38      9       0
156004  190     98492090610     19      3       0
```

- The Index is simple index of measurement line. This growing while measurement is running. If you append new measurement to existing file, new index is 0 again and again growing while new measurement is running.

Physical position of the x=0, y=0 pixel on the Minipix (1,1 in Pixet view)

- The Matrix Index is index of the pixel. On the Minipix Tpx3 is 0 at the left-down (see image)
- The ToA is time of arrival in units 25 ns, mod by limit specific by device type.

For example Minipix 2^{64} (14600y), Advapix-single 2^{30} (26s), Advapix-Quad 2^{28} (6.5s).

Note: The ToA on-chip implementation in the pixels is limited to 14 bits (409.6 µs).

The ToA in T3PA is extended by device. But there is inherent uncertainty around the borders. These values may be incorrectly assigned. Users not comfortable with our extension can apply AND with (uint64)16383 to extended ToA to get original ToA from the chip.

- The ToT is time over threshold in units 25 ns.
- The FToA stands for "fine ToA" and it is the finest step of the ToA measurement. To properly account for this step in the conversion of ToA to time, it is necessary to subtract the amount of counts of fToA in the following manner:

Time [ns] = 25*ToA - (25/16)*fToA

The original range of this fToA value in the chip is 4 bits, or 16 values. This is extended in the post-processing of the

data into 5 bits, or 32 values to include a correction for the delay of the clock propagation in the chip. The final value exported into t3pa files has a range of 5 bits, or 32 values, but the previous equation still stands.

- The Overflow is sign of data transfer overflow. If the line has this 1:

  index = 0x74: start of lost data
  index = 0x75: end of lost data, toa is length of the missing time
  (this can occurs with rates over megahits per seconds for Minipix)
  Note: In data from multichip devices, there is not Overflow, replaced by Chip index (But column name is still Overflow).

If saving of the T3PA repeated to the same file, new data will be append with new reset of record index and ToA and the file containing parts is like this:

```
507812     353         39993345         1022         15         0
507813   46177         39999843         159         2         0
507814   45921         39999843         159         2         0
0        421         2         13         29         0
1        297         2         22         27         0
2        297         145         62         17         0
3        297         283         19         13         0
```

T3P files details

Timepix3 Binary Pixels is similar to t3pa file without record index. And the numbers are not saved as ASCII, but binary. The file contains one pixel after each other. Each pixel in this format:

```
u32 matrixIdx;
u64 toa;
byte overflow;
byte ftoa;
u16 tot;
```

T3P file contents example:

As see in a HEX editor

