FOTONIC

FZ-API Reference Manual

Rev K
03/22/2013

1. Introduction to FZ-API

This document describes the FZ-API used to access the Fotonic cameras - how to set up the camera, and how to retrieve images. The interface provides a robust, efficient, and easy-to-use framework for application developers. The interface is kept simple to promote efficiency, readability, and usability. The API design is hierarchical, resulting in an efficient and extensible interface that provides access to sensor data. This architecture can be extended in the future by Fotonic to implement application libraries or other more complex image processing services.

Since the volume of data generated by a real-time sensor can be large, the implementation of the API must provide efficient frame transfer. The purpose of the API is to make frames available to the application with a minimum delay from the moment that the data is output from the sensor.

The application uses the API to:

- Open the sensor and define the initial parameters
- Send commands to control the sensor and set its operational parameters
- Read X, Y, Z and Brightness frames from the camera.
- Close the sensor.

The API is delivered as libraries. On PC with Windows, the libraries are in the form of a DLLs. On Linux it is one static library (`libfz_api.a`).

1.1 Programming model

The programming model for using FZ-API is as follows:

- Establish connection to a sensor.
- Configure the sensor with desired parameters such as shutter, frame rate, etc.
- Start the sensor.
- Get x,y,z and b frame.

o Consume the data.

- Repeat until done.
- Stop the sensor.
- Close connection to sensor.

These steps and other information relevant to the programmer are described below.

1.2 Includes

Include header files are

- `fz_api.h`, which in turn includes `fz_types.h` and `fz_commands.h`.

`fz_api.h` contains all functions and data structures for the API.
`fz_types.h` contains definitions for simple data types used in the API.
`fz_commands.h` contains definitions for all commands and values to pass to the `FZ_Ioctl()` function.

Include libraries are

- Windows: `fz_api.lib` (for `fz_api.dll`
- Linux: `libfz_api.a`

1.3 Sample application

The example application FZExample works on windows (compiled with Visual Studio 2008 or 2010), and linux (compiled with g++).

2. API Documentation

This describes the interface on the PC used to give control of a sensor to the application. The main purpose of this API is to program the sensor to return brightness, and depth images with desired properties. In addition, this API provides a sensor-independent way of configuring a Fotonic sensor. Using the same API, one can write an application that with no or minimal change, would work with another Fotonic sensor.

2.1 FZ_Init

• Synopsis

Initializes the FZ API. Should be run first in the user application. (Not needed in the Windows-version of the API)

• Syntax

```
FZ_Result FZ_Init()
```

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

2.2 FZ_Exit

• Synopsis

Uninitializes the API. Must be called by the application after finishing using the API. No more API-calls may be done before another call to `FZ_Init`.

• Syntax

```
FZ_Result FZ_Exit()
```

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

2.3 FZ_EnumDevices2

• Synopsis

Enumerates the FZ devices that are connected to the system.

• Syntax

```
FZ_Result FZ_EnumDevices2(
           FZ_DEVICE_INFO* pDeviceInfo,
           int* piNumDevices)
```

• Parameters

[in/out] `pDeviceInfo`: Receives an array of `FZ_DEVICE_INFO` of the found devices.
[in/out] `piNumDevices`: Contains the max number of devices that `pDeviceInfo` can receive. The actual number of devices found is returned here (up to the provided value).

The format of pDeviceInfo is

```
struct FZ_DEVICE_INFO
{
   uint32_t iDeviceType;          Currently this reports 0 for all canesta devices, and 1 for
                                  panasonic.
   char     szPath[512];          Device path to use in FZ_Open
   char     szShortName[32];      A more user friendly name than szPath
   char     szSerial[16];         Device serial number string. “N/A” for USB devices as it can not
                                  be retreived in the enumeration process.
   uint32_t iReserved[64];        Reserved for future use
};
```

• Return value

`FZ_Success`, The call was completed successfully.
`FZ_TOO_MANY_DEVICES`, More devices are found than the number that can be returned.
Refer to the `FZ_Result` declaration for other values which indicate failure.

