

PMDSDK 2

Version 2.1.1

PROGRAMMING MANUAL

For 3D time-of-flight cameras

Technical information subject to change without notice.
This document may also be changed without notice.
July 2008

Version:	2.1.1-1
Created:	31/July/2008
Changed:	15/Jun/2009
Author:	Pro,Frd
© PMDTec GmbH	

All texts, pictures and other contents published in this instruction manual are subject to the copyright of PMDTec, Siegen unless otherwise noticed. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher PMDTec.

We would like to advise you that all equipment related to the camera demonstrator, including the camera itself, is intended for internal use on an experimental basis only. Please note, that PMDTechnologies has assigned exclusive rights to third party companies for the use of PMDTechnologies systems. Permission must be granted by these companies for any further use or development of these systems, and lack thereof may result in PMDTechnologies being unable to supply further revisions of the equipment in the future.

Table of Contents

1	INTRODUCTION.....	4
1.1	DESCRIPTION	4
1.2	SYSTEM REQUIREMENTS	4
1.3	CHANGES.....	4
2	BUILDING APPLICATIONS.....	5
2.1	BUILDING APPLICATIONS FOR MICROSOFT WINDOWS	5
2.2	BUILDING APPLICATIONS FOR LINUX.....	5
3	GENERAL API DESCRIPTION.....	6
4	CONNECTION AND PROCESSING WITH PLUGINS	8
4.1	SEPARATE PLUGIN INSTANTIATION.....	9
5	RETRIEVING IMAGE DATA	11
5.1	SEPARATE CALCULATION OF IMAGE DATA	12
6	CONFIGURING THE PMD CAMERA	14
6.1	QUERYING VALID VALUES	14
7	RETRIEVING CAMERA INFORMATION.....	16
7.1	GETTING THE SOURCE DATA PROPERTIES.....	16
7.2	GETTING OTHER PROPERTIES	16
8	CONFIGURING PLUGINS AT RUNTIME	17
9	ERROR HANDLING.....	18
A.	REFERENCE	19
A.	STATUS CODES	19
B.	TYPES AND DATA STRUCTURES.....	20
C.	FUNCTION DOCUMENTATION.....	21

1 Introduction

1.1 Description

This manual describes the programming API (application programming interface) for PMD[vision][®] time-of-flight cameras.

The PMDSDK 2.1 supplies functionality to access PMD[vision][®] cameras and other data sources. It is possible to set camera parameters, as well as retrieve 2D and 3D images from the data source. This allows simple and fast application development supporting all available PMD[vision][®] cameras.

All functionality is provided through an API for the C programming language.

1.2 System requirements

For Windows:

- Microsoft Windows XP
- Microsoft Visual Studio .net 2003/2005/2008

Other Compilers/IDEs have been reported to work with the PMDSDK 2.1 as well.

For Linux:

- Linux Operating System
- x86 architecture or compatible
- GCC 4.1.2 or compatible
- glibc 2.3.6 or compatible, libstdc++ 6

1.3 Changes

2.1.0 → 2.1.1

- Added data type for the PMD[vision][®] CamCube 2.0

2.0.0 → 2.1.0

- Added pmdGet3DCoordinates function
- Added pmdCalc* functions
- Renamed *std* to *img* in *PMDDataDescription* structure. *std* is still available for source code compatibility if *PMD_NO_DEPRECATED* is not defined.

2 Building applications

2.1 Building applications for Microsoft Windows

To create Windows applications using the PMDSDK 2.1, several files from this SDK are needed.

- `pmdsdk2.h`: This file contains the declarations of all available PMDSDK 2.1 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`
- `pmdadatadescription.h`: This file is automatically included from `pmdsdk2.h`
- A source plugin file (`*.W32.pap` or `*.W32.pcp`)
- A processing plugin file (`*.W32.ppp` or `*.W32.pcp`)

2.2 Building applications for Linux

To create Linux applications using the PMDSDK 2.1, several files from this SDK are needed.

- `pmdsdk2.h`: This file contains the declarations of all available PMDSDK 2.1 functions. This must be included in the application source code.
- `libpmdaccess2.so`: This is the library.
- `pmdsdk2common.h`: This file is automatically included from `pmdsdk2.h`.
- `pmdadatadescription.h`: This file is automatically included from `pmdsdk2.h`.
- A source plugin file (`*.L32.pap` or `*.L32.pcp`).
- A processing plugin file (`*.L32.ppp` or `*.L32.pcp`).

