# Streaming format ThermoVision A320G

## Revision history

Rev A 2007-09-05 Original version.

## General

This document describes the format of a streamed image in the ThermoVision A320G camera. A streamed image is an entity of image data transported from a camera device to a host application without any intermediate storage of the image on a non-volatile media (i.e. hard disk). The image consists of:

- pixel data
- ancillary data

Ancillary data consists of:

- FPGA header data (32 bytes)
- Radiometric data formatted according to the FFF Tagged File Format. Size information is embedded in the FFF header.

Ancillary data is only transmitted when radiometric image format is activated. When linear temperature image format is selected, no ancillary data is transmitted.

## Streamed image format

A streaming session is established using GigEVision control and streaming protocols. Ancillary image data is added on as additional image lines.

The ancillary data consisting of FPGA header data and FFF header, tags and tag data blocks immediately follow the pixel data. A tag describing the pixel data should be present to describe the preceeding pixel data block. The data pointer of the pixel data tag must consequently be ignored since the pixel data always preceeds the FFF header.

## *FPGA Header data*

The FPGA header data consists of 32 bytes.

| Position | Size (in bytes) | Description | Value |
|---|---|---|---|
| 0 | 1 | Header major revision | 1 |
| 1 | 1 | Header minor revision | 0 |
| 2 | 1 | Digital port 1 trig type | Bit mask (bit pos. 0-7)<br>Bit 0=General Mark (0x01)<br>Bit 1=Start Mark (0x02)<br>Bit 2=Stop Mark (0x04)<br>Bit 3=Enable flow Mark (0x08)<br>Bit 4=Disable flow Mark (0x10)<br>Bit 5-7=Reserved (not set) |
| 3 | 1 | Digital port 2 trig type | Bit mask (bit pos. 0-7)<br>Bit 0=General Mark (0x01)<br>Bit 1=Start Mark (0x02)<br>Bit 2=Stop Mark (0x04)<br>Bit 3=Enable flow Mark (0x08)<br>Bit 4=Disable flow Mark (0x10)<br>Bit 5-7=Reserved (not set) |
| 4 | 1 | Dig. port 1 frame time stamp | 0 to 15 |
| 5 | 1 | Dig. port 1 line time stamp MSB | |
| 6 | 1 | Dig. port 1 line time stamp LSB | |
| 7 | 1 | Dig. port 2 frame time stamp | 0 to 15 |
| 8 | 1 | Dig. port 2 line time stamp MSB | |
| 9 | 1 | Dig. port 2 line time stamp LSB | |
| 10 | 1 | Digital port 1 trig state | 0=Deasserted<br>1=Asserted |
| 11 | 1 | Digital port 2 trig state | 0=Deasserted<br>1=Asserted |
| 12 | 1 | Frame count MSB | |
| 13 | 1 | Frame count LSB | |
| 14-31 | 18 | Reserved | Reserved. Set to 0 |

The digital port trig type is a bit mask. Bit 0 is the LSB bit.

The IR image flow has a fixed rate of 60 Hz. The frame rate of the transmitted images can be either 60 Hz, 30 Hz, 15 Hz, 7.5 Hz or 3.75 Hz. In order to properly determine the exact time when the trig event occurred in the IR image flow, we need to have time stamps to be able to calculate the trig event time stamp with respect to the time stamp of the image actually transmitted. For this purpose we have the frame time stamp and line time stamp values.