• Notes

- For Ethernet cameras the camera must be able to see broadcasts sent from the PC (port 1288), and the PC must see the camera UDP reply for a camera to be detectable by this function.
- For Ethernet cameras the UDP reply is sent to a port in a range starting at 60000, the first free port is always selected. The max number will depend on how much this method is run simultaneously from different threads/processes on the PC, and if other programs have ports occupied. A good number for the range would be 60000-60050.

2.4 FZ_Open

• Synopsis

Establishes a link with the device and returns a handle for further references.

• Syntax

```
FZ_Result FZ_Open(
           char* szDevicePath,
           unsigned int iFlags,
           FZ_Device_Handle_t* phDev)
```

• Parameters

[in] `szDevicePath`: Either a device path returned from `FZ_EnumDevices2` or a known device path. For Ethernet cameras, this is the IP number followed by an optional port (a.b.c.d[:port]).
[in] `iFlags`: Must be set to 0.
[in/out] `phDev`: Receives a handle to the device.

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

2.5 FZ_Close

• Synopsis

Closes the link between the device and the application.

• Syntax

```
FZ_Result FZ_Close(
           FZ_Device_Handle_t hDev)
```

• Parameters

[in] `hDev`: Handle to an open device.

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

2.6 FZ_Ioctl

• Synopsis

Sends an IO Control code to the device to set or get parameters. This command is used to start and stop a device, and set parameters of the device. It is also used to query the values of parameters of the device. The explanation of the I/O codes is given in the table below. This list is subject to expand/change in future releases.

• Syntax

```
FZ_Result FZ_IOCtl(
            FZ_Device_Handle_t hDev,
            FZ_CmdCode_t iCmd,
            void *pParam,
            int iCmdByteLen,
            FZ_CmdRespCode_t* piRespCode,
            void *pResp,
            int *piRespByteLen)
```

• Parameters

[in] `hDev`: Handle to an open device.
[in] `iCmd`: IO Control code. See below.
[in] `pParam`: Pointer to the input parameter. (Input value)
[in] `iCmdByteLen`: Size of the input parameter.
[in/out] `piRespCode`: Receives the response code (typically R_CMD_DE_ACK or R_CMD_DE_NACK).
[in/out] `pResp`: Receives the response, the response length depends on the `iCmd`. Can be NULL. (Output value)
[in/out] `piRespByteLen`: Contains the size of the memory allocated to `pResp` on input and is filled with the number of bytes written to `pResp` on output. Can be NULL if `pResp` is NULL.

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

• Notes

- This function is thread safe, i.e it can be used from another thread than other functions used on the handle. All functions are thread safe when operating on different handles.

• Explanation of IO Control codes

Jaguar sensors				
Control Code	Input value	Output value	Explanation	Range
CMD_DE_SENSOR_START	NULL	NULL	Starts the sensor	N/A
CMD_DE_SENSOR_STOP	NULL	NULL	Stops the sensor	N/A
CMD_DE_SET_SHUTTER	short	NULL	Sets the sensor shutter time. This parameter dominates the frame rate. If the shutter time is too high for a given frame rate, the frame rate is reduced accordingly. Use CMD_DE_GET_FPS to see current frame rate.	Min = 2 (0.2 ms) Max = 400 (40 ms) Default 20 (2.0 ms)
CMD_DE_SET_SHUTTER_EXT	FZ_SHUTTER_EXT	NULL	Useful for multishuter, can set one or many shutters at once. See fz_commands.h for declaration and use of the FZ_SHUTTER_EXT structure.	See CMD_DE_SET_SHUTTER Shutter with index 2 must be set lower than or equal to shutter with index 1.
CMD_DE_SET_FPS	short	NULL	Sets the sensor frame rate. This parameter is dominated by the shutter time. If the shutter time is too high for a given frame rate, the frame rate is reduced accordingly. Use CMD_DE_GET_FPS to see current frame rate.	Min = 1 Max = Mode dependent
CMD_DE_SET_FPS_DIVISOR	short	NULL	Sets a divisor to the FPS value, so that the camera outputs images at a lower rate, but runs the higher frame rate internally.	Min = 1 Max = 60 Default 1
CMD_DE_SET_LSSWITCH	short	NULL	Sets the state of the sensor light source (on/off)	0 (off) 1 (on) Default = 0
CMD_DE_SET_MODE	short	NULL	Sets the sensor operation mode	DE_MODE_TEMPORAL DE_MODE_SPATIO_TEMPORAL DE_MODE_MULTI_ST DE_MODE_ZFAST DE_MODE_ZFINE

