Technical Notes for heliotis gen4 Devices

Release 23.03

Heliotis AG

Mar 04, 2024

OVERVIEW

This is a collection of technical notes for the heliotis gen4 product generation.

CHAPTER

ONE

TN_15.1.0.1: HELIBRD B4 SD CARD RE-WRITING

1.1 Challenge: Corrupt SD Card Content

heliBrd B4 based devices (like heliInspect H8, heliCam C4) store a file system on an SD Card. Updates of this file system are handled using the UpdateTool application (comes with C4Utility installer) and image package or update package files (available for download on heliotis webpage). The file system is robust, i.e. it tolerates conditions such as power interruption during use. Only in exceptional failure cases the file system may be corrupted to the point of boot failures. In such rare cases, however, the device becomes inaccessible throught its ethernet interface even after attempts to reboot it (power cycle).

Note: If you suspect your file system to be corrupt contact support@heliotis.ch before proceeding with solution described below.

1.2 Solution: SD-Card Re-Writing

To recover from a corrupt file system load a full SD-Card image to the device. The required steps are as follows.

1.2.1 Remove SD-Card from heliInspect H8 or heliCam C4

Required tools (figure Fig. 1.1):

- 1.5mm hexagonal socket wrench (Allen key)
- tweezers

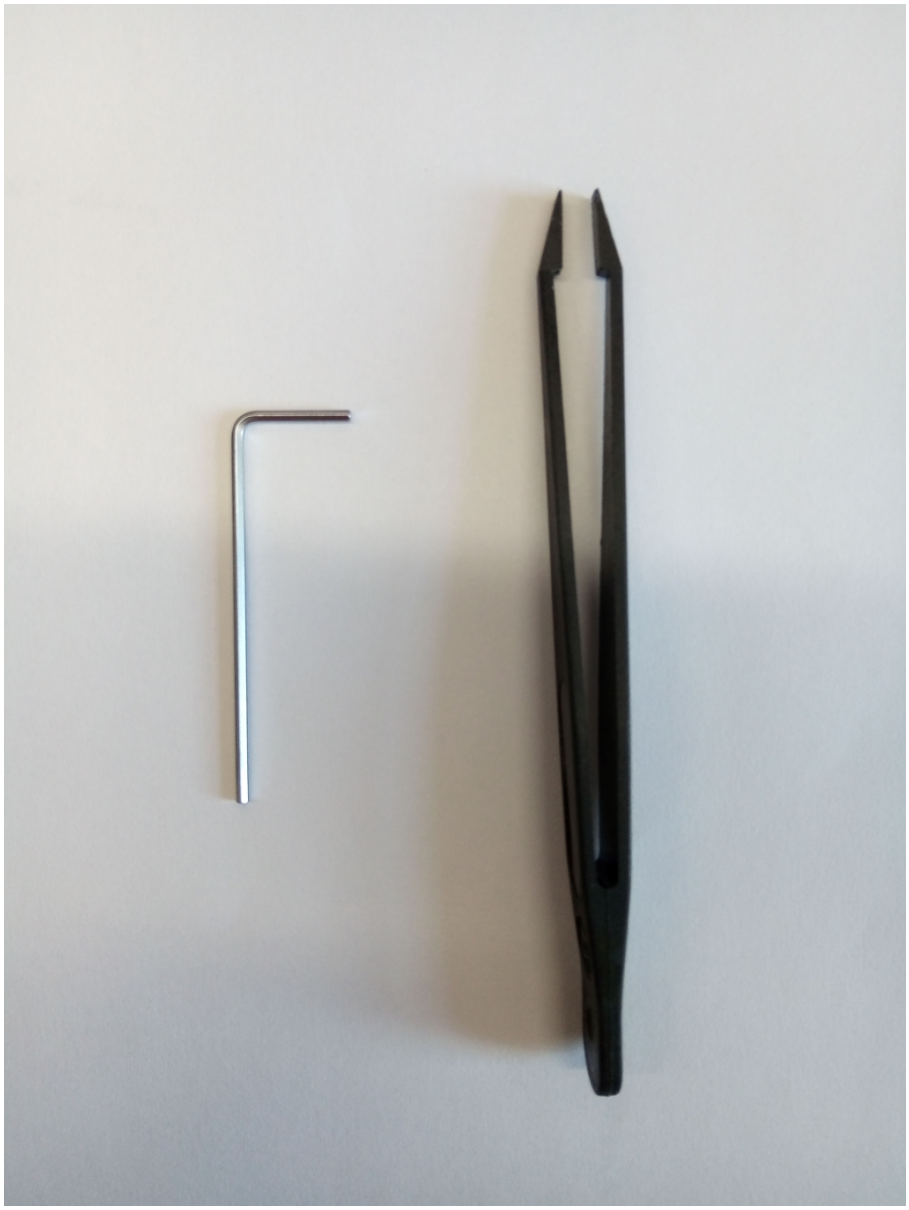

Fig. 1.1: Required Tools

Steps:

1. Switch off the power supply.
2. Remove SD-Card cover. Unscrew the two screws on the right hand side of heliInspect H8 (figure Fig. 1.2). Remove the cover.
3. Remove the SD-Card. The SD-Card holder has a push/pull mechanism. To remove the SD-Card, push the card with a thin rod (e.g. the back of your tweezers). A soft click should be audible. Afterwards, the card pops out a bit and can be removed carefully. (e.g. with the tweezers)

Fig. 1.2: SD-Card Cover Screws

Note: SD card is not accessible for heliInspect H9 devices. Don't attempt to open H9 devices witout heliotis assistance.

