

# **plFGI**

## **MatLAB Interface for PixeLINK FireWire Cameras**

### ***User Manual & Technical Guide***

Last updated: September, 2002

**MatLAB Interface  
for  
PixeLINK FireWire Cameras**

*User Manual & Technical Guide*

M.A.E.Bakker ([maarten@panic.et.tudelft.nl](mailto:maarten@panic.et.tudelft.nl))  
L.I.Oei ([L.I.Oei@its.tudelft.nl](mailto:L.I.Oei@its.tudelft.nl))  
September, 2002

**Copyright © 2002 Czech Technical University in Prague, Czech Republic  
M.A.E.Bakker, L.I.Oei**

# Index

<b>PART I</b>	<b>USER MANUAL.....</b>	<b>1</b>
<b>1</b>	<b>GENERAL INTRODUCTION.....</b>	<b>1</b>
1.1	SHORT DESCRIPTION OF MATLAB.....	1
1.2	SHORT DESCRIPTION OF THE PIXELINK CAMERA.....	1
1.3	SHORT DESCRIPTION OF PLFGI.....	1
1.4	HARDWARE AND SOFTWARE REQUIREMENTS.....	2
<b>2</b>	<b>INTRODUCTION TO PLFGL.....</b>	<b>3</b>
2.1	ABOUT DATA TYPES .....	3
2.2	ABOUT THE CAMERA'S SERIAL NUMBER.....	3
2.3	THE PLFGI FUNCTIONS.....	4
2.4	NOTES ON PLGRAB.....	8
2.5	NOTES ON PLGET AND PLSET.....	9
2.6	NOTE ON USING HANDLES.....	9
<b>3</b>	<b>WORKING WITH PLFGI.....</b>	<b>10</b>
3.1	GETTING STARTED .....	10
3.2	SOME EXAMPLES.....	10
<b>PART II</b>	<b>TECHNICAL GUIDE.....</b>	<b>12</b>
<b>1</b>	<b>INTRODUCTION TO PLFGI DEVELOPMENT.....</b>	<b>12</b>
1.1	THE PLFGI DISTRIBUTION PACKAGE.....	12
1.2	BUILDING PLFGI.....	12
1.3	GENERAL NOTES ON PLFGI DEVELOPMENT.....	14
<b>2</b>	<b>TECHNICAL DESCRIPTION OF PLFGL.....</b>	<b>15</b>
2.1	STRUCTURE DIAGRAM .....	15
2.2	DESCRIPTION OF PLCLOSE.....	16
2.3	DESCRIPTION OF PLGET.....	16
2.4	DESCRIPTION OF PLGRAB .....	16
2.5	DESCRIPTION OF PLISOPEN.....	18
2.6	DESCRIPTION OF PLOPEN.....	18
2.7	DESCRIPTION OF PLSET .....	19
2.8	DESCRIPTION OF PLDEVICES .....	20
2.9	DESCRIPTION OF PLCREATEDEVICEHANDLE.....	24
2.10	DESCRIPTION OF PLERROR.....	24
2.11	DESCRIPTION OF PLGETVALUE.....	25
2.12	DESCRIPTION OF PLSETVALUE.....	25
2.13	DESCRIPTION OF PLPRINTPOSSIBLEVALUES .....	26
2.14	DESCRIPTION OF PLTYPES .....	26
2.15	GENERAL HINTS ON MODIFYING.....	26
	<b>BIBLIOGRAPHY.....</b>	<b>27</b>

**APPENDIX A PARAMETERS IN THE DEVICE'S HANDLE STRUCTURE.....28**

**APPENDIX B PARAMETERS FOR USE WITH GET AND SET .....29**

**APPENDIX C SOURCE FILE DEPENDENCIES.....31**

**APPENDIX D TOPICS FOR IMPROVEMENT .....32**

# ***PART I      USER MANUAL***

## **1    General introduction**

This manual describes the Matlab interface 'pIFGI' for Vitana PixelINK digital cameras. The name pIFGI stands for 'PixelINK Frame Grabber Interface'. In this document you will find described the functionality of this interface and information about the working conditions. This document can be used as a reference while working with the interface.

### **1.1    *Short description of Matlab***

Matlab is a mathematical program for technical computing. It is used by research institutes and universities all over the world, mostly to solve problems, which involve matrix and vector calculations. It can, for example, be used to make analyses and computations on images, which are stored as matrices. This is the main purpose of the pIFGI software. One of the important features of Matlab is the ability to use external compiled programs from within Matlab, called MEX-files. The pIFGI consists of such external programs. In a windows environment, the native .dll format is used for MEX-files. Variables are passed between Matlab and the MEX-files using mxArray's in which all required variables can be stored and returned.

### **1.2    *Short description of the PixelINK camera***

The PixelINK PL-A6xx camera series consists of megapixel cameras with a CMOS image sensor and integrated image processing electronics. The IEEE-1394 'FireWire' interface is used for communicating with the computer. The cameras are available in both in colour and black & white versions. Some of the camera models are equipped with a machine vision connector to connect to external shutters etcetera.

### **1.3    *Short description of pIFGI***

The pIFGI is a simple interface for grabbing images directly from the PixelINK camera into a Matlab mxArray. It follows the concept of the existing Data Translation FGI closely. Several picture formats are supported. Currently, the interface features grabbing an image from a videostream and changing most settings of the camera. To accomplish this, functionality of the PixelINK API library - which is supplied with the camera - is used.

## **1.4 Hardware and software requirements**

To use this interface, the following is required, newer versions may also work, older versions probably won't give good results:

- A Vitana PixeLINK PL-A6xx series camera.
- PixeLINK drivers / API version 3.0
- A PC with at least a Pentium II processor, 64MB of RAM and a IEEE-1394 'FireWire' interface.
- MS Windows 98SE, Windows ME, Windows 2000 SP1 or Windows XP.
- Mathworks Matlab 6.

## 2 Introduction to pIFGI

In this chapter, some background information and a description of the pIFGI functions are given..

### 2.1 About data types

The following Matlab data types are used when working with the pIFGI:

- **string**  
A vector of which the components are ASCII codes. Used with the pIFGI for passing names of camera parameters to plGet and plSet.
- **double**  
The default data type, used by Matlab to store numbers in the double precision floating point format. Range from 1.7E-308 to 1.7E+308. Used with the pIFGI for passing all variables that are marked U32, int or double. The (U32) and (int) designations used below in this document are only used for indicating the allowed range (32 bit unsigned or signed integer) of the passed variable.
- **uint8**  
8 bit unsigned integer. Used by the pIFGI within an mxArray for returning a grabbed image.
- **uint16**  
16 bit unsigned integer. Used by the pIFGI within an mxArray for returning a grabbed image.
- **struct**  
An array containing fieldnames (string) and their values (any type). Used by pIFGI for the device handle structure.
- **mxArray**  
The default way of storing arrays of any data type in Matlab. Data is stored in column major format. Used by the pIFGI for returning a grabbed image and for storing the device's handle structure.

### 2.2 About the camera's serial number

The camera's serial number is used as a unique identifier for opening and using the camera. The PixelINK camera we used for testing the pIFGI had as many as 3 different serial numbers. One serial number on a sticker on the board the image sensor was mounted on (this number is returned in the HardwareInfo structure, and also used in the PixelINK application that comes with the camera). Another serial number is on the board on which the other electronics of the camera are located. The third serial number is returned when you get the serial number using the function `\PimMegaGetSerialNumber'`. This last number is the one we decided to use as 'official' serial number, on basis of the PixelINK documentation suggesting this. If you don't know what serial number the camera you want to use has, call plOpen with a non-existing number to get a listing of the serial numbers of all cameras that are attached to the computer.

## 2.3 The plFGI functions

The plFGI supports the following functions (in alphabetical order):

- **plClose** closes a PixeLINK camera.
- **plGet** gets the value of a camera or image processing parameter or the complete handle structure of the camera.
- **plGrab** grabs an image from a PixeLINK camera.
- **plIsOpen** checks if a PixeLINK camera is already open.
- **plOpen** opens a PixeLINK camera and return its handle structure or only returns its handle if the camera was already open.
- **plSet** sets the value of a camera or image processing parameter or shows the possible parameters or values.

