FSF File Parser Utility

High Level Design Document

|  |  |
| --- | --- |
| Project Name | TOF Software |
| Document Version | 0.1 |
| Document Date | 06-01-2019 |
| Document ID | ADI-HLDD-FSF-PARSER |
| Development Group | Consumer Software Engineering |

|  |  |  |
| --- | --- | --- |
| Srinivas/Mark - 09/06 |  |  |
| **Prepared By/ Date** | **Reviewed By/ Date** | **Approved By/ Date** |

# Abstract

Storing and retrieving different video and image streams is required across different ToF software components. The unified parser is to provide a library that can be utilized across all components. The parser design initially focuses on the FSF format**,** but the API shall be easily portable to other formats. This document describes the design details of the FSF parser and API for applications to construct, retrieve video data efficiently.

# Document Modification History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Date of Revision** | **Description of Change** | **Reason for Change** | **Affected Sections** | **Approved By** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Affected Groups

|  |
| --- |
|  |
|  |
|  |

# Table of Contents

[FSF File Parser Utility 1](#_Toc48064111)

[Abstract 1](#_Toc48064112)

[Document Modification History 1](#_Toc48064113)

[Affected Groups 1](#_Toc48064114)

[Table of Contents 2](#_Toc48064115)

[1 Introduction 3](#_Toc48064116)

[1.1 Purpose 3](#_Toc48064117)

[1.2 Scope 3](#_Toc48064118)

[1.3 Definitions, Acronyms, and Abbreviations 3](#_Toc48064119)

[1.4 References 4](#_Toc48064120)

[1.5 Overview 5](#_Toc48064121)

[2 High Level Design Overview 6](#_Toc48064122)

[2.1 FSF Class Diagram 6](#_Toc48064123)

[2.2 FSF Base Class 6](#_Toc48064124)

[2.3 FSF Platform Agnostic Class 7](#_Toc48064125)

[2.4 FSF Definitions Header 8](#_Toc48064126)

[3 High Level Data Architecture 11](#_Toc48064127)

[3.1 Enumerated Types 11](#_Toc48064128)

[3.1.1 FsfMode 11](#_Toc48064129)

[3.1.2 FsfStatus 11](#_Toc48064130)

[3.1.3 StreamType 11](#_Toc48064131)

[3.2 Data Structures 11](#_Toc48064132)

[3.2.1 FileHeader 13](#_Toc48064133)

[3.2.2 StreamInfo 13](#_Toc48064134)

[3.2.3 StreamHeader 13](#_Toc48064135)

[3.2.4 Stream 13](#_Toc48064136)

[4 API Definitions 14](#_Toc48064137)

[4.1 Constructor 14](#_Toc48064138)

[4.2 CreateFile() 14](#_Toc48064139)

[4.3 OpenFile() 14](#_Toc48064140)

[4.4 SaveFile() 14](#_Toc48064141)

[4.5 CloseFile() 15](#_Toc48064142)

[4.6 SetFileHeader() 15](#_Toc48064143)

[4.7 GetFileHeader() 15](#_Toc48064144)

[4.8 SetStreamInfo() 15](#_Toc48064145)

[4.9 GetStreamInfo() 16](#_Toc48064146)

[4.10 SetOptionalFileHeader() 16](#_Toc48064147)

[4.11 GetOptionalFileHeader() 16](#_Toc48064148)

[4.12 SetFileComment() 17](#_Toc48064149)

[4.13 GetFileComment() 17](#_Toc48064150)

[4.14 SetStream() 17](#_Toc48064151)

[4.15 GetStream() 18](#_Toc48064152)

[5 C Wrapper API Definitions 18](#_Toc48064153)

[5.1 CreateFSFFile() 18](#_Toc48064154)

[5.2 ReadFSFFile() 18](#_Toc48064155)

[5.3 ReadFileHeader() 19](#_Toc48064156)

[5.4 ReadStreamInfo() 19](#_Toc48064157)

[5.5 WriteOptionalFileHdr() 19](#_Toc48064158)

[5.6 ReadOptionalFileHdr() 19](#_Toc48064159)

[5.7 WriteFileComments () 19](#_Toc48064160)

[5.8 ReadFileComments() 19](#_Toc48064161)