3 General API description

All functions in the PMDSDK 2.1 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 PMDSDK 2.1 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 PMDSDK 2.1. The program attempts to connect to a PMD camera using the plugins called *a.X32.pap* and *b.X32.ppp* (where X is either W or L, depending on the operating system) and retrieve the number of pixel columns in the camera data. For more information on plugins, see the next section.

```
#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 Connection and processing with plugins

The PMDSDK 2.1 uses plugins to connect to the different camera models (or other data sources) and to do the processing that is needed to generate distances and other types of data.

The first thing a program using the PMDSDK 2.1 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 camera and a processing plugin for the calculation). Each plugin can take a string parameter for initialization. After this initialization, the other functions can be used in conjunction with the handle. If the plugin's behaviour shall be changed during runtime in a way that is not provided by the standard functions, the PMDSDK 2.1 provides functions for issuing plugin dependent commands as well (*pmdSourceCommand()*, *pmdProcessingCommand()*). 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 *mycam.pap* and a processing plugin called *myproc.ppp*. The source plugin takes an IP address as its parameter, while the processing plugin takes its own 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,
                  "mycam", "10.0.0.1",
                  "myproc", "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;
}
```


4.1 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 PMDSDK 2.1 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,
                              "mycam", "10.0.0.1");

    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*
- *pmdGetValidIntegrationTime*
- *pmdSetModulationFrequency*
- *pmdGetModulationFrequency*
- *pmdGetValidModulationFrequency*
- *pmdSourceCommand*

These functions need a processing plugin:

- *pmdCalcDistances*
- *pmdCalcAmplitudes*
- *pmdCalcIntensities*
- *pmdCalc3DCoordinates*
- *pmdProcessingCommand*

The following functions need both plugins on the same *PMDHandle*:

- *pmdGetDistances*
- *pmdGetAmplitudes*
- *pmdGetIntensities*
- *pmdGet3DCoordinates*

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

5 Retrieving image data

The PMDSDK 2.1 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)
- *pmdGetIntensities* (**PMDHandle** hnd, **float** * data, **size_t** size)
- *pmdGetSourceData* (**PMDHandle** hnd, **void** * data, **size_t** size)
- *pmdGet3DCoordinates* (**PMDHandle** hnd, **float** * 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 phase images) of the camera. Although in most cases, the source data will not be needed, some applications might have a need for it. 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.

pmdGetIntensities() will return a greyscale image. Not all cameras support this. In fact, none of the above functions is guaranteed to work with every device. Upon failure, they will return a value other than *PMD_OK*.

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 ($x_1\ y_1\ z_1\ x_2\ y_2\ z_2\ \dots$).

Before any of those functions can be called, the actual image data must be transferred from the PMD camera. 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. Error checking is kept simple (and crude) to keep the example short.

```
void showFirstPixel (PMDHandle hnd)
{
    int res;
    float dat[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]);
}
```

5.1 Separate calculation of image data

Additionally, the PMDSDK 2.1 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)
- *pmdCalcIntensities* (**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)

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.

6 Configuring the PMD camera

There are two important configuration parameters that control the PMD camera's image acquisition:

- The integration time.
- The modulation frequency.

Some cameras use only one integration time and one modulation frequency, others support multiple settings. Some cameras use fixed settings, others can be configured.

The PMDSDK 2.1 provides four functions to access these settings:

- *pmdGetIntegrationTime()*
- *pmdSetIntegrationTime()*
- *pmdGetModulationFrequency()*
- *pmdSetModulationFrequency()*

The function below will display the current settings of the first integration time (index 0) and the third modulation frequency (index 2) on the screen.

