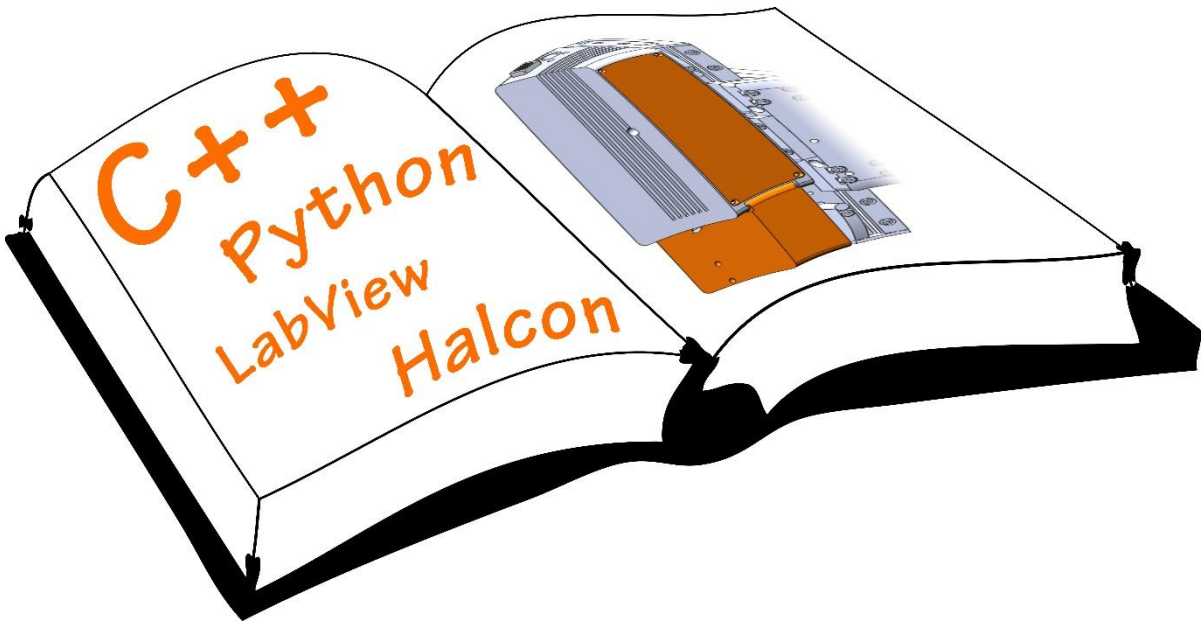
heliotis
advancing 3D metrology

heliSDK 1.x

Programmer's Manual

Document Number	UM.SDK.0001.EN
Release Version	1.2
Release Date	03.05.2016



How to contact Heliotis

Americas:

Heliotis AG	voice	+41-41-455-6700
Längenbold 5	fax	+41-41-455-6701
CH 6037 Root (Luzerne)	e-mail	support@heliotis.ch
Switzerland	web	www.heliotis.ch

Europe:

Heliotis AG	voice	+41-41-455-6700
Längenbold 5	fax	+41-41-455-6701
CH 6037 Root (Luzerne)	e-mail	support@heliotis.ch
Switzerland	web	www.heliotis.ch

Japan:

LinX Corporation	voice	+81-45-979-0731
1-13-11	fax	+81-45-979-0732
eda-nishi Aoba-ku	e-mail	info@linx.jp
Yokohama 225-0014	web	www.linx.jp
Japan		

Korea:

IOVIS	voice	+82-2-424-8832
1305	Fax	+82-2-6455-8832
Hyundai Knowledge Industrial	e-mail	sales@iovis.co.kr
Center C-dong, 7	web	www.iovis.co.kr
Seoul, Korea Zip Code 05836		

Asia Pacific:

Heliotis AG	voice	+41-41-455-6700
Längenbold 5	fax	+41-41-455-6701
CH 6037 Root (Luzerne)	e-mail	support@heliotis.ch
Switzerland	web	www.heliotis.ch

Contents

Preface

The Team

heliotis AG

About this document

This document is designed to support the installation and usage of the heliSDK to build own software application based on the libHeLIC library. The most important information for each supported programming language are included.

Please check the website www.heliotis.ch for updates on this document and related software components.

Related documentation

- *heliCam™ C3 User's Manual*
  This manual provides additional background information on the sensing technology of the heliInspect™ and heliCam™ products. In particular, this manual explains the meaning and relationships of all register parameters used to configure the available acquisition modes.
- *heliInspect™ H6 User's Manual*
  This manual support the installation, configuration and efficient operation of the heliInspect™ H6 measurement head as well as the LINAX/XENAX standard scanner.
- *Register description file*
  This manual include a description of all registers which can set in the heliCam C3.
- *Programmer's Manual XENAX*
  This manual covers advanced programming features, interface options and optimization of the XENAX servo controller. It also contains a section on trouble shooting.

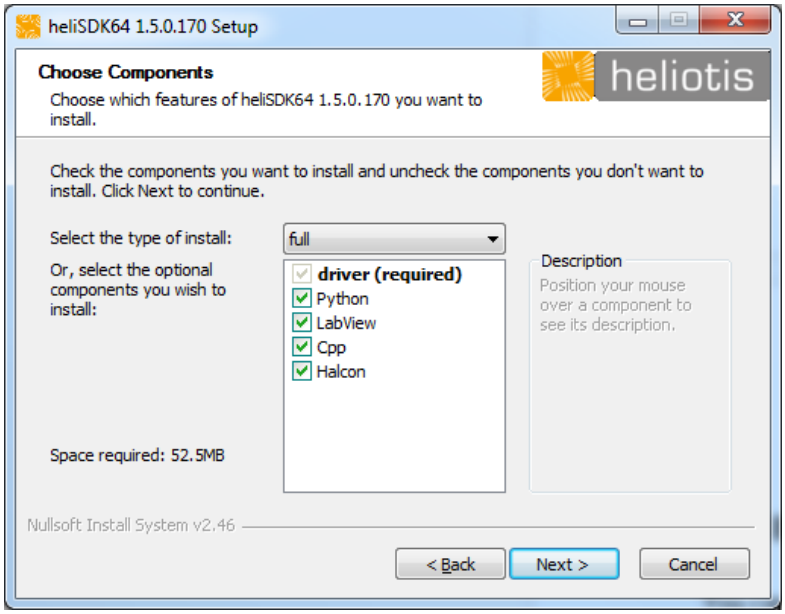
Conventions

>IMPORTANT< …
>WARNING< …
>HINT<…

Chapter I. Installation

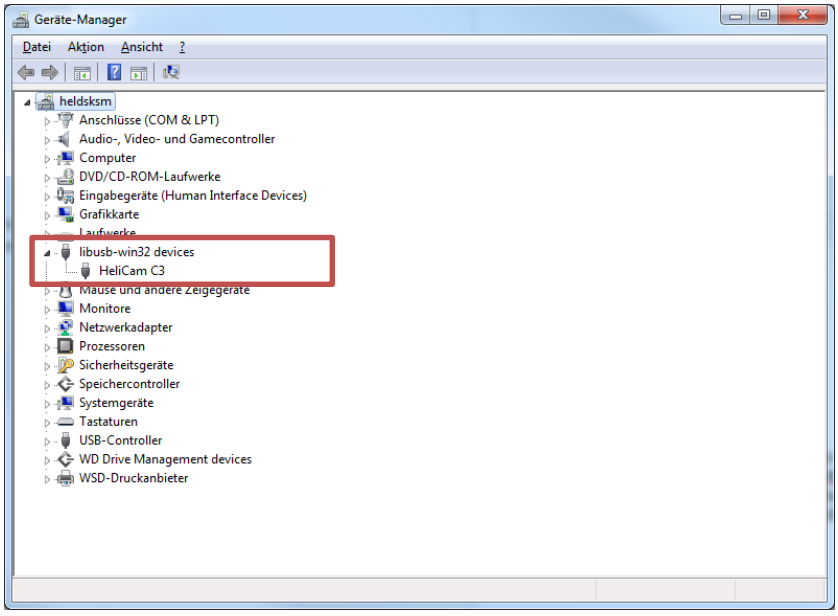
The heliSDK is available for 32 and 64bit Windows operating systems since Windows XP. The latest version can be downloaded from our website www.heliotis.ch

By default, the setup install all language interface. For normal use and high flexibility, the default (full) installation will be recommended.