[5.9 WriteStream() 19](#_Toc48064162)

[5.10 ReadStream() 20](#_Toc48064163)

[5.11 IsFileOpen() 20](#_Toc48064164)

[5.12 Close() 20](#_Toc48064165)

[5.13 GetWindowsErrorCode() 20](#_Toc48064166)

[6 C Wrapper Comparison with Microsoft FSF C APIs 20](#_Toc48064167)

[7 Example 22](#_Toc48064168)

[7.1 FSF Template Writer 22](#_Toc48064169)

[8 Limitations 22](#_Toc48064170)

[9 Traceability to Requirements 22](#_Toc48064171)

[10 Others 22](#_Toc48064172)

# Introduction

FSF is a video file format that is used by Microsoft to store various video data such as RAW, DEPTH, AB and XYZ. It supports the concept of streams for each of the data type. This format allows us to capture all video information in a single file or separate files per stream. This document provides the high-level design of the APIs to facilitate access and creation of the FSF files. The goal of the FSF parser is to have a simple interface with compatibility to different platforms yet heavily optimized.

ADI Frame Grabber

ToF Sensor

FSF Parser Utility

## Purpose

The purpose of this document is to give the design details of the components that are developed for the FSF utility. The intended audience are the software developers, test development engineers, and application engineers.

## Scope

The following software components are produced for windows platform

* FSF library that supports reading and writer functionality

## Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| SDK | Software Development Kit |
| SPI | Serial Peripheral Interface |
| ToF or TOF | Time of Flight |
| MIPI | Mobile Industry Processor Interface |
| GUI | Graphical User Interface |

## References

## Overview

File Header

Optional File Header

File Comment

Stream Info [ 1..m]

Frames [ 1..n ]

.HeaderSize,

MagicNumber,

FileFormatVersion,

.FileHeaderSize,

StreamInfoSize,

StreamHeaderSize,

OptionalFileHdrSize, FileCommentSize, FrameOffsetInfoLoc, nFrames,

nStreams

.SystemID

Stream Type,

ChannelFormat,

.BytesPerPixel,

nRowsPerStream,

nColumnsPerStream,

OptionalStreamHdrSize, StreamCommentSize,

CompressionScheme

Stream Header

OptionalStreamHdr

Stream Comment

StreamData

Streams [ 1..m ]

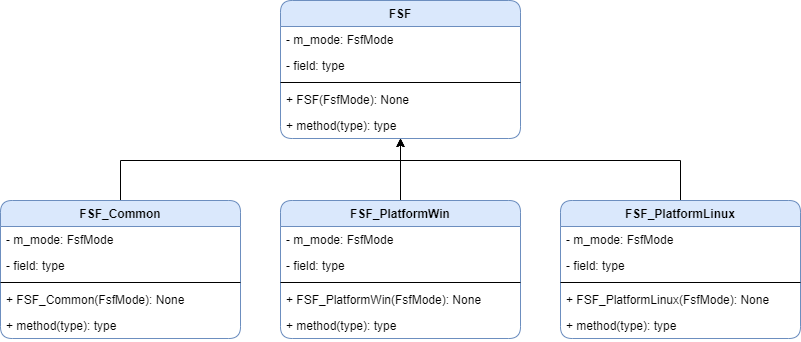
Frames [1..n]

Frame Offset

# High Level Design Overview

## FSF Class Diagram

*FSF* is an abstract base class which can be extended to have platform-optimized implementations (Windows or Linux) or to use the initial platform-agnostic implementation *FSF\_Common*.