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

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

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

    res = pmdGetModulationFrequency (hnd, &m, 2);

    if (res != PMD_OK) exit (8);

    printf ("Integration time: %d microseconds.\n", i);
    printf ("Modulation frequency: %d Hz.\n", m);
}
```

6.1 Querying valid values

The Set functions will fail if the supplied value does not exactly match an integration time/modulation frequency that is supported by the device. It is not always known which values are supported beforehand and the acceptable values might even change during the operation of the camera (e.g. some cameras might support different integration time settings for different

modulation frequencies). Therefore, there are two additional commands to query valid values:

- *pmdGetValidIntegrationTime()*
- *pmdGetValidModulationFrequency()*

With those commands it is possible to search for valid values that best match a given reference value. If, for example, you want to use an integration time of around 4 ms as the first integration time (index 0), you could use this command to get the supported integration time that is closest to it:

```
pmdGetValidIntegrationTime (hnd, &i, 0, CloseTo, 4000);
```

If the device supports 4 ms, the variable *i* will contain a value of 4000, otherwise it might contain values like 4100 or 3575, depending on what the camera can work with.

If 4 ms is the desired lower or upper limit, you can use

```
pmdGetValidIntegrationTime (hnd, &i, 0, AtLeast, 4000);
```

or

```
pmdGetValidIntegrationTime (hnd, &i, 0, AtMost, 4000);
```

Retrieving valid modulation frequencies works the same way.

7 Retrieving camera information

Several additional pieces of information can be read from the PMD camera. 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 camera's serial number or the number of the current frame. Some pieces of information are provided by all cameras/device, some are only available in certain models.

7.1 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 camera).

A *PMDDataDescription* structure is always exactly 128 bytes long. Unused bytes are padded.

This structure is provided by all camera models/data sources.

7.2 Getting other properties

The availability of other information depends on the model of the camera and the plugin that is used to connect to it. The functions *pmdSourceCommand()* and *pmdProcessingCommand()* can be used to issue plugin-dependent commands, including the retrieval of plugin-dependent information (see also the next chapter). The following code example demonstrates the retrieval of a serial number (if this is supported by the camera/data source):

```
void showSerialNumber (PMDHandle hnd)
{
    int res;
    char dat[MAX_LEN];

    res=pmdSourceCommand (hnd, dat, sizeof(dat),
                          "GetSerialNumber");
    if (res != PMD_OK) exit (4);

    printf ("The serial number is %s\n", dat);
}
```


8 Configuring plugins at runtime

Special configuration of the plugins can be achieved through two extra functions:

- *pmdSourceCommand()* to configure the source plugin
- *pmdProcessingCommand()* to configure the processing plugin

These functions take a textual command that is interpreted by the plugin. It is the plugin's responsibility to act accordingly. They can return a result message in a string.

Example:

```
void resetOffset (PMDHandle hnd)
{
    int res;
    char dat[MAX_LEN];
    dat[0] = 0; // init with empty string

    res=pmdProcessingCommand (hnd, dat, sizeof(dat),
                             "SetOffset 0");

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

    printf ("The result was: %s\n", dat);
}
```

9 Error handling

All functions of the PMDSDK 2.1 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 occurred: %s\n", dat);
    }
}
```

A. Reference

a. Status Codes

#define PMD_OK 0

Operation succeeded.

This value is returned when no error occurred during an operation.

#define PMD_RUNTIME_ERROR 1024

A runtime error occurred.

#define PMD_GENERIC_ERROR 1025

An unknown error occurred.

#define PMD_DISCONNECTED 1026

The camera was disconnected.

#define PMD_INVALID_VALUE 1027

The specified value/parameter is invalid.

#define PMD_LOGIC_ERROR 2048

The program did not behave correctly.

#define PMD_UNKNOWN_HANDLE 2049

The specified handle is invalid.

#define PMD_NOT_IMPLEMENTED 2050

The requested operation is not implemented.

#define PMD_OUT_OF_BOUNDS 2051

The specified value/parameter is not within the supported range.

#define PMD_RESOURCE_ERROR 4096

A resource could not be acquired.

#define PMD_FILE_NOT_FOUND 4097

The specified file could not be found.

#define PMD_COULD_NOT_OPEN 4098

Could not open data source.

#define PMD_DATA_NOT_FOUND 4099

The requested piece of information is not available or does not exist.

#define PMD_END_OF_DATA 4100

The data source has reached its end.

b. Types and data structures**typedef unsigned PMDHandle**

Handle for a camera connection. This handle is used for all operations that interact with a data source. It is initialized with the functions *pmdOpen()* or *pmdOpenSourcePlugin()* and is disposed when calling *pmdClose()*.

struct PMDDataDescription

Contains information about a data block. This is used to describe the source data of a camera. A *PMDDataDescription* is always 128 bytes long.

