FOTONIC

FZ-INTERNAL-API

Rev B
22/03/2013

1. Introduction to FZ-API

This document describes the FZ-INTERNAL-API used to access the Fotonic cameras from embedded applications.

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.

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 header files and libraries.

1.1 Embedded development

The camera is running Linaro Linux Distribution. To develop embedded application a host computer running Linux or Windows must be used.

The FZ INTERNAL SDK package contains installation packaged for ARM tool chain, Eclipse IDE and source code for a sample application.

The internal API is mostly compatible with the Windows/Linux x86/x64 APIs, so code can be reused as much as possible between platforms.

1.2 Programming model

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

- Establish connection to 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.

1.3 Includes

Include header files are

`fz_api.h`	contains all functions and data structures for the API. The header will in turn include `fz_commands.h`.

`fz_commands.h`	contains definitions for all commands and values to pass to the `FZ_Ioctl()` function.
`fz_camera.h`	contains camera specific defines and data types that are needed for the sample application and communication with the PC API and sample application.
`DE_Config.h`	contains camera specific definitions used in the sample application.

Include libraries are

- `libInternalAPI.a`

1.4 Sample application

The camera contains a sample application called ClientApplication whose source code is provided with the FZ INTERNAL SDK package.

This application can either be replaced with your own application or you can chose to modify it and add your own functionality and then keep the existing Ethernet communication protocol to the PC.

2. API Documentation

This describes the interface in the camera to get in control of the sensor. The main purpose of this API is to program the sensor to return brightness, and depth images with desired properties. This API is very similar to the Fotonic PC API to make it easy to port your PC software to embedded applications.

The API contains the following methods.

2.1 FZ_Open

• Synopsis

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

• Syntax

```
FZ_Result FZ_Open(
           FZ_Device_Handle_t* phDev,
           int iFlags)
```

• Parameters

[in/out] `phDev`: Receives a handle to the device.
[in] `iFlags`: Set 0.

• Return value

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

2.2 FZ_Close

• Synopsis

Closes the link between the sensor 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.

2.3 FZ_Ioctl

• Synopsis

Sends an IO Control code to the sensor to set or get parameters. This command is used to start and stop a sensor, and set parameters. It is also used to query the values of parameters of the sensor.

• 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`: Pointer to the handle returned from open.
[in] `iCmd`: IO Control code. See FZ_API.pdf.
[in] `pParam`: Pointer to the input parameter. (Input value)
[in] `iCmdByteLen`: Size of the input parameter.
[in/out] `piRespCode`: Receives the response code (typically CMD_DE_ACK or 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.

• Explanation of IO Control codes

See FZ-API.pdf

2.4 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_EXT *pHeader,
           void *pPixels, size_t *piPixelsByteLen)

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

• Parameters

[in] `hDev`: Pointer to the handle returned from open.
[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 contain brightness, Z, X and Y coordinates for each pixel of the frame. The frame is organized on a per-plane per-scanline (row) basis:

$B_0..B_N$	$Z_0..Z_N$	$X_0..X_N$	$Y_0..Y_N$
↕			
$B_0..B_N$	$Z_0..Z_N$	$X_0..X_N$	$Y_0..Y_N$

Where [0..N] is the number of columns, currently this is 160 or 320 for Panasonic 320 modes. The number of rows is 120, and not currently programmable. Each element is a 16-bit signed integer value.

Z, X, Y values are in millimeter.

The format of `pHeader` is:

```
struct FZ_FRAME_HEADER_EXT
{
int16_t magic_number;                mark the start of this frame
int8_t version;                      frame version
uint32_t ExpStartSec;                global time start of exposure in seconds
uint16_t ExpStartMSec;           global time start of exposure millisecond part
uint16_t bytesperpixel;          number of bytes in a pixel
uint16_t nrows;                  image height
uint16_t ncols;                  image width
uint32_t processedframecounter;  processed frame number
uint32_t shippedframecounter;    shipped frame number
uint16_t ExpDuration;            time in milliseconds from start to end of exposure
int32_t temperature;             chip temperature (10*celcius)
uint32_t captureid;              current frame in the order (useful for raw mode)
uint32_t shutter;                shutter time (10*ms)
int32_t cmr;
uint32_t reportedframerate;      camera frame rate
uint32_t frequency;              light pulse frequency
uint32_t mode;                   camera operation mode
uint32_t measuredframerate;      frame rate messured by pc
uint32_t lasterrorframe;         last processedframecounter with an error
uint16_t dll1;
uint16_t dll2;
uint16_t dllerr;
};
```

• Notes

- The function blocks if no frame is currently available. Use `FZ_FrameAvailable` if blocking is not desired.
- The API has an internal frame queue with space for 5 frames, the oldest is read first when more than one frame is available. When the queue becomes full the oldest frame is discarded.
- Use `FZ_GetFrameNewest` if the newest frame is wanted and the frame queuing feature is not needed.

2.5 FZ_FrameAvailable

• Synopsis

Use this to 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`: Pointer to the handle returned from open.

• 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.

3. Application development guide

See FZ-API.pdf.