Nota bene: In Matlab, the function names are not case sensitive, although it may be easier to read back code if cases are used.

Below, descriptions are given of each of the plFGI functions callable from Matlab. For a description of plDevices, which is used as an internal function, we refer to the technical documentation. In the descriptions below, prhs[0] is used to indicate the first input parameter, prhs[1] for the second input parameter, prhs[2] for the third, etcetera. Similarly, plhs[0] is used to indicate the (first) return value of the MEX-function.

- *plClose*

DLL name	plClose.dll
Syntax	plClose(handle) or plClose(serialnumber)
Description	plClose closes an open PixeLINK camera device.
Input	prhs[0] = (struct) handle of the device, or (U32) serial number of the device
Remarks	It will be verified whether the input is a valid handle or number. plClose will then try to close the device. If the device was not open, does not exist or returns an error code, an error message is printed and plClose exits.
Output	No return value, the device is closed and the internal list of open devices is updated.
Examples	plClose(m); plClose(75122);



- *plGet*

DLL name	plGet.dll
Syntax	plGet(handle), plGet(handle, parametername), plGet(serialnumber) or plGet(serialnumber, parametername)
Description	plGet returns the device's up-to-date handle if 1 argument is given, or returns the value of the given parameter if 2 arguments are given.
Input	prhs[0] = (struct) handle for the device, or (U32) serial number of the device prhs[1] = (string) name of the parameter (optional)
Remarks	It will be verified whether the input is a valid handle or number. plGet will then try to get the requested parameter. If the parameter is not valid, the device is not open, does not exist or returns an error code, and error message is printed and plGet exits.
Output	plhs[0] = (struct) handle for the device (with 1 input argument) plhs[0] = value(s) of the given parameter (with 2 input arguments)
Examples	m = plGet(m); value = plGet(m, 'Timeout'); m = plGet(75122); value = plGet(75122, 'Timeout');

- *plGrab*

DLL name	plGrab.dll
Syntax	imagematrix = plGrab(handle) or imagematrix = plGrab(serialnumber)
Description	plGrab grabs a frame from the PixeLINK camera device and places it into an mxArray.
Input	prhs[0] = (struct) handle for the device, or (U32) serial number of the device
Remarks	It will be verified whether the input is a valid handle or number. plGrab will then try to capture and return an image. If there is not enough free memory, the device is not open, does not exist or returns an error code, and error message is printed and plGrab exits.
Output	plhs[0] = (mxArray) grabbed frame
Examples	im = plGrab(m); im = plGrab(75122);

- *plIsOpen*

DLL name	plIsOpen.dll
Syntax	plIsOpen(handle) or plIsOpen(serialnumber)
Description	plIsOpen checks whether the device has been opened. It returns 0 if the device has not been opened, and 1 if it has been opened.
Input	prhs[0] = (struct) device handle, or (U32) serial number of the device
Remarks	It will be verified whether the input is a valid handle or number. If the camera was unplugged from the computer without being closed first, this function will still show it as being open.
Output	plhs[0] = (double) 0 if the camera device is not open 1 if the camera device is open
Examples	boolean = plIsOpen(m); boolean = plIsOpen(75122);

- *plOpen*

DLL name	plOpen.dll
Syntax	handle = plOpen(serialnumber)
Description	plOpen opens a PixeLINK camera device and returns the device's handle. If there is no camera with the given serial number, a list of available serial numbers is printed.
Input	prhs[0] = (int) serial number of the device to be opened
Remarks	It will be verified whether the input is a number. If it is a valid serial number plOpen will try to open the camera device and return a handle structure. If the camera is already open, a warning will be printed, and the handle structure will be returned. If the input was not a number, or the camera could not be opened, an error message is printed. If there is no camera with the given serial number, a list of available serial numbers is printed.
Output	plhs[0] = (struct) handle for the device
Example	m = plOpen(75122);

- *plSet*

DLL name	plSet.dll
Syntax	plSet(handle), plSet(handle, parametername), plSet(handle, parametername, value), plSet(serialnumber), plSet(serialnumber, parametername), or plSet(serialnumber, parametername, value)
Description	plSet sets the value of a given parameter. When 1 argument is given a list of all possible parameters is given. When 2 arguments are given, the possible values of the given parameter are given. When called with 3 or more arguments, the device's parameter given in the second argument will be set to the value(s) given in the other argument(s).
Input	prhs[0] = (struct) handle for the device, or (int) serial number of the device prhs[1] = (string) name of the parameter (optional) prhs[2...n] = value of the given parameter (optional)
Remarks	It will be verified whether the input is a valid handle or number. If it is not, an error message will be printed. plSet will then try to write the specified parameter. If the parameter does not exist, the value is out of range under the given circumstances or if the device returns an error code, an error message is printed and plSet exits.
Output	A list of all possible parameters is printed on the screen (with 1 input argument) The possible values of the parameter are printed on the screen (with 2 input arguments)
Examples	plSet(m); plSet(m, 'Timeout'); plSet(m, 'Timeout', value); plSet(75122); plSet(75122, 'Timeout'); plSet(75122, 'Timeout', 1024); plSet(m, 'SubWindowSize', 'PCS2112_NO_DECIMATION', 480, 640);

## 2.4 Notes on plGrab

The properties of the grabbed image depend on several parameters. For most parameter settings, a good overview is given in the PixelLINK manuals. Two parameters are not covered directly by those manuals: `GrabColorConversion` and `GrabOutputType`.

The setting of `GrabOutputType` (default: **RAW**) determines what image format should be returned. The following settings are possible:

- **RAW** returns the image exactly as produced by the camera (row major array, one position for every pixel). Colour images are returned as a Bayer pattern.
- **IMAGE** returns an mxArray that is suitable for direct viewing with the Matlab image command. In case of colour images, the Bayer pattern is first translated into a suitable RGB format. The conversion method is determined by the setting of `GrabColorConversion`.
- **RGB24** returns a single column mxArray that is filled according to the standardised RGB24 format. It can be used for example to save pictures to a file, as many image file formats are in one way or the other based on this format. In case of colour images, the Bayer pattern is first translated into a suitable RGB format. The conversion method is determined by the setting of `GrabColorConversion`.

The setting of `GrabColorConversion` (default: **BAYER\_3BY3\_COLOR**) determines, in case of colour images that are returned as **IMAGE** or **RGB24**, what method should be used to convert the Bayer pattern of the camera to a suitable image. Possible values are:

- |                                 |                                       |
|---------------------------------|---------------------------------------|
| • <b>BAYER_2BY2_COLOR</b>       | Fastest                               |
| • <b>BAYER_3BY3_COLOR</b>       | Fast—Default                          |
| • <b>BAYER_3BY3GGRAD_COLOR</b>  | Best quality for real-time            |
| • <b>BAYER_2PASSGRAD_COLOR</b>  | Best for captured images              |
| • <b>BAYER_2PASSADAPT_COLOR</b> | Best for captured images              |
| • <b>BAYER_VARGRAD_COLOR</b>    | Best for captured images              |
| • <b>BAYER_2BY2_MONO</b>        | Fastest (converts to monochrome)      |
| • <b>BAYER_3BY3_MONO</b>        | Best quality for real-time (to mono.) |
| • <b>BAYER_ADAPT_MONO</b>       | Best for captured images (to mono.)   |
| • <b>BAYER_NO_CONVERSION</b>    | No Bayer conversion                   |

The captured image is returned in an mxArray. To arrange things in a bit more convenient way, let's define 'Height' to be  $(\text{SubWindowSize.Height} / \text{SubWindowSize.Decimation})$  and define 'Width' to be  $(\text{SubWindowSize.Width} / \text{SubWindowSize.Decimation})$ . Then, the image is returned as described below:

- For **RAW** images an mxArray of size [Width, Height] filled with uint8 or uint16 is returned.  
Whether the contents of the mxArray consist of uint8 or uint16 depends on the setting of the `DataTransferSize` parameter.
- For **IMAGE** images an mxArray of size [Height, Width, 3] filled with uint8 is returned. This is true for colour and black & white images, both are returned in the Matlab colour image format.
- For **RGB24** format, an array of size [Height \* Width \* 3, 1] of uint8 is returned.

