DUO API

Overview

DUO Device API - Sensor Access & Control

Part of the DUO SDK, The DUO API provides low level access to device details, controls and configuration with the use of supported methods and parameters. This document outlines all functionality of the DUO API via the `DUOLib` library which provides a `C` interface paired with C++/C#/OpenCV/Qt Samples.

Here is a overview of the key methods and structures available in the API.

Control/Callback Methods

```
// Device Control
bool OpenDUO(DUOInstance *duo);
bool CloseDUO(DUOInstance *duo);
bool StartDUO(DUOInstance duo, DUOFrameCallback frameCallback, void *pUserData, bool masterMode = tr
ue);
bool StopDUO(DUOInstance duo);

// Device Information
int EnumerateResolutions(DUOResolutionInfo *resList,
                         int32_t resListSize,
                         int32_t width,
                         int32_t height,
                         int32_t binning,
                         float fps);

// Frame Callback
void (CALLBACK *DUOFrameCallback)(const PDUOFrame pFrameData, void *pUserData);
```

Parameters Methods

```
// Get DUO Parameters
bool GetDUODeviceName(duo, char *val);
bool GetDUOSerialNumber(duo, char *val);
bool GetDUOFirmwareVersion(duo, char *val);
bool GetDUOFirmwareBuild(duo, char *val);
bool GetDUOResolutionInfo(duo, DUOResolutionInfo &resList);
bool GetDUOExposure(duo, double *val);
bool GetDUOExposureMS(duo, double *val);
bool GetDUOGain(duo, double *val);
bool GetDUOHFlip(duo, int *val);
bool GetDUOVFlip(duo, int *val);
bool GetDUOCameraSwap(duo, int *val);
bool GetDUOLedPWM(duo, (double *val);
bool GetDUOFrameDimension(duo, uint32_t *w,  uint32_t *h);

// Get Calibration Parameters
bool GetDUOCalibrationPresent(duo);
bool GetDUOFOV(duo, DUOResolutionInfo &resList, double *val);
bool GetDUOUndistort(duo, int *val);
bool GetDUOIntrinsics(duo, doubl *val);
bool GetDUOExtrinsics(duo, double *val);

// Set DUO Parameters
bool SetDUOExposure(duo, double val);
bool SetDUOExposureMS(duo, double val);
bool SetDUOHFlip(duo, int val);
bool SetDUOResolutionInfo(duo, DUOResolutionInfo &resList);
bool SetDUOExposure(duo, double val);
bool SetDUOExposureMS(duo, double val);
bool SetDUOGain(duo, double val);
bool SetDUOHFlip(duo, int val);
bool SetDUOVFlip(duo, int val);
bool SetDUOCameraSwap(duo, int val);
bool SetDUOLedPWM(duo, double val);
bool SetDUOLedPWMSeq(duo, PDUOLEDSeq val, uint32_t size);
bool SetDUOUndistort(duo, int val);
```

DUOInstance

The DUO device instance is a DUO handle filled by the `OpenDUO()` and used in subsequent API calls.

```
void *DUOInstance
```

Methods

GetLibVersion

Retrieves the library version as a string.

```
char *GetLibVersion()
```

EnumerateResolutions

Enumerates supported resolutions. Requires an allocated DUOResolutionInfo structure and resListSize parameter indicating number of allocated elements in resList array. Enumeration can be filtered/restricted by fixing one or more parameters such as width, height, binning or fps. The function returns number of resolutions found.

```
int EnumerateResolutions(DUOResolutionInfo *resList,
                         int32_t resListSize,
                         int32_t width,
                         int32_t height,
                         int32_t binning,
                         float fps)
```

DUOFrameCallback

Called repeatedly by the DUOLib on successful fame capture. Returns PDUOFrame data with any specific user data (pUserData) passed to StartDUO function.

```
void (CALLBACK *DUOFrameCallback)(const PDUOFrame pFrameData, void *pUserData)
```

OpenDUO

Opens the DUO device and fills the pointer with the `DUOInstance` handle. The function returns `true` on success. This function must be called before using any API functions that require `DUOInstance` parameter. (All subsequent calls use DUOInstance to access the device.)

```
bool OpenDUO(DUOInstance *duo)
```

CloseDUO

Closes the DUO device. The function returns `true` on success.

```
bool CloseDUO(DUOInstance duo)
```

StartDUO

Starts capturing frames. Requires a frameCallback pointer to user defined DUOFrameCallback callback function. The frameCallback parameter can be `NULL` . The pUserData is any user data that is passed to the callback function. The function returns `true` on success.

```
bool StartDUO(DUOInstance duo, DUOFrameCallback frameCallback, void *pUserData, bool masterMode = tr
ue)
```

StopDUO

Stops capturing frames. The function returns `true` on success.

```
bool StopDUO(DUOInstance duo)
```

Parameters

Get Parameter Functions

Used to get various parameter values from the DUO device.