The default installation path is *C:\Program Files\Heliotis\heliCam* for 64bit operating systems and *C:\Program Files (x86)\Heliotis\heliCam* for 32bit operating system. The path descriptions from all Heliotis documentations based on this location.

After the SDK installation and with connected and powered heliCam, the device manager shows the *heliCam C3* under "*libusb-win32 devices*". If them not so, something was going wrong with the installation or the camera connection is broken.



a. Locations

After the heliSDK installation, a new folder is placed to *C:\Program Files\Heliotis\heliCam* or *C:\Program Files (x86)\Heliotis\heliCam* respectively. All files which are relevant for a user application are included in this folder and subfolders.

.\Cpp	If C++ was selected during the installation process, this folder exists and include some C++ examples.
.\documents	This folder include all documentation like this file.
.\Halcon	If Halcon was selected during the installation process, this folder exists and include all Halcon files. Updates are also available directly from MVTec.
.\LabView	If LabView was selected during the installation process, this folder exists and include the LabView library and examples.
.\libHeLIC	This folder include the API library. The documentation of this API is stored under *.\documents\heliCamAPI.chm*
.\Python	If Python was selected during the installation process, this folder exists and include the Python wrapper and examples.
.\sys	This folder include the Windows driver files for the heliCam C3
.\libHeLICTester.exe	Small test program, to check the connectivity and functionality from the heliCam.
.\Uninstall.exe	Uninstall the heliSDK and all files.

The important files are linked to the Windows start menu under *All programs -> Heliotis -> heliSDK* respectively *heliSDK64.*

If something missed, rerun the heliSDK installation and select the used or all components.

>HINT< *Before you start a new installation, please close all applications which use the heliSDK interface. Also the documentation files should be closed.*

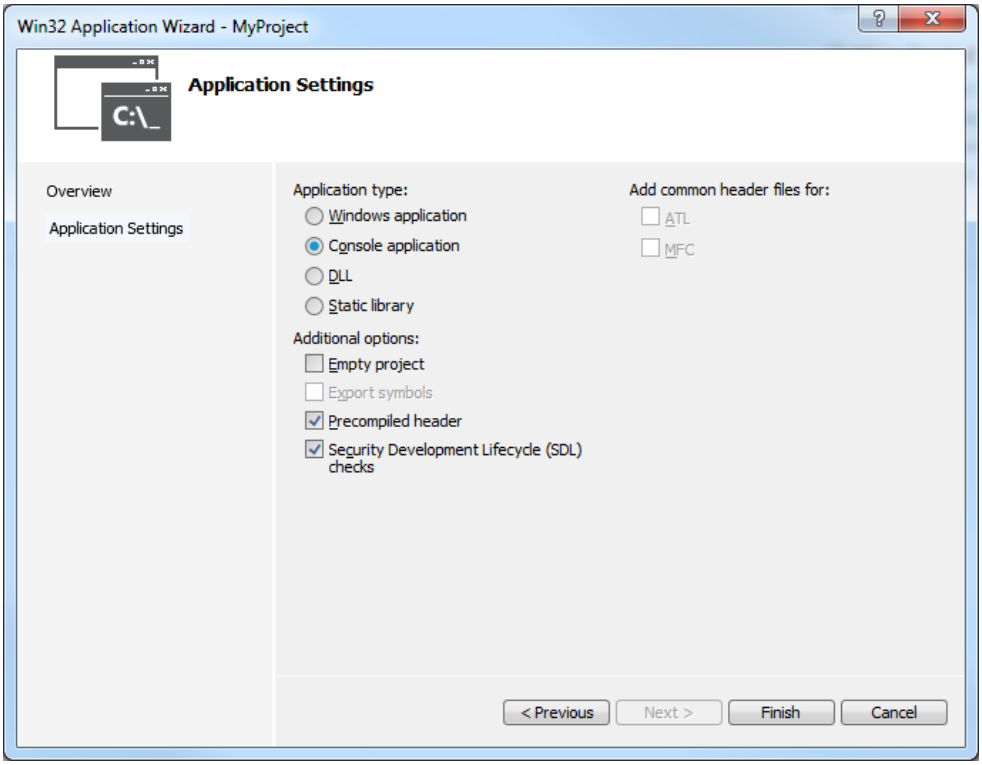
Chapter II. C++

The base of the application programmer interface from the heliCam is written in C++. An own software application can build up directly from this interface. A detailed API documentation was done with doxygen and is stored at .\documents\heliCamAPI.chm and linked to the start menu. In the following subchapters some helpful information and hints are listet.

a. Setup new Visual Studio project

To create an own Visual Studio project from scratch, do the following steps:

1. Create new project. *File -> New Project*
2. Choose "*Win32 Project*"
3. Create a project with "*Precompiled header*"



4. Configure the project settings: *Project -> Properties*

   For the **x86** platform set these settings: *Configuration Properties -> C/C++ -> General*
   Add the following path to "*Additional Include Directories*":
   C:\Program Files (x86)\Heliotis\heliCam\libHeLIC
   C:\Program Files (x86)\Heliotis\heliCam\Cpp\heliCam
   C:\Program Files (x86)\Heliotis\heliCam\Cpp\axis
   $(ProjectDir)

   Configuration Properties -> Linker -> General
   Add the following path to "*Additional Library Directories*"
   C:\Program Files (x86)\Heliotis\heliCam\libHeLIC

   Configuration Properties -> Linker -> Input
   Add the following line to "*Additional Dependencies*"

libHeLIC.lib

For the **x64** platform set these settings: *Configuration Properties -> C/C++ -> General*
Add the following path to "*Additional Include Directories*":
C:\Program Files\Heliotis\heliCam\libHeLIC
C:\Program Files\Heliotis\heliCam\Cpp\heliCam
C:\Program Files\Heliotis\heliCam\Cpp\axis
$(ProjectDir)

Configuration Properties -> Linker -> General
Add the following path to "*Additional Library Directories*"
C:\Program Files\Heliotis\heliCam\libHeLIC

Configuration Properties -> Linker -> Input
Add the following line to "*Additional Dependencies*"
libHeLIC.lib

5. Add the HeliCam class to your project: *Project -> Add Existing Item…*
   Choose the files *heliCamC3.h* and *heliCamC3.cpp* from
   C:\Program Files (x86)\Heliotis\heliCam\Cpp\heliCam or
   C:\Program Files\Heliotis\heliCam\Cpp\heliCam

   Add the axis class to your project (e.g. Jenny axis): *Project -> Add Existing Item…*
   Choose the files Xenax.h and Xenax.cpp from
   C:\Program Files (x86)\Heliotis\heliCam\Cpp\axis or
   C:\Program Files\Heliotis\heliCam\Cpp\axis
6. Include the h-files in your project
   ```
   #include "heliCamC3.h"
   #include "Xenax.h"
   ```
7. Use the heliCam and enjoy. A good starting point are also the sequences from the examples.

b. Functions

All available functions from the API are documented in a doxygen documentation. In the heliCamAPI.chm, which is linked to the start menu, all functions with a description are listed. A short overview about the functions and in which state, which functions are used gives the flow chart in the appendix from this document.