## **2.5 Notes on plGet and plSet**

Care should be taken on the following points:

- Not all combinations of parameter values are valid, for example setting ImagerClocking to 0x00 (16MHz, no division) while DataTransferSize is set to 'DATA\_16BIT\_SIZE' will result in an error message.
- Some values have a certain granularity, for example setting SubWindowSize.Height to 475 will result in a value of 472 being set.
- Parameter names are case sensitive.
- For a complete list of parameters, refer to Appendix B of this document and to the PixeLINK documentation.

## **2.6 Note on using handles**

The handle obtained when using plOpen will of course not automatically be updated to reflect the recent settings of the device. This is not important for the pIFGI functions, as they only use non- settable parameters from the handle. When you want to view the most recent settings, please use plOpen or plGet to obtain the most recent information. It is even possible to make use of the construction `m = plGet(m);` to update the handle to reflect changed settings, if one wants to.

## 3 Working with pIFGI

### 3.1 Getting started

To be able to use pIFGI functions, Matlab should be able to find and load the right files. First make sure the PixelINK API library file 'PimMegaApi.dll' is in a directory that is in the Windows path. If it is not, the following error message will occur whenever you try to use a pIFGI function:

```
Unable to load mex file: g:\fgi\bin\plOpen.dll.  
The specified module could not be found.  
  
??? Invalid MEX-file
```

This can be quite confusing, as it doesn't directly point to the actual cause of the error.

Also, Matlab should be able to find the pIFGI executable files. If the pIFGI distribution is stored on the network path Y:\Software\pIFGI for example, the following commands should be typed into the Matlab command window:

```
system_dependent RemotePathPolicy Reload;  
system_dependent RemoteCWDPolicy Reload;  
addpath Y:\software\pIFGI\bin;
```

To view black and white RAW images with the Matlab image function, make sure the right colour map is set by typing:

```
colormap(gray(255));
```

When viewing raw images in 16-bit format, it should be taken into consideration that values of 255 and above will be interpreted as maximum white level by the Matlab image function. To be able to view the image correctly, use some scaling.

### 3.2 Some examples

The following example will open a camera having serial number 75122, get the current value of the gamma correction parameter displaying it on the screen, set the gamma correction parameter, then get the new value of the gamma correction parameter displaying it on the screen, then close the camera. To have the parameter not display on the screen, just put a ; behind the plGet lines.

```
m = plOpen(75122);  
g = plGet(m, 'Gamma')  
plSet(m, 'Gamma', 0.65);  
g = plGet(m, 'Gamma')  
plClose(m);
```

The following example will open a (black & white) camera having serial number 75122, grab an image using the default settings (480 rows, 640 columns, 8-bit raw image), display it and then close the camera.

```
m = plOpen(75122);
im = plGrab(m);
colormap(gray(255));
image(im');
plClose(m);
```

The following example will open a (colour) camera having serial number 98765, set the image size to 200 rows, 320 columns, decimation factor 2, set the GrabOutputType, set the GrabColorConversion and capture an image. Then it will set the GrabColorconversion to another value and capture another image. It will then close the camera, display the first image, wait for a key to be pressed, then display the second image.

```
m = plOpen(98765);
plSet(m, 'SubWindowSize', 'PCS2112_DECIMATE_BY_2', 200, 320);
plSet(m, 'GrabOutputType', 'IMAGE');
plSet(m, 'GrabColorConversion', 'BAYER_VARGRAD_COLOR');
im = plGrab(m);
plSet(m, 'GrabColorConversion', 'BAYER_ADAPT_MONO');
im2 = plGrab(m);
plClose(m);
image(im);
waitforbuttonpress;
image(im2);
```

The following example will capture 100 images and display them on the screen one after the other as in a movie. It is presumed that the camera is already open with handle h, that all parameters are set correctly and that it will be closed later. Note: the tic and toc Matlab functions can be used to measure the average frame rate. To measure the pure grabbing speed without the time Matlab needs to display the image, just leave out the line with `image(i')`;

```
for im=1:100
    im = plgrab(m);
    image(im');
end
```

# **PART II      TECHNICAL GUIDE**

## **1 Introduction to pIFGI development**

### **1.1 The pIFGI distribution package**

The pIFGI distribution package currently consists of 3 folders:

/bin     contains all .dll and .m executable files  
 /doc     contains all documentation  
 /src     contains the source code and project files used for building the pIFGI

In every folder and subfolder, a readme.txt file is present, in which all files that should be in the folder are listed, along with the purpose they are for.

### **1.2 Building pIFGI**

pIFGI was built using the Microsoft Visual C++ 6.0 Integrated Development Environment. Every MEX-function has its own subfolder, containing the necessary files for building it. Double click on the appropriate workspace file (.dsw) to open the project. All relevant header and source files (see Appendix C for an overview of the dependencies) are included in the workspace. Before opening the project, you may want to check the pathnames used in the various project configuration files, or just place all files in the same folders as used during development:

G:\NewFGI\src\	pIFGI source files and project folders
Y:\software\framegrag\pixelink\api\	PixelINK libraries
Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\	Matlab libraries

At some point, you may want to create a new project, below is the procedure that was used to create pIFGI:

- Create a new DLL project (File → New → Projects → Win32 Dynamic-Link Library → A DLL that exports some symbols)
- Throw away StdAfx.cpp, StdAfx.h and Projectname.h
- Disable precompiled headers (Project → Settings → C/C++ → Category Precompiled Headers → Not using precompiled headers)
- Link the necessary libraries (Project → Settings → Link → Category General → Object/library modules):
  - Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib
  - Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib
  - Y:\software\framegrag\pixelink\api\pimmegaapi.lib
- Export the mexFunction (Project → Settings → Link → Category General → Project Options): /export:mexFunction
- Implement Projectname.cpp, starting with the following template:



---

```
#define WIN32_LEAN_AND_MEAN
/* This excludes rarely-used stuff from Windows headers. We don't
 * know what that means exactly, but it is generated by MS Visual C++
 * when you create a project and we just copied it.
 */

#include <windows.h>

#include "Y:\soft95\matlab6\extern\include\mex.h"
/* with Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmx.lib
 * and Y:\soft95\matlab6\extern\lib\win32\microsoft\msvc60\libmex.lib
 */

#include "Y:\software\framegrag\pixelink\api\pimmegaapiuser.h"
/* with Y:\software\framegrag\pixelink\api\pimmegaapi.lib */

#ifdef __cplusplus
    extern "C" {
#endif

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    /* implement the MEX function here */
}

#ifdef __cplusplus
    } /* extern "C" */
#endif
#endif
```

---

**Code example 1 *Template for a pIFGI MEX-function***

The libraries of Matlab version 6.1 and the PixelINK Camera API Release 3.0 were used for the development of the pIFGI.

### 1.3 General notes on pIFGI development

The MEX-files are implemented as C++ files (instead of C), because the PixeLINK Camera API is implemented in C++ and it uses language constructions, which cannot be used with C files. With the `extern "C"` command, the `mexFunction` is exported as a C function, so it will be possible to call the function from within Matlab, just like normal C MEX-files.

The PixeLINK dll is load-time linked. This means that when the MEX-file is loaded into memory, the PixeLINK dll is loaded at the same time. Therefore, to use the MEX-files, it is necessary that the file 'PimMegaApi.dll' is in the *Windows*-path (not the Matlab-path). Otherwise Matlab will give an error-message like:

```
Unable to load mex file: g:\fgi\bin\plOpen.dll.  
The specified module could not be found.  
  
??? Invalid MEX-file
```

This can be quite confusing, as it doesn't directly point to the actual cause of the error.

### 1.4 Testing

During development, frequent testing helped to find bugs in pIFGI. After the program reached a stable working state, a few other tests were run in the limited time that was still available.