1.2.2 Load an SD-Card Image

Required software:

- balenaEtcher https://www.balena.io/etcher/
- heliBoard B4 Image (e.g. b4brdImg_1.8.0.zip)

Steps:

1. Extract the *.zip file. (It contains a single *.img file)
2. Insert the card into PC's SD-Card slot and start balenaEtcher

3. Push the *Flash from file* button and select the extracted *.img file
4. Push the *Select target* button and select the SD-Card drive.
5. The final selections are shown in figure Fig. 1.3. Now push the *Flash!* button to write the image to the SD-Card (PC administrator privileges are required)
6. After a successful write procedure, remove the SD-Card from the PC and insert it into the heliInspect H8.

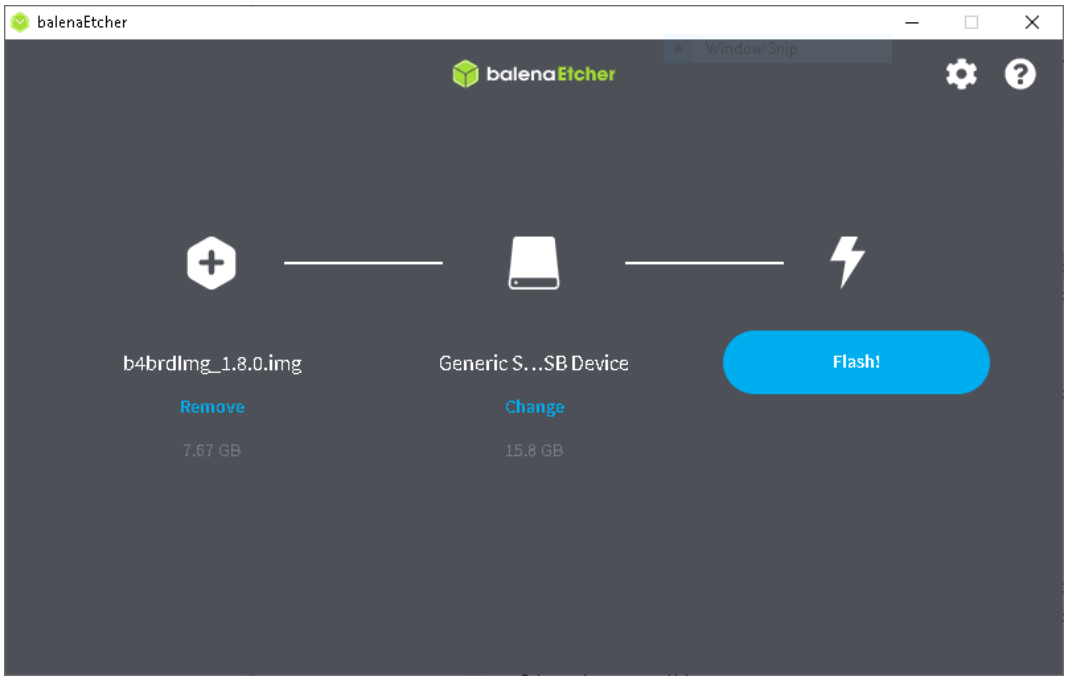

Fig. 1.3: balenaEtcher selection

1.2.3 Insert an SD-Card to heliInspect/heliCam Device

Steps:

1. Insert the SD-Card into heliInspect H8. Use tweezers to hold the card and insert it carefully into the SD-Card slot. The SD-Card slot has a push/pull mechanism. After inserting the SD-Card push it carefully until a soft click is audible. The SD-Card orientation is shown is figure Fig. 1.4.
2. Mount the cover with the two screws.

Fig. 1.4: SD-Card Orientation

1.3 Document History

Version	Date	Changes	Responsible
15.1.0.0	19-05-2022	initial version	sm
15.1.0.1	14-11-2022	review, typos	cl

CHAPTER

TWO

TN_15.2.0.0: ADJUST 3D DATA SCALING AND OFFSET

2.1 Introduction

heliInspect devices scale 3D output data in *um* and map the value to the current stage possition which is in many cases a negative position

This application note describes how to setup 3D data scaling and offset. This allows a well configured value range for *unsigned int16* (Mono16) surface output.