```
5E86 0000 1E0B 0000
0000 0000 0005 0300
6087 0000 1E0B 0000
0000 0000 0005 0400
6387 0000 1F0B 0000
0000 0000 001B 0100
6486 0000 1E0B 0000
0000 0000 0015 0400
5D84 0000 1F0B 0000
0000 0000 0010 0200
89BD 0000 240B 0000
0000 0000 0015 0D00
5F80 0000 1E0B 0000
0000 0000 0002 0600
```

Redistributed according the structure

pxIdx	ToA	Over	fToA	ToT
4 B	8 B	1 B	1 B	2 B
5E 86 00 00	1E 0B 00 00 00 00 00 00	00	05	03 00
60 87 00 00	1E 0B 00 00 00 00 00 00	00	05	04 00
63 87 00 00	1F 0B 00 00 00 00 00 00	00	1B	01 00
64 86 00 00	1E 0B 00 00 00 00 00 00	00	15	04 00
5D 84 00 00	1F 0B 00 00 00 00 00 00	00	10	02 00
89 BD 00 00	24 0B 00 00 00 00 00 00	00	15	0D 00

Corresponding start of T3PA

Index	Matrix Index	ToA	ToT	FToA	Overflow
0	34398	2846	3	5	0
1	34656	2846	4	5	0
2	34659	2847	1	27	0
3	34404	2846	4	21	0

T3P files with trgTimeStamp

Note: This is old internal testing feature and was not intended for mormal using. If you do want to use it, here's some info:

If the trgTimeStamp feature is enabled, file can contains lines of tab-divided ASCII numbers. Every record is six numbers divided by tabs (0x09) and ended with line end (0x0A). It is possible that older firmware versions have a different number and meaning of the numbers.

Every sync pulse cause creating of one line record. In the file, each sync record and each pixel are simply stored in the order as it arrived on the computer. Any combination of order and number of binary and text records can be expected.

This is a source of complications when using the file. The file must be browsed sequentially as binary pixels. At the first occurrence of faulty or suspicious values (eg high pixel index, high ToA, Overflow>1) assume that the current record is not a pixel, but that the ASCII/tab line starts here and that it ends at 0A.

T3R files

The Timepix3 Raw Data File is special format for testing purposes. This is a dump of raw communication from the device. The file format is device specific, binary, complex and files are very large. Use this only if you have no other option.

DSC/INFO metadata files

The metadata text files are saved beside the data files and containing informations about device and settings used for measuring the data. It can be usable while openning the data file in the Pixet program or in other working with the data.

If the API is used to saving the data, programmer can use callback like us "before saving data callback" to add Your specific metadata items or can remove items that will not need.

- DSC are files generated beside the frame data and cotaining information for each frame

- INFO are files generated beside pixel data and some special data formats

DSC files details

The first line is header line:

Some like as A123456789: B=binary / A=ASCII and number = count of frames in multiframe data file

Next are frames in format:

1. [Fn] - Frame with idx n start: [F0], [F1], ...
2. Frame type - Data type, pixel format and frame size: Some like as "Type=i16 [X,C] width=256 height=256"

   Pixels format options:

   matrix - Whole matrix saved. Number of saved pixels are allways width*height.

   Multiframe data file not contains frame separator.

   [X,C] - Hit pixels only. Every saved pixel has matrix index and data value.

   ASCII multiframe data file contains the frame separators.
   The IDX file must be used to find frame begins in binary multiframe file.

   [X,Y,C] - Hit pixels only. Every saved pixel has X,Y position and data value.

   ASCII multiframe data file contains the frame separators.
   The IDX file must be used to find frame begins in binary multiframe file.

3. Frame metadata - List of metadata items separated by blank lines:

   Each metadata item is line triplet:

1. "Item name" ("Item description"): Example: "Acq time" ("Acquisition time [s]"):
2. DataType[valCount] Example: double[1]
3. Values list Example: 0.500000
4. (blank line)

4. (blank line) - end of frame (there are two blank lines, the last metadata item end and the frame end)

In txt.dsc and pbf.dsc, end of the frame is end of the file.
In the pmf.dsc, next frames or subframes metadata follows.

Some example (PBF 1 frame, with BINARY and SPARSEXY – test_49_ToA.pbf.dsc):

```
B000000001                                    B=binary / A=ASCII and number = count of
frames in multiframe file
[F0]                                          Index of frame in the file = 0
Type=double [X,Y,C] width=256 height=256      Data type double, X,Y,C = only hit pixels
saved and has XY pos.
"Acq Serie Index" ("Acquisition serie index"): Some metadata item name and (description)
u32[1]                                        Type of the item data [number of values]
49                                            The value

(more metadata items separated by blank lines …)

"Frame name" ("Frame name"):
char[3]
ToA                                           This is the ToA frame

(more metadata items separated by blank lines …)
(end of the file)
```

Other example (PMF 10 frames, with BINARY+SPARSEX+ONEFILE – test.pmf.dsc):

```
B000000010
[F0]                                          Start of the first subframe
Type=double [X,C] width=256 height=256        Pixel index and double type pixel data (ToA
in ns)
"Acq Serie Index" ("Acquisition serie index"):
u32[1]
0

(more metadata items separated by blank lines …)

"Frame name" ("Frame name"):
char[3]
ToA
(more metadata items separated by blank lines …)


[F1]                                          Start of the second subframe
```

```
Type=i16 [X,C] width=256 height=256            Pixel index and i16 type pixel data (ToT in
ticks 40MHz)
"Acq Serie Index" ("Acquisition serie index"):
u32[1]
0
```

(and the ToT frame metadata, [F2] and ToA subframe, [F3] and ToT sfr, … [Fn] and ToT sfr of (n/2)th frame)

Complete one frame DSC example (PMF 1 frame, BINARY+SPARSEX – test_15_ToA.pbf.dsc):