				DE_MODE_MS_ST DE_MODE_BM_ZFINE DE_MODE_BM_TEMPORAL DE_MODE_BM_SPATIO_TEMPORAL DE_MODE_BM_MULTI_ST DE_MODE_BM_MS_ST DE_MODE_BM_MS_ZFINE Default DE_MODE_TEMPORAL
CMD_DE_SET_MS_SATURATION	short	NULL	Multi shutter saturation value for a pixel. If exceeded it will trigger a pixel swap to the exposure with lower shutter value.	Min = 0 Max = 2048 Default 1200
Control Code	Input value	Output value	Explanation	Range
CMD_DE_GET_SHUTTER	NULL	short	Returns the sensor shutter in 0.1 ms units	See CMD_DE_SET_SHUTTER
CMD_DE_SET_SHUTTER_EXT	FZ_SHUTTER_EXT	FZ_SHUTTER_EXT	Returns one or more sensor shutter times.	See CMD_DE_SET_SHUTTER_EXT
CMD_DE_GET_FPS	NULL	short	Returns the sensor frame rate	See CMD_DE_SET_FPS
CMD_DE_GET_FPS_DIVISOR	NULL	short	Returns the sensor frame rate divisor	See CMD_DE_SET_FPS_DIVISOR
CMD_DE_GET_LSSWITCH	NULL	short	Returns the state of the sensor light source	See CMD_DE_SET_LSSWITCH
CMD_DE_GET_MODE	NULL	short	Returns the sensor operating mode	See CMD_DE_SET_MODE
CMD_DE_GET_MS_SATURATION	NULL	short	Returns the multi shutter saturation value	See CMD_DE_SET_MS_SATURATION
CMD_DE_GET_WOI	NULL	2xshort	Returns W and H	W=160, H=120

Control Code	Input value	Output value	Explanation	Range
CMD_DE_SET_FILTERCONTROL	short	NULL	Enables or disables the depth filter. Filtered pixels will be given the Z value -1	0 (off) 1 (on) Default 0
CMD_DE_SET_LOWLIGHT_GT_PASSIVE	short	NULL	Sets the minimum threshold	Min = 0

			for the passive brightness of the depth filter	Max = 255 Default 0
CMD_DE_SET_LOWLIGHT_GT_ACTIVE	short	NULL	Sets the minimum threshold for the active brightness of the depth filter	Min = 0 Max = 32767 Default 8
CMD_DE_SET_SATURATION_LT_B	short	NULL	Sets the maximum threshold for the passive brightness of the depth filter	Min = 0 Max = 255 Default 255
CMD_DE_SET_SATURATION_LT_A_MINUS_B	short	NULL	Sets the maximum threshold for the differential read of the depth filter	Min = -4096 Max = 4096 Default 4096
CMD_DE_SET_SATURATION_GT_A_MINUS_B	short	NULL	Sets the minimum threshold for the differential read of the depth filter	Min = -4096 Max = 4096 Default -4096
CMD_DE_GET_FILTERCONTROL	NULL	short	Returns the sensor depth filter state	See CMD_DE_SET_FILTERCONTROL
CMD_DE_GET_LOWLIGHT_GT_PASSIVE	NULL	short	Returns the minimum threshold for the passive brightness of the depth filter	See CMD_DE_SET_LOWLIGHT_GT_PASSIVE
CMD_DE_GET_LOWLIGHT_GT_ACTIVE	NULL	short	Returns the minimum threshold for the active brightness of the depth filter	See CMD_DE_SET_LOWLIGHT_GT_ACTIVE
CMD_DE_GET_SATURATION_LT_B	NULL	short	Returns the maximum threshold for the passive brightness of the depth filter	See CMD_DE_SET_SATURATION_LT_B
CMD_DE_GET_SATURATION_LT_A_MINUS_B	NULL	short	Returns the maximum threshold for the differential read of the depth filter	See CMD_DE_SET_SATURATION_LT_A_MINUS_B
CMD_DE_GET_SATURATION_GT_A_MINUS_B	NULL	short	Returns the minimum threshold for the differential read of the depth filter	See CMD_DE_SET_SATURATION_GT_A_MINUS_B
CMD_DE_SET_EDGE_FILTER	short	NULL	Sets the edge filter operation	0 (off) 1 (on) Default 0

