O3D3xx Camera
Programming- and Reference-Manual

Version 0.1.134039 beta

Technical information subject to change without notice.
This document may also be changed without notice.
Revision: 134039
Created: 30.07.2014
Changed: 08.04.2015

ifm syntron

Contents

ifm syntron

Part 1

Introduction

1 Disclaimer of warranties

ifm disclaims to the fullest extent authorized by law any and all warranties, whether express or implied, including, without limitation, any implied warranties of title, non-infringement, quiet enjoyment, integration, merchantability or fitness for a particular purpose.

Without limitation of the foregoing, ifm expressly does not warrant that:

1. the software will meet your requirements or expectations;
2. the software or the software content will be free of bugs, errors, viruses or other defects;
3. any results, output, or data provided through or generated by the software will be accurate, up-to-date, complete or reliable;
4. the software will be compatible with third party software
5. any errors in the software will be corrected

DEMO SOFTWARE AND TEMPLATES

Demo software and templates are provided “as is” and “as available”, without any warranty of any kind, either express or implied. The User acknowledges and agrees to use the software at User’s own risk. In no event shall ifm be held liable for any direct, indirect, incidental or consequential damages arising out of the use of or inability to use the software. User may use the software solely for demonstration purposes and to assess the software functionalities and capabilities.

2 Introduction

This document serves as user manual, programming guide and reference manual for application development with O3D3xx cameras by ifm gmbh. These devices use the range imaging technology from pmdtechnologies gmbh (pmd[vision]® Time-of-Flight devices). The document is split in two parts, a generic programming guide and a reference manual. In part I, the programming manual, the PMD software development kit (PMDSDK2) is introduced and everything that is needed to start developing applications with O3D3xx cameras is explained. Part II, the reference manual, documents in detail the complete application programming interface (API) of the PMDSDK2 and provides information about the specific properties of the reference design. These specific properties include information such as value ranges for certain parameters like integration time and individual commands.

This part of the manual describes the API (application programming interface) for O3D3xx cameras. It is possible to set parameters of the camera, as well as retrieve 2D and 3D images from the data source. This allows simple and fast application development supporting O3D3xx cameras. All functionality is provided through an API for the C programming language.

2.1 System requirements

Some requirements have to be met for the host system to develop applications using the PMDSDK2.

For Windows:

- Microsoft Windows XP/Vista/7/8
- Microsoft Visual Studio .net 2003/2005/2008/2010/2012

Other Compilers/IDEs have been reported to work with the PMDSDK2 as well.

3 Building Applications

3.1 PMDSDK2 plugins

The PMDSDK2 uses plugins to connect to different devices (or other data sources) and to do the processing that is needed to generate distances and other types of data. Plugins are libraries which encapsulate hardware communication and processing algorithms. Plugins should neither be used directly by the programmer nor should the application binary link to them. Plugins should always be accessed using the provided SDK functions to ensure highest stability and flexibility.

Plugins have filename extensions of the form .X##.ext, where X describes the system, ## the bitness of the system and ext the type of plugin used. Possible systems are (**W**=Windows, **L**=Linux, **A**=Apple), bitness will result either in **32** or **64** and ext may be one of **pap**, **pcp**, **ppp**.

Note:

- Not all plugins are available for all possible combinations of system and bitness.
- O3D3xx camera plugins are available only for Windows platform.

3.2 Building applications for Microsoft Windows

To create Windows applications using the PMDSDK2, several files from the PMDSDK are needed.

- pmdsdk2.h: This file contains the declarations of all available PMDSDK2 functions. This must be included in the application source code
- pmdaccess2.lib: The import library to be linked to the application.
- pmdaccess2.dll: This is the library itself.
- pmdsdk2common.h: This file is automatically included from pmdsdk2.h
- pmddatadescription.h: This file is automatically included from pmdsdk2.h
- A source plugin file (∗.W32.pap)
- A processing plugin file (∗.W32.ppp)

4 General API description

All functions in the PMDSDK2 are prefixed by the letters pmd, followed by a descriptive command identifier. For example, the command to get the distance data from the image sensor is pmdGetDistances().

All commands require a handle of a camera connection. This handle is created when connecting to the camera and is disposed upon disconnection. Also, every command returns a code stating the success of its execution (PMD_OK) or giving an error code upon failure. See the reference section for information about the available return codes.

Almost all functions in the PMDSDK2 look like this:

```
int pmdFunctionName(PMDHandle hnd, Type1 * result, Type2 param1, Type2 param2);
```

The first parameter is always the handle. Then follow parameters that will be modified by the function (call-by-reference). At the end there are parameters that will not be changed.