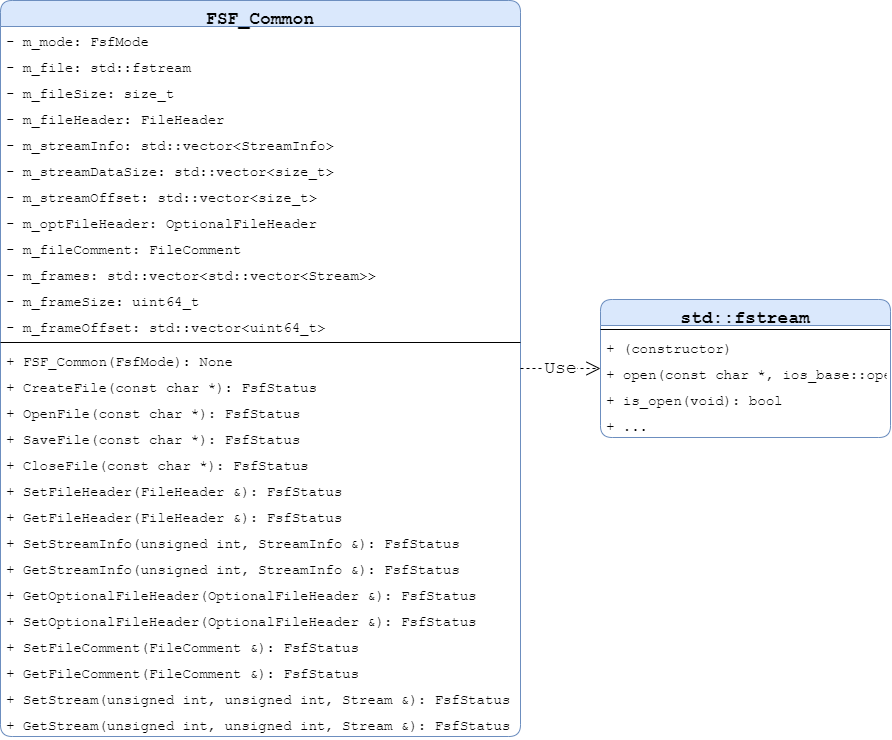
## FSF Base Class

The *FSF* base class contains the following virtual public methods which must implemented in the derived classes.



## FSF Platform Agnostic Class

The initial platform agnostic implementation of the FSF file parser utility derived from the *FSF* base class uses the C++ standard file stream library *fstream*.



## FSF Definitions Header

This section lists down all the macros, enumerated types and structures used throughout the FSF utility.

#ifndef FSF\_DEFINITIONS\_H

#define FSF\_DEFINITIONS\_H

#include <cstddef>

#include <cstdint>

#include <string>

#include <vector>

#define FSF\_MAGIC\_NUMBER 0x00465346 // "FSF"

#define FILE\_FORMAT\_MAJOR\_VERSION 1

#define FILE\_FORMAT\_MINOR\_VERSION 1

#define FILE\_HEADER\_SIZE sizeof(FileHeader)

#define STREAM\_INFO\_SIZE sizeof(StreamInfo)

#define STREAM\_HEADER\_SIZE sizeof(StreamHeader)

#define MAX\_NFRAMES 16

#define MAX\_NSTREAMS 16

/\*\*

\* @brief FSF file mode

\*/

enum class FsfMode {

READ,

WRITE

};

/\*\*

\* @brief FSF file processing status

\*/

enum class FsfStatus {

SUCCESS = 0,

FILE\_NOT\_OPEN,

FILE\_NOT\_CREATED,

FILE\_DOESNOT\_EXIST,

FILE\_FORMAT\_ERROR,

FILE\_HEADER\_ERROR,

INVALID\_OPERATION,

FAILED = -1

};

/\*\*

\* @brief FSF file stream data types

\*/

enum class StreamType {

STREAM\_TYPE\_UNKNOWN,

STREAM\_TYPE\_RAW,

STREAM\_TYPE\_ACTIVE\_BR,

STREAM\_TYPE\_PHASE,

STREAM\_TYPE\_X,

STREAM\_TYPE\_Y,

STREAM\_TYPE\_DEPTH,

STREAM\_TYPE\_RGB,

STREAM\_TYPE\_RAW\_NORM,

STREAM\_TYPE\_RAW\_REAL,

STREAM\_TYPE\_RAW\_IMAG,

STREAM\_TYPE\_RAW\_REAL\_FILT,

STREAM\_TYPE\_RAW\_IMAG\_FILT,

STREAM\_TYPE\_RADIAL,

STREAM\_TYPE\_RADIAL\_FILT,

STREAM\_TYPE\_COMMON\_MODE,

STREAM\_TYPE\_CONF,

STREAM\_TYPE\_VARIANCE,

STREAM\_TYPE\_REFLECTIVITY

};

struct FileHeader {

uint32\_t HeaderSize;

uint32\_t MagicNumber;

uint16\_t FileFormatMajorVersion;

uint16\_t FileFormatMinorVersion;

uint32\_t FileHeaderSize;

uint32\_t StreamInfoSize;

uint32\_t StreamHeaderSize;

uint32\_t OptionalFileHdrSize;

uint32\_t FileCommentSize;

uint64\_t FrameOffsetInfoLoc;

uint32\_t nFrames;

uint32\_t nStreams;