Note: This structure contains an anonymous union. Most C compilers support this feature, but it is not part of the ANSI C standard. If you need full ANSI C compatibility, define *PMD_ANSI_C* before including *pmdsdk2.h*. The union then has the name *u*, so in order to access its fields, write *myDD.u.img.numColumns* instead of *myDD.img.numColumns*. C++ does not have this problem as its standard includes anonymous unions.

Fields:

PID Identifier of the plugin that generated or modified the data block

DID Identifier of the data block. This is unique for a plugin

type Type of the data. This is used to describe what kind of data is in the block.

size Size in bytes of the data block

subHeaderType Identifier for the active sub header structure. This can be *PMD_IMAGE_DATA* or *PMD_GENERIC_DATA*.

Union Fields (only one of these fields is used at the same time):

gen *PMDGenericData*

img *PMDImageData*

struct PMDGenericData

Contains information about a generic data block. This structure is used as a field inside *PMDDataDescription* for special information.

Fields:

subType Sub-type of the data. This depends on *type* in *PMDDataDescription* and allows further refinement of the type.

numElem Number of data elements.

sizeOfElem Size in bytes of one data element.

struct PMDImageData

Contains information about a PMD- or image-data block. This structure is used as a field inside *PMDDataDescription* for special information.

Fields:

subType Sub-type of the data. This depends on *type* in *PMDDataDescription* and allows further refinement of the type.

numColumns Number of pixel columns in the image.

numRows Number of pixel rows in the image.

numSubImages Number of sub-images (e.g. phase images) in the data block.

integrationTime[] Up to four integration times used to generate the image.

modulationFrequency[] Up to four modulation frequencies used to generate the image.

c. Function Documentation

int pmdOpen (PMDHandle * *hnd*, const char * *pap*, const char * *rparam*, const char * *ppp*, const char * *pparam*)

Open a connection to a PMD camera or other data source.

Parameters:

hnd Empty PMDHandle. On success, this value will contain the handle for subsequent operations.

pap Filename of the source plugin (*.pap or *.pcp)

rparam Parameter string for the source plugin

ppp Filename of the processing plugin (*.ppp or *.pcp)

pparam Parameter string for the processing plugin

Returns:

PMD_OK on success, errorcode otherwise

int pmdOpenSourcePlugin (PMDHandle * *hnd*, const char * *pap*, const char * *rparam*)

Open a connection to a PMD camera or other data source without a processing plugin. Functions that depend on a processing plugin, like *pmdGetDistances*, will not be available.

Parameters:

hnd Empty PMDHandle. On success, this value will contain the handle for subsequent operations.

pap Filename of the source plugin (*.pap or *.pcp)

rparam Parameter string for the source plugin

Returns:

PMD_OK on success, errorcode otherwise

int pmdClose (PMDHandle *hnd*)

Disconnect and close the handle.

After this call, the handle is invalid and must not be used anymore.

Parameters:

hnd Handle of the connection.

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetAmplitudes (PMDHandle *hnd*, float * *data*, size_t *maxLen*)

Get the amplitude data from the current frame. *pmdUpdate()* must be called at least once before this function.

The amplitude value of a pixel relates to its signal strength. Higher amplitudes indicate better accuracy of the measurement.

Not all data sources support this call.

Parameters:

hnd Handle of the connection.

data Pointer to memory to contain the data.

maxLen Number of bytes available in the memory block.

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetDistances (PMDHandle *hnd*, float * *data*, size_t *maxLen*)

Get the distance data from the current frame. *pmdUpdate()* must be called at least once before this function.

The values in this matrix represent the distance between the camera and the object that the respective pixel observes.

For some PMD cameras, the distances may be shifted by a constant offset, whose value depends on the modulation frequency. Not all camera models have this behaviour.

Not all data sources support this call.

Parameters:

hnd Handle of the connection.

data Pointer to memory to contain the data.

maxLen Number of bytes available in the memory block.

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetIntensities (PMDHandle *hnd*, float * *data*, size_t *maxLen*)

Get the grayscale data from the current frame. *pmdUpdate()* must be called at least once before this function.

This function produces a 2D grayscale image, like a standard 2D camera would produce.