The following source code shows a very simple program using the PMDSDK2. The program attempts to connect to a pmd device using the plugins called a.W32.pap and b.W32.ppp and retrieve the number of pixel columns in the camera data. For more information on plugins, see the next section.

```c
#include <stdio.h>
#include "pmdsdk2.h"

int main (void)
{
    PMDHandle hnd;      // connection handle
    int res;
    PMDDataDescription dd;

    // connect to camera
    res = pmdOpen (&hnd, "a", "", "b", "");

    if (res != PMD_OK) {
        printf ("Could not connect\n");
        return 1;
    }

    res = pmdUpdate (hnd);

    if (res != PMD_OK) {
        printf ("Could not retrieve data\n");
        pmdClose (hnd);
        return 1;
    }

    res = pmdGetSourceDataDescription (hnd, &dd);

    if (res != PMD_OK) {
        printf ("Could not retrieve sensor width\n");
        pmdClose (hnd);
        return 2;
    }

    printf ("Sensor width: %d\n", dd.img.numColumns);

    pmdClose (hnd);

    return 0;
}
```

Some functions use output parameters of variable length, like image data or text messages. These functions will always take the maximum length in bytes as an additional parameter to prevent buffer overflows. If not enough memory is supplied for the data, the behaviour is as follows:

- Strings will be truncated to fit the available memory.
- Other commands will fail with an error code.

4.1 Features which do not apply to all pmd reference designs

The remaining part of section 2 describes the generic API of the PMDSDK2. **Some of the functions provided by the SDK may not apply to your pmd device**. Using these functions is not recommended, they are documented here for the sake of completeness. Please refer to chapter Plugin Documentation in the reference manual to see what functions from the PMDSDK2 are supported by your sensor.

5 Connection and processing with plugins

The first thing a program using the PMDSDK2 has to do is to use the function pmdOpen() to initialize a handle for the communication and load two plugins (a source plugin for accessing the device and a processing plugin for the calculation). Each plugin can take a string parameter for initialization. After this initialization, other functions can be used in conjunction with the handle. The last thing to do is to close the connection and unload the plugins with pmdClose().

In the following example, there is a source plugin called O3D3xxCamera.W32.pap and a processing plugin called O3D3xxProc.W32.ppp. The source plugin is initialized without parameters, while the processing plugin takes an initialization string.

The program loads both plugins, configures the integration time and immediately disconnects again.

```
#include <stdio.h>
#include "pmdsdk2.h"

int main (void)
{
    PMDHandle hnd;       // connection handle
    int res;

    // connect to camera
    res = pmdOpen (&hnd,
                   "O3D3xxCamera", "192.168.0.69:80:50010",
                   "O3D3xxProc", "offset=10:mult=1.0");

    if (res != PMD_OK) {
        printf ("Could not connect\n");
        return 1;
    }
    res = pmdSetIntegrationTime (hnd, 0, 1000);

    if (res != PMD_OK) {
        printf ("Could not set integration time\n");
    }

    pmdClose (hnd);

    return 0;
}
```

5.1 Separate plugin instantiation

In some rare cases, it can be desirable not to load a source plugin and a processing plugin at the same time. Instead of pmdOpen, which opens both a source plugin and a processing plugin, the functions pmdOpenSourcePlugin and pmdOpenProcessingPlugin can be called to open only one kind of plugin. Some of the functions in the PMDSDK2 will only work when a processing plugin is loaded, others rely on the availability of a source plugin and some need both at the same time. Other than that, the behaviour is basically the same.

Here is the example from above with only a source plugin. Only one statement has been changed (from pmdOpen to pmdOpenSourcePlugin), because there are no functions that actually need a processing plugin in this example.

```
#include <stdio.h>
#include "pmdsdk2.h"

int main (void)
{
    PMDHandle hnd;      // connection handle
    int res;

    // connect to camera without a processing plugin
    res = pmdOpenSourcePlugin (&hnd, "O3D3xxCamera", "192.168.0.69:80:50010");

    if (res != PMD_OK) {
        printf ("Could not connect\n");
        return 1;
    }
    res = pmdSetIntegrationTime (hnd, 0, 1000);

    if (res != PMD_OK) {
        printf ("Could not set integration time\n");
    }

    pmdClose (hnd);

    return 0;
}
```

The following functions need a source plugin:

- pmdUpdate()
- pmdGetSourceData()
- pmdGetSourceDataDescription()
- pmdGetSourceDataSize()
- pmdSetIntegrationTime()
- pmdGetIntegrationTime()

These functions need a processing plugin:

- pmdCalcDistances()
- pmdCalcAmplitudes()
- pmdCalcFlags()
- pmdCalc3DCoordinates()

The following functions need both plugins on the same PMDHandle:

- pmdGetDistances()

- pmdGetAmplitudes()
- pmdGetFlags()
- pmdGet3DCoordinates()

For a detailed description of these functions, see the following chapters and the reference in the appendix.

