Developer Guide

Phantom SDK for Matlab

June 2011

Software release

This document is based on software and SDK release 705 (version 12.0.705.0).

100 Dey Road
Wayne, NJ 07470 USA
+1.973.696.4500
phantom@visionresearch.com

www.visionresearch.com

Contents

1. Introduction

The Matlab SDK is designed to be used by Matlab developers who want to write their own code to control Phantom high speed cameras or to play, analyze and do measurements on cine files.

It contains header files needed to call Phantom SDK functions from Matlab, function wrappers, a simple object-orietended layer and demo scripts. This SDK uses the Matlab interface to shared libraries to call functions from Phantom libraries.

This Matlab SDK was written and tested in `Matlab 2009a`. For older versions of Matlab, the compatibility of the calls with calllib to Phantom dlls functions will be low and crashes may occur. The code should work with all Matlab versions after version 2009a.

This document provides a description of what you will find in the package, setup instructions, information about how to write functions wrappers for calls from Matlab, a description of the object oriented layer used by the demo application written in Matlab.

2. Package content

The package for MatlabSDK contains the following items:

- C header files adjusted for Matlab interface to shared libraries
- Matlab wrappers for a large number of functions from Phantom SDK
- An object-oriented Matlab layer for PhantomSDK
- Matlab demo scripts
  - ✓ A demo application for camera control and cine player: `PhDemoMatlab`
  - ✓ A simple function to read images from a cine file: ReadCineFileImage
- Helper functions and classes
- This documentation file and a copy of all documentation from the Phantom SDK.
- The Phantom dlls and the redist files for the release 705. Newer releases of Phantom dlls will be compatible with 705 so you can replace them with the latest version you have, to get access to new cameras or features.

3. Installation

Do the following steps prior to the use of this SDK.

- If you have a software installer for Phantom camera please install it. You can try to run the applications from this SDK and connect to standard Gigabit Ethernet cameras or to simulated cameras without installing the Phantom cameras software
- Unzip the PhMatlabSDK to your computer in a folder at your choice. Update Phantom dlls if you have a newer version.
- Make sure you have your appropriate Phantom dlls (32 or 64 bit) in the Matlab search path. Do this by File -> Set Path... -> Add Folder… -> browse for your desired version of phantom dlls. Pay attention to the case when there are other directories in Matlab search path that include phantom dlls - remove those search paths.
- Include the PhMatlabSDK directory into Matlab search path. Do this by using File -> Set Path... -> Add with Subfolders... -> then select the PhMatlabSDK directory from your computer.

Note: You can find more information for Matlab search path on Mathworks site: http://www.mathworks.com/help/techdoc/matlab_env/br5tea6-6.html#br5tea6-7

4. SDK description

4.1. C header files for Matlab interface to shared libraries

The C header files are variants of the ones included in Phantom SDK, adjusted for compatibility with Matlab. They are used in Matlab external interface for shared libraries to define the data structures and functions prototypes from Phantom SDK libraries.

The following files are located in the "PhMatlabSDK\PhMatlab\Inc" directory:

- PhConML.h
- PhIntML.h
- PhFileML.h

The constants declared with #define in the C headers are redefined as Constant properties of some classes in:

- PhFileConst.m
- PhConConst.m
- PhIntConst.m

The way the class is declared simulates an enum. The files with the constant definitions are stored in "PhMatlabSDK\PhMatlab\Constants".

4.2. Matlab function wrappers for PhantomSDK

The function wrappers make use of Matlab interface for shared libraries, mainly the calllib function, to call PhantomSDK functions. You can find information on these functions in the "Phantom SDK Reference Manual.pdf".

The function wrappers are located under the "PhMatlabSDK\PhMatlab\" directory in "PhCon", "PhInt" and "PhFile" directories.

4.3. Object-oriented layer for PhantomSDK

The object-oriented layer written for Matlab is build on top of function wrappers for Phantom SDK. It provides basic functionality to setup and control a high speed Phantom camera and access the recordings from camera or files. The classes are similar to those used in Phantom SDK demos as described in the document "PhDemo Developer Guide.pdf" available in the folder of the Doc\PhantomSDK. The scripts for the object oriented layer are located in the "PhMatlabSDK\PhMatlab\OOP" directory.

4.4. Matlab demo scripts

4.4.1. PhDemoMatlab

PhDemoMatlab contains a graphical user interface with the basic controls you need to get images from a camera, to set and get acquisition parameters and image parameters, to record and save cines to files and to playback the recordings either from camera or from file.
PhDemoMatlab script is based on the included object-oriented layer. Its interface structure and functionality is similar to the one used for the two demo versions in the main SDK – one written in C++ (PhDemoCPP) and the other in C Sharp

(PhDemoCS). These demos are described in the document "PhDemo Developer Guide.doc" from the "Doc\PhantomSDK Doc".
The PhDemoMatlab files are located under the "PhMatlabSDK\PhDemoMatlab" directory.