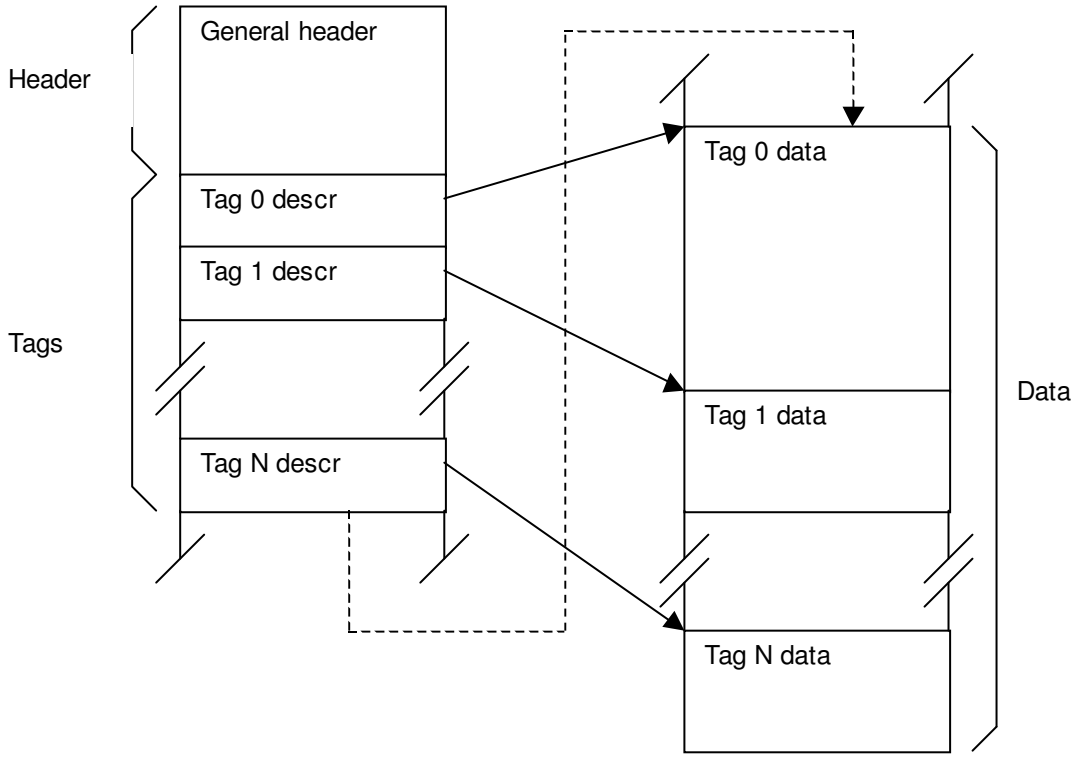
The digital port frame time stamp denotes the zero based frame number where the trig event occurred. If more than one trig event occurs during the sample time, the frame time stamp corresponds to the first trig event.

The digital port line time stamp denotes the zero based line number in the frame where the trig event occurred. If more than one trig event occurs during the sample time, the line time stamp corresponds to the first trig event.

The digital port states indicates the state of the digital ports just before the image is delivered to the GigE interface board. For example if the digital port is configured to be active low, then if the port is grounded the state should be set to Asserted.

## *FFF Tagged File Format*

The “FFF” format has all its tag location information in the file header.




The general header contains a unique identifier, “Magic number”, and some more information including offset to tags and number of tags in tag section

The tag section is variable in size and contains tag descriptions; type of data, version of data tag, file offset and size, optional checksum for tag data. It also contain an index ID that is used for collection of different tag sections to each other.

The data section contains the actual data blocks referred to by tag descriptors. The tag data block start is aligned to a longword boundary. Data size doesn’t have to be size aligned (data size of 1 byte is allowed, data block (N+1)will then start at offset(N)+4).

There is a header file “fff.h” included (“C syntax”) that describes the format in detail.

Tag section is normally initiated to 14 tag descriptors (size of header + tags is then 512 bytes). When extending file to more tags, new tag descriptors are normally added in chunks of 16 (adds 512 bytes). But the file format itself does not require this tag descriptor extension scheme.

The general header contains a checksum for data+all tags descriptors. (not data blocks)
Data blocks that want to use checksum for data should enter a data tag checksum in the corresponding tag.

Header and index tag endian order can be in both Little and Big Endian order. The header record contains a flag which indicates endian type. Data tag endian order might be tag type dependent. You should assume the same endian order in the data tags as in the header tag unless the tag type indicates a specific endian order.