On suggestion of Assistant Professor Smutný, it was tested how much time the capturing of multiple frames takes. This could possibly be used as an indication of the quality of the code (memory leaks and resources that are not released would have become apparent this way). Also, the performance of the code, the PixeLINK API and the PixeLINK camera could be verified.

First, some basic tests were performed, showing that capturing the images resulted in a frame rate a bit lower than that of the demo application, but that could be contributed to the overhead of calling functions from Matlab, and the lack of loop optimisation in pIFGI. When testing different resolutions, the effect on overall capturing speed of the camera itself became apparent: reducing the number of columns in the picture did not affect the capture time very much, as the camera scans row by row. Reducing the number of rows reduced the time needed for capturing accordingly. As a last experiment, the exposure time was varied. Strangely enough, capturing 100 images would take less time than 100 times the exposure time.

As there was not much time to take a closer look at these tests and do some more advanced testing, this might be a good subject for future examination.

In Appendix E, the test results are included.

## 2 Technical description of plFGI

This chapter describes the implementation of the plFGI, the structure of which is similar to the original FGI system for the DataTranslation frame grabbers. The system is composed of several MEX-functions, which are callable from Matlab. It supports functionality to open and close a device, check whether a device is opened, get and set a number of parameters and grab an image from the camera. This chapter starts with an overview of the structure of the plFGI, after that follow descriptions of the individual source files. For a functional specification from the user's point of view, please refer to the first section of this document.

### 2.1 Structure diagram

The system is composed of several functions (plIsOpen, plGrab, plGet, plOpen, plClose and plSet), which can be called from Matlab. These functions call the appropriate functions from the PixelINK Camera Application Programming Interface (API). plDevices maintains the 'open device array', in which the device ID's (handles to open devices) are stored. In the diagram below, the relations between the internal and external plFGI functions are shown. An arrow indicates that a function is called by another function. The ↔ arrow between plDevices and the open device array means that data is exchanged in two directions.

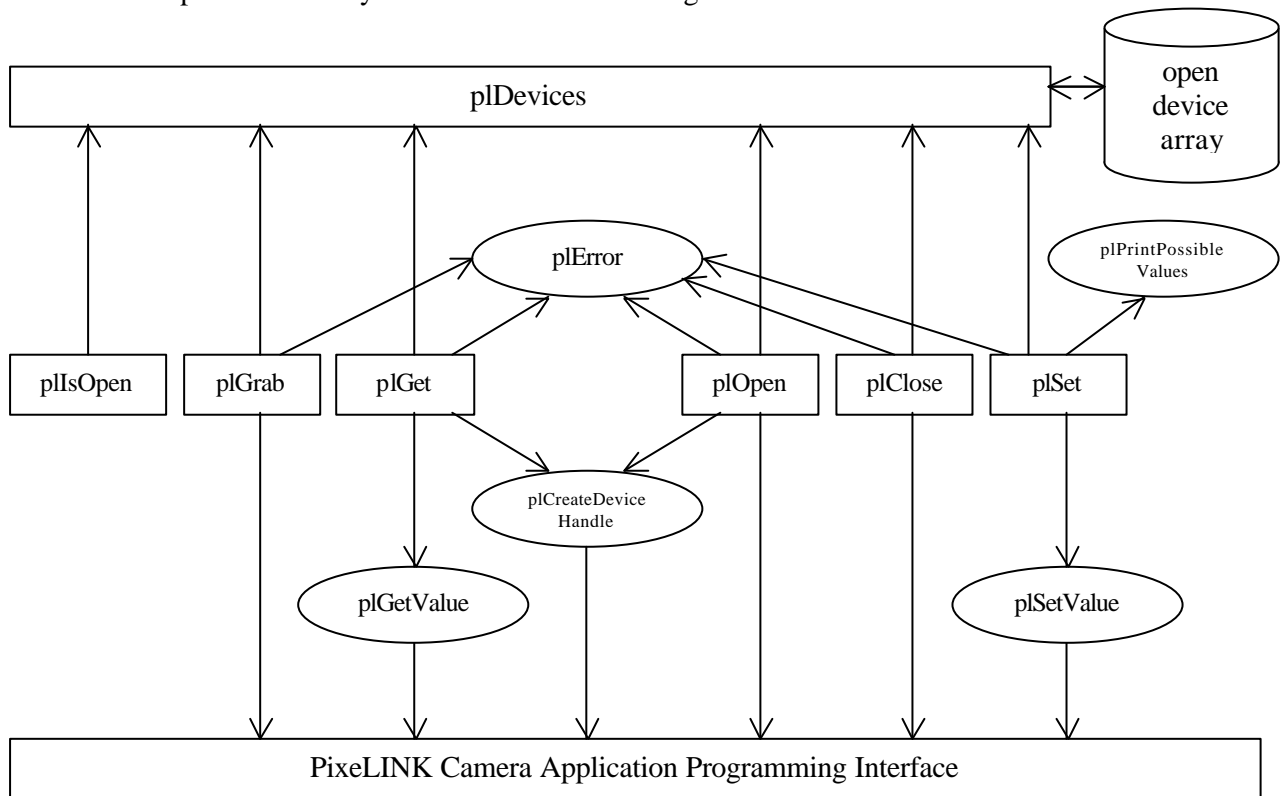


Figure 1 Structure of plFGI

## 2.2 Description of *plClose*

*plClose* is the MEX-function which takes care of closing an opened PixeLINK device. First it is tested whether one and only one argument was given, using 'if (nrhs == 1)'. If not, the else part is executed, printing an error message with `mexPrintf` and exiting with `mexErrMsgTxt`. If the argument is a device handle structure, the device's serial number is extracted from it. If the argument is a number, this is treated as the serial number. If the argument is something else, again an error message will be shown and the function will exit.

Now, it can be assumed that a serial number is present. It is checked whether the device is registered as open, with a call to `plDevices`. If it is not open, the else-part of the 'if (isOpen)'-construction is executed and the function terminates with an error message. If it is open, again a call to `plDevices` is made, this time to obtain the PixeLINK API's device-ID. Then the PixeLINK API function 'pimMegaUninitialize' is made to close the device. When an error occurs closing the device, the standard error handling code in `plError` returns a non-false value and the *plClose* function exits. If all went well, the device should now be closed and a call to `plDevices` is made, to update the list of open devices.

## 2.3 Description of *plGet*

*plGet* is the MEX-function which is used for obtaining the value of a parameter, a set of parameters, or a complete device handle for a PixeLINK device. First, it is tested whether 1 or 2 arguments were given. If not, the else part of the check is executed, resulting in an error message and the termination of *plGet* by using the Matlab call `mexErrMsgTxt`. The first argument is either a serial number, or a device handle structure. If it is a double, it is stored in the `serialNumber` variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the `serialNumber` variable (without further checking). If the first argument is another data type, an error message is given and *plGet* is terminated.

After the serial number has been obtained, it is checked whether the device is open by calling `plDevices`. If not, an error message is given and *plGet* is terminated. If the device is open, it is determined whether one or two arguments were given when calling *plGet*. If only one argument was given, a call to the `plCreateDeviceHandle` subroutine is made and the result of that is returned. If two arguments were given, it is checked whether the second argument is a string. If it is not, an error message is given and *plGet* is terminated. If it is, the name of the parameter is copied into a string variable and the subroutine `plGetValue` is called and the result of that is returned.

## 2.4 Description of *plGrab*

*plGrab* is the MEX-function that grabs a frame from the PixeLINK device and takes care of the required postprocessing before delivering it to Matlab. Currently only one calling syntax is supported, so it is checked whether there is exactly one input argument and one output argument. If not, the else part of the check is executed, resulting in an error message and the termination of *plGrab* by using the Matlab call `mexErrMsgTxt`. The input argument is either a serial number, or a device handle structure. If it is a double, it is stored in the `serialNumber` variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the `serialNumber` variable (without further checking). If the first argument is another data type, an error message is given and *plGrab* is terminated.

The serial number is now used to obtain a PixelINK API device ID by calling `plDevices`. Using `PimMegaStartVideoStream`, the video stream is opened. Behind every call to a PixelINK API function, `plError` is used to check for an error. If an error occurs while the videostream is supposed to be active, `PimMegaStopVideoStream` is called before terminating.