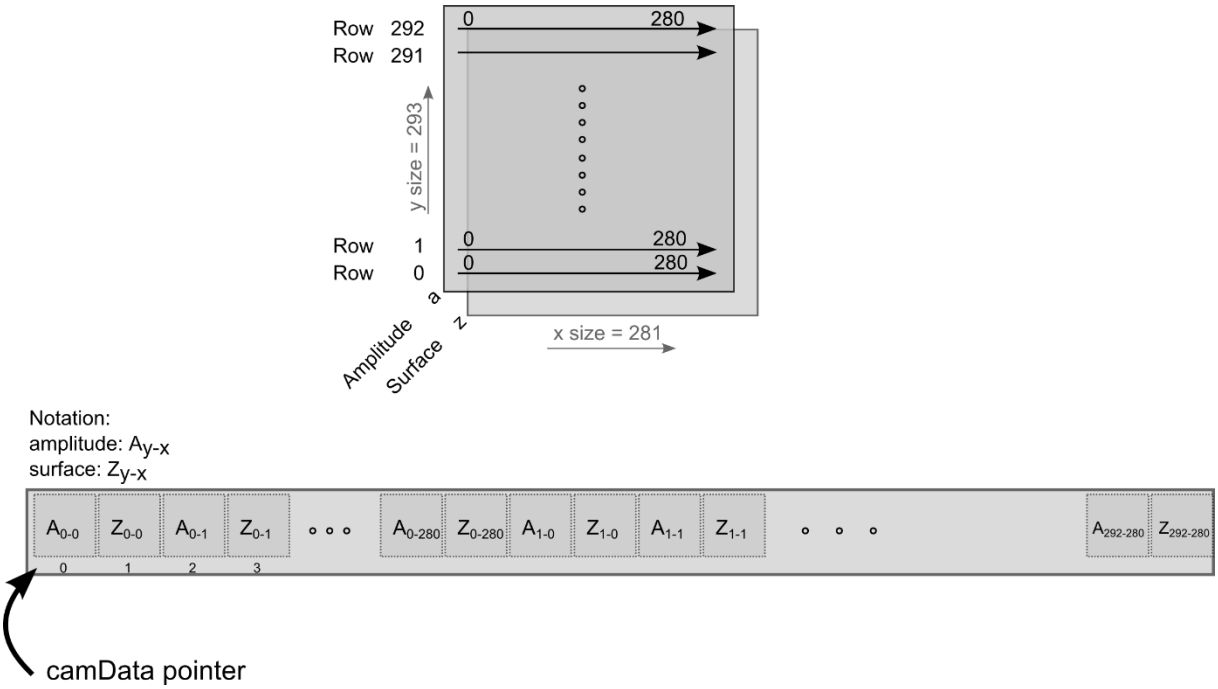
The basic application flow for an own C++ application can be found in the example projects.

c. Data format

The data format change depending from the used camera mode. This means also, that for different camera modes, different allocation types are used. For more information show the doxygen documentation for the function *HE_AllocCamData* and the function parameter 'format' of type CamDataFmt.

CamMode = 4, CamMode = 7

For this camera mode, the CamDataFmt '**DF_A16Z16**' is used. After processing the previous acquired camera data with the function *HE_ProcessCamData*, the function *HE_GetCamData* return a pointer to the measurement result which is constructed as follows:

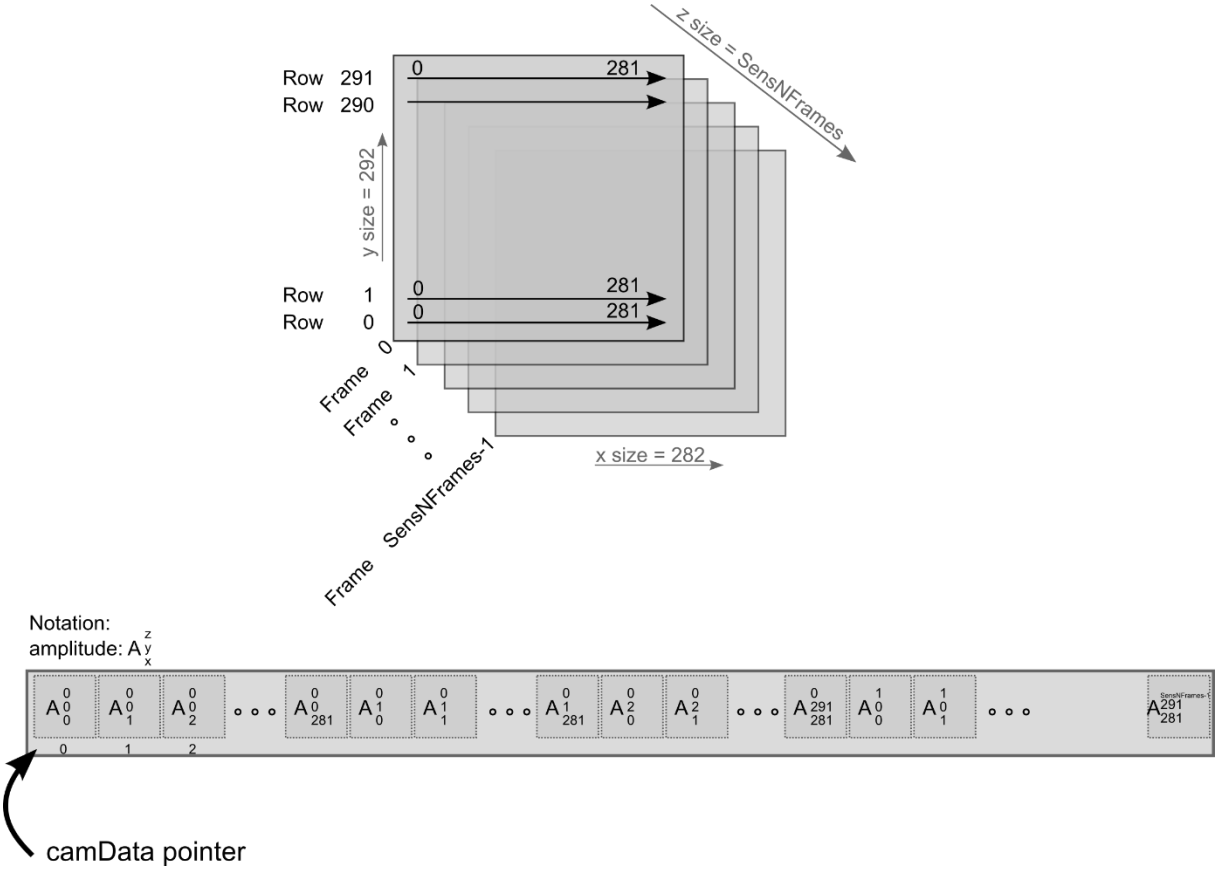


C++ example:

```
cd = HE_ProcessCamData(currInst->heHdl, 1, 0, 0);
cd = HE_GetCamData(currInst->heHdl, 1, 0, metadata);
ushort* camData = (ushort*)cd->data;

uint ySize = metadata->dimSz[2]; // y - 293
uint xSize = metadata->dimSz[1]; // x - 281
uint arraySize = metadata->dimSz[0]; // z or a - 2

const ULONG zStrideRead = 1;
const ULONG yStrideRead = arraySize * xSize;
const ULONG xStrideRead = arraySize;

float imgFixedPoint;

for (uint arrayId = 0; arrayId < (arraySize); arrayId++) {
  for (uint y = 0; y < (ySize); y++) {
    for (uint x = 0; x < (xSize); x++) {
      if (arrayId == 0) {
      // aValue
      aScaledData[y][x] = camData[arrayId * zStrideRead + y * yStrideRead + x * xStrideRead] / (float)16;
      }
      else {
      // zValue
      imgFixedPoint = camData[arrayId * zStrideRead + y * yStrideRead + x * xStrideRead] / (float)32;
      zScaledData[y][x] = (float)imgFixedPoint*frameThickness;
      }
    }
  }
}
```

CamMode = 1, CamMode = 2

For this camera mode, the CamDataFmt '**DF_A16**' is used. After processing the previous acquired camera data with the function *HE_ProcessCamData*, the function *HE_GetCamData* return a pointer to the measurement result which is constructed as follows:

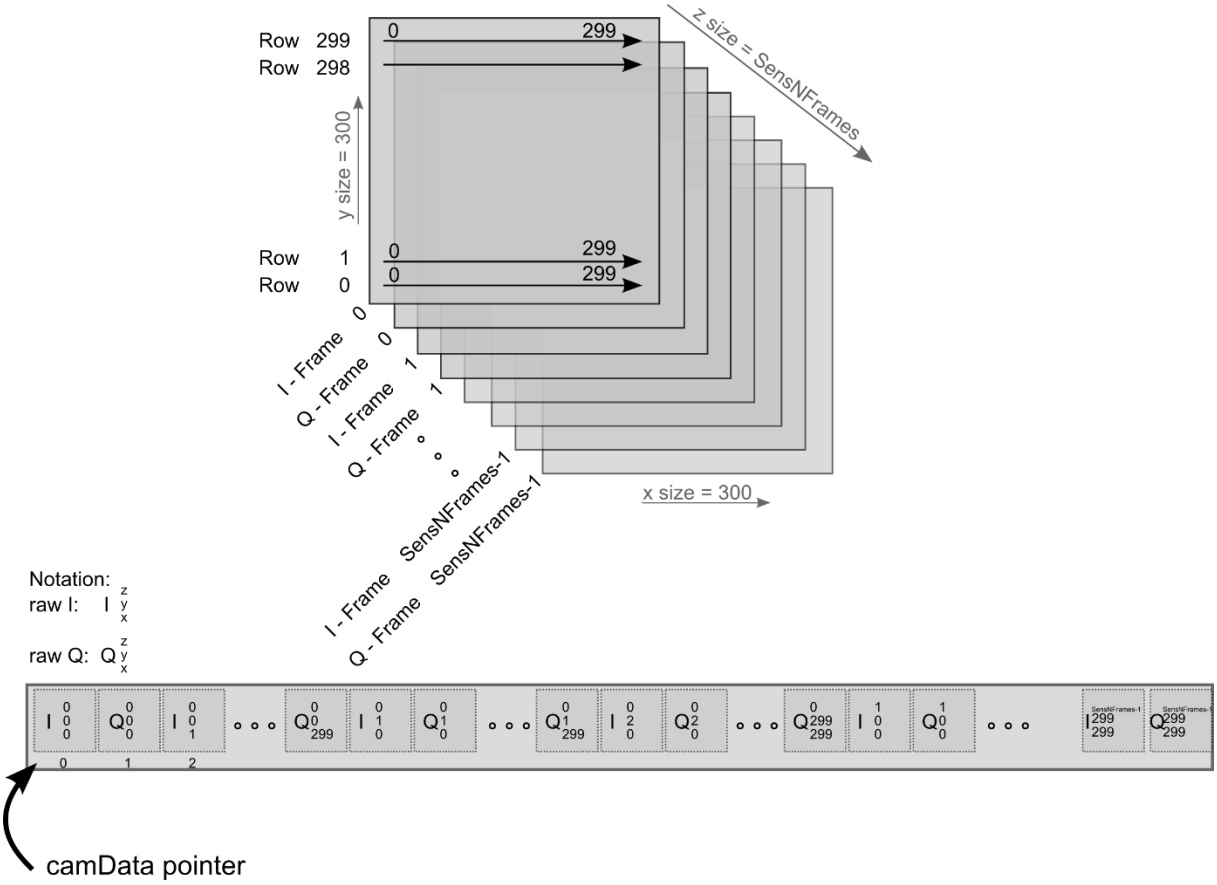


C++ example:

```
cd = HE_ProcessCamData(currInst->heHdl, 1, 0, 0);
cd = HE_GetCamData(currInst->heHdl, 1, 0, metadata);
ushort* camData = (ushort*)cd->data; //16 bit Data

uint zSize = metadata->dimSz[2]; // z - SensNFrames
uint ySize = metadata->dimSz[1]; // y - 292
uint xSize = metadata->dimSz[0]; // x - 282

const ULONG zStrideRead = xSize * ySize;
const ULONG yStrideRead = xSize;

float *dataBuffer;

for (uint x = 0; x < (xSize); x++) {
  for (uint y = 0; y < (ySize); y++) {
    for (uint z = 0; z < (zSize); z++) {
      aScaledData[z][y][x] = camData[x + y * yStrideRead + z * zStrideRead] / (float)16;
    }
  }
}
```

CamMode = 0

For this camera mode, the CamDataFmt '**DF_I16Q16**' is used. After processing the previous acquired camera data with the function *HE_ProcessCamData*, the function *HE_GetCamData* return a pointer to the measurement result which is constructed as follows:

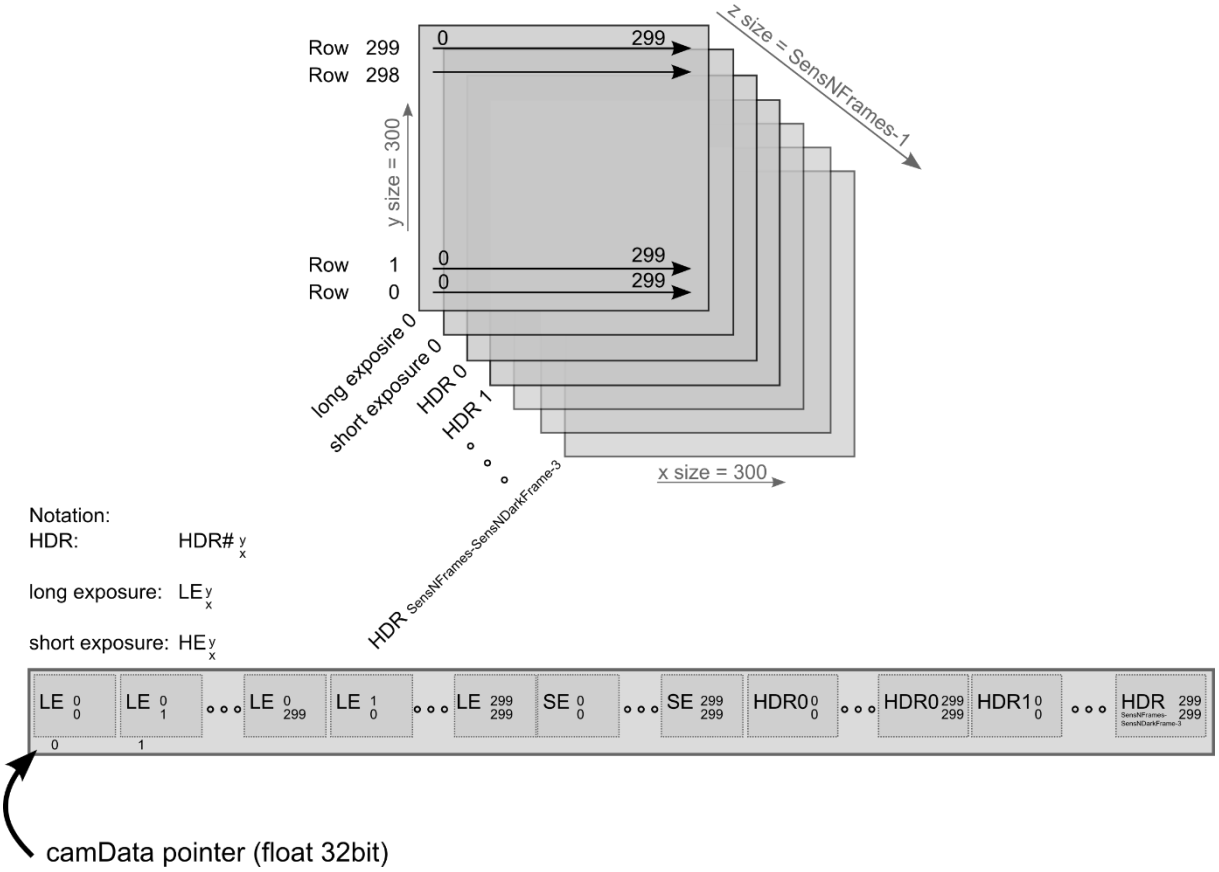


C++ example:

```
cd = HE_ProcessCamData(currInst->heHdl, 1, 0, 0);
cd = HE_GetCamData(currInst->heHdl, 1, 0, metadata);
ushort* camData = (ushort*)cd->data; //16 bit Data

uint zSize = metadata->dimSz[3]; // z - SensNFrames
uint ySize = metadata->dimSz[2]; // y - 300
uint xSize = metadata->dimSz[1]; // x - 300
uint arraySize = metadata->dimSz[0]; // I or Q - 2

const ULONG zStrideRead = xSize * ySize * arraySize;
const ULONG yStrideRead = xSize * arraySize;
const ULONG xStrideRead = arraySize;

for (uint x = 0; x < (xSize); x++) {
  for (uint y = 0; y < (ySize); y++) {
    for (uint z = 0; z < (zSize); z++) {
      for (uint arrayId = 0; arrayId < (arraySize); arrayId++){
              if (arrayId == 0) {
        IData[z][y][x] = camData[x * xStrideRead + y * yStrideRead + z * zStrideRead + arrayId];
        }
              else {
        QData[z][y][x] = camData[x * xStrideRead + y * yStrideRead + z * zStrideRead + arrayId];
        }
      }
    }
  }
}
```