};

struct StreamInfo {

uint32\_t SystemID;

uint32\_t StreamType;

uint32\_t ChannelFormat;

uint32\_t BytesPerPixel;

uint32\_t nRowsPerStream;

uint32\_t nColsPerStream;

uint32\_t OptionalStreamHdrSize;

uint32\_t StreamCommentSize;

uint32\_t CompressionScheme;

};

typedef std::string OptionalFileHeader;

typedef std::string FileComment;

struct StreamHeader {

uint32\_t TimeStamp;

uint32\_t CompressedStreamSize;

};

typedef std::string OptionalStreamHeader;

typedef std::string StreamComment;

typedef std::vector<uint8\_t> StreamData;

struct Stream {

StreamHeader streamHeader;

OptionalStreamHeader optionalStreamHeader;

StreamComment streamComment;

StreamData streamData;

};

#endif // FSF\_DEFINITIONS\_H

# High Level Data Architecture

## Enumerated Types

### FsfMode

The constructor argument type for creating FSF class objects.

|  |  |
| --- | --- |
| Fsfmode | Description |
| READ | This value must be passed to the FSF class constructor for FSF objects used for reading from FSF files. |
| WRITE | This value must be passed to the FSF class constructor for FSF objects used for writing to FSF files. |

### FsfStatus

The return status of FSF API calls.

|  |  |
| --- | --- |
| FsfStatus | Description |
| SUCCESS | Indicates successful API operation. |
| FILE\_NOT\_OPEN | Indicates the FSF file is not opened for reading or writing. |
| FILE\_NOT\_CREATED | Indicates the FSF file is not created on WRITE mode. |
| FILE\_DOESNOT\_EXIST | Indicates the FSF file to read does not exist. |
| FILE\_FORMAT\_ERROR | Indicates the opened file is not a valid FSF file. |
| FILE\_HEADER\_ERROR | Indicates an invalid file header is detected which prevents further API operation. |
| INVALID\_OPERATION | Indicates invalid use of the FSF API. See API descriptions. |
| FAILED | Indicates generic API failure. See API descriptions. |

### StreamType

<TODO>

## Data Structures

The figure below shows the detailed structure of the contents of an FSF file. Such format is stream type agnostic. This means that the format remains the same whether the stored stream data bytes is a stream of float, signed, or raw data byte values. The FSF data structures are patterned closely from this format.

For reference, whenever *size* is mentioned in this section, it refers to the number of bytes in the referred object.



### FileHeader

Data structure for the file header.

|  |  |
| --- | --- |
| FileHeader | Description |
| HeaderSize | The size of the FileHeader structure plus the size of the StreamInfo structure multiplied by the number of streams. |
| MagicNumber | The hexadecimal representation of the string “FSF” which is 0x00465346. |
| FileFormatMajorVersion | File format major revision number. |
| FileFormatMinorVersion | File format minor revision number. |
| FileHeaderSize | The size of the FileHeader structure. |
| StreamInfoSize | The size of the StreamInfo structure. |
| StreamHeaderSize | The size of the StreamHeader structure. |
| OptionalFileHdrSize | The size of the optional file header string. |
| FileCommentSize | The size of the file comment string. |
| FrameOffsetInfoLoc | The offset of the FrameOffsetInfo structure from the start of file. |
| nFrames | The number of frames that are contained within the FSF file. |
| nStreams | The number of streams within each frame in the FSF file. |

### StreamInfo

Data structure for the stream information.

|  |  |
| --- | --- |
| StreamInfo | Description |
| SystemID | The combined processor and sensor identification. |
| StreamType | The types of streams e.g. Active Brightness, Phase, RAW, X, Y, Z. |
| ChannelFormat |  |
| BytesPerPixel |  |
| nRowsPerStream |  |
| nColsPerStream |  |
| OptionalStreamHdrSize |  |
| StreamCommentSize |  |
| CompressionScheme |  |