CMD_DE_GET_EDGE_FILTER	NULL	short	Returns the edge filter state	See CMD_DE_SET_EDGE_FILTER

Control Code	Input value	Output value	Explanation	Range
CMD_API_GET_VERSION	NULL	char [128]	Returns the API version (version of FZ_API.dll)	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_VERSION	NULL	char [128]	Returns the camera depth engine version	ASCII, variable length up to 128, NULL terminated
CMD_CA_GET_VERSION	NULL	char [128]	Returns the camera client application version	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_PCODE	NULL	char [128]	Returns the camera product code	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_UNIT_NO	NULL	char [128]	Returns the camera serial number	10 digits, NULL terminated

Panasonic sensors				
Control Code	Input value	Output value	Explanation	Range
CMD_DE_SENSOR_START	NULL	NULL	Starts the sensor	N/A
CMD_DE_SENSOR_STOP	NULL	NULL	Stops the sensor	N/A
CMD_DE_SET_SHUTTER	short	NULL	Sets the sensor shutter time. This parameter dominates the frame rate. If the shutter time is too high for a given frame rate, the frame rate is reduced accordingly. Use CMD_DE_GET_FPS to see current frame rate.	Min = 0 (auto) Max = 30 (3.0 ms) Default 8 (0.8 ms)
CMD_DE_SET_SHUTTER_EXT	FZ_SHUTTER_EXT	NULL	Useful for multishuter, can set one or many shutters at once. See fz_commands.h for declaration and	See CMD_DE_SET_SHUTTER

			use of the FZ_SHUTTER_EXT structure.	
CMD_DE_SET_FPS	short	NULL	Sets the sensor frame rate. This parameter is dominated by the shutter time. If the shutter time is too high for a given frame rate, the frame rate is reduced accordingly. Use CMD_DE_GET_FPS to see current frame rate.	Min = 30 Max = 60
CMD_DE_SET_FPS_DIVISOR	short	NULL	Sets a divisor to the FPS value, so that the camera outputs images at a lower rate, but runs the higher frame rate internally.	Min = 1 Max = 60 Default 1
CMD_DE_SET_MODE	short	NULL	Sets the sensor operation mode	DE_MODE_PA_Z DE_MODE_PA_Z_MS DE_MODE_PA_RAW DE_MODE_PA_RAW_320 Default DE_MODE_PA_Z
Control Code	Input value	Output value	Explanation	Range
CMD_DE_GET_SHUTTER	NULL	short	Returns the sensor shutter in 0.1 ms units	See CMD_DE_SET_SHUTTER
CMD_DE_GET_FPS	NULL	short	Returns the sensor frame rate	See CMD_DE_SET_FPS
CMD_DE_GET_FPS_DIVISOR	NULL	short	Returns the sensor frame rate divisor	See CMD_DE_SET_FPS_DIVISOR
CMD_DE_GET_MODE	NULL	short	Returns the sensor operating mode	See CMD_DE_SET_MODE
CMD_DE_GET_WOI	NULL	2xshort	Returns W and H	W=160, H=120. W is doubled in 320 modes.
CMD_DE_GET_LIGHT_FRQ	NULL	short	Returns the sensor light frequency.	Min = 0 Max = 3
CMD_DE_GET_LIGHT_PWR	NULL	short	Returns the sensor light power.	Min = 0 Max = 3