```
B000000001
[F0]
Type=double [X,C] width=256 height=256
"Acq Serie Index" ("Acquisition serie index"):
u32[1]
15

"Acq Serie Start time" ("Acquisition serie start time"):
double[1]
1639059034.903085

"Acq time" ("Acquisition time [s]"):
double[1]
0.500000

"ChipboardID" ("Chipboard ID"):
char[9]
I08-W0060

"DACs" ("DACs"):
u16[19]
16 8 128 10 120 1301 501 5 16 8 16 8 40 128 128 128 256 128 128

"Frame name" ("Frame name"):
char[3]
ToA

"HV" ("High voltage [V]"):
double[1]
-500

"Interface" ("Readout interface"):
```

```
char[7]
MiniPIX

"Mpx type" ("Medipix type (1-MXR, 2-TPX, 3-MPX3, 4-TPX3, 5-TPX2)"):
i32[1]
4

"Pixet version" ("Pixet version"):
char[5]
1.7.8

"Start time" ("Acquisition start time"):
double[1]
1639059042.934810

"Start time (string)" ("Acquisition start time (string)"):
char[64]
Thu Dec 9 15:10:42.934809 2021

"Threshold" ("Threshold [keV]"):
double[1]
5.026744
```

INFO files details

- The T3PA.INFO containing metadata in format very similar to one frame of DSC file.
- Some other INFO files can containing simpliest formated metadata

The T3PA.INFO example:

```
[FileInfo]
"Acq Serie Index" ("Acquisition serie index"):
u32[1]
0

"Acq Serie Start time" ("Acquisition serie start time"):
double[1]
1704809538.719000

"Acq time" ("Acquisition time [s]"):
double[1]
1.000000
```

```
"ChipboardID" ("Chipboard ID"):
char[9]
D06-W0065

"DACs" ("DACs"):
u16[19]
16 8 128 10 120 1237 437 5 16 8 16 8 40 128 128 128 256 128 128

"HV" ("High voltage [V]"):
double[1]
-450

"Interface" ("Readout interface"):
char[7]
MiniPIX

"Mpx type" ("Medipix type (1-MXR, 2-TPX, 3-MPX3, 4-TPX3, 5-TPX2)"):
i32[1]
4

"Pixet version" ("Pixet version"):
char[5]
1.8.1

"Shutter open time" ("Shutter open timestamp"):
double[1]
1704809538.867000

"Start time" ("Acquisition start time"):
double[1]
1704809538.867000

"Start time (string)" ("Acquisition start time (string)"):
char[64]
Tue Jan  9 15:12:18.867000 2024

"Threshold" ("Threshold [keV]"):
double[1]
5.015797
```

The BMF.INFO example:

```
[File Meta Data]
```

```
Acq Serie Index:0
Acq Serie Start time:1704813831.469
Acq time:0.001
ChipboardID:G03-W0259
DACs:10 100 255 127 127 0 153 6 130 100 80 85 128 128
HV:-450
Interface:AdvaPIX
Mpx type:2
Pixet version:1.8.1
Start time:1704813831.633
Start time (string):Tue Jan  9 16:23:51.633000 2024
Threshold:5.02649397407217
Timepix clock:50
```

IDX files details

The IDX files are generated with multiframe files to help with fast seeking frames in files. Each frame except first has the basic structure in the IDX file:

```
struct IndexItem {
 i64 dscPos;    // frame position in the DSC file
 i64 dataPos;   // frame position in the main data file
 i64 sfPos;     // subframe position if exist subframes file next to the main data file
(usually not and =0)
};
// Note: CLOG.IDX has no this structure, this in only i64 pointers to frames
```

The PMF.IDX files generated beside the PMFs. Contains the simple binary array of structs of 3 little-endian qwords with addresses associated to the start of each frame except first: DSC, frame and subframe.

.pmf.idx with BINARY+ONEFILE, ToA+ToT example
main data contains
- ToA subframes (double*0x10000 = len 0x80000)
- ToT subframes (i16*0x10000 = len 0x20000)

The IDX contains
1. Pointers to frames in DSC file at 0 (not in idx), 0x03B5, 0x075D, 0x0B08, 0x0BE0, ... (points to an empty line before [Fx] line)
2. Pointers to frames in main data file at 0 (not in idx), 0x080000, 0x0A0000, 0x120000, 0x140000, 0x1C0000, ...
3. Pointers to frames in additional subframes file (not exist -> all=0)

	0001	0203	0405	0607
0000	B503	0000	0000	0000
0008	0000	0800	0000	0000
0010	0000	0000	0000	0000
0018	5D07	0000	0000	0000
0020	0000	0A00	0000	0000
0028	0000	0000	0000	0000
0030	080B	0000	0000	0000
0038	0000	1200	0000	0000
0040	0000	0000	0000	0000

.pmf.idx file example

HDF5 files

The HDF5 (.H5) files are general standard binary containers for structured data. If used to save, contains both measured data and metadata.
• To access theese files, use third party tools like as:
• HDFview from HDF Group
• h5py python library from HDF Group
See: Python API: Examples for reading using Python
• HDF5 C++ API from HDF Group
• If saving from API without the Pixet program, the hdf5io.dll plugin must be found and listed in the [plugins] section of the pixet.ini file.
See example right >>>
See Files and directories of the Pixet and SDK: pixet.ini

Pixet.ini example with the hdf5io plugin:

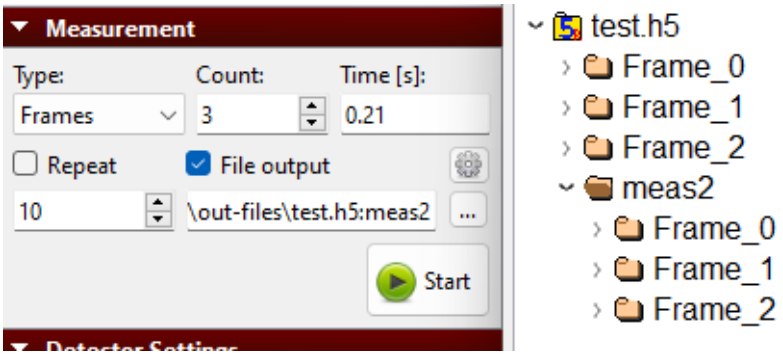
```
[Settings]
UseAppDataDir=false
FactoryDir=C:\Advacam\_factory

[Hwlibs]
hwlibs\minipix.dll
hwlibs\zem.dll
hwlibs\zest.dll

[Plugins]
plugins\hdf5io.dll
```

Saving a HDF5 files

- The files can be saved from the Pixet program or by API.



Saving HDF5 from the Pixet program:
First was saved the test.h5, second the test.h5:meas2

- When saving to an existing file, the data is added to the existing structure in it.
- Use the AUTODETECT filetype in the API functions using filetype.
- Saving flags will be ignored.

Next image showing 3 files in the HDFview program

1. File test1.h5 saved by acquisition of 10 frames with no additional filename settings.
2. File test2.h5 saved by tripple acquisitions of 10 frames, with filename settings "test1.h5:set0", "test1.h5:set1" and "test1.h5:set2".
3. File test1.h5, existing from first acq., saved again in next acquisition of 10 frames with no additional filename settings.

HDF5 files examples in HDFview: Single acq. with 10 frames, triple with structure, first file after second acq.

The files was saved from the PY script:

```
fName = out_dir + "test1.h5"
print("doSimpleAcquisition", fName, "...")
```

```
rc = dev.doSimpleAcquisition(10, 0.1, pixet.PX_FTYPE_AUTODETECT, fName)
if rc==0: print("OK")
else:     print("error:", rc, dev.lastError())
fName = out_dir + "test1.h5"
print("doSimpleAcquisition", fName, "...")
rc = dev.doSimpleAcquisition(10, 0.1, pixet.PX_FTYPE_AUTODETECT, fName)
if rc==0: print("OK")
else:     print("error:", rc, dev.lastError())