Now, it is checked whether the camera is in 'video mode', if it is not, a warning is printed to the screen and the camera is set to video mode. Video mode is used because still mode requires special lighting conditions or a shutter to control the exposure time<sup>1</sup>. After this the program gets some camera parameters and settings:

- Get `imagerType` (colour or monochrome)
- Get `dataTransferSize` (8 bit or 16 bit format)
- Get decimation, width and height of the current subwindow<sup>2</sup>
- Calculate `pixelWidth`, `pixelHeight` according to the rules in the PixelINK Megapixel FireWire Camera Developer's Manual<sup>3</sup>
- Get `GrabColorConversion` and `GrabOutputType` parameters using `plDevices`

Using the above parameters, 3 arrays must be set up:

- The capture array, in which the PixelINK API stores the raw image returned by the camera. This array has the same number of elements (either 8 or 16 bit unsigned integers) as the image has pixels (`pixelWidth * pixelHeight`) and is arranged as an mxArray of [`pixelWidth`, `pixelHeight`].
- The matlab array, which is returned to Matlab. In case of a RAW image, this array has the same size and arrangement as the capture array. In case of a 'Matlab IMAGE', it is an array of unsigned 8 bit integers three times the size of (`pixelWidth * pixelHeight`). It is arranged as an mxArray of [`pixelHeight`, `pixelWidth`, 3]; In memory, this means that 3 column-major arrays of [`pixelHeight`, `pixelWidth`] are stored consecutively (like this: RRR...GGG...BBB...). In case of an RGB24 image, an array of size [`Height * Width * 3`, 1] of `uint8` is returned.
- The RGB24 array is used as an intermediate array to store the result of `PimMegaConvert` in case of a non-RAW image, before it is converted to fit into the matlab array. Its size is always (`3 * pixelWidth * pixelHeight`) and it is arranged as a normal C-type row-major array (like this: RGBRGBRGB...).

First, a distinction between the various 'GrabOutputType' settings is made, using a case construction. The size and dim variables are set to the required size and dimensions of the matlab array. After that, the sizes of the capture array and the RGB24 array are defined. A case construction is used to allocate memory for the matlab and capture arrays, differentiating between 8 and 16 bits data size. After this, it is checked whether an error occurred, if so the program is terminated. If everything went all right, the pointer to the matlab array is set. When needed, memory is allocated for the RGB24 array.

Now, the image is captured by making a call to `PimMegaReturnVideoData` and the videostream is closed using `PimMegaStopVideoStream`.

For postprocessing, two possibilities are distinguished using an if/else construction.

- The first possibility occurs in case of a RAW image: An 8 bit RAW image is copied into the matlab array using `memcpy`, a 16 bit RAW image is converted to have all 10

---

<sup>1</sup> [Vitana, 2002-1], §1.5.5

<sup>2</sup> [Vitana, 2002-1], p.92, 'PimMegaSetSubWindow'

<sup>3</sup> [Vitana, 2002-1], p.92, 'PimMegaSetSubWindow'

significant bits in the right order, according to the PixelINK Megapixel FireWire Camera User's Manual<sup>4</sup>: The 2 most significant bits of the first byte, are shifted to be the 2 least significant bits. The second byte is shifted 2 bits to the left (multiplied by 4), then the two bytes are added (equivalent to a logical 'and' operation).

- The second possibility occurs in case of an IMAGE or RGB24 output format. Depending on 8 versus 16 bits and black/white versus colour, one of the PimMegaConvert\* functions is called to convert the captured image to a 24 bits per pixel image. For colour images, the GrabColorConversion parameter is used to determine the algorithm to be used by the PimMegaConvert\* function. The result of PimMegaConvert\* is stored into the RGB24 array. If the GrabOutputType is set to RGB24, the RGB24 array is directly copied into the matlab array, using memcopy. In case of an IMAGE type, however, the RGB24 array must be transposed in order for Matlab to display it properly. This is done using two nested for loops. Within the for loops the bytes for R, G and B are copied to the appropriate locations in the matlab array.

All above mentioned case and if constructions contain a default case resulting in an appropriate error message when the distinguishing variable is not recognised.

- Suggestions for improvement:
  - Implementation of the 5 other ways to call plGrab:
    - moviematrix = plGrab(handle, imgmatrix)
    - [imgmatrix, moviematrix] = plGrab(handle)
    - plGrab(handle, imgmatrix)
    - plGrab(handle, imgmatrix, moviematrix)
    - plGrab(handle, imgmatrix, moviematrix, scaling factor)
  - Addition of RGB48 format for 16-bit captures
  - Testing the plGrab function with a colour camera (this should work)
  - Loop optimisations
  - Capture a RAW image directly into the matlab array.

## 2.5 Description of plIsOpen

plIsOpen is the MEX-function for checking if certain PixelINK device is open.

First, it is tested whether only 1 argument was given. If not, the else part of the check is executed, resulting in an error message and the termination of plIsOpen by using the Matlab call mexErrMsgTxt. The argument is either a serial number, or a device handle structure. If it is a double, it is stored in the serialNumber variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the serialNumber variable (without further checking). If the first argument is another data type, an error message is given and plGet is terminated.

After the serial number has been obtained, it is checked whether the device is open by calling plDevices. The result of the call to plDevices is stored in the return array of plIsOpen.

## 2.6 Description of plOpen

plOpen is the MEX-function which takes care of opening a PixelINK device for use with the other plFGI functions. First it is tested whether the input argument is a number, if it is not, the function terminates printing an appropriate error message using mexErrMsgTxt. Then it is

---

<sup>4</sup> [Vitana 2002-2], Appendix C

checked whether the device is open by calling `plDevices`. If the device is already open, the else-part of the if-construction is executed printing a warning message, then calling `plCreateDeviceHandle` and returning the handle.

If the device was not open, a call to `PimMegaGetNumberDevices` is made to get the amount of available devices. When no devices are attached, `plOpen` exits with an appropriate error message. When one or more devices are attached, a for loop is entered to open every device, store the serial number in the array of available serial numbers, check if the serial number matches the requested serial number, then close it and loop. If the serial number matches the requested serial number, the variable `found` is set to true and the loop is exited by using a break statement.

If `found` is not true, the array containing all available serial numbers is printed to the screen and the function is terminated using `mexErrMsgTxt`.

Finally, the open device array is updated by calling `plDevices`.

- Suggestions for improvement: Use `plError` after the call to `PimMegaGetNumberDevices` instead of non standard error code.

## 2.7 Description of `plSet`

`plSet` is the MEX-function used to change the parameter settings of a PixeLINK device. First, it is checked whether any arguments are given, if not, an appropriate error message is printed to the screen using `mexPrintf`, and the `plSet` is terminated using `mexErrMsgTxt`.

If one or more arguments were given, the first argument should be either a handle structure or a serial number. If it is a double, it is stored in the `serialNumber` variable. If it is a struct containing a field with the name 'SerialNumber', the value of this field is stored in the `serialNumber` variable (without further checking). If the first argument is another data type, an error message is given and `plGet` is terminated. After the serial number has been obtained, it is checked whether the device is open by calling `plDevices`. If not, an error message is given and `plGet` is terminated.

If only one argument was given, a list of parameter names is printed on the screen and `plSet` terminates. If a second argument was given, it is checked whether this is a string. If not `plSet` is terminated with an appropriate error message. If it is a string, it is stored in the `parametername` variable. Depending on whether 2 or 3 arguments were given, `plPrintPossibleValues` is called (2 parameters), or `plSetValue` is called (3 arguments).