Not all data sources support this call.

Parameters:

hnd Handle of the connection.

data Pointer to memory to contain the data.

maxLen Number of bytes available in the memory block.

Returns:

PMD_OK on success, errorcode otherwise

int pmdGet3DCoordinates (PMDHandle *hnd*, float * *data*, size_t *maxLen*)

Get the cartesian coordinates of the current frame. *pmdUpdate()* must be called at least once before this function.

The values in this matrix represent the coordinates between the camera and the object that the respective pixel observes. The x,y,z values are aligned pixelwise in the data array (*x₁ y₁ z₁ x₂ y₂ z₂ ..*).

For some PMD cameras, the coordinates may be shifted by a constant offset, whose value depends on the modulation frequency. Not all camera models have this behaviour.

Not all data sources support this call. Therefor consult your camera source plugin documentation.

Parameters:

hnd Handle of the connection.

data Pointer to memory to contain the data.

maxLen Number of bytes available in the memory block.

Returns:

PMD_OK on success, errorcode otherwise

int pmdCalcAmplitudes (PMDHandle *hnd*, float * *data*, size_t *maxLen*, PMDDatadescription *dd*, void * *data*)

Calculate the amplitude data from a given frame.

The amplitude value of a pixel relates to its signal strength. Higher amplitudes indicate better accuracy of the measurement.

Not all data sources support this call.

Parameters:

hnd Handle of the connection.

data Pointer to memory to contain the data.

maxLen Number of bytes available in the memory block.

dd Description of the source data frame

data The source data to use to calculate the amplitudes

Returns:

PMD_OK on success, errorcode otherwise

int pmdCalcDistances (PMDHandle *hnd*, float * *data*, size_t *maxLen*, PMDDatadescription *dd*, void * *data*)

Calculate the distance data from a given frame.

The values in this matrix represent the distance between the camera and the object that the respective pixel observes.

For some PMD cameras, the distances may be shifted by a constant offset, whose value depends on the modulation frequency. Not all camera models have this behavior.

Not all data sources support this call.

Parameters:

hnd Handle of the connection.

data Pointer to memory to contain the data.
maxLen Number of bytes available in the memory block.
dd Description of the source data frame
data The source data to use to calculate the distances

Returns:

PMD_OK on success, errorcode otherwise

int pmdCalcIntensities (PMDHandle *hnd*, float * *data*, size_t *maxLen*, PMDDatadescription *dd*, void * *data*)

Calculate the grayscale data from a given frame.

This function produces a 2D grayscale image, like a standard 2D camera would produce.

Not all data sources support this call.

Parameters:

hnd Handle of the connection.
data Pointer to memory to contain the data.
maxLen Number of bytes available in the memory block.
dd Description of the source data frame
data The source data to use to calculate the intensities

Returns:

PMD_OK on success, errorcode otherwise

int pmdCalc3DCoordinates (PMDHandle *hnd*, float * *data*, size_t *maxLen*, PMDDatadescription *dd*, void * *data*)

Calculate the cartesian coordinates from a given frame.

The values in this matrix represent the coordinates between the camera and the object that the respective pixel observes. The x,y,z values are aligned pixelwise in the data array (*x₁ y₁ z₁ x₂ y₂ z₂ ..*).

For some PMD cameras, the coordinates may be shifted by a constant offset, whose value depends on the modulation frequency. Not all camera models have this behaviour.

Not all data sources support this call.

Parameters:

hnd Handle of the connection.
data Pointer to memory to contain the data.
maxLen Number of bytes available in the memory block.
dd Description of the source data frame
data The source data to use to calculate the 3D coordinates

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetIntegrationTime (PMDHandle *hnd*, unsigned * *t*, unsigned *idx*)

Get an integration time of the camera

Parameters

hnd Handle of the connection.
t Pointer to a variable to contain the integration time in microseconds.

idx Index of the integration time to be retrieved (starting with 0).

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetModulationFrequency (PMDHandle *hnd*, unsigned * *t*, unsigned *idx*)

Get a modulation frequency of the camera

Parameters:

hnd Handle of the connection.

t Pointer to a variable to contain the modulation frequency in Hz.

idx Index of the modulation frequency to be retrieved (starting with 0).