CamMode = 3

For this camera mode, the CamDataFmt '**DF_Hf**' is used. After processing the previous acquired camera data with the function *HE_ProcessCamData*, the function *HE_GetCamData* return a pointer to the measurement result which is constructed as follows:



C++ example:

```
cd = HE_ProcessCamData(currInst->heHdl, 1, 0, 0);
cd = HE_GetCamData(currInst->heHdl, 1, 0, metadata);
float* camData = (float*)cd->data; //32 bit Data

uint zSize = metadata->dimSz[2]; // z – SensNFrames-SensNDarkFrames-3
uint ySize = metadata->dimSz[1]; // y - 300
uint xSize = metadata->dimSz[0]; // x - 300

const ULONG zStrideRead = xSize * ySize;
const ULONG yStrideRead = xSize;

for (uint x = 0; x < (xSize); x++) {
  for (uint y = 0; y < (ySize); y++) {
    for (uint z = 0; z < (zSize); z++) {
      Image[z][y][x] = camData[x + y * yStrideRead + z * zStrideRead];
    }
  }
}
```

CamMode = 5

For this camera mode, the CamDataFmt '**DF_Z16A16P16**' is used. After processing the previous acquired camera data with the function *HE_ProcessCamData*, the function *HE_GetCamData* return a pointer to the measurement result which is constructed as follows:

```
cd = HE_ProcessCamData(currInst->heHdl, 1, 0, 0);
cd = HE_GetCamData(currInst->heHdl, 1, 0, metadata);
ushort* camData = (ushort*)cd->data; // 16 bit Data

//read the measurement size from the metadata.
uint ySize = metadata->dimSz[3]; // 292
uint xSize = metadata->dimSz[2]; // 280
uint zSize = metadata->dimSz[1]; // 2 * exSimpHwin + 1
uint typeSize = metadata->dimSz[0]; // 3
// type=> 0: surface (ushort 11.5) 1: amplitude (ushort 12.4) 2: phase phi (ushort 3.13)

const ULONG zStrideRead = typeSize;
const ULONG yStrideRead = typeSize * zSize * xSize;
const ULONG xStrideRead = typeSize * zSize;

for (uint x = 0; x < (xSize); x++) { // x - 280
  for (uint y = 0; y < (ySize); y++) { // y - 292
    for (uint z = 0; z < (zSize); z++) { // z - 2 * ExSimpMaxHwin + 1
      for (uint t = 0; t < (typeSize); t++) { // 3
        if (t == 0) {
          Surface[z][y][x] = camData[t + x * xStrideRead + y * yStrideRead + z * zStrideRead] / (float)32;
        } else if (t == 1) {
          Amplitude[z][y][x] = camData[t + x * xStrideRead + y * yStrideRead + z * zStrideRead] / (float)16;
        } else {
          Phi[z][y][x] = camData[t + x * xStrideRead + y * yStrideRead + z * zStrideRead] / (float)8192;
        }
      }
    }
  }
}
```

d. zTag calculation

The measured surface provided in the surface mode (CamMode = 4 or 7) refers to the frame number with a sub-frame resolution. The result is provided in an 11.5 unsigned fixed point representation. A detailed description about the sub-frame calculation is provided in the heliCam C3 manual in chapter 5.3.3

Basic surface calculation

A first basic calculation from the surface can be done with the frame thickness. For this, each Pixel value should be multiplied with the frame thickness.

The frame thickness depends from the camera settings and can be calculated with the following formula:

If: BSEnable=1 and CalDur1Cyc=1 $d_{frame} = 0.5 * (2 * SensNavM2 + 3) * \lambda$

If: BSEnable=0 $d_{frame} = 0.5 * (2 * SensNavM2 + 2) * \lambda$

In the previously shown formula, the SensNavM2 is the used camera parameter and lambda (λ) is the wavelength from the light source. (Usually 640nm with red LED) The unit given by lambda is the unit from the frame thickness. (For representation in mm, you can set lambda=0.00064 [mm])

The resulting calculation in C/C++ is something like this:
(The camData pointer is described in the previous chapter c. data format)

```
float frameThickness = 0.5*(2 * SensNavM2 + 3)*lambda;
float imgFixedPoint = camData[arrayId * zStrideRead + y * yStrideRead + x * xStrideRead] / (float)32;
float zScaledData[y][x] = (float)imgFixedPoint*frameThickness;
```

surface calculation with zTags

Each measurement contains some Meta (header) information which are including “zTag” values. The zTag values represent a counter value which is incremented by the linear axis encoder ticks. During the measurement, this value is stored at the beginning from each frame readout of image sensor.

The counter value is a 16bit unsigned value. With a 100nm encoder resolution the counter can be incremented over 6.55mm without a wrap around. In order to don’t take care about the wrap around, a reset position at the start position from the measurement (around 300um before the trigger position) is a good solution. The reset procedure can be found in the C/C++ example. (heliCam Method resetCamZCounter(..))Please don’t move the axis during this process.

After acquiring a measurement, the header information are included in the measurement buffer 0. The full header content is described in the register description in chapter 3. In the following code sequence, the access to the zTag information is done with the pointer “zTags”

After this, add the reset position (in encoder ticks) to each zTag value in the header. To prevent a wrap around in the next calculation, add also the value 2^16 = 65536 to each zTag value. A wrap around (in zTagsReferenced) is possible when the zTag counter was decremented by the axis encoder. (This is dependent from the scan and trigger direction)

Furthermore, a scaling into mm is possible. The zTag unit is encoder ticks. When you prefer a representation in mm, each zTag value should be divided by the “Ticks per mm” value. With a 100nm encoder (1 tick per 100nm), this value is 10000. In the following code, the new values are stored in the zTagReferenced variable. This Variable include the position in mm for each integer Frame number.

```
// This buffer include the header with zTags.
cd = camera.getCamData(0, 0, 0);

//set a pointer to the first zTag in the header
ushort* zTags = (ushort*)cd->data + 0x10;

//allocate memory for the zTag reference
float *zTagsReferenced;
zTagsReferenced = new float[SensNFrames + 1];

//calculate the zTag reference values
for (int i = 0; i < SensNFrames; i++) {
        zTagsReferenced[i] = ((pow(2, 16) + resetPosition) - zTags[i]) / TicksPerMM;
}
//duplicate the last value. Need for a simple calculate from the zScaledData values
zTagsReferenced[SensNFrames] = zTagsReferenced[SensNFrames - 1];
```

As described in the introduction to this chapter, the zValues from the surface are in sub-frame resolution. But zTagsReferenced values are only in (integer) frame resolution.