6 Retrieving image data

The PMDSDK2 provides several commands to retrieve image information from the camera. These commands are:

- pmdGetDistances(PMDHandle hnd, float * data, size_t size)
- pmdGetAmplitudes(PMDHandle hnd, float * data, size_t size)
- pmdGetSourceData(PMDHandle hnd, void * data, size_t size)
- pmdGet3DCoordinates(PMDHandle hnd, float * data, size_t size)
- pmdGetFlags(PMDHandle hnd, unsigned * data, size_t size)

All of these functions take three parameters. The first one is the handle of the connection that was initialized with a call to pmdOpen(). The second one is a pointer to a block of memory. The third one is the size of the memory block (to prevent buffer overruns). If the call succeeds, this memory block will contain the requested data. The data type of the data depends on the function.

pmdGetSourceData()

will return the source data (e.g. the processed images like distance, amplitude, X, Y, Z coordinate and flags images) of the camera. The size of the data block depends on the type of the sensor. pmdGetSourceDataSize() can determine this. The PMDDataDescription structure that can be obtained through pmdGetSourceDataDescription() includes, among other information, also the size of the data. The other functions use values of the type float. There will always be one value for each pixel, except for pmdGet3DCoordinates, which will return three values for each pixel.

pmdGetDistances()

will return a matrix of distance values in meters. The fields contain the distance between the PMD camera and the object (or part thereof) that the respective pixel observes.

pmdGetAmplitudes()

will return the signal strength of the active illumination. This can be used to determine the quality of the distance value. Very low amplitudes indicate a low accuracy of the measured distance in a pixel.

pmdGet3DCoordinates()

will return the range data in cartesian coordinates. Not all cameras or source plugins support this. Upon failure, it will return a value other than PMD_OK. Remember that the coordinate of every pixel is described by 3 values (X, Y and Z), thus the data size is three times the data size of distance values, for example. The x,y,z data is stored pixelwise within a float array (x1 y1 z1 x2 y2 z2 ...).

pmdGetFlags()

will return an "image" with a 32 bit value for each pixel. Each bit in this value contains additional information about the pixel. For example, there is a flag for invalid pixels. If the corresponding bit is set, the distance value of the pixel cannot be trusted to be correct. Other bits might provide information about the reason for the invalidity (e.g. saturation or low signal) or other additional information. To check whether a flag is set, use a bitwise AND operator in conjunction with one of the following identifiers:

- PMD_FLAG_INVALID The pixel's depth value should not be used because it does not represent a reliable distance.
- PMD_FLAG_SATURATED The pixel was overexposed.

It is likely that when one of the other flags is set, PMD_FLAG_INVALID is also set.

Note that not all cameras generate all kinds of flag data. Some cameras are not capable to detect certain kinds of situations.

Before any of those functions can be called, the actual image data must be transferred from the pmd device. This is done by calling pmdUpdate(). Subsequent calls to the above "Get"-functions will produce the same values until pmdUpdate() is called again. This way, calls to two of the above functions (between two pmdUpdate() calls) will always produce data from the same frame.

The following example shows a function which displays the distance of the first pixel on the screen and checks if the distance is valid. Error checking is kept simple (and crude) to keep the example short.

```
void showFirstPixel (PMDHandle hnd)
{
    int res;
    float dat[NUM_OF_PIXELS];
    unsigned flags[NUM_OF_PIXELS];

    res = pmdUpdate (hnd);

    if (res != PMD_OK) exit (3);

    res = pmdGetDistances (hnd, &dat, sizeof(dat));

    if (res != PMD_OK) exit (4);

    printf ("The first pixel measured %f m\n", dat[0]);

    res = pmdGetFlags (hnd, &flags, sizeof(flags));

    if (res != PMD_OK) exit (4);

    if (flags[0] & PMD_FLAG_INVALID)
      {
        printf ("The first pixel is invalid\n");
      }
    else
      {
        printf ("The first pixel is valid\n");
      }
}
```

6.1 Separate calculation of image data

Additionally, the PMDSDK2 provides functions to calculate distances, amplitudes etc. from source data, without the need of an active connection to a data source. Only a processing plugin has to be loaded through pmdOpenProcessingPlugin to use them. These functions are:

- pmdCalcDistances(PMDHandle hnd, float * data, size_t size, struct PMDDataDescription dd, void * sourceData)
- pmdCalcAmplitudes(PMDHandle hnd, float * data, size_t size, struct PMDDataDescription dd, void * sourceData)
- pmdCalc3DCoordinates(PMDHandle hnd, float * data, size_t size, struct PMDDataDescription dd, void * sourceData)
- pmdCalcFlags(PMDHandle hnd, unsigned * data, size_t size, struct PMDDataDescription dd, void * sourceData)

They behave like their respective pmdGet∗ counterparts. The only difference is that they need two additional parameters: The PMDDataDescription structure as retrieved by pmdGetSourceDataDescription() and the actual source data as retrieved by pmdGetSourceData().

These functions are useful to perform offline processing or to use separate threads for data acquisition and calculation in order to improve the frame rate.

7 Configuring the pmd device

There are two important configuration parameters that control the pmd device's image acquisition:

- The integration time.
- The modulation frequency.

Some reference designs use only one integration time and one modulation frequency, others support multiple settings. Some reference designs use fixed settings, others can be configured.
In O3D3xx camera, only integration time setting is configurable. Modulation frequency configuration is not supported.