Control Code	Input value	Output value	Explanation	Range
CMD_DE_SET_EDGE_FILTER	short	NULL	Sets the edge filter operation	0 (off) 1 (on) Default 0
CMD_DE_GET_EDGE_FILTER	NULL	short	Returns the edge filter state	See CMD_DE_SET_EDGE_FILTER

Control Code	Input value	Output value	Explanation	Range
CMD_API_GET_VERSION	NULL	char [128]	Returns the API version (version of FZ_API.dll)	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_VERSION	NULL	char [128]	Returns the camera depth engine version	ASCII, variable length up to 128, NULL terminated
CMD_CA_GET_VERSION	NULL	char [128]	Returns the camera client application version	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_PCODE	NULL	char [128]	Returns the camera product code	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_UNIT_NO	NULL	char [128]	Returns the camera serial number	10 digits, NULL terminated

Asus sensors				
Control Code	Input value	Output value	Explanation	Range
CMD_DE_SENSOR_START	NULL	NULL	Starts the sensor	N/A
CMD_DE_SENSOR_STOP	NULL	NULL	Stops the sensor	N/A
CMD_DE_SET_SHUTTER	short	NULL	Sets the sensor shutter time. Ignored on this sensor.	N/A
CMD_DE_SET_SHUTTER_EXT	FZ_SHUTTER_EXT	NULL	Sets the sensor shutter time. Ignored on this sensor.	N/A
CMD_DE_SET_FPS	short	NULL	Sets the sensor frame rate. Ignored on this sensor, use CMD_DE_SET_MODE to control frame rate.	N/A

CMD_DE_SET_FPS_DIVISOR	short	NULL	Sets a divisor to the FPS value, so that the camera outputs images at a lower rate, but runs the higher frame rate internally.	Min = 1 Max = 60 Default 1
CMD_DE_SET_MODE	short	NULL	Sets the sensor operation mode	DE_MODE_320X240_30_RAW DE_MODE_320X240_60_RAW DE_MODE_640X480_30_RAW DE_MODE_320X240_30 DE_MODE_320X240_60 DE_MODE_640X480_30 Default DE_MODE_640X480_30
Control Code	Input value	Output value	Explanation	Range
CMD_DE_GET_SHUTTER	NULL	short	Returns the sensor shutter in 0.1 ms units	See CMD_DE_SET_SHUTTER
CMD_DE_GET_FPS	NULL	short	Returns the sensor frame rate	See CMD_DE_SET_FPS
CMD_DE_GET_FPS_DIVISOR	NULL	short	Returns the sensor frame rate divisor	See CMD_DE_SET_FPS_DIVISOR
CMD_DE_GET_MODE	NULL	short	Returns the sensor operating mode	See CMD_DE_SET_MODE
CMD_DE_GET_WOI	NULL	2xshort	Returns W and H	W=640, H=480

Control Code	Input value	Output value	Explanation	Range
CMD_API_GET_VERSION	NULL	char [128]	Returns the API version (version of FZ_API.dll)	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_VERSION	NULL	char [128]	Returns the camera depth engine version	ASCII, variable length up to 128, NULL terminated
CMD_CA_GET_VERSION	NULL	char [128]	Returns the camera client application version	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_PCODE	NULL	char [128]	Returns the camera product code	ASCII, variable length up to 128, NULL terminated
CMD_DE_GET_UNIT_NO	NULL	char [128]	Returns the camera serial number	10 igits, NULL terminated

2.7 FZ_SetFrameDataFmt

• Synopsis

Sets the frame output format for `FZ_GetFrame` functions.