### StreamHeader

Data structure for stream header information.

|  |  |
| --- | --- |
| StreamHeader | Description |
| TimeStamp | TODO |
| CompressedStreamSize | When no compression is applied this would be the actual stream size. |

### Stream

Stream data structure.

|  |  |
| --- | --- |
| Stream | Description |
| streamHeader | Stream header time stamp and compressed stream size. |
| optionalStreamHeader |  |
| streamComment |  |
| streamData |  |

# API Definitions

## Constructor

|  |  |
| --- | --- |
| Prototype | FSF\_Common(FsfMode mode); |
| Description | Create an FSF file utility object in the given mode. |
| Return | Pointer to the class object. |

## CreateFile()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus CreateFile(const char \*pFilename); |
| Description | This creates a new file with the specified file name for writing internally allocated FSF data structures. Internal structures are created through the corresponding Setxxx methods. |
| Return  SUCCESS  FILE\_NOT\_CREATED  INVALID\_OPERATION | If able to successfully create a new file  If the file cannot be created with the given name  If API is invoked with the FSF file utility in READ mode |

## OpenFile()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus OpenFile(const char \*pFilename); |
| Description | The specified FSF file is opened for reading. Contents of the file can be read into allocated FSF data structures through the corresponding Getxxx methods. |
| Return  SUCCESS  FILE\_DOESNOT\_EXIST  FILE\_FORMAT\_ERROR  INVALID\_OPERATION | If able to successfully open the file  If tried to open a file that does not exist  If the opened file is of invalid format  If API is invoked with the FSF file utility in WRITE mode |

## SaveFile()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus SaveFile(const char \*pFilename); |
| Description | This populates the created FSF file with the contents of the internally allocated FSF data structures. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FILE\_NOT\_OPEN  FAILED  INVALID\_OPERATION | If able to successfully save FSF contents to file  If the file header structure is invalid  If file is not created before saving contents to file  If there are errors while saving structures into file  If API is invoked with the FSF file utility in READ mode |

## CloseFile()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus CloseFile(void); |
| Description | Close the created/opened FSF file. |
| Return | Always returns successfully |

## SetFileHeader()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus SetFileHeader(FileHeader &fileHeader); |
| Description | The internally allocated FileHeader structure is populate with the contents of the referenced FileHeader structure. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  INVALID\_OPERATION | If internal structures are updated successfully  If the file header structure is invalid  If API is invoked with the FSF file utility in READ mode |

## GetFileHeader()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus GetFileHeader(FileHeader &fileHeader); |
| Description | The referenced FileHeader structure is populated with the file header information from the opened FSF file. |
| Return  SUCCESS  FILE\_NOT\_OPEN  FAILED  FILE\_HEADER\_ERROR  INVALID\_OPERATION | If the file header is retrieved successfully  If tried to read file header with the file not open  If there are errors encountered while reading from file  If the retrieved file header is invalid  If API is invoked with the FSF file utility in WRITE mode |

## SetStreamInfo()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus SetStreamInfo(const unsigned int streamIndex, StreamInfo &streamInfo); |
| Description | The internally allocated StreamInfo structure for the selected stream is populated with the contents of the referenced StreamInfo structure. The FileHeader must be set first prior to setting the StreamInfo structures. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FAILED  INVALID\_OPERATION | If internal structures are updated successfully  If the file header structure is invalid  If the stream info structure index is invalid  If API is invoked with the FSF file utility in READ mode |

## GetStreamInfo()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus GetStreamInfo(const unsigned int streamIndex, StreamInfo &streamInfo); |
| Description | The referenced StreamInfo structure is populated with the stream information for the selected stream from the opened FSF file. The FileHeader must be retrieved first prior to getting the stream information. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FILE\_NOT\_OPEN  FAILED  INVALID\_OPERATION | If the stream info is retrieved successfully  If the file header structure is invalid  If tried to read stream info with the file not open  If there are errors encountered while reading from file or the stream info structure index is invalid  If API is invoked with the FSF file utility in WRITE mode |