GetDUODeviceName

Fills the user allocated char array pointer with human-readable device name. The array size should be 260 bytes. The function returns `true` on success.

```
bool GetDUODeviceName(DUOInstance, char *val)
```

GetDUOSerialNumber

Fills the user allocated char array pointer with serial number. This array size should be 260 bytes. The function returns `true` on success.

```
bool GetDUOSerialNumber(DUOInstance, char *val)
```

GetDUOFirmwareVersion

Fills the user allocated char array pointer with the firmware version of the device. Array size should be 260 bytes. The function returns `true` on success.

```
bool GetDUOFirmwareVersion(DUOInstance, char *val)
```

GetDUOFirmwareBuild

Fills the user allocated char array pointer with the firmware build of the device. The array size should be 260 bytes. The function returns `true` on success.

```
bool GetDUOFirmwareBuild(DUOInstance, char *val)
```

GetDUOResolutionInfo

Fills the user supplied `DUOResolutionInfo` variable with the currently selected resolution info. The function returns `true` on success.

```
bool GetDUOResolutionInfo(DUOInstance, DUOResolutionInfo &resList)
```

GetDUOExposure

Fills the user supplied double variable with current exposure value in percentage (range [0,100]). The function returns `true` on success.

```
bool GetDUOExposure(DUOInstance, double *val)
```

GetDUOExposureMS

Fills the user supplied double variable with current exposure value in milliseconds. The function returns `true` on success.

```
bool GetDUOExposureMS(DUOInstance, double *val)
```

GetDUOGain

Fills the user supplied double variable with current gain value in percentage (range [0,100]). The function returns `true` on success.

```
bool GetDUOGain(DUOInstance, double *val)
```

GetDUOHFlip

Fills the user supplied int variable with current horizontal flip value (range [0,1]). The function returns `true` on success.

```
bool GetDUOHFlip(DUOInstance, int *val)
```

GetDUOVFlip

Fills the user supplied int variable with current vertical flip value (range [0,1]). The function returns `true` on success.

```
bool GetDUOVFlip(DUOInstance, int *val)
```

GetDUOCameraSwap

Fills the user supplied int variable with current camera swap value (range [0,1]). The function returns `true` on success.

```
bool GetDUOCameraSwap(DUOInstance, int *val)
```

GetDUOLedPWM

Fills the user supplied double variable with current LED PWM value in percentage (range [0,100]). The function returns `true` on success.

```
bool GetDUOLedPWM(DUOInstance, double *val)
```

GetDUOFrameDimension

Fills the user supplied int variables with current width and height of the image. The function returns `true` on success.

```
bool GetDUOFrameDimension(DUOInstance, uint32_t *w, uint32_t *h)
```

GetDUOCalibrationPresent

Checks the user supplied `DUOInstance` variable to see if CalibrationData is available on the device. The function returns `true` on success.

```
bool GetDUOCalibrationPresent(DUOInstance)
```

GetDUOFOV

Accepts the user supplied `DUOResolutionInfo` and fills the double variable with FOV of selected resolution. The function returns `true` on success.

```
bool GetDUOFOV(DUOInstance, DUOResolutionInfo &resList, double *val)
```

GetDUOUndistort

Fills the user supplied boolean variable with status of undistortion operation. The function returns `true` on success.

```
bool GetDUOUndistort(DUOInstance, int *val)
```

GetDUOIntrinsics

Fills the user supplied INTRINSICS structure with the calibration intrinsics data. The function returns `true` on success.

```
typedef struct
{
    unsigned short w, h;
    double left[12];
    double right[12];
}INTRINSICS;

bool GetDUOIntrinsics(DUOInstance, INTRINSICS *val)
```

GetDUOExtrinsics

Fills the user supplied EXTRINSICS structure with the calibration extrinsics data. The function returns `true` on success.

```
typedef struct
{
    double rotation[9];
    double translation[3];
}EXTRINSICS;

bool GetDUOExtrinsics(DUOInstance, EXTRINSICS *val)
```

Set Parameter Functions

Used to set various parameter values for the DUO device.