## *fff.h*

```
/**********************************************************************
 *
 * Description of file:
 *   Definition of FFF fileformat
 *
 *
 *    Copyright:   FLIR Systems AB
 **********************************************************************/

#ifndef FFF_H
#define FFF_H

/*====================================================================*/

#define FLIR_FILE_FORMAT           0x46464600              /* FFF\0 */
#define FILE_FORMAT_VERSION        100                     /* 1.00  */

#define FFF_EXT                   "FFF"           /* Default file name
                                                     extention */
/* main types */

typedef enum {
    /* General tags */
    FFF_TAGID_FREE = 0,                /* Marks unused tag descriptor */
    FFF_TAGID_Pixels = 1,
    FFF_TAGID_GainMap = 2,
    FFF_TAGID_OffsMap = 3,
    FFF_TAGID_DeadMap = 4,
    FFF_TAGID_GainDeadMap = 5,
    FFF_TAGID_CoarseMap = 6,
    FFF_TAGID_ImageMap = 7,
    FFF_general_high = 0x1f,           /* Reserve space for other general
                                          tags */

    /* FLIR Matrix TAGs */
    FFF_TAGID_BasicData = 0x20,
    FFF_TAGID_Measure,
    FFF_TAGID_ColorPal,

    FFF_TAGID_matrix_high = 0x3f,      /* reserve space for other system
                                          image blocks */

    /* FLIR Boston reserved TAG number series */
    FFF_TAGID_Boston_reserved = 0x40,
    FFF_TAGID_Boston_reserved_high = 0x5f,

    FFF_highnum = 0x100                /* Guarantee 2 bytes enum */
} TAG_MAIN_T;

/* Sub Tags for FFF_TAGID_Pixels */

enum {FFF_Pixels_BE = 1,               /* Big endian pixel data block */
      FFF_Pixels_LE = 2,               /* Little endian pixel data block */
      FFF_Pixels_PNG = 3               /* PNG compressed pixel data block
*/
};

/* When subtype isn't used, fill subtype with FFF_SubID_default */
enum {FFF_SubID_default = 1};

/* When appropriate, add subID enums for other TAG_MAIN_T too */

typedef struct tagFLIRFILEHEAD
{
    char  szFormatID[4];           /* Fileformat ID 'FFF\0'  4   4       */
    char  szOrigin[16];            /* File origin           16  20       */
    unsigned long dwVersion;       /* File format version    4  24       */
    unsigned long dwIndexOff;      /* Pointer to indexes     4  28       */
    unsigned long dwNumUsedIndex;  /* Number of indexes      4  32       */
    unsigned long dwNextID;        /* Next free index ID     2  36       */
    unsigned short usSwapPattern;/* Swap pattern             2  38      */
    unsigned short Spare[7];      /* Spare                  14  52       */
    unsigned long reserved[2];    /* reserved                8  60       */
```

```
    unsigned long dwChecksum;     /* Head & index checksum  4  64 bytes */
} FLIRFILEHEAD;


typedef struct tagFLIRFILEINDEX
{
   TAG_MAIN_T     wMainType;     /* Main type of index     2   2       */
   unsigned short wSubType;      /* Sub type of index      2   4       */
   unsigned long  dwVersion;     /* Version for data       4   8       */
   unsigned long  dwIndexID;     /* Index ID               4  12       */
   unsigned long  dwDataPtr;     /* Pointer to data        4  16       */
   unsigned long  dwDataSize;    /* Size of data           4  20       */
   unsigned long  dwParent;      /* Parentnr               4  24       */
   unsigned long  dwObjectNr;    /* This object nr         4  28       */
   unsigned long  dwChecksum;    /* Data checksum          4  32 bytes */

   /*  dwParent and dwObjectNr may be set to 0 when not used,

       dwChecksum is checksum of tag data block. It may be set to
       0 when not used.
       When used, 2 types of checksums are allowed:
       1) Type 1 for longword aligned data:
          dwChecksum & 0xC0000000 == 0x80000000.
          Checksum is calculated as the sum of all 32-bit longwords in the
          tag data block given by dwDataPtr and dwDataSize modulo 0x20000000

          or'ed with 0x8000 0000 (MSB always set, MSB-1 always 0).

       2) For possible odd number of bytes in tag data, algorithm 2 exists:
          dwChecksum & 0xC0000000 == 0x40000000.
          Checksum is calculated as the sum of all bytes in the
          tag data block given by dwDataPtr and dwDataSize modulo 0x20000000
          or'ed with 0x4000 0000 (MSB always set, MSB-1 always 0).
   */

} FLIRFILEINDEX;

#define FFF_HeadSize() (sizeof(FLIRFILEHEAD) + \
                  sizeof(FLIRFILEINDEX) * FFF_DEFAULT_NUM_IDX)

/*--------------------------------------------------------------------------*/
/* #endif for fff.h include                                                 */
/*--------------------------------------------------------------------------*/
#endif
```