## SetOptionalFileHeader()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus SetOptionalFileHeader(OptionalFileHeader &optFileHeader); |
| Description | If the OptionalStreamHdrSize in the FileHeader is non-zero, a buffer enough to hold the OptionalFileHeader string is internally allocated. The allocated buffer is populated with the contents of the referenced OptionalFileHeader string. The FileHeader must be set first prior to setting the OptionalFileHeader string. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FAILED  INVALID\_OPERATION | If internal structures are updated successfully  If the file header structure is invalid  If OptionalFileHdrSize is different with actual string size  If API is invoked with the FSF file utility in READ mode |

## GetOptionalFileHeader()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus GetOptionalFileHeader(OptionalFileHeader &optFileHeader); |
| Description | If the OptionalStreamHdrSize in the FileHeader is non-zero, the referenced OptionalFileHeader string buffer is populated with the contents of the OptionalFileHeader string from the opened FSF file. The FileHeader must be retrieved first prior to getting the OptionalFileHeader string. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FILE\_NOT\_OPEN  FAILED  INVALID\_OPERATION | If the optional file header is retrieved successfully  If the file header structure is invalid  If tried to read optional file header with the file not open  If there are errors encountered while reading from file  If API is invoked with the FSF file utility in WRITE mode |

## SetFileComment()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus SetFileComment(FileComment &fileComment); |
| Description | If the FileCommentSize in the FileHeader is non-zero, a buffer enough to hold the FileComment string is internally allocated. The allocated buffer is populated with the contents of the referenced FileComment string. The FileHeader must be set first prior to setting the FileComment string. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FAILED  INVALID\_OPERATION | If internal structures are updated successfully  If the file header structure is invalid  If FileCommentSize is different with actual string size  If API is invoked with the FSF file utility in READ mode |

## GetFileComment()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus GetFileComment(FileComment &fileComment); |
| Description | If the FileCommentSize in the FileHeader is non-zero, the referenced FileComment string buffer is populated with the contents of the FileComment string from the opened FSF file. The FileHeader must be retrieved first prior to getting the FileComment string. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FILE\_NOT\_OPEN  FAILED  INVALID\_OPERATION | If the file comment string is retrieved successfully  If the file header structure is invalid  If tried to read file comment with the file not open  If there are errors encountered while reading from file  If API is invoked with the FSF file utility in WRITE mode |

## SetStream()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus SetStream(const unsigned int frameIndex, const unsigned int streamIndex, Stream &stream); |
| Description | The internally allocated Stream structure for the selected stream is populated with the contents of the referenced Stream structure. Stream structure are structures are internally allocated based on the the number of frames (nFrames) and the number of streams for each frame (nStreams). The FileHeader must be set first prior to setting the Stream structures. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FAILED  INVALID\_OPERATION | If internal structures are updated successfully  If the file header structure is invalid  If stream index is invalid or the actual data/string sizes does not match those in the stream info  If API is invoked with the FSF file utility in READ mode |

## GetStream()

|  |  |
| --- | --- |
| Prototype | virtual FsfStatus GetStream(const unsigned int frameIndex, const unsigned int streamIndex, Stream &stream); |
| Description | The referenced Stream structure is populated with the contents of the selected stream from the opened FSF file. The FileHeader must be retrieved first prior to retrieving streams. |
| Return  SUCCESS  FILE\_HEADER\_ERROR  FILE\_NOT\_OPEN  FAILED  INVALID\_OPERATION | If the stream is retrieved successfully  If the file header structure is invalid  If tried to read stream contents with the file not open  If there are errors encountered while reading from file or the stream index is invalid  If API is invoked with the FSF file utility in WRITE mode |

# C Wrapper API Definitions

The FSF utility is written primarily in C++. For usability with C-based applications, wrapper APIs are provided. The API return types are Boolean values. Boolean t*rue* corresponds to successful API operations, *false* otherwise.