4.4.1.1. Running the PhDemoMatlab

After installation of the SDK on your computer, open the PhDemoMatlab.m from the PhDemoMatlab folder and use F5 to run the demo. You can connect to Phantom cameras or you can work on simulated cameras in the absence of the real devices.

4.4.2. Function to read images from a cine file

Most applications of the Matlab are related to the play and analyze of the cine recordings stored in files. If your interest is in this area you can skip the complex, object oriented PhDemoMatlab and start from a simpler example: ReadCineFileImage function.

`ReadCineFileImage` is a simple function to read cine images from a file. The function implementation is based only on the PhantomSDK function wrappers.

4.4.2.1. Function parameters

- fileName – the file path where the cine file is located
- imageNo – the image number to be read from the cine file
- showImage – if true the function will create a figure where the image is shown

4.4.2.2. Function outputs

- matlabIm – a 1D Gray/3D RGB matrix with pixel values left aligned to 16b ready to be displayed with Matlab `image` function
- unshiftedIm – a 1D Gray/3D RGB matrix with pixel values unshifted. For example, if the camera provides a pixel bitdepth of 12 you will get pixel values as 16 bit integers from 0 to 4095.

4.4.2.3. Function usage

- Load the phantom libraries
  ```
  LoadPhantomLibraries();
  ```
- Register the Phantom dlls ignoring connected cameras. Use this function once at the begining of your work
  ```
  RegisterPhantom(true);
  ```
- Read the cine image
  ```
  [matlabIm, origIm] = ReadCineFileImage('D:\Cine\test.cine', -3000, true);
  ```
- Do other work with cine files
- When you finished your work unregister Phantom dlls
  ```
  UnregisterPhantom();
  ```
- Finally unload the Phantom libraries
  ```
  UnloadPhantomLibraries();
  ```

5. Tips for the Matlab developers

This section describes a few tips that will help programmers in their development work and in understanding of the actual code.

5.1. Calling functions which have a pointer to an array of data

Some functions like: `PhGetCameraID`, `PhGetCameraResolutions`, `PhGetOpenCineName`, `PhGetCineImage` have as parameters a pointer to an array of data.

The steps to call such a function are:

- Allocate sufficient memory by creating a large enough array of data with the type required by the parameter.

Examples:

- ✓ Allocating uint8 array used to read an image buffer with dll function `PhGetCineImage`:
  ```
  pixels = zeros(BufferSize, 1, 'uint8');
  ```
- ✓ Allocating a 256 char array which will be used to read a file path with function `PhGetOpenCineName`:
  ```
  charVector = char(ones(256,1,'int8'));
  ```
  or
  ```
  charVector = blanks(256);
  ```

- Create a `libpointer` to this array.
  ```
  pPixel = libpointer('uint8Ptr', pixels);
  ```
- Pass the pointer using `calllib` function.
  ```
  [HRES, allOutPointers] = calllib('phfile', 'PhGetCineImage',
  CH, pImRng, pPixel, uint32(BufferSize), pIH);
  ```
- Access returned data using Value member of the pointer. Use `Plus` operator and/or `setdatatype` function when needed.
  ```
  readPixels = pPixel.Value;
  ```

Note: Only an array of simple structures (containing simple data types as `INT` or `float`) are supported for these calls. Array of structures containing different kind of fields (or pointers) cannot be used.

5.2. Working with C structures and arrays not supported in Matlab

Some structures declared in the original Phantom C headers are not compatible with `calllib` function use from the 2009a Matlab version. As a result separate compatible headers were created.

5.2.1. Array of structures inside structure

These structures are not supported by calllib. Array fields must be converted to an array of basic C type. Examples of such structures: `CAMERAOPTIONS`, `BITMAPINFO`, `IMPARAMS`.

5.2.2. Multidimensional array inside structure

These arrays are not supported for calllib calls. These fields must be converted to a single dimensional array.

Example: `char AnaUnit[8][6],` `a` member of the `SETUP` structure, will become char `AnaUnit[8*6]`

5.2.3. Structure with array or pointer fields inside another structure

These fields can be supported in MatLAB2009a if the below patch is applied:
http://www.mathworks.com/support/bugreports/537829

5.3. Compatibility with Matlab versions older than 2009a

Arrays of structures are not supported for function calls in versions older than Matlab 2009a. If the structure contains simple data (only int for example) you may change the function declaration from the header file and use simple data pointers (like int pointers) and then parse the content returned in the pointer.
Example:
`PhGetResolutions (UINT CN, PPOINT pRes, PUINT pCnt, PBOOL pCAR, PUINT pADCBits)` function declaration.
You may change the function declaration using PINT. One int pair value represent a POINT structure.
Note: Other compatibility issues may rise in older versions.