• Syntax

```
FZ_Result FZ_SetFrameDataFmt(
           FZ_Device_Handle_t hDev,
           int x, int y,
           int w, int h,
           int iFlags)
```

• Parameters
[in] `hDev`: Handle to an open device.
[in] `x`: The start x coordinate of the output in comparison to the sensor frame.
[in] `y`: The start y coordinate of the output in comparison to the sensor frame.
[in] `w`: The width of the output. -1 to get the sensor frame size, overriding `x,y,w,h`.
[in] `h`: The height of the output, -1 to get the sensor frame size, overriding `x,y,w,h`.
[in] `iFlags`: The wanted components and processing when transferring the sensor frame to the output buffer. Any combination of the flags below can be specified by using the | operator.

```
FZ_FMT_COMPONENT_B       //1 short, order 1 if selected
FZ_FMT_COMPONENT_Z       //1 short, order 2 if selected
FZ_FMT_COMPONENT_XY      //2 short, order 3 if selected
FZ_FMT_COMPONENT_RADIALZ //1 short, order 4 if selected
FZ_FMT_COMPONENT_RGB     //4 uint8 (255,b,g,r), order 5 if selected
FZ_FMT_PIXEL_INTERLEAVED //components are grouped per pixel ([BZ...][BZ...]...)
FZ_FMT_PIXEL_PER_PLANE   //components are grouped per plane ([160xB][160xZ]...)
FZ_FMT_PROCESS_MIRROR    //mirrors all data positions
FZ_FMT_PROCESS_Z_FULLSCALE5M // Scale Z-values to: Z = Z_mm * 65535 / 5000
FZ_FMT_PROCESS_INVERTY   //all Y values multiplied with -1
```

• Return value
`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

• Frame data format

`FZ_FMT_PIXEL_INTERLEAVED`

Components	B(x)	Z(x)	X(x)	Y(x)	rZ(x)	RGB(x)	...	B(x+w-1)	Z(x+w-1)	X(x+w-1)	Y(x+w-1)	rZ(x+w-1)	RGB(x+w-1)
r(y)	2	2	2	2	2		...	2	2	2	2	2	4
r...	...	...	...	...	...		...	...	...	...	...	...	
r(y+h-1)	2	2	2	2	2		...	2	2	2	2	2	4

`FZ_FMT_PIXEL_PER_PLANE`

Components	B(x)..B(x+w-1)	Z(x)..Z(x+w-1)	X(x)..X(x+w-1)	Y(x)..Y(x+w-1)	rZ(x)..rZ(x+w-1)	RGB(x)..RGB(x

						+w-1)
r(y)	w*2	w*2	w*2	w*2	w*2	w*2
r...	...	...	...	...	...	
r(y+h-1)	w*2	w*2	w*2	w*2	w*2	w*2

The component order is always the order above, but only the selected components are present in the result. For example if B and rZ are selected the number of bytes per pixel will be 4.

Default after open (and if this function is never run) is
`x = -1, y = -1, w = -1, h = -1` (=using sensor dimension),
`iFlags = FZ_FMT_COMPONENT_B|FZ_FMT_COMPONENT_Z|FZ_FMT_COMPONENT_XY`

• Notes

- Set `w, h` to -1 to get the sensors full dimension (no clipping or extending of the image is done).
- If the output dimension does not equal the the sensor frame, output is either clipped, or expanded by adding 0 pixel data to pixels not present in the sensor.
- When `FZ_FMT_PROCESS_MIRROR` is used clipping may not be done the way that is expected.

2.8 FZ_GetFrame

• Synopsis

Gets one frame containing X, Y, Z and brightness data.