## CreateFSFFile()

|  |  |
| --- | --- |
| Prototype | bool CreateFSFFile(unsigned int \*fileHandle, LPCSTR fileName, unsigned int nStreams, FSFSTREAMINFO \*streamInfo, unsigned int optFileHdrSize, unsigned int fileCommSize, unsigned int nFrames); |
| Description | Create a new FSF file for writing stream data. It creates a new file with the given filename if it does not exist otherwise, the existing file contents are overwritten. The number of frames, number of streams within each frame, optional file header strings size, and file comment string size must be known prior to calling this API. Additionally, the stream information structure for all the streams must also be constructed beforehand.  The referenced file handle is updated with the handle to the created FSF file and internal FSF utility object. |

## ReadFSFFile()

|  |  |
| --- | --- |
| Prototype | bool ReadFSFFile(LPCSTR fileName, unsigned int \*fileHandle); |
| Description | Open an FSF file for reading. The given filename should be an existing file.  The referenced file handle is updated with the handle to the opened FSF file and internal FSF utility object. |

## ReadFileHeader()

|  |  |
| --- | --- |
| Prototype | bool ReadFileHeader(unsigned int fileHandle, FSFFILEHEADER \*fileHeader); |
| Description | Read the FSF file header. The referenced file header structure is updated with the file header information coming from the opened FSF file. |

## ReadStreamInfo()

|  |  |
| --- | --- |
| Prototype | bool ReadStreamInfo(unsigned int fileHandle, unsigned int streamID, FSFSTREAMINFO \*streamInfo); |
| Description | Read the FSF stream information. The referenced stream information structure is updated with the information for the given stream index coming from the opened FSF file. |

## WriteOptionalFileHdr()

|  |  |
| --- | --- |
| Prototype | WriteOptionalFileHdr(unsigned int fileHandle, void \*optionalFileHeader); |
| Description | Set the optional file header string for writing into an FSF file. The internal optional file header string buffer is updated with the contents of the referenced string. |

## ReadOptionalFileHdr()

|  |  |
| --- | --- |
| Prototype | ReadOptionalFileHdr(unsigned int fileHandle, void \*optionalFileHeader); |
| Description | Read the optional file header string from the opened FSF file. The referenced optional file header string buffer is updated with the optional file header information from the opened FSF file. |

## WriteFileComments ()

|  |  |
| --- | --- |
| Prototype | WriteFileComments(unsigned int fileHandle, void \*fileComments); |
| Description | Set the file comment string for writing into an FSF file. The internal file comment string buffer is updated with the contents of the referenced string. |

## ReadFileComments()

|  |  |
| --- | --- |
| Prototype | ReadFileComments(unsigned int fileHandle, void \*fileComments); |
| Description | Read the file comment string from the opened FSF file. The referenced file comment string buffer is updated with the file comment information from the opened FSF file. |

## WriteStream()

|  |  |
| --- | --- |
| Prototype | bool WriteStream(unsigned int fileHandle, unsigned int frameID, unsigned int streamID, unsigned int timeStampInmsecs, void \*optStreamHdr, void \*comments, void \*frameData); |
| Description | Set the stream information for the given frame and stream ID. The internal stream structure along with the buffer for the stream data, optional stream header information, and stream comment are updated with the contents of the respective referenced buffers. |

## ReadStream()

|  |  |
| --- | --- |
| Prototype | bool ReadStream(unsigned int fileHandle, unsigned int frameID, unsigned int streamID, unsigned int \*timeStampInmsecs, void \*optStreamHdr, void \*comments, void \*frameData); |
| Description | Read the stream information for the given frame and stream ID. The referenced stream structure along with the referenced buffer for the stream data, optional stream header information, and stream comment are updated with the contents coming from the opened FSF file. |

## IsFileOpen()

|  |  |
| --- | --- |
| Prototype | bool IsFileOpen(unsigned int fileHandle); |
| Description | Returns true if the file for a given file handle was created/opened successfully. False, otherwise. |

## Close()

|  |  |
| --- | --- |
| Prototype | bool Close(unsigned int fileHandle); |
| Description | Closes the file (and the FSF utility object) referenced by the given file handle. If the filehandle is for an FSF file opened for writing, the buffers are saved into the file prior to closing the file. |

## GetWindowsErrorCode()

|  |  |
| --- | --- |
| Prototype | unsigned int GetWindowsErrorCode(void); |
| Description | This API returns the error code (0 means SUCCESS) of the last API call that was made. |