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetSourceData (PMDHandle *hnd*, void * *data*, size_t *maxLen*)

Get the source data from the current frame. *pmdUpdate()* must be called at least once before this function.

The structure of the data depends on the data source.

Parameters:

hnd Handle of the connection.

data Pointer to memory to contain the data.

maxLen Number of bytes available in the memory block.

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetSourceDataDescription (PMDHandle *hnd*, PMDDataDescription * *dd*)

Get the source data description structure of the current frame. *pmdUpdate()* must be called at least once before this function.

Parameters:

hnd Handle of the connection.

dd Pointer to a PMDDataDescription structure to be filled

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetSourceDataSize (PMDHandle *hnd*, size_t * *size*)

Get the size of the source data from the current frame. *pmdUpdate()* must be called at least once before this function.

Parameters:

hnd Handle of the connection.

size Pointer to memory to contain the size.

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetValidIntegrationTime (PMDHandle *hnd*, unsigned * *result*, unsigned *idx*, Proximity *w*, unsigned *t*)

Get a valid integration time of the camera. This function is used to determine which integration time settings a camera supports. It is possible to search for valid integration times that best match a user-specified time.

Parameters:

hnd Handle of the connection.

result Returns a valid integration time according to specification in microseconds.

idx Index of the integration time to be queried (starting with 0).

w Where to look for a valid integration time. Either *CloseTo*, *AtLeast* or *AtMost*.

t The desired integration time in microseconds. The result will be close to this value.

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetValidModulationFrequency (PMDHandle *hnd*, unsigned * *result*, unsigned *idx*, Proximity *w*, unsigned *f*)

Get a valid modulation frequency of the camera. This function is used to determine which modulation frequency settings a camera supports. It is possible to search for valid modulation frequencies that best match a user-specified frequency.

Parameters:

hnd Handle of the connection.

result Returns a valid modulation frequency according to specification in Hz.

idx Index of the modulation frequency to be queried (starting with 0).

w Where to look for a valid modulation frequency. Either *CloseTo*, *AtLeast* or *AtMost*.

f The desired modulation frequency in Hz. The result will be close to this value.

Returns:

PMD_OK on success, errorcode otherwise

int pmdSetIntegrationTime (PMDHandle *hnd*, unsigned *idx*, unsigned *t*)

Set an integration time of the camera

Parameters:

hnd Handle of the connection.

idx Index of the integration time to be set (starting with 0).

t Integration time in microseconds.

Returns:

PMD_OK on success, errorcode otherwise

int pmdSetModulationFrequency (PMDHandle *hnd*, unsigned *idx*, unsigned *f*)

Set a modulation frequency of the camera.

Parameters:

hnd Handle of the connection.

idx Index of the modulation frequency to be set (starting with 0).

f Modulation frequency in Hz.

Returns:

PMD_OK on success, errorcode otherwise

int pmdUpdate (PMDHandle *hnd*)

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

Parameters:

hnd Handle of the connection.

Returns:

PMD_OK on success, errorcode otherwise

int pmdSourceCommand (PMDHandle *hnd*, char * *result*, size_t *len*, const char * *cmd*)

Issue a device-dependent command.

Parameters:

hnd Handle of the connection.

result Pointer to memory to contain the result of the command.

len Number of bytes available in the memory block.

cmd String identifying the command.

Returns:

PMD_OK on success, errorcode otherwise

int pmdProcessingCommand (PMDHandle *hnd*, char * *result*, size_t *len*, const char * *cmd*)

Issue an arbitrary command to the processing plugin.

Parameters:

hnd Handle of the connection.

result Pointer to memory to contain the result of the command.

len Number of bytes available in the memory block.

cmd String identifying the command.

Returns:

PMD_OK on success, errorcode otherwise

int pmdGetLastError (PMDHandle *hnd*, char * *dest*, size_t *len*)

Get a textual description of the last error associated with the given handle.

Parameters:

hnd Handle of the connection.

desc Description of the last error.

len Number of bytes available for the error string.

Returns:

PMD_OK on success, errorcode otherwise

PMDTechnologies GmbH
Am Eichenhang 50

57076 Siegen
Germany

Phone +49(0)271 / 238 538- 800

Fax +49(0)271 / 238 538- 809

info@PMDTec.com

www.PMDTec.com