The required features described in this documentation are available since software version **1.9.0**.

2.1.1 Challange

Some applications could not handle the 3D data output as 32bit floating point value. For such applications, the heliInspect H8/H9 as well as any other heliBrd B4 based camera systems from heliotis supports an 16bit unsignet integer data output. Within this mode, the output values are integers and its range is limited from *0 - 65535*.

As a default configuration, the camera scales the 3D output data in *um* and maps the value to the current stage possition which is in many cases a negative position. In section *Configure Custom Surface Scaling* the configuration for custom output data scaling is described. Section *Configure Custom Surface Offset* contains the configuration for a custom surface shift allong the z dirrection and finally, the section *Calculate Real-World Coordinates Data* contains the calculations and additional information how the submitted 3D data are transfered back to the *real-world* data.

Note: The Scaling and Offset configuration is applied to any 3D Data output data (16bit integer as well as 32bit floating point).

2.2 Configure Custom Surface Scaling

A rough scaling setting is applied with the feature *Scan3dDistanceUnit* which allows to select the data output unit as *mm*, *um* or *nm*. The default value, and in almost all cases the best sellection is *um*.

To adjust a custom scaling the feature *Scan3dCoordinateScale[Scan3dCoordinateSelector]* is avalable. The reciprocal value of the *CoordinateC* configuration is internally multiplied with the real-world z value. As an example, the following code listening shows the configuration to adjust the z values to 100nm per integer step.

Listing 2.1: Scan3dCoordinateScale example

```
Scan3dDistanceUnit = "um"
Scan3dCoordinateSelector = "CoordinateC"
Scan3dCoordinateScale = 0.1
```

2.3 Configure Custom Surface Offset

The feature *Scan3dCoordinateOffset[Scan3dCoordinateSelector]* allows the shifting of the 3D output data along the z axis. The value configured within the *CoordinateC* selector is internally subtracted from the real-world z values. As an example, the following code listening shows the configuration to shift the the surface 14.1mm upper. This place a real-world pixelvalue from -13900um up to 200um.

Listing 2.2: Scan3dCoordinateOffset example

```
Scan3dDistanceUnit = "um"
Scan3dCoordinateSelector = "CoordinateC"
Scan3dCoordinateOffset = -14100.0
```

Note: The unit of the *Scan3dCoordinateOffset* feature is configured with the *Scan3dDistanceUnit* feature. It is recommended to select a well *Scan3dDistanceUnit* before tuning the *Scan3dCoordinateOffset* value.

2.4 Calculate Real-World Coordinates Data

The calculation of the 3d data to real-world data requires the configuration value of the *Scan3dCoordinateScale[Scan3dCoordinateSelector]* and *Scan3dCoordinateOffset[Scan3dCoordinateSelector]* features. The values are readable from the device configuration by reading the two features directly or optionally included in the data buffers meta data as so-called chunk data.

The first listening Listing 2.3 shows how to read the information from the device configuration and the second listening Listing 2.4 how to enable the chunk data and read the information from the buffer chunk data. For the final calculation one of this two approaches could be selected.

Listing 2.3: read Scan3dCoordinateScale and Scan3dCoordinateOffset example

```
Scan3dCoordinateSelector = "CoordinateC"
scaleC = Scan3dCoordinateScale
offsetC = Scan3dCoordinateOffset
```

Listing 2.4: enable Chunk data and read Scan3dCoordinateScale and Scan3dCoordinateOffset from the data buffer

```
# enable the chunk data
ChunkModeActive = True
ChunkSelector = "Scan3dCoordinateScale"
ChunkEnable = True
ChunkSelector = "Scan3dCoordinateOffset"
ChunkEnable = True

# after an acquisition, the configuration is readable from the buffers chunk data
ChunkScan3dCoordinateSelector = "CoordinateC"
scaleC = ChunkScan3dCoordinateScale
offsetC = ChunkScan3dCoordinateOffset
```

The following pseudo code shows the transformation of the submitted 3d data to real-world cordinate.

Listing 2.5: pseudo code how to transform the 3d Data to real-world coordinate data

```
for(row = 0; row < Height; row++)
{
```

(continues on next page)

(continued from previous page)

```
  for(col = 0; col < Width; col++)
  {
    coordC[row,col] = imageC[row,col] * scaleC + offsetC;
  }
}
```