# C Wrapper Comparison with Microsoft FSF C APIs

The following table summarizes the comparison of the FSF utility’s C wrappers with the existing Microsoft FSF utility APIs which was distributed as a windows library.

|  |  |  |
| --- | --- | --- |
| **API** | **Microsoft C API Parameters** | **FSF - C Interface Parameters** |
| **CreateFSFFile** | unsigned int\* fileHandle | unsigned int \*fileHandle |
| LPCSTR fileName | LPCSTR fileName |
| unsigned int nStreams | unsigned int nStreams |
| FSFSTREAMINFO\* streamInfo | FSFSTREAMINFO \*streamInfo |
| unsigned int optFileHdrSize | unsigned int optFileHdrSize |
| unsigned int fileCommSize | unsigned int fileCommSize |
| FILE\_CREATION\_FLAGS mode | unsigned int nFrames |
|  | | |
| **ReadFSFFile** | LPCSTR fileName | LPCSTR fileName |
| unsigned int\* fileHandle | unsigned int \*fileHandle |
|  | | |
| **ReadFileHeader** | unsigned int fileHandle | unsigned int fileHandle |
| FSFFILEHEADER\* fileHeader | FSFFILEHEADER \*fileHeader |
|  | | |
| **ReadStreamInfo** | unsigned int fileHandle | unsigned int fileHandle |
| unsigned int streamID | unsigned int streamID |
| FSFSTREAMINFO\* streamInfo | FSFSTREAMINFO \*streamInfo |
|  | | |
| **WriteOptionalFileHdr** | unsigned int fileHandle | unsigned int fileHandle |
| void\* optionalFileHeader | void \*optionalFileHeader |
|  | | |
| **ReadOptionalFileHdr** | unsigned int fileHandle | unsigned int fileHandle |
| void\* optionalFileHeader | void \*optionalFileHeader |
|  | | |
| **WriteFileComments** | unsigned int fileHandle | unsigned int fileHandle |
| void\* fileComments | void \*fileComments |
|  | | |
| **ReadFileComments** | unsigned int fileHandle | unsigned int fileHandle |
| void\* fileComments | void \*fileComments |
|  | | |
| **WriteStream** | unsigned int fileHandle | unsigned int fileHandle |
| unsigned int frameID | unsigned int frameID |
| unsigned int streamID | unsigned int streamID |
| unsigned int timeStampInmsecs | unsigned int timeStampInmsecs |
| void\* optStreamHdr | void \*optStreamHdr |
| void\* comments | void \*comments |
| void\* frameData | void \*frameData |
|  | | |
| **ReadStream** | unsigned int fileHandle | unsigned int fileHandle |
| unsigned int frameID | unsigned int frameID |
| unsigned int streamID | unsigned int streamID |
| unsigned int\* timeStampInmsecs | unsigned int \*timeStampInmsecs |
| void\* optStreamHdr | void \*optStreamHdr |
| void\* comments | void \*comments |
| void\* frameData | void \*frameData |
|  | | |
| **IsFileOpen** | unsigned int fileHandle | unsigned int fileHandle |
|  | | |
| **Close** | unsigned int fileHandle | unsigned int fileHandle |
|  | | |
| **GetWindowsErrorCode** | void | void |

# Example

## FSF Template Writer

The following example shows how to use the FSF utility in writing stream data from the sensor.

...

void main(int argc, char \*argv[])

{

FSF \*pFSF = new FSF\_Common { FsfMode::WRITE };

if (pFSF != NULL)

{

pFSF->CreateFile(“Write.fsf”);

pFSF->SetFileHeader(...);

pFSF->SetStreamInfo(...);

pFSF->SetStreamInfo(...);

pFSF-> SetOptionalFileHeader(...);

pFSF-> SetFileComment(...);

pFSF->SetStream(...);

pFSF->SetStream(...);

pFSF->SaveFile(...);

pFSF->CloseFile(...);

}

}

# Limitations

# Traceability to Requirements

|  |  |  |
| --- | --- | --- |
| **Document reference ID & Description:** <Typically requirement specification document.> | | |
| **Serial No.** | **Reference document**  **Requirement (Section ID/Name)** | **Design document**  **(Section ID/Name)** |
|  |  |  |
|  |  |  |

# Others