A simple solution is to do a linear interpolation between the zTagsReferenced values and the sub-Frame value from the measurement surface. This is done and described in the following code:

```
for (uint arrayId = 0; arrayId < (metadata->dimSz[0]); arrayId++) {
        for (uint y = 0; y < (metadata->dimSz[1]); y++) {
                for (uint x = 0; x < (metadata->dimSz[2]); x++) {
                        if (arrayId == 0) {
                        // calculate amplitude value (12.4 fixed point value)
                        aData[y][x] = camData[arrayId*arrayIdStride + y*yStride + x*xStride] / (float)16;
                        }
                        else {
                        // interpolate the z value
                        // raw z-value is 11.5 fixed point value
                        imgFixedPoint = camData[arrayId*arrayIdStride + y*yStride + x*xStride] / (float)32;

                        // round off the fixed point value
                        imgFloor = (int)imgFixedPoint;

                        // calculate the residue from round off. (Required for the linear interpolation)
                        imgResidue = imgFixedPoint - imgFloor

                        // define the z value from rounded value
                        imgFloorScaled = zTagsReferenced[imgFloor];

                        // calculate the different between rounded zTag value and the next zTag value
                        imgStepsScaled = zTagsReferenced[imgFloor + 1] - imgFloorScaled;

                        // lineare interpolation from zTag values for this Pixel
                        zScaledData[y][x] = imgFloorScaled + imgStepsScaled*imgResidue;
                        }
                }
        }
}
```

Advanced surface calculation with zTags

In the previously descripted calculation, the zTag values are based on the measurement end from each frame. Depends on the measurement direction, the frame end can be on the top or the bottom side from the measurement. To calculate the zTagReference value independently from the scan direction, add an approximate offset from the frame start to the frame middle. This calculation dependents from the measurement speed (in ticks per second) and the camera parameters “SensNavM2” and “SensTqp”. Add this value to the zTagReferenced for each z Tag. In the following code sequence this changes are added.

```
// This buffer include the header with zTags.
cd = camera.getCamData(0, 0, 0);

// set a pointer to the first zTag in the header
ushort* zTags = (ushort*)cd->data + 0x10;

// calculate the offset to the middle of a frame.
// Do this with the speed in [ticks/s] and the camera parameters SensNavM2 and SensTqp,
// 35000000 means 35MHz and is an internal frequency
float zTagOffset = measurementSpeed * 4 * (SensNavM2 + 1) * (SensTqp + 6) / 35000000;

//allocate memory for the zTag reference
float *zTagsReferenced;
zTagsReferenced = new float[SensNFrames + 1];

//calculate the zTag reference values
for (int i = 0; i < SensNFrames; i++) {
        zTagsReferenced[i] = ((pow(2, 16) + resetPosition - zTagOffset) - zTags[i]) / 10000;
}
//duplicate the last value. Need for a simple calculate from the zScaledData values
zTagsReferenced[SensNFrames] = zTagsReferenced[SensNFrames - 1];
```

The surface interpolation should be done on the same way like the description before.

e. Examples

The folder .*Cpp**example* includes the Visual Studio 2013 solution 'Example'. This solution includes two different example project. Each project is independently and can be build and run separately.

Minimize_Energy

The project 'Minimize_Energy' includes four different examples for a simple use of the camera in minimize energy mode. Each example include the full measurement sequence and control a linear motor. In examples which ends with a number (e.g. *sample1()* or *sample2()*) the measurement result is stored in a text file. In examples which ends with a letter (e.g. *sampleA()* or *sampleB()*) the measurement result is rescaled for a graphical output with openCV.

In the *sample1()* and *sampleA()* the topology is calculated with the zTag information which are included in the header from the camera data. In the most cases, this is approach is more accurate but harder to implement.

The shorter and simpler examples are *sample2()* and *sampleB()* which calculate the topology with the frame thickness. For a first own implementation is this approach faster and less error-prone.

All examples contain a lot of comments with information about the functions and measurement sequences. To configure the examples for the own system, all significant parameters are defined at the beginning of the main file (Minimize_Energy.cpp). The main parameters are the motor parameter. Define the IP address, Port and measurement area for your system. Please be careful with the measurement area parameters, make sure that the optical components don't can crash in the sample!

Change_Mode

The project 'Change_Mode' include one application example which include an initialization sequence and a measurement loop. In the initialization sequence the heliCam and the motor will be set up with default parameters (in this case for simple max). In the measurement loop, the application do first a measurement in the minimize energy mode and plot this result. Then, the heliCam will be reconfigured for the intensity mode and making a 2D image. This result will be plotted and after them the camera will be reconfigure back for minimize energy mode. This loop run 'NofLoops' times which can be configured at the beginning of the file (Change_Mode.cpp).

To configure the examples for the own system, all significant parameters are defined at the beginning of the main file (Change_Mode.cpp). The main parameters are the motor parameter. Define the IP address, Port and measurement area for your system. Please be careful with the measurement area parameters, make sure that the optical components don't can crash in the sample!

Plot color mapped image

The C++ examples plot the measurement result as a black and white picture. The black and white plot use a small part from the openCV library. For color mapped images, the full openCV library is required which isn't included in the heliSDK installation.

The example ‘Minimize_Energy’ contains the define *USE_FULL_OPENCV* to create a color mapped image. Set this define to 1 and add the full openCV library (tested with 2.49) to the project. The file location is:

include files: *$(SolutionDir)\openCV\include\opencv2*
library files: *$(SolutionDir)\openCV\x86\vc12\lib*
$(SolutionDir)\openCV\x64\vc12\lib
dll files: *$(SolutionDir)\openCV\x86\vc12\bin*
$(SolutionDir)\openCV\x64\vc12\bin

The color map needs the library *opencv_contrib249d.lib*. Add this filename to the linker options: *Configuration Properties -> Linker -> Input*

“*Additional Dependencies*” should include the following lines:

Debug	**Release**
libHeLIC.lib	*libHeLIC.lib*
opencv_core249d.lib	*opencv_core249.lib*
opencv_highgui249d.lib	*opencv_highgui249.lib*
opencv_contrib249d.lib	*opencv_contrib249.lib*

Chapter III. Python

With Python a fast application implementation is possibility. Heliotis provide a Python wrapper for the C++ based libHeLIC library. Through this wrapper, each function which is described in the doxygen documentation (heliCamAPI.chm) is available in a Python script. For a fast and simply use in Python, some helper function are implemented in the wrapper.

The wrapper, which must be imported in each Pythons script which use the heliCam, is stored on *.\Python\wrapper*

a. Basic script

To use the functions from the python wrapper, import the wrapper in an own Python script. For this, add the Path (*.\Python\wrapper*) to the path description in Python. Use the following lines at the beginning from the Python script:

```
import os, sys
prgPath=os.environ["PROGRAMFILES"]
sys.path.insert(0,prgPath+r'\Heliotis\heliCam\Python\wrapper')
from libHeLIC import *
```

b. Examples

In the Python examples, the basic functions can be found and used. The example *libHeLIC_Simple.py* which is stored on *.\Python\example* is a good starting point for an own Python application.

This example includes a simple measurement sequence in the surface mode which initialize and configure the camera and the z-axis. All relevant parameters are defined at the beginning from this Python script.

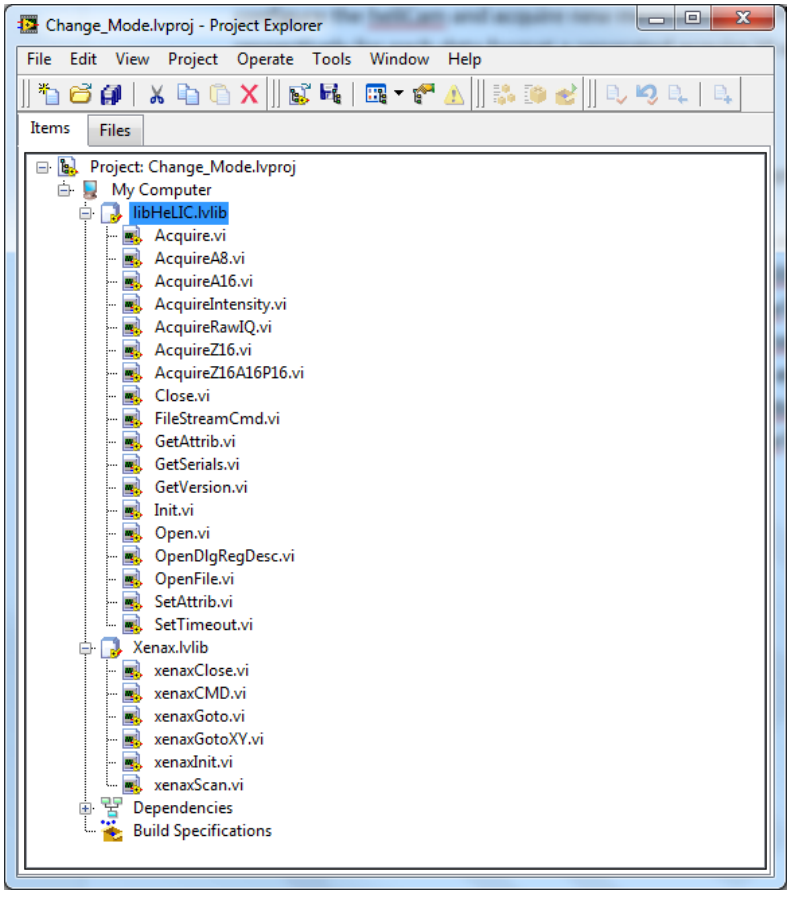
In the example *libHeLICTester.py* different tests are available and more advanced functionalities are implemented. Please check this source code for more information about the sequences and configuration controlled with Python.