Listing 2.6: test 42

```
```

Listing 2.7: Python example demonstrate this application node

```
##
# \file         an_15.2.x.y.py
# \author       Silvan Murer
# \date         04 Aug 2022
# \copyright    Heliotis, 2022
#
# \brief        Example for an_15.2.x.y
import os, sys
import time
from harvesters.core import Harvester
import genicam.genapi
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import numpy as np
import statistics
from collections import OrderedDict, Mapping
import json

import warnings
warnings.simplefilter('always') # always, ignore


##
# Scan for devices and let the user select one
#
# \param   h harvesters object
# \return  harvesters camera object
def selectDevice(h):
    h.update()
    print(str(len(h.device_info_list)) + " devices detected on the Network\n\n")
    cnt = 0
    for dev in h.device_info_list:
        cnt = cnt + 1
        msg = str(cnt) + ") " + dev.id_ + " (sn:" + dev.serial_number + ")"
        print(msg)

    selection_str = input("Select a device (0=exit): ")
    selection_int = int(selection_str)
    if((selection_int <= 0) or (selection_int > cnt)):
        exit('No device selected - exit script')

    deviceID = h.device_info_list[selection_int-1].id_
    print('selected device:', deviceID)
    return h.create(selection_int-1)


##
# initialize the camera with some default parameters
```

(continues on next page)

(continued from previous page)

```
#
# \param camera camera handler
def initializeDevice(camera):
    # enable required components
    camera.remote_device.node_map.ComponentSelector.value = "Intensity"
    camera.remote_device.node_map.ComponentEnable.value = False
    camera.remote_device.node_map.ComponentSelector.value = "Range"
    camera.remote_device.node_map.ComponentEnable.value = True
    camera.remote_device.node_map.ComponentSelector.value = "Reflectance"
    camera.remote_device.node_map.ComponentEnable.value = False
    camera.remote_device.node_map.ComponentSelector.value = "Phase"
    camera.remote_device.node_map.ComponentEnable.value = False

    # trigger configuration
    camera.remote_device.node_map.TriggerSelector.value = "RecordingStart"
    camera.remote_device.node_map.TriggerMode.value = "On"
    camera.remote_device.node_map.TriggerSource.value = "Stage"
    camera.remote_device.node_map.TriggerSelector.value = "AcquisitionStart"
    camera.remote_device.node_map.TriggerMode.value = "Off"
    camera.remote_device.node_map.TriggerSelector.value = "FrameStart"
    camera.remote_device.node_map.TriggerMode.value = "On"
    camera.remote_device.node_map.TriggerSource.value = "Software"

    # motion and position configuration
    camera.remote_device.node_map.ScanPosition.value = -14.5
    camera.remote_device.node_map.ScanRange.value = 0.5
    camera.remote_device.node_map.ScanSpeed.value = 5.0
    camera.remote_device.node_map.GeneralSpeed.value = 10.0

    camera.remote_device.node_map.ScanMode.value = "Down"
    camera.remote_device.node_map.StageInit.execute()

    # additional camera and processing configuration
    camera.remote_device.node_map.Scan3dExtractionMethod.value =
↪"AcceleratedCenterOfMassIQCorrection"
    camera.remote_device.node_map.Scan3dScalingMethod.value = "zTags"
    camera.remote_device.node_map.Scan3dDistanceUnit.value = "um"

    camera.remote_device.node_map.TargetVerticalSpacing.value = 2.5
    camera.remote_device.node_map.ExposureRatio.value = 1.0

    camera.remote_device.node_map.FPNCorrection.value = "AverageLastFrames"
    camera.remote_device.node_map.FPNCorrectionNFrames.value = 8
    camera.remote_device.node_map.ExtSimpMaxHWin.value = 7

    heliDriverFamily = camera.remote_device.node_map.HeliDriverFamily.value
    # illumination control (D3)
    if (heliDriverFamily == "D3"):
        camera.remote_device.node_map.LightControllerSelector.value =
↪"LightController0"
        camera.remote_device.node_map.LightControllerSource.value = "UserOutput0"
        camera.remote_device.node_map.LightBrightness.value = 100.0
    # illumination control (D2)
    elif (heliDriverFamily == "D2"):
        camera.remote_device.node_map.LineSelector.value = "Line2"
        camera.remote_device.node_map.LineSource.value = "UserOutput0"
        camera.remote_device.node_map.LineInverter.value = 1
    # switch on the illumination (D2/D3)
    camera.remote_device.node_map.UserOutputSelector.value = "UserOutput0"
    camera.remote_device.node_map.UserOutputValue.value = True

```

(continues on next page)