fName = out_dir + "test2.h5:set0"
print("doSimpleAcquisition", fName, "...")
rc = dev.doSimpleAcquisition(10, 0.1, pixet.PX_FTYPE_AUTODETECT, fName)
if rc==0: print("OK")
else:     print("error:", rc, dev.lastError())
fName = out_dir + "test2.h5:set1"
print("doSimpleAcquisition", fName, "...")
rc = dev.doSimpleAcquisition(10, 0.1, pixet.PX_FTYPE_AUTODETECT, fName)
if rc==0: print("OK")
else:     print("error:", rc, dev.lastError())
fName = out_dir + "test2.h5:set2"
print("doSimpleAcquisition", fName, "...")
rc = dev.doSimpleAcquisition(10, 0.1, pixet.PX_FTYPE_AUTODETECT, fName)
if rc==0: print("OK")
else:     print("error:", rc, dev.lastError())
```

Pixet structures in HDF5

As see at the "HDF5 files examples" image in the previous chapter, the acquisition creates the file with structure (or adds to existing):

1. Root name or path if defined by adding :hdfpath at end of filename (optional)
2. Frame list: Frame_0, Frame_1, ...
3. Main frame data: The Data item
4. Basic informations items: AcqTime, Width, Height, StartTime
5. MetaData directory containing same data as saved to the dsc files alongise classic simple data files.
6. SubFrames directory with subframes subdirs named by subframe names (ToA, ToT, Event, iToT, ...) containing same structures as the main frame.

TIFF images

The TIFF (.TIF) "Tag Image File Format", files are image format for high-resolution and high-bitDepth data and has multi-page support. If used to save, contains measured data in 32 bit depth format.
• If saving from API without the Pixet program, the tiffio.dll plugin must be found and listed in the [plugins] section of the pixet.ini file.
See example right >>>
See Files and directories of the Pixet and SDK: pixet.ini
• Note: Multipage format is not yet supported in Pixet.
• To simplify further data processing, Pixet saving an integer data only in 32bit format without any conversion. Or if data is double/float, the conversion factor is used (change it in the File output tab on Measurement settings dialog).
• It very often happens that integer data has negligible values against the range of 32 bits (over 4 billion). This can cause the image looks like completely black when viewed in some programs, even though it contains data.

Pixet.ini example with the tiffio plugin:

```
[Settings]
UseAppDataDir=false
FactoryDir=C:\Advacam\_factory

[Hwlibs]
hwlibs\minipix.dll
hwlibs\zem.dll
hwlibs\zest.dll

[Plugins]
plugins\tiffio.dll
```

Pixel matrix configuration files

		Overview
bpc	Binary Pixel Configuration	All PM config in one file, meaning of the bits depends on the chip.
txt	Ascii Mask Matrix	Text file with pixel mask
txt	Ascii Test Bit Matrix	Text file with test bits
txt	Ascii THL adj. bits Matrix	Text file with threshold values adjustment

Obsolete files

All formats in this chapter are obsolete and, with the exception of CLOG, has not used for long time and it is possible that their support will be removed. Pay attention to possible ambiguities when using CLOG.

CLOG and PLOG files

Old text formats from age of the first Timepix chips. Due to the new use with Tpx3, new ambiguities in CLOG have arisen.

- CLOG (clusters log) has remained popular in the context of cluster processing.
- PLOG (pixels log) is currently no longer used.

CLOG and CLOG.IDX files details

The CLOG format was developed to facilitate further processing of cluster data by the user programs. This is a text file divided to the frame records and the records can contain a clusters. Frames and clusters are separated by the line breaks. Frames can be separated by whole free line.

The record format
  Frame FN (frameStart, frameAcqTime s)
  [x, y, energy, ToA] [x, y, energy, ToA] [x, y, energy, ToA] ...

FN	Frame index number. First 0 or 1.
frameStart	Start time of the frame. There are variants: 1. If it from measuring or from replay frame-based data with metadata available: Linux format, frame starting time from PC's getPrecisionTime. 2. If it from pixel-based data with metadata available (file.t3pa + file.t3pa.info): Linux format, acq. starting time from PC's getPrecisionTime with added time from data. 3. If it from replay data and metadata not available: Nanoseconds from the input data. Periodic increments if source is frame-based, random increments if source is data-driven.
frameAcqTime	Duration of the frame, float in seconds. Always 0.000000 in data from data-driven sources.
x, y	Position of the pixel.
energy*	Energy deposited in the pixel. Integer ToT counter value if not calibrated, float in keV if calibrated.
ToA*	Time of arrival, relative to frameStart. Integer in CLK ticks if ToA conversion is disabled, float in ns if ToA conversion is enabled.

*ToA+energy records can be created from source that supports combined ToA+ToT modes, like as OPM_TOATOT on the Timepix3. If the data source supports only single modes, only one value is in this position.

Clog from data-driven source not contains free frames.
Clog from frame-based source can contains free frames.

Example records (Timepix3, Frame2 with two clusters by 2 and 4 pixels, Frame3 with single 2-pixel cluster)