Chapter IV. LabView

To use the heliCam in a LabView application, Heliotis provide a LabView library which includes VI's to configure the heliCam and acquire new measurements. Please note, that for each camera mode respectively for each data format a separated acquire VI was created. Select the correct Acquire VI dependent from the used CamMode.

a. Setup new LabView project

1. Create new LabView Project. *File -> New… -> Empty Project*
2. Add libHeLIC.lvlib library to the project:
   Right-click on "*My Computer*" than *Add -> File…* and select
   C:\Program Files\Heliotis\heliCam\LabView\lib\libHeLIC.lvlib or
   C:\Program Files (x86)\Heliotis\heliCam\LabView\lib\libHeLIC.lvlib for 32bit
3. Optional: To use Xenax motors, add also the Xenax library to the project:
   Right-click on "*My Computer*" than *Add -> File…* and select
   C:\Program Files\Heliotis\heliCam\LabView\lib\Xenax\Xenax.lvlib or
   C:\Program Files (x86)\Heliotis\heliCam\LabView\lib\Xenax\Xenax.lvlib for 32bit



4. Now the project is ready to use. Create a new VI and implement your own application.

b. Library

The following is a list of the most important and most used VI's included in the library and their description.

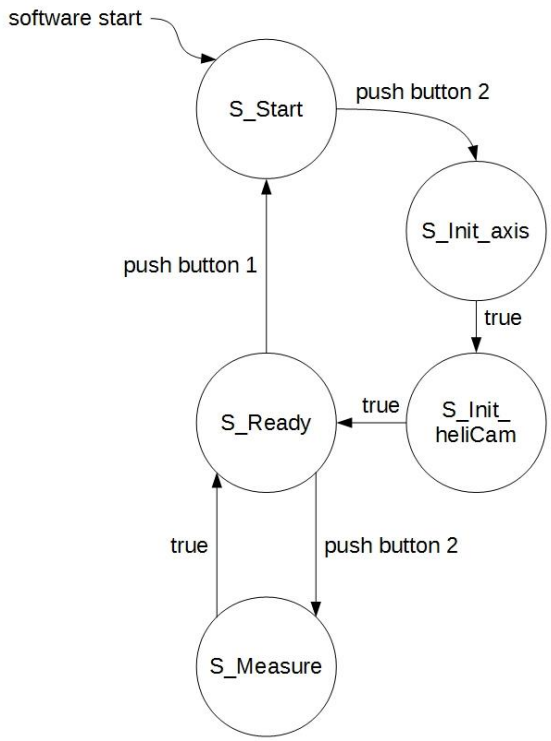
AcquireA8.vi	Acquire a measurement from heliCam in **CamMode=1** and **Comp11to8=1**
AcquireA16.vi	Acquire a measurement from heliCam in **CamMode=1, 2**
AcquireIntensity.vi	Acquire a measurement from heliCam in **CamMode=3**
AcquireRawIQ.vi	Acquire a measurement from heliCam in **CamMode=0**
AcquireZ16.vi	Acquire a measurement from heliCam in **CamMode=4, 7**
AcquireZ16A16P16.vi	Acquire a measurement from heliCam in **CamMode=5**
Close.vi	Close the connection to the heliCam.
GetAttrib.vi	Read the current value from a register.
GetSerials.vi	Returns the number of connected heliCams and the first serial number.
Init.vi	Initialize the driver for use it with LabView. Call this VI before each other call from this library!
Open.vi	Open the connection to a heliCam
SetAttrib.vi	Set a Register. The register name and value range can be found in the register description file
SetTimeout.vi	Change the timeout for all other VIs calls.

c. Examples

For LabView two different examples are available. A simple example exist in the project 'Change_Config'. This example is structured as a "Flat sequence" which included the parts: *Parameter definition*, *axis configuration*, *camera configuration*, *measurement loop* and *termination*.

This five parts should be the basic subdivision from a own LabView application.

The second example 'Minimize_Energy' is implemented with a state machine which is shown below.



S_Start: Start state in which you can set parameters for the axis. You can also set the configuration for the heliCam. For a first use, the given heliCam settings should be okay.

S_Init_axis: Configure the axis with the previously defined parameters.

S_Init_heliCam: Configure the heliCam with the previously defined settings.

S_Ready: The system is ready for a measurement. By pushing the Button "Measure" it will run a measurement. By clicking on the button "Configure" it jumps back to state S_Start and you can reconfigure the axis.

S_Measure: In this state, a measurement will be started. After the measurement is finished, the result will be show on the display and the software goes back to the S_Ready state.

Chapter V. HALCON

Heliotis provide an interface to HALCON from MVTec Software GmbH. Please check the HALCON website www.halcon.com for updates from this interfaces. It is recommended to use the newest HALCON interface with the newest heliSDK.

a. Installation

Update the Acquisition interface from HALCON by downloading the newest files from the HALCON website. Copy the received library (*.dll files) into the folder:
%HALCONROOT%\bin\%HALCONARCH%.

%HALCONROOT% means the base directory from HALCON which was selected during the HALCON installation. %HALCONARCH% means the system architecture. x86sse2-win32 is for 32bit HALCON versions and x64-win64 for 64bit HALCON versions.

Please note, that for the use of HALCON 32bit also the heliSDK 32bit (heliSDK-x.x.x.x-win32.exe) must be installed. And for HALCON 64bit is the heliSDK 64bit (heliSDK-x.x.x.x-win64.exe) required. To work with both HALCON versions (32 and 64bit) install both heliSDK versions.

b. Examples

For HALCON five different examples are available. In each example the most important parameters are defined at the beginning from the script. Information about the sequence and called function can found in the script. In the following table a short description about the example and his function is listen.

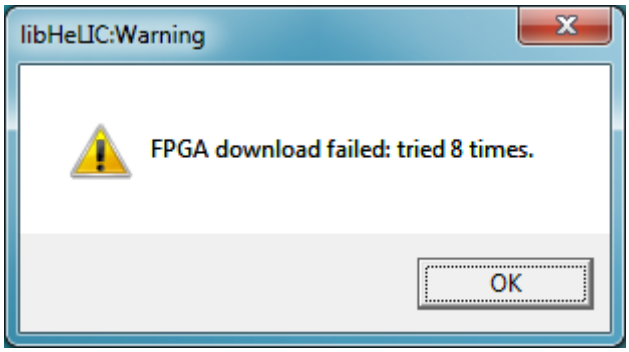
Name	Description
helicamc3_simple	Very simple example which define only the most important parameters. A fast topology measurement is possibility.
helicamc3_bidirectional	Scan alternately in up and down moving the axis. A special example for fast measurement sequences.
helicamc3_modes	Example with all possible camera modes. (topology and tomography) For each mode a procedure is implemented which set the special parameter for this mode.
helicamc3_motor_control	This examples don't control the linear motor. The user must implement and control the moving and triggering sequence.
helicamc3_2cameras	HALCON can control up to 127 heliCams from one script. This example control two cameras with different settings.