(continued from previous page)

```
108
109 print('*************************************************')
110 print('* ' + str(os.path.basename(__file__)))
111 print('*************************************************')
112
113 try:
114     ctiFile = os.environ['DIAPHUS_GENTL64_FILE']
115
116     h = Harvester()
117     h.add_file(ctiFile)
118
119     camera = selectDevice(h)
120     initializeDevice(camera)
121     print("INIT DONE")
122
123     # configure application node specific features
124     camera.remote_device.node_map.ChunkModeActive.value = True
125     camera.remote_device.node_map.ChunkSelector.value = "Scan3dCoordinateScale"
126     camera.remote_device.node_map.ChunkEnable.value = True
127     camera.remote_device.node_map.ChunkSelector.value = "Scan3dCoordinateOffset"
128     camera.remote_device.node_map.ChunkEnable.value = True
129
130     camera.remote_device.node_map.Scan3dCoordinateSelector.value = "CoordinateC"
131     camera.remote_device.node_map.Scan3dCoordinateScale.value = 0.1
132     camera.remote_device.node_map.Scan3dCoordinateOffset.value = -15000.0
133
134     # setup plot
135     gridsize = (1, 2)
136     fig = plt.figure()
137     axOrgSurf = plt.subplot2grid(gridsize, (0, 0))
138     axRWSurf = plt.subplot2grid(gridsize, (0, 1))
139     fig.suptitle('Measurements')
140     axOrgSurf.set_title('Original Surface')
141     imOrgSurf = axOrgSurf.imshow(np.ones([542, 512]))
142     axRWSurf.set_title('Real-World Surface')
143     imRWSurf = axRWSurf.imshow(np.ones([542, 512]))
144     plt.show(block=False)
145     plt.draw()
146     plt.pause(0.1)
147
148     # acquire data
149     print('start acquisition')
150     camera.start()
151     for itr in range(0, 5):
152         camera.remote_device.node_map.TriggerSelector.value = "FrameStart"
153         camera.remote_device.node_map.TriggerSoftware.execute()
154
155         buffer = camera.fetch(timeout=10.0)
156
157         camera.remote_device.node_map.ChunkScan3dCoordinateSelector.value =
↪"CoordinateC"
158         coordCScale = camera.remote_device.node_map.ChunkScan3dCoordinateScale.
↪value
159         coordCOffset = camera.remote_device.node_map.ChunkScan3dCoordinateOffset.
↪value
160         print(f"ChunkScan3dCoordinateScale: {coordCScale},
↪ChunkScan3dCoordinateOffset: {coordCOffset}")
161
162         height = buffer.payload.components[0].height
163         width = buffer.payload.components[0].width
164         orgSurf = buffer.payload.components[0].data.reshape(height, width)
```

(continues on next page)

(continued from previous page)

```
        orgSurf = np.copy(orgSurf)
        rwSurf = orgSurf * coordCScale + coordCOffset
        orgSurfMean = np.mean(orgSurf)
        rwSurfMean = np.mean(rwSurf)
        print(f"Surface Mean Values: Original {orgSurfMean}, Real-Word:
↪{rwSurfMean}")

        imOrgSurf.set_data(orgSurf)
        imOrgSurf.set_clim(vmin=np.min(orgSurf), vmax=np.max(orgSurf))
        imOrgSurf.set_extent((0, width, height, 0))

        imRWSurf.set_data(rwSurf)
        imRWSurf.set_clim(vmin=np.min(rwSurf), vmax=np.max(rwSurf))
        imRWSurf.set_extent((0, width, height, 0))

        plt.draw()
        plt.pause(0.1)

        buffer.queue()
    camera.stop()
except Exception as error:
    print(error)

camera.destroy()
print("SCRIPT END")
```

2.5 Document History

Version	Date	Changes
15.2.0.0	19-05-2022	initial version

CHAPTER

THREE

TN_15.3.0.1: COLLECTING AND INTERPRETING MONITORING INFORMATION OF GEN4 DEVICES

3.1 Context and Challenge: Finding Root Causes of Non-Systematic Issues

HeliInspect H8, H9, or heliCam C4 devices may suddenly, sporadically, or gradually fail or deliver poor quality data. Hardware defects or environmental issues, including interactions on their interfaces, are possible root causes.

The devices and software tools do, for the time being, not provide accessible log files or interactive monitoring of internal or external operating conditions. Finding root causes of such issues, therefore, requires high debugging efforts.

3.2 Solution: Client-Based System Monitoring

Debugging of issues is simplified or even prevented by collecting and possibly monitoring status information from the device. This should be implemented in the client (consumer) application that controls the heliCam/heliInspect device.

The below tables point at useful status information. Client applications read it throught the standard camera interface at the stages of configuring the camera, after performing a measurement, or in case of an exception. Depending on the type of information and on requirements the application may implement monitoring and/or logging.

3.2.1 Permanently Available Status Information

The client application can read the below device and status information any time as long as it has a connection to the device.

Feature	Value (expected / healthy)
DeviceFirmwareVersion	camera embedded SW version (format e.g. 1.x.y)
DeviceFPGAFwVersion	camera fpga firmware version in cryptic format
HeliDriverFirmwareVersion	heliDriver firmware version (format: e.g. 1.x)
HeliDriverStatus	status of the heliDriver in cryptic format
DeviceTemperature [DeviceTemperatureSelector= *Processor*]	< 80
DeviceMonitorActualValue [DeviceMonitorSelector = *VddVoltage*]	between 21.5 and 26.5
DeviceMonitorActualValue [DeviceMonitorSelector = *VddCurrent*]	< 1.5
DeviceMonitorActualValue [DeviceMonitorSelector = *UserVddVoltage*]	between 21.5 and 26.5
DeviceMonitorActualValue [DeviceMonitorSelector = *UserVddCurrent*]	< 0.15

Note: Refer to the feature description document for data types and detailed information abou the mentioned features.