The PMDSDK2 provides two functions to access integration time setting:

- pmdGetIntegrationTime()
- pmdSetIntegrationTime()

The function below will display the current settings of the first integration time (index 0) on the screen.

```
void showParameters (PMDHandle hnd)
{
    int res;
    unsigned i;

    res = pmdGetIntegrationTime (hnd, &i, 0);

    if (res != PMD_OK) exit (7);

    printf ("Integration time: %d microseconds\n", i);
}
```

8 Retrieving camera information

Several additional pieces of information can be read from the pmd device. The most important ones are the properties of the sensor, like the type of data it produces or its dimensions in pixels. Others might include the serial number or the number of the current frame from the reference design. Some pieces of information are provided by all devices, some are only available in certain models.

8.1 Getting the source data properties

The function pmdGetSourceDataDescription() returns a PMDDataDescription structure which includes all necessary information about the source data. A PMDDataDescription contains the type of the data (e.g. a code for "16-Bit Difference data" or "precalculated floating point distances"), the size of the data block in bytes and a union of sub-structures with additional information, depending on the type. It also includes a unique ID for the data block. The PMDDataDescription always refers to the data generated by the last pmdUpdate() call.

The most important sub-structure is called img (of the type PMDImageData). It includes fields for the number of columns and rows of the sensor. It also includes additional information like a sub-type or the number of sub-images in the data block (e.g. 4 phase images from a standard pmd reference design). A PMDDataDescription structure is always exactly 128 bytes long. Unused bytes are padded.

This structure is provided by all reference designs/data sources.

9 Error handling

All functions of the PMDSDK2 return a status code of the type int. If the command was executed successfully, this status code is PMD_OK. If there was an error, the code contains a value describing the type of error that occurred. For example, if a source plugin is supposed to read data from a file which does not exist, the call to pmdOpen() will return PMD_FILE_NOT_FOUND.

To make it easier to deal with errors, the function pmdGetLastError() exists. It will return a textual description of the last error that was associated with the given handle.

Example:

```
void  doSomething(PMDHandle  hnd)
{
  int  res;

  res  =  pmdUpdate  (hnd);

  if  (res  ==  PMD_OK)
    {
      printf  ("Everything  went  ok\n");
    }
  else
    {
      char  err[128];
      pmdGetLastError  (hnd,  err,  128);
      fprintf  (stderr,  "An  error  occured:  %s\n",  dat);
    }
}
```

Part 4

Reference Manual

10 Plugin Documentation

This section describes the SDK plugins shipped with your software package in detail. The content of this section may vary depending on the hardware you have.

10.1 O3D3xx camera Plugins

This section provides all the documentation that is specific for the source and processing plugins of the PMD reference design called O3D3xx camera. The O3D3xx camera design is based on the PG250 platform where most of the raw data processing is done.

10.2 Source Plugin

The source plugin for the O3D3xx camera reference design is able to connect to the sensor via *ethernet*. Also, it is able to captures frames from camera and configures integration time of camera.

10.3 Processing Plugin

The processing plugin for the O3D3xx camera reference design applies some additional data processing on the data delivered by the sensor board. All processing within the processing plugin is done on the host system, i.e. filtering etc. is done on the host CPU.

11 Module Documentation

11.1 PMDSDK

Definitions for the PMDSDK2.

Files

- file pmddatadescription.h
  Definitions and structs which describe the data format of the PMDSDK2.
- file pmdsdk2.h
  The main include file for working with pmd sensors.
- file pmdsdk2common.h
  Common definitions and structs used by the PMDSDK2.

Functions

- int pmdOpen (PMDHandle _hnd, const char _rplugin, const char _rparam, const char _pplugin, const char _pparam)
  Handle for a camera connection. This handle is used for all subsequent operations.
- int pmdOpenSourcePlugin (PMDHandle _hnd, const char _rplugin, const char _rparam)
  Connect to a pmd device or other data source without processing.
- int pmdClose (PMDHandle hnd)
  Disconnect and close the handle.
- int pmdCloseAll ()
  Disconnect and close all handles.
- int pmdGetLastError (PMDHandle hnd, char _error, size_t maxLen)
  Get an error description from the last error. Error messages are stored per handle. A new error associated with PMDHandle A does not overwrite the last error message associated with PMDHandle B.
- int pmdUpdate (PMDHandle hnd)
  Retrieve the a new frame from the camera. To obtain the actual data, use pmdGetSourceData, pmdGetDistances, pmdGetAmplitudes etc.
- int pmdSetIntegrationTime (PMDHandle hnd, unsigned idx, unsigned t)
  Set the integration time of the camera.
- int pmdGetIntegrationTime (PMDHandle hnd, unsigned _t, unsigned idx)
  Get the integration time of the camera.
- int pmdGetSourceData (PMDHandle hnd, void _data, size_t maxLen)
  Get the raw data from the current frame.
- int pmdGetSourceDataSize (PMDHandle hnd, size_t _size)
  Get the size in bytes of the current raw data frame.
- int pmdGetSourceDataDescription (PMDHandle hnd, struct PMDDataDescription _dd)
  Get the description of the current raw data frame.
- int pmdGetDistances (PMDHandle hnd, float _data, size_t maxLen)
  Get the distance data from the current frame.
- int pmdGet3DCoordinates (PMDHandle hnd, float _data, size_t maxLen)
  Get 3d-coordinates of the current frame.
- int pmdGetAmplitudes (PMDHandle hnd, float _data, size_t maxLen)
  Get the amplitude data from the current frame.
- int pmdGetFlags (PMDHandle hnd, unsigned _data, size_t maxLen)

Get the flags from the current frame.