```
Frame 2 (273697060.937500, 0.000000 s)
[214, 195, 43.1598, 0] [220, 191, 20.6515, 7.8125]
[224, 182, 21.8018, 31.25] [223, 186, 4.58576, 31.25] [222, 183, 38.2381, 31.25] [226, 185,
14.7623, 34.375]

Frame 3 (371034565.625000, 0.000000 s)
[151, 33, 32.5745, 0] [151, 34, 13.8135, 17.1875]
```

Example records (Timepix)

```
Frame 6 (1639143482.765164, 0.200000 s)
[87, 134, 5.75352] [217, 58, 14.8396]
Frame 7 (1639143483.019154, 0.200000 s)
Frame 8 (1639143483.261158, 0.200000 s)
Frame 9 (1639143483.513150, 0.200000 s)
```

The CLOG.IDX files generated beside the CLOGs. Contains the simple binary array of little-endian qword addresses of the "F" at each record start.
.clog.idx example
Pointers to records at 0, 0x29, 0x52, 0x7b, 0xA4, 0xCD, ...

Note: The CLOG.IDX is different from ordinary ones IDX files.

	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F
000	0000	0000	0000	0000	2900	0000	0000	0000
010	5200	0000	0000	0000	7B00	0000	0000	0000
020	A400	0000	0000	0000	CD00	0000	0000	0000
030	F600	0000	0000	0000	3401	0000	0000	0000

Example .clog.idx file

PLOG files details

- Like as CLOG, but with simple lists of hit pixels of a frames.
- Metatdata section at start.
- Obsolete format usable only with Timepix (first generation) chips.

Recommended to use PMF with SPARSEX(Y) flag instead it.

Advapix specific data files

New data formats were created for early AdvaPIX variants based on special requirements, but their use was minimal. This chapter serves mainly in case you have such data from the past and need to process it with your software.

BMF details

This special file contains a binary matrix data from fast measurements (AdvaPIX-Tpx and ModuPIX devices).

Note

Obsolete format for obsolete devices

BMF files details

To save theese files the Advapix-Timepix must be used, set the fast mode by setting acq. time 0.01 sec or shorter and frames count divisible by 100.

The file starts with 13 bytes long header and then is followed by pixel values of each frame. Each frame has a few dummy bytes at the beginning. So the layout of the file is:

```
[HEADER][Frame 1][Frame 2][Frame 3] ...
```

where header is 13 bytes:

```
u32 width;
u32 height;
u32 offset;
char frameType;
```

- witdh and height is the dimensions of each frame.
- Each frame data is prepended by offset number of dummy bytes.
- The frameType specifies the type (variable type) of pixel values. It can be one of the following:

```
CHAR = 0 (1 byte size)
BYTE = 1 (1 byte size)
I16 = 2 (2 bytes size)
U16 = 3 (2 bytes size)
I32 = 4 (4 bytes size)
U32 = 5 (4 bytes size)
I64 = 6 (8 bytes size)
U64 = 7 (8 bytes size)
FLOAT = 8 (4 bytes size)
DOUBLE = 9 (8 bytes size)
```

AMF details

Notes

Obsolete, special, rare format.

This is the output of a mechanical assembly of four AdvaPIX-TPX devices, with four USB cables, which was then presented as the AdvaPIX TPX Quad. Not to be confused with the current AdvaPIX-Quad which has a single USB cable. Even if you have this set, don't save AMF unless you have a very special reason.

AMF files details

The amf is a binary file that contains data from all the devices combined into one stream of frame matrixes. The file consists of two parts a header (1000 bytes) and the data.

There are two versions of the file. Version 1 and Version 2. Version 1 has only one offset parameter, but had a bug, where frames were shifted in the file by 8 bytes. Version 2 has to frame data offsets - before frame data and after frame data.

Header (version 1):