- Suggestions for improvement:
  - Implementation of `PimMegaSetOverlayCallBack`
  - Implementation of `PimMegaSetPreviewWindow`
  - Implementation of `PimMegaAutoExposure`

## 2.8 Description of plDevices

plDevices is a MEX-function, just like the 6 functions described above. However, it only contains plFGI internal functionality and should not be called by the user. It is used to keep track of open devices. This is its functional specification:

DLL name	plDevices.dll
Syntax	<code>mexCallMATLAB(nlhs, *plhs[], nrhs, *prhs[], "plDevices");</code>
Description	plDevices maintains the 'open device array', in which the device ID's (handles to open devices) are stored. Also some settings, which are not stored in the PixeLINK Camera API itself, are stored in plDevices.
Input	<p>prhs[0] = (string) name of task to be performed: print, remove, isopen, get, add, getpar, setpar</p> <p>prhs[1] = (U32) serial number of device; Only for remove, isopen, get, add, getpar, setpar</p> <p>prhs[2] = (int) deviceID; Only for add (string) name of the parameter; Only for getpar, setpar</p> <p>prhs[3] = value of the parameter; Only for setpar</p>
Output	<p>plhs[0] = (double) 1 if device open, 0 if closed; Only for isopen (int) deviceID if open, -1 if closed; Only for get parameter value, -1 if not found; Only for getpar</p>

plDevices is implemented as a MEX-function, because of two reasons. First this makes it possible to use the function `mexLock()` to keep the file in memory, even after typing 'clear all' or 'clear mex'. This also eliminates the need of having to do the memory management ourselves. The other reason is that, for testing purposes, it is convenient to be able to call plDevices directly from Matlab. The users of the plFGI, however, should not use this feature.

### The open device array

plDevices maintains the 'open device array'. This is an array of structures of type:

```
struct {U32 serialNumber;
       int deviceID;
       U32 grabColorConversion;
       int grabOutputType;}
```

In this global array of structs the device ID (handle to the device) and the parameters `GrabColorConversion` and `GrabOutputType` are stored for every open device. The serial number of the device is used to uniquely identify the device. This is the number returned by `PimMegaGetSerialNumber`, which is called by `plOpen` immediately after initialising the device.

### The workings of plDevices

For every open device the following information is stored in the open device array described above: The device's serial number, the device's ID which is used by the PixeLINK API to identify the device, the `GrabColorConversion` parameter and the `GrabOutputType` parameter. Another global variable, `deviceCount`, is incremented every time a device is opened and decremented every time a device is closed, thus counting the number of open devices.

First `mexLock` is called, to make sure plDevices (containing the open device array) stays in memory. Then it is checked whether any arguments are given and if the first argument is a



text string. If not, the error message in the else part of the check is displayed and plDevices terminates. If it is, the first argument is stored in the task variable. If there is a second argument, it should always be the serial number of a device. If it is not a number, plDevices is terminated with an error message.

A case construction is used to make a first selection between the different tasks based on the number of arguments given. For every number of arguments from 1 to 4, a different case exists; the default case gives an error message. Inside a case, the different tasks are selected using an if/else if construction together with strcmp. When the strcmp fails, an appropriate error message is given and plDevices terminates.

Although plDevices is not meant to be called directly from Matlab, in the next part the syntaxes and examples are written down as if it were, because this way it is easier to denote and understand the usage of the left-hand and right-hand side arguments.

- **1 argument**

### **print**

Syntax	plDevices('print')
Description	prints a list of open devices, with SerialNumber, DeviceID, GrabColorConversion and GrabOutputType.
Input	prhs[0] = (string) 'print'
Output	None
Example	plDevices('print')

This task is mainly intended for testing and debugging purposes: It prints all entries of the open device array to the screen, one device per line, by using a for-loop (for i = 0 to deviceCount - 1).

- **2 arguments**

### **remove**

Syntax	plDevices('remove', serialnumber)
Description	removes a device from the 'open device array'.
Input	prhs[0] = (string) 'remove' prhs[1] = (U32) serialnumber
Output	None
Example	plDevices('remove', 75122)

This task removes a device from the open device list. Using a for-loop (for i = 0 to deviceCount - 1), the given serial number is compared to every serial number in the open device array. As soon as a matching entry is found, a new for-loop is entered (for t = <number of matching entry> to deviceCount - 1) to overwrite every entry, from the matching entry up to the last entry, with the entry immediately following it (this goes wrong if the last entry is 31; see 'Suggestions for improvement'). After the inner for-loop finishes, deviceCount is decremented.

**isopen**

Syntax	<code>plDevices('isopen', serialnumber)</code>
Description	checks whether a device has been opened.
Input	<code>prhs[0] = (string) 'isopen'</code> <code>prhs[1] = (U32) serialnumber</code>
Output	<code>plhs[0] = (double) 1 if the device is open</code> <code>0 if the device is closed</code>
Example	<code>plDevices('isopen', 75122)</code>

This task returns 0 if the device is not open, 1 if it is open. First the return value is set to 0, then a for-loop (for  $i = 0$  to  $\text{deviceCount} - 1$ ) is entered. The given serial number is compared to every serial number in the open device array. As soon as a matching entry is found, the return value is set to 1, and the loop is left using a break statement.

**get**

Syntax	<code>deviceId = plDevices('get', serialnumber)</code>
Description	returns the device ID (handle to the device) of a camera..
Input	<code>prhs[0] = (string) 'get'</code> <code>prhs[1] = (U32) serialnumber</code>
Output	<code>plhs[0] = (int) deviceId if the device is open</code> <code>-1 if the device is closed</code>
Example	<code>plDevices('remove', 75122)</code>

This task returns the device ID for a given serial number. It works the same way as `isopen`, except that the return value is set to -1, and as soon as a matching entry is found, the return value is set to the appropriate device ID.

- *3 arguments*

**add**

Syntax	<code>plDevices('add', serialnumber, deviceId)</code>
Description	adds a device to the 'open device array'.
Input	<code>prhs[0] = (string) 'add'</code> <code>prhs[1] = (U32) serialnumber</code> <code>prhs[2] = (int) deviceId</code>
Output	None
Example	<code>plDevices('add', 75122, 256085040)</code>

This task adds a new device to the open device array. First, it is checked whether there is still room in the array to store the information of one more open device. If the array is full, an error message is printed and `plDevices` is terminated. If there still is room, the entry after the last entry of the open device array (index:  $\text{deviceCount}$ , because array indexing starts at  $\text{deviceCount} - 1$ ) is filled with the device ID given in the third argument, and suitable default values for the parameters. After that,  $\text{deviceCount}$  is incremented.

**getpar**

Syntax	<code>parameterValue = plDevices('getpar', serialnumber, parametername)</code>
Description	returns the value of the specified parameter.
Input	<pre>prhs[0] = (string) 'getpar' prhs[1] = (U32)  serialnumber prhs[2] = (string) parametername</pre>
Output	<code>plhs[0] = value of the specified parameter</code>
Example	<code>parameterValue = plDevices('getpar', 75122, 'GrabOutputType')</code>

This task reads the value of the given device parameter from the open device array. First, the third argument, which should be a string with the name of the requested parameter, is loaded into the `parName` variable. Then the return value (which should be a double) is initialised. It is determined which parameter should be returned by using `strcmp` in an if/then/else if/else construction. The procedure is the same for every parameter: A temporary boolean variable `t` is used to indicate whether a matching serial number was not found. It is set to 1, then a (for `i = 0` to `deviceCount - 1`) loop is entered. As soon as a matching serial number is found, the requested parameter is stored as return value, `t` is set to 0, and the for loop is terminated using a `break` statement. Finally, if `t` still has value 1, an error message is printed stating that the requested serial number could not be found.

- *4 arguments*

**setpar**

Syntax	<code>plDevices('setpar', serialnumber, parametername, parametervalue)</code>
Description	sets the value of the specified parameter.
Input	<pre>prhs[0] = (string) 'setpar' prhs[1] = (U32)  serialnumber prhs[2] = (string) parametername prhs[3] = value of the specified parameter</pre>
Output	None
Example	<code>plDevices('setpar', 75122, 'GrabOutputType', RAW)</code> (in which case <code>RAW</code> is defined as <code>0x0</code> )

This task writes the given value of the given device parameter in the open device array. Its internal structure is almost the same as that of `getpar`, but the parameter value is written, not read.