- int pmdOpenProcessingPlugin (PMDHandle _hnd, const char _pplugin, const char _pparam)
  Open a PMDSDK processing plugin.
- int pmdCalcDistances (PMDHandle hnd, float _data, size_t maxLen, struct PMDDataDescription sourceDD, void _sourceData)
  Calculate the distances based on the source data.
- int pmdCalcAmplitudes (PMDHandle hnd, float _data, size_t maxLen, struct PMDDataDescription sourceDD, void _sourceData)
  Calculate the amplitudes based on the source data.
- int pmdCalc3DCoordinates (PMDHandle hnd, float _data, size_t maxLen, struct PMDDataDescription sourceDD, void _sourceData)
  Calculate the 3D coordinates based on the source data.
- int pmdCalcFlags (PMDHandle hnd, unsigned _data, size_t maxLen, struct PMDDataDescription sourceDD, void _sourceData)
  Calculate the flags based on the source data.

11.1.1 Detailed Description

Definitions for the PMDSDK2.

11.1.2 Function Documentation

11.1.2.1 int pmdCalc3DCoordinates (PMDHandle hnd, float data, size t maxLen, struct PMDDataDescription sourceDD, void sourceData)

Calculate the 3D coordinates based on the source data.

Parameters

hnd Handle of the connection.
data Array for the calculation result.
maxLen Size of the result array in bytes.
sourceDD Source data description.
sourceData Source data.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.2 int pmdCalcAmplitudes (PMDHandle hnd, float data, size t maxLen, struct PMDDataDescription sourceDD, void sourceData)

Calculate the amplitudes based on the source data.

Parameters

hnd Handle of the connection.
data Array for the calculation result.
maxLen Size of the result array in bytes.

sourceDD Source data description.
sourceData Source data.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.3 int pmdCalcDistances (PMDHandle hnd, float data, size t maxLen, struct PMDDataDescription sourceDD, void sourceData)

Calculate the distances based on the source data.

Parameters

hnd Handle of the connection.
data Array for the calculation result.
maxLen Size of the result array in bytes.
sourceDD Source data description.
sourceData Source data.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.4 int pmdCalcFlags (PMDHandle hnd, unsigned data, size t maxLen, struct PMDDataDescription sourceDD, void sourceData)

Calculate the flags based on the source data.

Parameters

hnd Handle of the connection.
data Array for the calculation result.
maxLen Size of the result array in bytes.
sourceDD Source data description.
sourceData Source data.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.5 int pmdClose (PMDHandle hnd)

Disconnect and close the handle.

Parameters

hnd Handle of the connection.

Returns PMD_OK on success, errorcode otherwise

11.1.2.6 int pmdCloseAll ( )

Disconnect and close all handles.

Returns

PMD_OK on success, errorcode otherwise\

11.1.2.7 int pmdGet3DCoordinates (PMDHandle hnd, float data, size t maxLen)

Get 3d-coordinates of the current frame.

Parameters

hnd Handle of the connection.
data Pointer to a block of memory to contain the data.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.8 int pmdGetAmplitudes (PMDHandle hnd, float data, size t maxLen)

Get the amplitude data from the current frame.

Parameters

hnd Handle of the connection.
data Pointer to a block of memory to contain the data.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.9 int pmdGetDistances (PMDHandle hnd, float data, size t maxLen)

Get the distance data from the current frame.

Parameters

hnd Handle of the connection.
data Pointer to a block of memory to contain the data.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.10 int pmdGetFlags (PMDHandle hnd, unsigned data, size t maxLen)

Get the flags from the current frame.

Parameters

hnd Handle of the connection.
data Pointer to a block of memory to contain the data.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.11 int pmdGetIntegrationTime (PMDHandle hnd, unsigned time, unsigned idx)

Get the integration time of the camera.

Parameters

hnd Handle of the connection.

idx Index of the integration time.
time Pointer to a variable to contain the integration time in microseconds.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.12 int pmdGetLastError (PMDHandle hnd, char error, size_t maxLen)

Get an error description from the last error. Error messages are stored per handle. A new error associated with PMDHandle A does not overwrite the last error message associated with PMDHandle B.

Parameters

hnd Handle of the connection/plugin.
error Memory to hold the error message.
maxLen Maximum length of the error message, including the terminating zero byte. Longer messages will be truncated.

11.1.2.13 int pmdGetSourceData (PMDHandle hnd, void data, size t maxLen)

Get the raw data from the current frame.

Parameters

hnd Handle of the connection.
data Pointer to the memory to contain the address of the data.
maxLen Maximum length in bytes for the data

Returns

PMD_OK on success, errorcode otherwise

11.1.2.14 int pmdGetSourceDataDescription (PMDHandle hnd, struct PMDDataDescription dd)

Get the description of the current raw data frame.