```
struct header {
 byte magic[3]; // AMF
 byte ver; // 1
 u32 channelCount;
 u32 offset; // offset of each frame data in the frame block
 u32 chipsWidth; // number of chips in x coordinate
 u32 chipsHeight; // number of chips in y coordinate
 byte chipLayout[256]; // order of chips
 byte chipAngles[256]; // rotation of chips
}
```

Header (version 2):

```
struct header {
 byte magic[3]; // AMF
 byte ver; // 2
 u32 channelCount;
 u32 offsetBefore; // offset of the beginning of frame data in frame block
 u32 offsetAfter; // offset after frame data
 u32 chipsWidth; // number of chips in x coordinate
 u32 chipsHeight; // number of chips in y coordinate
 byte chipLayout[256]; // order of chips
 byte chipAngles[256]; // rotation of chips
}
```

The file may contain variable number of chips (not only data from AdvaPIX Quad = 4 chips).

- channelCount - How many chip are present in the file.
- chipsWidth and chipsHeight - How many chips are in x and y coordinate. For example for AdvaPIX Quad it is 2 by 2 (chipsWith = 2, chipsHeight = 2).
- chipLayout and chipAngles - When the device is read the order of chips is different than shown on the screen (depending on the layout of the internal chip interconnection). Therefore it is necessary to know AdvaPIX QUAD Multi-Frame Format (*.amf) order of the chips and they rotation to create correct image. chipLayout specifies order of the chip (the indexes starts from 0 to the index of last chip, from the top left to the right bottom). The chipAngles specifies rotation of each chip (0 = no rotation, 1 = 90 deg, 2 = 180, 3 = 270, all clockwise).

After the header file the frame data folows. The frame data are saved in frames blocks. Each block contain frames from each detector.

```
[FrameBlock1][FrameBlock2][FrameBlock3]...
```

Frame Block contains:

```
[FrameData1][FrameData2][FrameData3][FrameData4]....
```

Each frame contains:

```
[Offset][MatrixData(65536*2)] // Version 1 of the file
[OffsetBefore][MatrixData(65536*2)][OffsetAfter] // Version 2 of the file
```

Each frame is prepended by an offset (specified in header, offsetBefore) and appended byt some dummy data of length offsetAfter. The frame pixels are saved as 16 bit unsigned integer. Each chip has 256x256 pixels. Therefore - 65536 * 2 bytes.

Bug in Version 1 of the AMF File: The version 1.0 of the AMF file contains bug, where the first frame in the data is missing first 8 bytes. To compensate in the code, when reading make the length of HEADER smaller by 8 bytes => 992 bytes.
Version 2 has size of offset before and after frame data instead.

Version 1.0 Example:

```
#define HEADER_SIZE 1000
frameSizeInBytes = 65536 * 2 + offset;
numberOfFramesInFile = (fileSizeInBytes - HEADER_SIZE) / frameSizeInBytes / channelCount
firstFrameDataPosition = (HEADER_SIZE - 8) + offset
secondFrameDataPosition = (HEADER_SIZE - 8) + offset + frameSizeInBytes * 1
```

Version 2.0 Example:

```
#define HEADER_SIZE 1000
frameSizeInBytes = 65536 * 2 + offsetBefore + offsetAfter;
numberOfFramesInFile = (fileSizeInBytes - HEADER_SIZE) / frameSizeInBytes / channelCount
firstFrameDataPosition = HEADER_SIZE + offsetBefore
secondFrameDataPosition = HEADER_SIZE + offsetBefore + frameSizeInBytes * 1
```

Other files

- See: Files and directories of the Pixet and SDK: pixet.ini
- See: Files and directories of the Pixet and SDK: Configuration XML files
- See: Files and directories of the Pixet and SDK: Device configuration ini files
- See: Files and directories of the Pixet and SDK: Device firmware files
- See: Binary Spectral Imaging API: BSTG files or see the Spectraimg and data files chapter in the Python API manual
- User XML settings: See: The ISetting object chapter in the Python API manual
- ASCII vertical *.vtxt: CSV-like file used in PIXet Basic and Clustering plugin for saving histograms

Related

- Files and directories of the Pixet and SDK
- Pixet SDK overview
- The PIXet program