SetDUOExposure

Sets exposure value to the user supplied double value in percentage (range [0,100]). The function returns `true` on success.

```
bool SetDUOExposure(DUOInstance, double val)
```

SetDUOExposureMS

Sets exposure value to the user supplied double value in milliseconds. The function returns `true` on success.

```
bool SetDUOExposureMS(DUOInstance, double val)
```

SetDUOGain

Sets gain value to the user supplied double value in percentage (range [0,100]). The function returns `true` on success.

```
bool SetDUOGain(DUOInstance, double val)
```

SetDUOHFlip

Sets horizontal flip value to the user supplied int value (range [0,1]). The function returns `true` on success.

```
bool SetDUOHFlip(DUOInstance, int val)
```

SetDUOVFlip

Sets vertical flip value to the user supplied int value (range [0,1]). The function returns `true` on success.

```
bool SetDUOVFlip(DUOInstance, int val)
```

SetDUOCameraSwap

Sets camera swap value to the user supplied int value (range [0,1]). The function returns `true` on success.

```
bool SetDUOCameraSwap(DUOInstance, int val)
```

SetDUOLedPWM

Sets LED PWM value to the user supplied double value in percentage (range [0,100]). The function returns `true` on success.

```
bool SetDUOLedPWM(DUOInstance, double val)
```

SetDUOLedPWMSeq

Sets LED PWM sequence to the user supplied `DUOLEDSeq` array. The maximum size of the sequence is 64 steps. The user must supply the desired number of steps via `size` parameter. This function can only be called while the DUO is not capturing (i.e. before StartDUO call). The function returns `true` on success.

```
bool SetDUOLedPWMSeq(DUOInstance, PDUOLEDSeq val, uint32_t size)
```

SetDUOResolutionInfo

Sets the desired resolution, binning and the frame rate. The user supplied `DUOResolutionInfo` parameter is obtained by calling EnumerateResolutions function. The function returns `true` on success.

```
bool SetDUOResolutionInfo(DUOInstance, DUOResolutionInfo &resList)
```

SetDUOUndistort

Sets the undistortion operation to the user supplied boolean value (range [true, false]). The function returns `true` on success.

```
bool SetDUOUndistort(DUOInstance, int val)
```

Parameter Values/Structures

DUOParameter

Used **internally** to identify device parameters, should be used only through defined parameter methods above.

```
enum DUOParameter
{
    // DUO Parameter Units
    DUO_PERCENTAGE,
    DUO_MILLISECONDS,

    // DUO Camera Parameters
    DUO_DEVICE_NAME,               // Get only: (string allocated by user min size 252 bytes)
    DUO_SERIAL_NUMBER,             // Get only: (string allocated by user min size 252 bytes)
    DUO_FIRMWARE_VERSION,          // Get only: (string allocated by user min size 252 bytes)
    DUO_FIRMWARE_BUILD,            // Get only: (string allocated by user min size 252 bytes)
    DUO_RESOLUTION_INFO,           // Set/Get:  (PDUOResolutionInfo) - must be first parameter to s
et
    DUO_FRAME_DIMENSION,           // Get only: (uint32_t, uint32_t)
    DUO_EXPOSURE,                  // Set/Get:  (double [0,100], DUO_PERCENTAGE) or (double in mill
iseconds, DUO_MILLISECONDS)
    DUO_GAIN,                      // Set/Get:  (double [0,100])
    DUO_HFLIP,                     // Set/Get:  (bool [false,true])
    DUO_VFLIP,                     // Set/Get:  (bool [false,true])
    DUO_SWAP_CAMERAS,              // Set/Get:  (bool [false,true])

    // DUO LED Control Parameters
    DUO_LED_PWM,                   // Set/Get:  (double [0,100])
    DUO_LED_PWM_SEQ,               // Set only: (PDUOLEDSeq, int) - number of LED sequence steps (m
ax 64)

    // DUO Calibration Parameters
    DUO_CALIBRATION_PRESENT,       // Get Only: return true if calibration data is present
    DUO_FOV,                       // Get Only: (PDUOResolutionInfo, double* (leftHFOV, leftVFOV, r
ightHFOV, rightVFOV)
    DUO_UNDISTORT,                 // Set/Get:  (bool [false,true])
    DUO_INRINSICS,                 // Get Only: (pointer to ushort16_t resolution w, h, intrinsics
2*12 double parameters)
    DUO_EXTRINSICS,                // Get Only: (pointer to extrinsics rotation matrix 9 double, tr
anslation vector 3 double)
};
```

DUOBinning

Contains binning options used in resolution configuration. (1x1, 1x2, 1x4, 2x1, 2x2, 2x4)

```
DUO_BIN_ANY = -1,               // Any binning mode available
DUO_BIN_NONE = 0,               // No horizontal or vertical binning
DUO_BIN_HORIZONTAL2 = 1,        // Horizontal binning by factor of 2
DUO_BIN_HORIZONTAL4 = 2,        // Horizontal binning by factor of 4
DUO_BIN_VERTICAL2 = 4,          // Vertical binning by factor of 2
DUO_BIN_VERTICAL4 = 8           // Vertical binning by factor of 4
```