Parameters

hnd Handle of the connection.
dd Will contain the PMDDataDescription after the call.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.15 int pmdGetSourceDataSize (PMDHandle hnd, size t size)

Get the size in bytes of the current raw data frame.

Parameters

hnd Handle of the connection.
size Will contain the size after the call

Returns

PMD_OK on success, errorcode otherwise

11.1.2.16 int pmdOpen (PMDHandle hnd, const char rplugin, const char rparam, const char pplugin, const char pparam)

Handle for a camera connection. This handle is used for all subsequent operations. Connect to a PMD sensor or other data source

Parameters

hnd Empty PMDHandle structure. On success, this value will contain the handle for subsequent operations.
rplugin Path of the camera plugin
rparam Parameter for the camera plugin
pplugin Path of the processing plugin. If this is NULL, no processing plugin will be loaded.
pparam Parameter for the processing plugin

Returns

PMD_OK on success, errorcode otherwise

11.1.2.17 int pmdOpenProcessingPlugin (PMDHandle hnd, const char pplugin, const char pparam)

Open a PMDSDK processing plugin.

Parameters

hnd Handle of the connection.

pplugin Filename of the plugin.
pparam Parameters for the plugin.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.18 int pmdOpenSourcePlugin (PMDHandle hnd, const char rplugin, const char rparam)

Connect to a pmd device or other data source without processing.

Parameters

hnd Empty PMDHandle structure. On success, this value will contain the handle for subsequent operations.
rplugin Path of the camera plugin
rparam Parameter for the camera plugin

Returns

PMD_OK on success, errorcode otherwise

11.1.2.19 int pmdSetIntegrationTime (PMDHandle hnd, unsigned idx, unsigned time)

Set the integration time of the camera.

Parameters

hnd Handle of the connection.
idx Index of the integration time.
time Integration time in microseconds.

Returns

PMD_OK on success, errorcode otherwise

11.1.2.20 int pmdUpdate (PMDHandle hnd)

Retrieve the a new frame from the camera. To obtain the actual data, use pmdGetSourceData, pmdGetDistances, pmdGetAmplitudes etc.

Parameters

hnd Handle of the connection.

Returns

PMD_OK on success, errorcode otherwise

11.2 pmdCommon

Common definitions used by the PMDSDK2.

Structs

- struct PMDGenericData
  Generic data description.
- struct PMDImageData
  Standard PMD image data

Typedefs

- typedef float PMDFloat32
  Single precision float.
- typedef double PMDFloat64
  Double precision float.
- typedef PMDFloat32 PMDFloat
  Standard float.
- typedef unsigned PMDUInt32
  Unsigned 32-bit integer.
- typedef signed int PMDSInt32
  Signed 32-bit integer.
- typedef unsigned short PMDUInt16
  Unsigned 16-bit integer.
- typedef signed short PMDSInt16
  Signed 16-bit integer.
- typedef unsigned char PMDUInt8
  Unsigned 8-bit integer.
- typedef signed char PMDSInt8
  Signed 8-bit integer.

Variables

- unsigned PMDGenericData::subType
  Specific type of the data.
- unsigned PMDGenericData::numElem
  Number of elements in the data.
- unsigned PMDGenericData::sizeOfElem
  Size of one element in bytes.
- unsignedPMDImageData::subType
  Specific type of the data.
- unsignedPMDImageData::numColumns
  Number of columns in the image.
- unsignedPMDImageData::numRows
  Number of rows in the image.
- unsignedPMDImageData::numSubImages
  Number of sub images.
- intPMDImageData::integrationTime[4]
  Integration times at which the data was captured.
- intPMDImageData::modulationFrequency[4]
  Modulation frequencies at which the data was captured.

•intPMDImageData::offset[4]
Offsets for up to four separate measurements in mm.
•intPMDImageData::pixelAspectRatio
Pixel aspect ratio. The ratio is pixelAspectRatio / 1000.0.
•intPMDImageData::pixelOrigin
Position/direction of the first pixel.
•unsignedPMDImageData::timeStampHi
Time at which the data was captured. Most significant word.
•unsignedPMDImageData::timeStampLo
Time at which the data was captured. Least significant word.
•charPMDImageData::reserved[24]
Reserved for future use.
•unsignedPMDImageData::userData0
Contains user-defined information. This will not be evaluated by the PMDSDK.

11.2.1 Detailed Description

Common definitions used by the PMDSDK2.

11.3 StatusCodes

Status codes used by the PMDSDK2.