Note: Some features are sensitive to values of selector features. Refer to the programmer's manual for more information on the concept.

Note: Providing logged information in support requests to heliotis is very helpful. In particular, please contact heliotis for interpreting information in cryptic format.

3.2.2 Status Information Relevant after Configuration

The status information below is readable any time but its content is sensitive to configurations features. The client application should log it after configuring the device.

Feature	Value (expected / healthy)
ActualVerticalSpacing	near **TargetVerticalSpacing**
FrameRate	H8,H9,C4 (542x512 pixel) devices: < 4000 H8M, H9M, C4M (1.1M pixel) devices: < 1000
RecordingNFrame	appoximately: $\frac{ScanRange}{ActualVerticalSpacing}$
TriggerPosition	defined by **ScanMode**, **ScanPosition**, **ScanRange**
LightBrightness [LightControllerSelector = *LightController0*]	10% to 100%

Note: FrameRate maximum values are approximate. The exact maximum depends on further settings.

3.2.3 Status Information of the Acquisition Engine

The status information below represent the device internal states of the acquisition engine. In case of an acquisition timeout the client application could figure out where the acquisition engine got stuck.

Feature	Value
AcquisitionStatus [AcquisitionStatusSelector = *FrameTriggerWait*]	*True* : state active, *False* : state inactive
AcquisitionStatus [AcquisitionStatusSelector = *RecordingTriggerWait*]	*True* : state active, *False* : state inactive
AcquisitionStatus [AcquisitionStatusSelector = *FrameActive*]	*True* : state active, *False* : state inactive
AcquisitionStatus [AcquisitionStatusSelector = *RecordingActive*]	*True* : state active, *False* : state inactive
TransferStatus [TransferStatusSelector = *Streaming*]	*True* : state active, *False* : state inactive
StageStatus [StageStatusSelector = *InMotion*]	*True* : state active, *False* : state inactive

Note: The acquisition engine states are documented in the appendix of the feature description.

3.2.4 Status Information Related to Measurement Data

The following status information is related to measurement data. It should be read after successful measurement and always relates to the latest measuremnt.

Feature	Value (expected / healthy)
LightCurrentMeasured [LightControllerSelector = *LightController0*]	$LightCurrentRating[..] * \frac{LightBrightness[..]}{100\%}$
ChunkZTagCount	equal to **RecordingNFrame**
ChunkZTagValue [ChunkZTagSelector = *1*]	$\frac{TriggerPosition}{EncoderResolution[Camera]}$
ChunkProcessingString	(.json string) dump from on-camera data processing
ChunkConfigString	(.json string) dump of camera configuration

Note: Make sure illumination is enabled when reading the value of **LightCurrentMeasured**. Typical values for LED current rating are 1A (H8) or 2A (H9).

Many of the measurement related pieces of information are transmitted as GenTL ChunkData along with measurement data. The host application must enable transmission by setting the following configuration features already before measurement, e.g. when initializing the camera:

Listing 3.1: Enable relevant chunk data

```
ChunkModeActive = True
ChunkSelector = "ZTagCount"
ChunkEnable = True
ChunkSelector = "ZTagValue"
ChunkEnable = True
ChunkSelector = "ProcessingString"
ChunkEnable = True
ChunkSelector = "ConfigString"
ChunkEnable = True
```

Note: Some of the debugging chunk data like *ConfigurationString* could significantly increase the measurement cycle time. General enabling is not recommended.

3.3 Document History

Version	Date	Changes	Responsible
15.3.0.0	14-11-2022	initial version	cl
15.3.0.1	01-03-2024	review/correction	sm

CHAPTER

FOUR

TN_51.1.0.0: HELIOTIS GEN4 DEVICE FIELDBUS REFERENCE FOR D3

4.1 Introduction

The Heliotis gen4 device family (heliCam™ C4, heliInspect™ H8/H9) offers optional fieldbus capability in combination with a fieldbus module build into the heliDriver™ D3A. Heliotis currently offers modules supporting either PROFINET RT or EtherCAT®. This capability enables the gen4 device to be integrated into industrial networks for data exchange and control. The heliDriver poses as a fieldbus device or slave and can be controlled by a programmable logic controller (PLC). To integrate the device into the fieldbus, both the device and the PLC that controls it need to be configured correctly. This document serves as a reference for the *configuration of both sides* with the use of features on the camera and the description file (ESI/GSD) inclusion and demo projects for the PLC. The document also points to further *information on Heliotis gen4 devices and fieldbuses*.