• Syntax

```
FZ_Result FZ_GetFrame(
            FZ_Device_Handle_t hDev,
            FZ_FRAME_HEADER *pHeader,
            void *pPixels, size_t *piPixelsByteLen)

FZ_Result FZ_GetFrameNewest(
            FZ_Device_Handle_t hDev,
            FZ_FRAME_HEADER *pHeader,
            void *pPixels, size_t *piPixelsByteLen)
```

• Parameters

[in] `hDev`: Handle to an open device.
[in/out] `pHeader`: Receives the frame header for the image.
[in/out] `pPixels`: Receives the pixels for the image.
[in/out] `piPixelsByteLen`: Contains the size of the memory allocated to `pPixels` on input, and is filled with the number of bytes written to `pPixels` on output.

• Return value

FZ_Success, The call was completed successfully.
Refer to the FZ_Result declaration for other values which indicate failure.

• Frame data format

pPixels will be filled with pixel data as set with FZ_SetFrameDataFmt. (see that function)

2.9 FZ_FrameAvailable

• Synopsis

Checks if at least one frame is available directly to FZ_GetFrame without it having to block.

• Syntax

```
FZ_Result FZ_FrameAvailable(
           FZ_Device_Handle_t hDev)
```

• Parameters

[in] hDev: Handle to an open device.

• Return value

FZ_Success, At least one frame can be read from the given device.
Refer to the FZ_Result declaration for other values which indicate failure.

2.10 FZ_SetLogging

• Synopsis

Enables or disables different kinds of API logging. Can be useful during debugging.

• Syntax

```
FZ_Result FZ_SetLogging(
           int iFlags,
           const char *szFilename
           FZ_LOGCALLBACKTYPE pFunction = NULL)
```

• Parameters

[in] iFlags: Sets output type and log level. See all "FZ_LOG_. . ." in fz_api.h for valid flags. Any combination of flags can be set together by using the binary or operator "|".
[in] szFilename: Set directory and file name of text file to log to. Used only for flag FZ_LOG_TO_FILE, can be NULL otherwise.
[in] pFunction: Callback function that receives all log entries.

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

2.11 FZ_OpenFrameChannel

• Synopsis

Opens a local TCP/IP network frame channel to be able to share received frames between processes. Open is done either as send or receive.

• Syntax

```
FZ_Result FZ_OpenFrameChannel(
           int iChannel,
           int iFlags)
```

• Parameters

[in] `iChannel`: Number of the wanted channel.
[in] `iFlags`: Sets the function behavior. See all "`FZ_FRAME_CHAN_...`" in fz_api.h for valid flags. Only one may be specified.

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

• Notes

- For a send-channel the function is blocking with a 5 sec timeout.
- For a recv-channel the function is blocking with a 10 sec timeout.

2.12 FZ_CloseFrameChannel

• Synopsis

Closes an open frame channel.

• Syntax

```
FZ_Result FZ_CloseFrameChannel(
           int iChannel)
```

• Parameters

[in] `iChannel`: Number of an open frame channel.

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

2.13 FZ_SendFrameToChannel

• Synopsis

Sends a frame on an open frame channel.

• Syntax

```
FZ_Result FZ_SendFrameToChannel(
           int iChannel,
           FZ_FRAME_HEADER *pHeader,
           void *pPixels)
```

• Parameters

[in] `iChannel`: Number of an open send-channel.
[in] `pHeader`: The header data.
[in] `pPixels`: The pixel data.

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

2.14 FZ_GetFrameFromChannel

• Synopsis

Receives a frame on an open frame channel.

• Syntax

```
FZ_Result FZ_GetFrameFromChannel(
           int iChannel,
           FZ_FRAME_HEADER *pHeader,
           void *pPixels,
           size_t *piPixelsByteLen)
```

• Parameters

[in] `iChannel`: Number of an open receive-channel.
[in/out] `pHeader`: Receives the frame header for the image.
[in/out] `pPixels`: Receives the pixels for the image.
[in/out] `piPixelsByteLen`: Contains the size of the memory allocated to `pPixels` on input, and is filled with the number of bytes written to `pPixels` on output.