Macros

• #define PMD_OK 0
Success.
• #define PMD_RUNTIME_ERROR 1024
Runtime error.
• #define PMD_GENERIC_ERROR 1025
Generic error.
• #define PMD_DISCONNECTED 1026
Connection lost.
• #define PMD_INVALID_VALUE 1027
An invalid value was given.
• #define PMD_TIMEOUT_ERROR 1028
A timeout occurred.
• #define PMD_LOGIC_ERROR 2048
Program error.
• #define PMD_UNKNOWN_HANDLE 2049
Handle not known (internal error)
• #define PMD_NOT_IMPLEMENTED 2050
Requested functionality not implemented.
• #define PMD_OUT_OF_BOUNDS 2051
Index or value out of range.
• #define PMD_RESOURCE_ERROR 4096
Could not get resource.
• #define PMD_FILE_NOT_FOUND 4097
Could not open file.
• #define PMD_COULD_NOT_OPEN 4098
Could not connect to or open the data source.
• #define PMD_DATA_NOT_FOUND 4099

Could not retrieve the requested data.
- #define PMD_END_OF_DATA 4100
  There is no more data left.
- #define PMD_DEVICE_IS_BUSY 4101
  The device currently used and can not be accessed.

11.4 Flags

Flag definitions.

Macros

- #define PMD_FLAG_HIDE_PIXEL 0x00000001u
  invalid flag
- #define PMD_FLAG_INVALID 0x00000001u
  invalid flag
- #define PMD_FLAG_SATURATED 0x00000002u
  saturation flag
- #define PMD_FLAG_INCONSISTENT 0x00000004u
  inconsistency flag
- #define PMD_FLAG_LOW_SIGNAL 0x00000008u
  low signal flag
- #define PMD_FLAG_SBI_ACTIVE 0x00000010u
  SBI flag.

11.4.1 Detailed Description

For O3D3xx camera, the confidence image provides status information for each pixel. The information is bit coded and the meaning and values of the bits are shown in the table below.

Flag data type: 8 bit unsigned integer (Bits 0-7)

Bit	Value	Description
0	1 = pixel invalid	Pixel invalid (NV) The pixel is invalid. To determine, whether a pixel is valid or not only this bit needs to be checked. For a reason why the bit is invalid the other confidence bits may be checked.
1	1 = pixel saturated	Pixel is saturated (SA) If binning is active and if all 4 original pixels are invalid and if at least one is saturated, then this bit is set. Contributes to pixel validity: yes

2	1 = bad A-B symmetry	A-B pixel symmetry (SY) The A-B symmetry value of the four phase measurements is above threshold. Remark: This symmetry value is used to detect motion artefacts. Noise (e.g. due to strong ambient light or very short integration times) or PMD interference may also contribute. If binning is active and if all 4 original pixels are invalid and if at least one is asymmetric, then this bit is set. Contributes to pixel validity: yes, unless motion artefact correction is switched on and the pixel has been corrected.
3	1 = amplitude below minimum amplitude threshold	Amplitude limits (AM) The amplitude value is below minimum amplitude threshold. If binning is active and if all 4 original pixels are invalid and if at least one is underexposed, then this bit is set. Contributes to pixel validity: yes
4+5	Bit 5, Bit 4 0 0 unused 0 1 shortest exposure time (only used in 3 exposure mode) 1 0 middle exposure time in 3 exposure mode, short exposure in double exposure mode 1 1 longest exposure time (always 1 in single exposure mode)	Exposure time indicator The two bits indicate, which exposure time was used in a multiple exposure measurement. Contributes to pixel validity: no
6	1= motion artefact compensated	Motion artefact compensated (MC) A motion artefact has been detected in this pixel and its value has been replaced by a compensated value. Contributes to pixel validity: no
7	1 = pixel suspect/defect	Suspect pixel (SU) This pixel has been marked as "suspect" or "defect" and values have been replaced by interpolated values from the surrounding. Contributes to pixel validity: no

12 Class Documentation

12.1 PMDGenericData Struct Reference

Generic data description.

Public Attributes

- unsigned subType
  Specific type of the data.
- unsigned numElem
  Number of elements in the data.
- unsigned sizeOfElem
  Size of one element in bytes.

12.1.1 Detailed Description

Generic data description.

12.2 PMDImageData Struct Reference

Standard PMD image data.

Public Attributes

- unsigned subType
  Specific type of the data.
- unsigned numColumns
  Number of columns in the image.
- unsigned numRows
  Number of rows in the image.
- unsigned numSubImages
  Number of sub images.
- int integrationTime[4]
  Integration times at which the data was captured.
- int modulationFrequency[4]
  Modulation frequencies at which the data was captured.
- int offset[4]
  Offsets for up to four separate measurements in mm.
- int pixelAspectRatio
  Pixel aspect ratio. The ratio is pixelAspectRatio / 1000.0.
- int pixelOrigin
  Position/direction of the first pixel.
- unsigned timeStampHi
  Time at which the data was captured. Most significant word.
- unsigned timeStampLo
  Time at which the data was captured. Least significant word.
- char reserved[24]

Reserved for future use.

- unsigned userData0

  Contains user-defined information. This will not be evaluated by the PMDSDK.

12.2.1 Detailed Description

Standard PMD image data.

13 File Documentation

13.1 include/pmdsdk2.h File Reference

The main include file for working with pmd sensors.