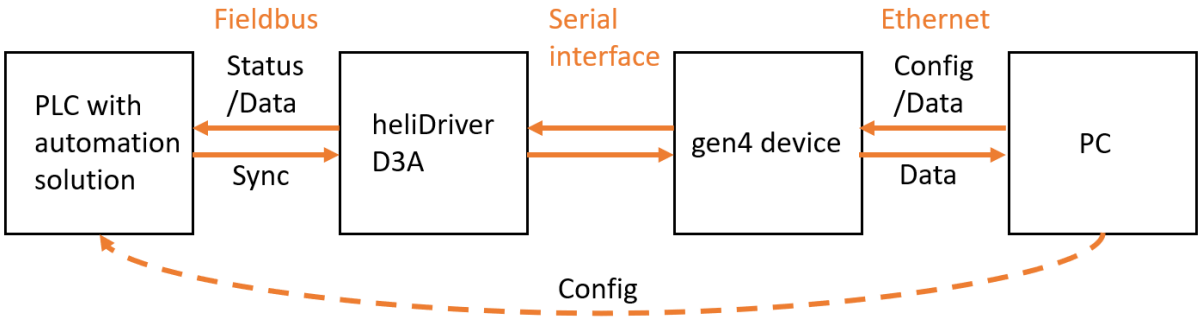

Fig. 4.1: System interconnection

Note: This document only covers gen4 device control via fieldbus but control and status features can also be used with physical I/Os or a combination of both

The following process data objects (PDOs) are available on the network:

Fieldbus names	Camera feature names	Direction
Bool0..7	FieldbusBool0..7	device in / PLC out
Bool8..15	FieldbusBool8..15	device out / PLC in
ApplDataInt0..15	FieldbusIntData0..15	device in / PLC out
ApplDataInt16..31	FieldbusIntData16..31	device out / PLC in
ApplDataFloat0..15	FieldbusFloatData0..15	device in / PLC out
ApplDataFloat16..31	FieldbusFloatData16..31	device out / PLC in

4.2 Example Use Cases

The following chapters describe some common use cases and how to implement them on both the device and the PLC controlling it.

Note: The link to the device description files that need to be imported to the PLC development environment can be found in the *Further Reading* section of this document

4.2.1 "Hello World"

It is recommended to begin with a very basic application to verify that the data exchange between the device and the PLC works in both directions. The following pseudo-code snippets configure a camera output via GenICam features and verify the value of the input PDO on the PLC.

Listing 4.1: Setting an output PDO via camera features

```
FieldbusBoolSelector = 8        #Or any value 8..15
FieldbusBoolSource[FieldbusBoolSelector] = "UserOutput0"  #Or any of UserOutput0..
↪UserOutput7
UserOutputSelector = "UserOutput0" #Must match with the configuration above
UserOutputValue[UserOutputSelector] = True
#True enables (high) the output, False disables (low) the output
```

Listing 4.2: Reading an input PDO on the PLC

```
input_value = Bool8             #According to FieldbusBool from above Listing
↪(Bool8..Bool15)
```

To check the other direction, an output PDO value is set by the PLC and read via the FieldbusBoolStatus feature on the device.

Listing 4.3: Setting an output PDO on the PLC

```
Bool0 = 1                       #Or any bool PDO Bool0..Bool7
```

Listing 4.4: Reading an input PDO via camera features

```
FieldbusBoolSelector = 0        #Or any value according to the listing above (0..7)
input_value = FieldbusBoolStatus[FieldbusBoolSelector]
```

4.2.2 Camera Status

Heliotis gen4 devices offer a range of status flags (e.g. RecordingActive) that can be mapped to output PDOs to relay status information to the PLC.

Listing 4.5: Mapping the RecordingActive status flag to a PDO via camera features

```
FieldbusBoolSelector = 8        #Or any value 8..15
FieldbusBoolSource[FieldbusBoolSelector] = "RecordingActive"  #Maps the
↪"RecordingActive" information to the selected PDO (e.g. 8)
```

Listing 4.6: Reading an input PDO on the PLC

```
recording_active = Bool8        #According to the selected FieldbusBool from above
↪listing (Bool8..Bool15)
```

The diagram below shows the different status flags and how they fit into the acquisition cycle.