Troubleshoot

Warnings and error message

In the case of a problem, the heliSDK returns more information in form of a warning or error message. Dependent on the interface used, this information shows differently.

By using the C++ or Python interface, the heliSDK returns popup windows with a message like the figure below. To suppress the popup windows in a own application, use the API function HE_SetCallback(..).



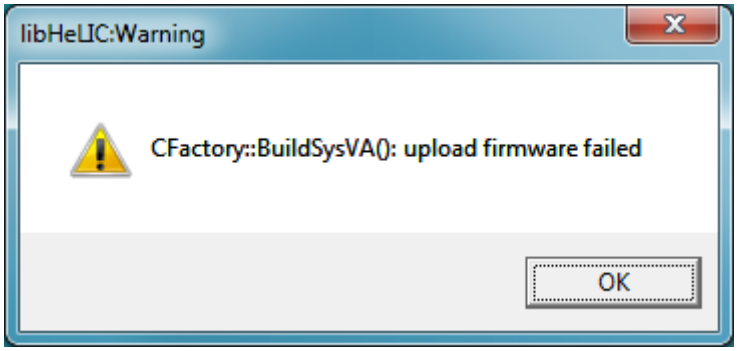
By using the LabView interface, the warnings and error messages are redirected into the LabView error handling. In the case of some problems, the error flag will be set and the same message is included in the error structure.

By using the HALCON interface, the default error messages from HALCON will be popup. In this case, the error message from the heliSDK is written into the output console from HALCON as message type “HALCON-low-level-error”.

Couldn't load FPGA firmware

Symptoms:

By opening the connection to the heliCam device (e.g. HE_Open(..) in C++, open VI in LabView or open_framegrabber in HALCON) the FPGA firmware will be downloaded. In this case, the following message can be returned from the heliSDK: "FPGA download failed: tried 8 times." or "CFactory::BuildSysVA(): upload firmware failed"



Solution:

To opening the connection to the heliCam, the type of built-up FPGA must be passed. In normally cases the type "c3cam_sl70" is the right one. By using the previous version (sl50), change the camera type to "c3cam". (This must be done in each example and also in the heliViewer)

By using the current version (sl70), the connectivity to the heliCam can be checked with the "libHeLICTester" which was installed with the heliSDK. To run this application click on "Start menu -> all programs -> heliotis -> heliSDK(64) -> libHeLICTester". A console application is started and several tests can be selected.

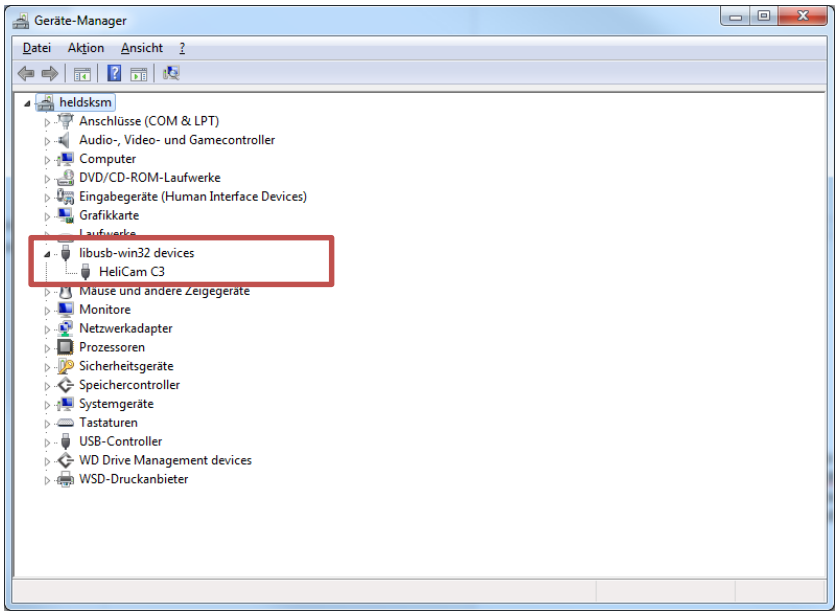
Select "Test6: GetSerial numbers from all connected heliCams!" by type number 6. In this test, each heliCam device which is connected to the computer and correctly detected is listed. If one device missed, please read the section "heliCam detection failed".

To check the data transfer between heliCam and Computer, the test number 3 is a well start. In this test, ….

heliCam detection failed

If no heliCam C3 listed with the thest number 6 from “libHeLICTester”, please check all wires and the power supply from the whole measurement system. The camera system without motor controller can consume up to 24W. Please check, if that power supply have enough power.

Check also the USB wire from the heliCam to the computer. Whit connected heliCam, power up the computer. Now in the device manager the heliCam C2 should be listed:

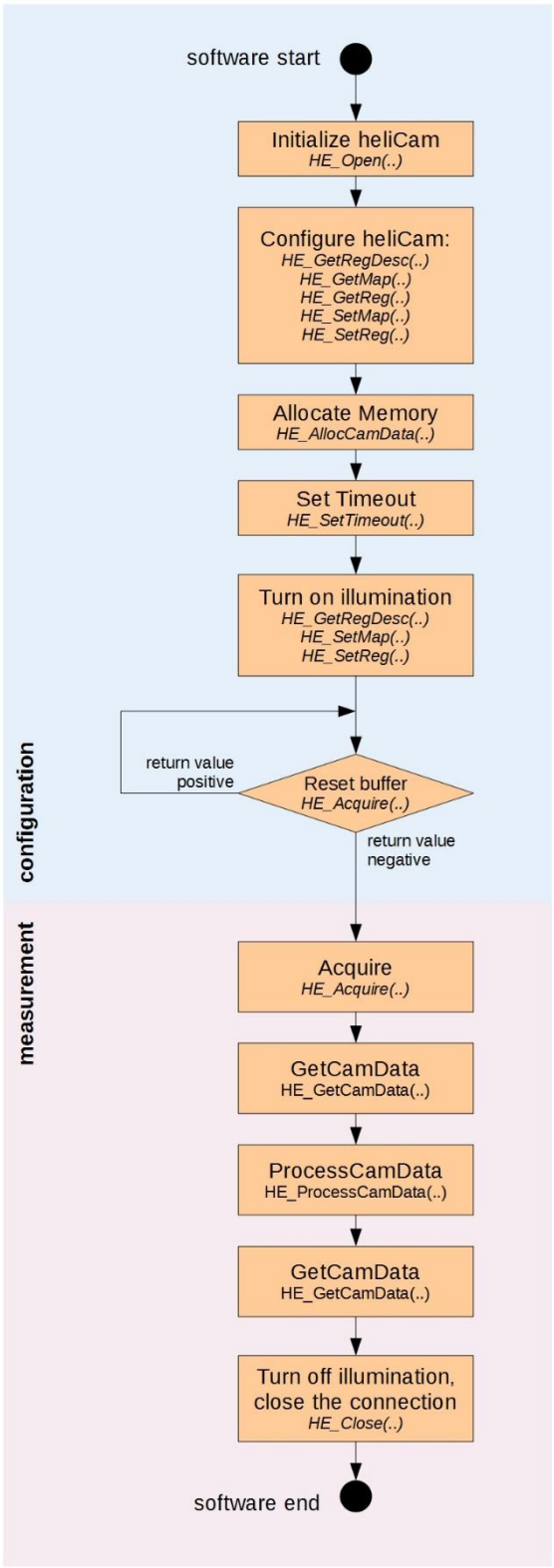


If no heliCam listed, please install the heliSDK with administrator rights and reboot your computer after the installation. Repower also the heliCam device. A description about the installation process, is listed in the “chapter 1 Installation” from this manual.

Appendix

Flow chart

This visualization shows an example software flow which is divided in two areas and includes only the camera communication. One section is for the camera configuration and one for the measurement sequence. In advanced implementation we can step alternate between this two sections. This simple flow chart should be a help to see, in which state which Heliotis functions are to be used.



Index