- Suggestions for improvement: In 'remove', the inner for loop tries to copy one entry more than needed. When having 32 camera's open, this could result in a segfault. The inner loop for condition should be: (`t = i; t < deviceCount - 1; t++`)

## 2.9 Description of *plCreateDeviceHandle*

*plCreateDeviceHandle* is a subroutine which is linked to the *plOpen* and *plGet* MEX-functions. When given the serial number of a PixelINK device, it will create a Matlab struct matrix containing all the device's parameter names with their respective values. This is the so-called handle structure, which is returned by *plOpen* and *plGet*.

After declaration of the necessary variables and structures, the device-ID for calling the PixelINK API functions is obtained by calling *plDevices*. Then, for every parameter, the appropriate API call is made to obtain its value. If the API call returns the value for unsupported function, the string 'Unsupported' is stored as parameter value. If the API returns any other error code, the string 'Could not get value' is stored as parameter value. No other error checking is being done. If everything went okay, the value of the requested parameter is stored in a temporary variable by the PixelINK API and immediately after that it is stored into the handle structure using *mxSetField*.

The only exception to this are the parameters 'GrabColorConversion' and 'GrabOutputType', which are not stored in the PixelINK API, but in *plDevices*. They are obtained in a similar way to the device-ID by calling *plDevices* using *mexCallMatlab*.

- Suggestions for improvement: Convert some of the if-constructions to switch constructions (for example the one used for *SubWindowSize*).

## 2.10 Description of *plError*

*plError* is an error-checking function which is linked to every MEX-function that uses PixelINK API calls. It is called after every call to such an API function. Required inputs are the PixelINK API return-code returned by the called API function and a string describing in max. 33 characters what the program was doing when the possible error occurred.

First, the string input of the *plError* function is used to create an error message 'The device's API encountered a problem while <string>:'. When the PixelINK API return code (which is defined in *PimMegaApiUser.h*) indicates an error, this string is printed on the screen.

After that, a case construction is used to determine the nature of the PixelINK API return code. When everything is okay, *plError* returns a value of 0 (false). When an error occurred, *plError* prints an appropriate descriptive message on the screen using *mexPrintf* and returns a non-zero value (true). To make it possible to distinguish between critical and non-critical errors, a value of 1 is returned on an error that might not be critical. A value of 2 is returned on errors that are always considered critical. Where appropriate, the program can decide on this information to continue or to terminate.

- Suggestions for improvement: The entire building of the string used in 'The device's API encountered...' can be moved within the following 'if (result != *ApiSuccess*)' condition.

## 2.11 Description of plGetValue

plGetValue is a subroutine which is linked to the plGet MEX-function. When given an mxArray for returning the parameter value, the device's serial number and the name of the parameter to be returned, it will return the parameter's value or values in the given mxArray.

plGetValue is very similar to plCreateDeviceHandle. The most important difference is that not all possible parameters of the device are obtained, but only the parameter or structure of which the name was given. This is accomplished by using an if/elseif construction together with multiple strcmp statements. An obvious difference is that the mxArray m can differ in size according to the parameter that is stored in it.

When an unknown parameter name is given, plGetValue prints a list of known parameter names to the screen.

## 2.12 Description of plSetValue

plSetValue is a subroutine linked to the plSet MEX-function. It needs 4 or more input arguments: the serial number of the device, the name of the parameter to be set, the number of values that are passed, all values that are needed to set the parameter.

First, the serial number is used to obtain the PixelINK device-ID by calling plDevices. For every parameter name, it is checked using an if/strcmp construction whether the right number and type of values are given. If not, a descriptive error message is printed and plSetValue is terminated using a call to mexErrMsgTxt.

After this, another if/elseif/strcmp construction is used to distinguish between the different parameter names. For every parameter name, the passed values are copied into a variable, parsed if necessary and a call is made to the relevant PimMegaSet\* function. For the 'GrabColorConversion' and the 'GrabOutputType' parameters, a call is made to plDevices as these are stored there instead of in the PixelINK API.

- Errors found in the PixelINK API documentation:

[*Vitana 2002-1*], p.92, 'PimMegaSetSubWindow'

(uStartColumn + uNumberColumns) must be less than PCS2112\_MAX\_WIDTH.  
(uStartRow + uNumberRows) must be less than PCS2112\_MAX\_HEIGHT.

'less than' should be 'less than or equal to'.

--

[*Vitana 2002-1*], p.94, 'PimMegaSetSubWindowPos'

(uStartColumn + current number of columns) must be less than PCS2112\_MAX\_WIDTH.  
(uStartRow + current number of rows) must be less than PCS2112\_MAX\_HEIGHT.

'less than' should be 'less than or equal to'.

--

[Vitana 2002-1], p.95, 'PimMegaSetSubWindowSize'

(Current start column + Width) must be less than PCS2112\_MAX\_WIDTH.  
(Current start row + Height) must be less than PCS2112\_MAX\_HEIGHT.

'less than' should be 'less than or equal to'.

### **2.13 Description of plPrintPossibleValues**

plPrintPossibleValues is a subroutine linked to the plGet and plSet MEX-functions. When given the name (string) of a parameter, it will use strcmp in an if/else if construction to determine what should be printed. Then a short description of the given parametername and its possible values is printed to the screen using mexPrintf.

### **2.14 Description of plTypes**

plTypes.h contains all definitions that are used by the plFGI program. Currently, the only values that are defined here are those used to store the 'GrabOutputType' variable.

### **2.15 General hints on modifying**

- When adding or removing a device parameter, this should be done in the obvious places in the following files: plCreateDeviceHandle.cpp, plGetValue.cpp, plPrintPossibleValues.cpp, plSetValue.cpp and plSet.cpp. Also the 'nof' variable and the 'fieldnames' structure in plCreateDeviceHandle should be updated accordingly. Please also don't forget to update the parameter overview in plGetValue.

## Bibliography

- [Vitana, 2002-1] VITANA CORPORATION (2000-2002). *PixeLINK Megapixel FireWire Camera Developer's Manual, Release 3.0 (online version)*. Vitana Corporation, Ottawa, Ontario, Canada.
- [Vitana, 2002-2] VITANA CORPORATION (2000-2002). *PixeLINK Megapixel FireWire Camera User's Manual, Release 3.0 (online version)*. Vitana Corporation, Ottawa, Ontario, Canada.

## Appendix A Parameters in the device's handle structure

DeviceID
BlueGain
CurrentFrameRate
DataTransferSize
Exposure
ExposureTime
Gamma
Gpo
GreenGain
HardwareVersion.ProductID
HardwareVersion.SerialNumber
HardwareVersion.FirmwareVersion
HardwareVersion.FpgaVersion
ImagerChipId
ImagerClocking
ImagerName
ImagerType
MonoGain
PreviewWindowPos.Top
PreviewWindowPos.Left
PreviewWindowSize.Height
PreviewWindowSize.Width
RedGain
Saturation
SerialNumber
SoftwareVersion
SubWindow.Decimation
SubWindow.StartRow
SubWindow.StartColumn
SubWindow.NumberRows
SubWindow.NumberColumns
SubWindowPos.StartRow
SubWindowPos.StartColumn
SubWindowSize.Decimation
SubWindowSize.Height
SubWindowSize.Width
Timeout
VideoMode
GrabColorConversion
GrabOutputType