Functions

- int pmdOpen (PMDHandle _hnd, const char _rplugin, const char _rparam, const char _pplugin, const char _pparam)
  Handle for a camera connection. This handle is used for all subsequent operations.
- int pmdOpenSourcePlugin (PMDHandle _hnd, const char _rplugin, const char _rparam)
  Connect to a pmd device or other data source without processing.
- int pmdClose (PMDHandle hnd)
  Disconnect and close the handle.
- int pmdCloseAll ()
  Disconnect and close all handles.
- int pmdGetLastError (PMDHandle hnd, char _error, size_t maxLen)
  Get an error description from the last error. Error messages are stored per handle. A new error associated with PMDHandle A does not overwrite the last error message associated with PMDHandle B.
- int pmdUpdate (PMDHandle hnd)
  Retrieve the a new frame from the camera. To obtain the actual data, use pmdGetSourceData, pmdGetDistances, pmdGetAmplitudes etc.
- int pmdSetIntegrationTime (PMDHandle hnd, unsigned idx, unsigned t)
  Set the integration time of the camera.
- int pmdGetIntegrationTime (PMDHandle hnd, unsigned _t, unsigned idx)
  Get the integration time of the camera.
- int pmdGetSourceData (PMDHandle hnd, void _data, size_t maxLen)
  Get the image data from the current frame.
- int pmdGetSourceDataSize (PMDHandle hnd, size_t _size)
  Get the size in bytes of the current raw data frame.
- int pmdGetSourceDataDescription (PMDHandle hnd, struct PMDDataDescription _dd)
  Get the description of the current raw data frame.
- int pmdGetDistances (PMDHandle hnd, float _data, size_t maxLen)
  Get the distance data from the current frame.
- int pmdGet3DCoordinates (PMDHandle hnd, float _data, size_t maxLen)
  Get 3d-coordinates of the current frame.
- int pmdGetAmplitudes (PMDHandle hnd, float _data, size_t maxLen)
  Get the amplitude data from the current frame.
- int pmdGetFlags (PMDHandle hnd, unsigned _data, size_t maxLen)
  Get the flags from the current frame.
- int pmdOpenProcessingPlugin (PMDHandle _hnd, const char _pplugin, const char _pparam)
  Open a PMDSDK processing plugin.
- int pmdCalcDistances (PMDHandle hnd, float _data, size_t maxLen, struct PMDDataDescription sourceDD, void _sourceData)
  Calculate the distances based on the source data.
- int pmdCalcAmplitudes (PMDHandle hnd, float _data, size_t maxLen, struct PMDDataDescription

sourceDD, void _sourceData)
Calculate the amplitudes based on the source data.
• int pmdCalc3DCoordinates (PMDHandle hnd, float _data, size_t maxLen, struct PMDDataDescription sourceDD, void _sourceData)
Calculate the 3D coordinates based on the source data.
• int pmdCalcFlags (PMDHandle hnd, unsigned _data, size_t maxLen, struct PMDDataDescription sourceDD, void _sourceData)
Calculate the flags based on the source data.

13.1.1 Detailed Description

The main include file for working with pmd sensors.

13.2 include/pmdsdk2common.h File Reference

Common definitions and structs used by the PMDSDK2.

Macros

• #define PMD_OK 0
Success.
• #define PMD_RUNTIME_ERROR 1024
Runtime error.
• #define PMD_GENERIC_ERROR 1025
Generic error.
• #define PMD_DISCONNECTED 1026
Connection lost.
• #define PMD_INVALID_VALUE 1027
An invalid value was given.
• #define PMD_TIMEOUT_ERROR 1028
A timeout occurred.
• #define PMD_LOGIC_ERROR 2048
Program error.
• #define PMD_UNKNOWN_HANDLE 2049
Handle not known (internal error)
• #define PMD_NOT_IMPLEMENTED 2050
Requested functionality not implemented.
• #define PMD_OUT_OF_BOUNDS 2051
Index or value out of range.
• #define PMD_RESOURCE_ERROR 4096
Could not get resource.
• #define PMD_FILE_NOT_FOUND 4097
Could not open file.
• #define PMD_COULD_NOT_OPEN 4098
Could not connect to or open the data source.
• #define PMD_DATA_NOT_FOUND 4099
Could not retrieve the requested data.
• #define PMD_END_OF_DATA 4100
There is no more data left.
• #define PMD_DEVICE_IS_BUSY 4101
The device currently used and can not be accesse

Typedefs

- typedef float PMDFloat32
  Single precision float.
- typedef double PMDFloat64
  Double precision float.
- typedef PMDFloat32 PMDFloat
  Standard float.
- typedef unsigned PMDUInt32
  Unsigned 32-bit integer.
- typedef signed int PMDSInt32
  Signed 32-bit integer.
- typedef unsigned short PMDUInt16
  Unsigned 16-bit integer.
- typedef signed short PMDSInt16
  Signed 16-bit integer.
- typedef unsigned char PMDUInt8
  Unsigned 8-bit integer.
- typedef signed char PMDSInt8
  Signed 8-bit integer.

Enumerations

- enum Proximity
  Enumeration for retrieving valid integration times and modulation frequencies.

13.2.1 Detailed Description

Common definitions and structs used by the PMDSDK2.