• Return value

`FZ_Success`, The call was completed successfully.
Refer to the `FZ_Result` declaration for other values which indicate failure.

• Notes

- The function is blocking with a 5 sec timeout.

3. TimeStamp

Cameras with DepthEngine version above 3.2.1 will store the time of capture and capture duration in the frame header. To be able to get accurate timestamp information the camera needs to be connected to a network with internet access so that the camera can synchronize it's time with NTP servers.

The timestamp is stored in the following way in the `uint32_t timestamp[3]` field in the `FZ_FRAME_HEADER` struct.

`timestamp[0]`	Contains start of exposure in POSIX time (seconds)
`timestamp[1]`	Contains the milliseconds part of start of exposure.
`timestamp[2]`	Contains the duration of the complete exposure in milliseconds.

Example, A frame is captured in Temporal mode, 20 fps with a exposure time of 10ms on the 4:th September 2011 22:34.44 and 50 milliseconds.

`timestamp[0]` is **1315175684** (4:th September 2011 22:34.44)
`timestamp[1]` is 50 (milliseconds part)
`timestamp[2]` is the total exposure duration which will be the total time of exposure from first to last image needed to process the frame. In this case it is 70ms.

4. Application development guide

This section provides guidelines to help application developers use the Fotonic camera and the FZ-API software in their applications.

The camera has various parameters that need to be set to appropriate values in accordance with the requirements of the application. Below is a list of some important parameters along with information on how they should be used.

4.1 Shutter time

This parameter is analogous to exposure setting in a regular camera. It determines the total amount of time (in milliseconds) that the light source is turned on during each frame. During this time the shutter of the sensor is also turned on such that the reflected light can be sensed by the pixels. The longer the shutter time, the more light gets projected to the scene. Therefore, the application developers should set the shutter time based on their light power requirements. As an example, for a close range application, e.g. gesture recognition, a relatively small shutter time (4-16 milliseconds) is usually sufficient. Setting the shutter time to large values may cause some pixels to saturate, especially if the scene includes objects that are either highly reflective or are very close to the camera.

For long range applications, we recommend using longer shutter times, perhaps in the range of 16-40 milliseconds. For smaller shutter times, far objects in the field of view may not reflect enough light, and therefore may not appear in the range images.

The shutter time and the frame rate are closely-related parameters. As the shutter time is increased, the frame rate has to (and will automatically) be reduced if the current frame rate cannot be sustained at the given shutter speed.

Shutter on the Panasonic sensor works the same way, except the range is 0.1 to 0.8 ms.

4.2 Frame rate

Certain applications may require a high frame rate, for example to capture moving objects. The sensor supports frame rate that is a function of the shutter time, speed of the camera and the host processor. Please refer to the release notes for more information on maximum frame rate. However, the frame rate has to be reduced as the shutter time is increased. The application developers should consider the tradeoff between the frame rate and shutter time, and set these values based on the requirements of their applications.

Frame rate on the Panasonic sensor works the same way, but there are only normal and fast rates. 30-33 in normal, and 51-57 in fast, depending on shutter. The threshold is 35 for normal/fast.

4.3 Increasing resolution

In order to increase the resolution of the <XYZ> data, the application can use either temporal or spatial or both types of averaging methods. The temporal averaging uses the values of the same pixel from a running set of successive frames. The spatial averaging combines the values of several pixels in a small locality of the target. A combination of both averaging methods can be used to obtain other application-specific effects.

N/A on the Panasonic sensor.

4.4 Pixel saturation condition

If a pixel saturates because it receives too much light (from any light source) then the <XYZ> measurement would not be correct. There are several methods to avoid or deal with this issue. The <XYZ> filters can be used to identify and screen the saturated pixels. The application can check the brightness pixel values and decide on the confidence level of the corresponding <XYZ> value. The application can reduce the light power or reduce the shutter time if supported.

The application may use the value of other neighboring non-saturated pixels to correct the saturated pixels. Or use any combination of these strategies to avoid or deal with pixel saturation.