**Table 1** A list of all parameters that are present in the handle structure of the device



## Appendix B Parameters for use with get and set

Parameter name	Type	Range	Default	Granularity	availability	
					get	set
BlueGain	U32	0 .. 63	31	1	+	+
CurrentFrameRate	FLOAT		-	-	+	-
DataTransferSize	STRING	DATA_8BIT_SIZE, DATA_16BIT_SIZE	8BIT	-	+	+
Exposure	U32	0 .. 2046	2046		+	+
ExposureTime	FLOAT				+	+
Time ChangeClockSpeed	STRING	TRUE, FALSE	FALSE	-	-	+
Gamma	FLOAT	0 .. 1	1		+	+
Gpo	U32	0, 1	0	1	+	+
GreenGain	U32	0 .. 63	31	1	+	+
HardwareVersion	STRING	-	-	-	+	-
ProductID	STRING	-	-	-	+	-
SerialNumber	STRING	-	-	-	+	-
FirmwareVersion	STRING	-	-	-	+	-
FpgaVersion	STRING	-	-	-	+	-
ImagerChipId	U32	-	-	-	+	-
ImagerClocking	STRING	0x00, 0x01, 0x02, 0x80, 0x81, 0x82	0x00	-	+	+
ImagerName	STRING	-	ImagerX	-	+	+
ImagerType	STRING	PCS2112M_IMAGER, PCS2112C_IMAGER	-	-	+	-
MonoGain	U32	0 .. 63	31	1	+	+
OverlayCallBack	-	-	-	-	-	N/P <sup>5</sup>
PreviewColorConversion	STRING	(see table 3)	(table 3)	-	-	+
PreviewWindow	-	-	-	-	-	N/I
PreviewWindowPos	Top Left	LONG LONG	200 200		+	-
PreviewWindowSize	Height Width	U32 U32	0 .. 1024 0 .. 1280	8 8	+	-
RedGain	U32	0 .. 63	31	1	+	+
Saturation	U32		96		+	+
SerialNumber	U32	-	-	-	+	-
SoftwareVersion	U32	-	-	-	+	-
SubWindow	Decimation	STRING	(see table 4)	(table 4)	-	+
	StartRow	U32		0	+	+
	StartColumn	U32		0	+	+
	NumberRows	U32	0 .. 1024	480	8	+
	NumberColumns	U32	0 .. 1280	640	8	+
SubWindowPos	StartRow	U32		0	+	+
	StartColumn	U32		0		
SubWindowSize	Decimation	STRING	(see table 4)	(table 4)	-	+
	Height	U32		480	8	
	Width	U32		640	8	
Timeout	U32		1000		+	+
VideoMode	STRING	STILL_MODE, VIDEO_MODE	VIDEO	-	+	+
GrabColorConversion	STRING	(see table 3)	(table 3)	-	+	+

<sup>5</sup> N/I means 'Not Implemented'

GrabOutputType	STRING	RAW, IMAGE, RGB24	RAW	-	+	+
----------------	--------	-------------------	-----	---	---	---

**Table 2**

BAYER_2BY2_COLOR	Fastest
BAYER_3BY3_COLOR	Fast – Default
BAYER_3BY3GGRAD_COLOR	Best quality for real-time
BAYER_2PASSGRAD_COLOR	Best for captured images
BAYER_2PASSADAPT_COLOR	Best for captured images
BAYER_VARGRAD_COLOR	Best for captured images
BAYER_2BY2_MONO	Fastest (converts to monochrome)
BAYER_3BY3_MONO	Best quality for real-time (to monochrome)
BAYER_ADAPT_MONO	Best for captured images (to monochrome)
BAYER_NO_CONVERSION	No Bayer conversion

**Table 3**

PCS2112_NO_DECIMATION	Original resolution – Default
PCS2112_DECIMATE_BY_2	Resolution / 2, resulting height and width may be rounded down to accommodate the camera’s decimation algorithms
PCS2112_DECIMATE_BY_4	Resolution / 4, resulting height and width may be rounded down to accommodate the camera’s decimation algorithms

**Table 4**

## Appendix C Source file dependencies

plClose	plClose.cpp, plError.cpp, plError.h
plDevices	plDevices.cpp, plTypes.h
plGet	plGet.cpp, plCreateDeviceHandle.cpp, plGetValue.cpp, plError.cpp, plCreateDeviceHandle.h, plGetValue.h, plError.h, plTypes.h
plGrab	plGrab.cpp, plError.cpp, plError.h, plTypes.h
plIsOpen	plIsOpen.cpp
plOpen	plOpen.cpp, plCreateDeviceHandle.cpp, plError.cpp, plCreateDeviceHandle.h, plError.h
plSet	plSet.cpp, plPrintPossibleValues.cpp, plSetValue.cpp, plError.cpp, plPrintPossibleValues.h, plSetValue.h, plError.h, plTypes.h

**Table 5**

## Appendix D Topics for improvement

This is a full list of all topics for improvement on plFGI:

- plGrab: Implementation of the 5 other ways to call plGrab:
  - `moviematrix = plGrab(handle, imgmatrix)`
  - `[imgmatrix, moviematrix] = plGrab(handle)`
  - `plGrab(handle, imgmatrix)`
  - `plGrab(handle, imgmatrix, moviematrix)`
  - `plGrab(handle, imgmatrix, moviematrix, scaling factor)`
- plGrab: Addition of RGB48 format for 16-bit captures
- plGrab: Testing the plGrab function with a colour camera (this should work)
- plGrab: Loop optimisations
- plGrab: Capture a RAW image directly into the left hand array (\*plhs[]) of the mexFunction
- plOpen: Use plError after the call to PimMegaGetNumberDevices instead of non standard error code
- plSet: Implementation of PimMegaSetOverlayCallBack
- plSet: Implementation of PimMegaSetPreviewWindow
- plSet: Implementation of PimMegaAutoExposure
- plDevices: In 'remove', the inner for loop tries to copy one entry more than needed. When having 32 camera's open, this could result in a segfault. The inner loop for condition should be: `(t = i; t < deviceCount - 1; t++)`
- plCreateDeviceHandle: Convert some of the if-constructions to switch constructions (for example the one used for SubWindowSize)
- plError: The entire building of the string used in 'The device's API encountered...' can be moved within the following 'if (result != ApiSuccess)' condition.
- Function to check whether the camera is connected to the computer
- Reset function, to set all parameters to their default value
- Demo M-files
- Additional tests on the performance of plFGI

## Appendix E Test results

Below are the screen printouts of the tests performed on plFGI:

```
>> plset(m,'DataTransferSize','DATA_8BIT_SIZE');
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

    65.8000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 512, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

    29.6600

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 256, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

    17.3000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 1024, 640);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

    55.8000

>> plset(m,'SubWindowSize','PCS2112_NO_DECIMATION', 1024, 320);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

    52.7800

>> plset(m,'ExposureTime',1);
>> tic,for i=1:100,y=plgrab(m);end,t=toc

t =

    51.3500

>> plget(m)

ans =

                DeviceID: 256106896
                BlueGain: 'Unsupported'
    CurrentFrameRate: 'Could not get value'
    DataTransferSize: 'DATA_8BIT_SIZE'
                Exposure: 1028
    ExposureTime: 1.0400
                Gamma: 1
                Gpo: 'Off'
                GreenGain: 'Unsupported'
    HardwareVersion.ProductID: 'PL-A630 Series Monochrome EC Module'
    HardwareVersion.SerialNumber: '12572'
    HardwareVersion.FirmwareVersion: '1.0.3.0'
    HardwareVersion.FpgaVersion: '4'
    ImagerChipId: 805306368
    ImagerClocking: '0x01 External (16Mhz) Division by 2'
    ImagerName: 'Imager0'
    ImagerType: 'PCS2112M_IMAGER (Monochrome Camera)'
    MonoGain: 0
    PreviewWindowPos.Top: 'Could not get value'
    PreviewWindowPos.Left: 'Could not get value'
    PreviewWindowSize.Height: 'Could not get value'
```

```
PreviewWindowSize.Width: 'Could not get value'  
    RedGain: 'Unsupported'  
    Saturation: 'Unsupported'  
    SerialNumber: 75122  
    SoftwareVersion: 16777984  
SubWindow.Decimation: 'PCS2112_NO_DECIMATION'  
    SubWindow.StartRow: 0  
SubWindow.StartColumn: 0  
    SubWindow.NumberRows: 1024  
SubWindow.NumberColumns: 320  
    SubWindowPos.StartRow: 0  
SubWindowPos.StartColumn: 0  
SubWindowSize.Decimation: 'PCS2112_NO_DECIMATION'  
    SubWindowSize.Height: 1024  
    SubWindowSize.Width: 320  
    Timeout: 1000  
    VideoMode: 'VIDEO_MODE'  
GrabColorConversion: 'BAYER_3BY3_COLOR'  
GrabOutputType: 'RAW'
```