DUOFrame

DUOFrame structure holds and describes the sensor data that is passed to user via DUOFrameCallback function.

```
uint32_t width;                 // DUO frame width
uint32_t height;                // DUO frame height
uint8_t ledSeqTag;              // DUO frame LED tag
uint32_t timeStamp;             // DUO frame time stamp in 100us increments
uint8_t* leftData;              // DUO left frame image data (aligned to 64 byte boundary)
uint8_t* rightData;             // DUO right frame image data (aligned to 64 byte boundary)
uint8_t accelerometerPresent;   // True if accelerometer chip is present
float accelData[3];             // DUO accelerometer data (x, y, z)
float gyroData[3];              // DUO gyroscope data (x, y, z)
float magData[3];               // DUO magnetometer data (x, y, z)
float tempData;                 // DUO temperature data in degrees °C
```

DUOResolutionInfo

DUOResolutionInfo structure describes DUO frame size, binning and frame-rate. This structure is populated by the EnumerateResolutions function.

```
int   width;                // Desired DUO frame width or -1 for any
int   height;               // Desired DUO frame height or -1 for any
int   binning;              // Horz/Vertical Binning od BIN_ANY for any
float fps;                  // Desired Framerate or -1 for any
float minFps;               // Minimum Framerate for this resolution
float maxFps;               // Maximum Framerate for this resolution
```

DUOLEDSeq

DUOLEDSeq structure describes the individual LED power values. The ledPwmValue[0] applies to the leftmost LED, ledPwmValue[1] applies to center LED and ledPwmValue[2] applies to right-most LED on the DUO.

```
    uint8_t ledPwmValue[4];       // LED PWM values are in percentage [0,100]
```

Sample Usage

You can easily integrate the DUO into application or system by using a portable `c` interface that allows for full access and control to the DUO device. Here is a quick summary of the steps for using the API:

- **Step 1** - Include the DUOLib headers and link against the library.
- **Step 2** - Get available resolutions via EnumerateResolutions.
- **Step 3** - Open and retrieve a DUO instance by calling OpenDUO function.
- **Step 4** - Implement a DUOFrameCallback function to be able to retrieve the DUOFrame data.
- **Step 5** - Set the desired resolution, you frame callback function and any user data by calling StartDUO function.
- **Step 6** - Process DUO frame data that is repeatedly passed to you via DUOFrameCallback function.
- **Step 7** - Stop capture by calling StopDUO function.
- **Step 8** - Close the DUO by calling CloseDUO function.

The code snippets below demonstrate each step in more detail:

Include/Link

```
#include "../include/DUOLib.h"              // Include DUO API header file
#pragma comment(lib, "../lib/DUOLib.lib")   // Link against DUOLib
```

Configure/Start Device

```
DUOResolutionInfo ri;
// Select 320x240 resolution with 2x2 binning capturing at 30FPS
if(EnumerateResolutions(&ri, 1, 320, 240, DUO_BIN_HORIZONTAL2+DUO_BIN_VERTICAL2, 30))
{
    DUOInstance duo;    // Creates instance of DUO
    if(OpenDUO(&duo))    // Open DUO
    {
        // Set selected resolution
        SetDUOResolutionInfo(duo, ri);
        // Start capture and pass DUOCallback function that will be called on every frame captured
        if(StartDUO(duo, DUOCallback, NULL))
        {
            ...
        }
    }
}
```

Implement Callback

```
void CALLBACK DUOCallback(const PDUOFrame pFrameData, void *pUserData)
{
    ...
}
```

Close/Shutdown

```
    StopDUO(duo);     // Stop capture
    CloseDUO(duo);    // Close DUO
```

Get Device Info

```
char tmp[260];
GetDUODeviceName(duo, tmp);
printf("DUO Device Name:      '%s'\n", tmp);
GetDUODeviceName(duo, tmp);
printf("DUO Serial Number:    %s\n", tmp);
GetDUOFirmwareVersion(duo, tmp);
printf("DUO Firmware Version: v%s\n", tmp);
GetDUOFirmwareBuild(duo, tmp);
printf("DUO Firmware Build:   %s\n", tmp);
```

Resources

Samples

Review the samples provided with the SDK:

- Capturing Motion Data
- Capturing Image Data
- Configuring Parameters
- Configuring LED Sequences
- Capture frames using polling mechanism
- Capture frames using polling mechanism (OpenCV)
- Using the Dense3D API to process depth map (Dense3D)

Tips

- Most API calls return a true or false if the action was successful.
- Make sure you USB Hub meets requirments stated in the product datasheet.
- Always make sure you have the latest DUO SDK, Dashboard and Driver.

Related

Relevant articles and links:

- DUO SDK
- DUO Dense3D
- DUO Developers
- DUO Devices
- DUO Downloads