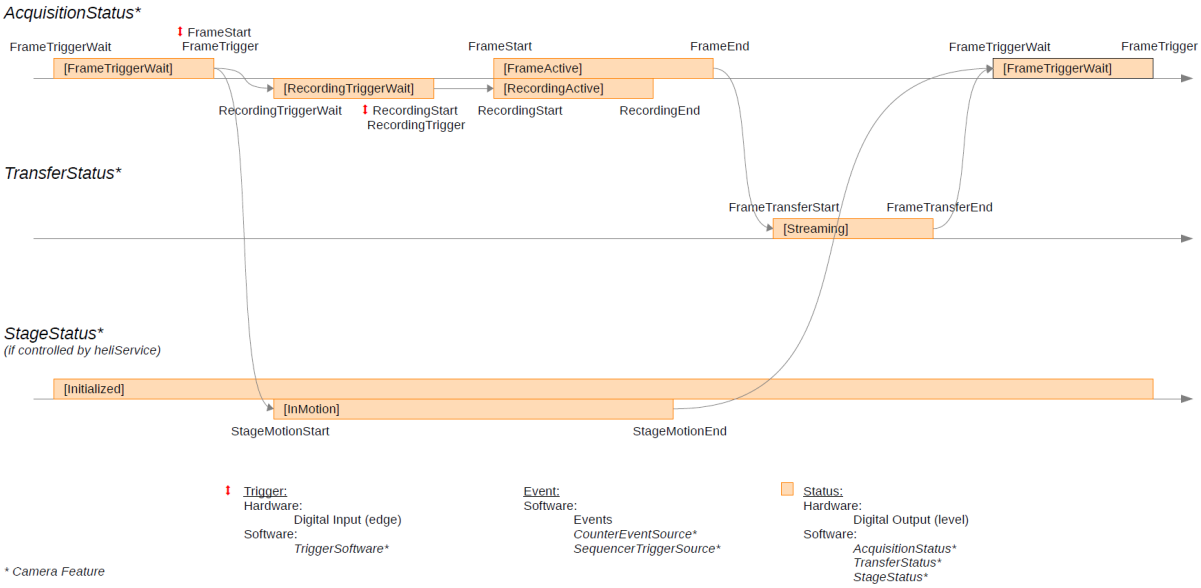

Fig. 4.2: Event and status diagram

Note: The current version of the Event and Status Diagram can be found in the Feature Description appendix (see *Further Reading*) and may differ depending on the camera firmware version.

4.2.3 Acquisition Control

To start a measurement on a Heliotis gen4 device, the TriggerSource feature can be used to map a PDO to a triggersource.

Listing 4.7: Mapping the FrameStart trigger to a PDO via camera features

```
TriggerSelector = "FrameStart"
TriggerSource[TriggerSelector] = "FieldbusBool0"  #Maps the "FrameStart" trigger
↪to the FieldbusBool0 PDO (or any of FieldbusBool0..FieldbusBool7)
```

Listing 4.8: Setting an output PDO on the PLC to trigger the frame start

```
Bool0 = 1       #Select the PDO according to the camera configuration (Bool0..
↪Bool7)

Bool0 = 0       #By default, the camera is triggered on the "RisingEdge" of the
↪signal
#(See TriggerActivation in feature description for more information)
```

4.2.4 Camera Control, Light ON/OFF

The light can be controlled from the PLC via the LightControllerSource feature.

Listing 4.9: Mapping the light controller source to a PDO via camera features

```
LightControllerSelector = "LightController0"

LightControllerSource[LightControllerSelector] = "FieldbusBool0"  #Maps the light
↪control to a PDO (any of FieldbusBool0..FieldbusBool7)
```

Listing 4.10: Setting an output PDO on the PLC to turn on the light

```
Bool0 = 1               #Enables the light

Bool0 = 0               #Disables the light
```

4.2.5 Camera Control, Sequencer Sets

Sequencer sets can be used on Heliotis gen4 devices to switch between multiple camera configuration states triggered by events (see *Event and Status Diagram*). The event source can be mapped onto an edge of a PDO signal.

Listing 4.11: Mapping the sequencer set trigger source to a PDO via camera features

```
SequencerSetSelector = 0
SequencerPathSelector = 0
SequencerTriggerSource[SequencerSetSelector][SequencerPathSelector] =
↪"FieldbusBool0" #Maps the sequencer trigger to FieldbusBool0 (or any of
↪FieldbusBool0..FieldbusBool7)
```

Listing 4.12: Setting an output PDO on the PLC to trigger the sequencer set execution

```
Bool0 = 1

Bool0 = 0               #By default the sequencer set transition is triggered by
↪the "RisingEdge" of the signal. (see SequencerTriggerActivation)
```

4.2.6 Camera Control, Counters

The counter feature set provides the option to count the occurences of specific events on the device. This can be helpful for testing and debugging of an application. It can for example be used to count transitions on a PDO.

Listing 4.13: Mapping the counter event source to a PDO via camera features and reading the counter value

```
CounterSelector = "Counter0"    #Or Counter1
CounterEventSource[CounterSelector] = "FieldbusBool0"  #Or any of FieldbusBool0..
↪FieldbusBool7
CounterEventActivation[CounterSelector] = "RisingEdge" #Can also be "FallingEdge"
↪or "AnyEdge"

cnt_value = CounterValue[CounterSelector]
```

Listing 4.14: Toggling an output PDO on the PLC to increment the counter

```
Bool0 = 1

Bool0 = 0
```

4.2.7 Vision Application, Parameterisation

The use of integer or float PDOs also enables the option to transfer data either to or from the vision application on the gen4 device. For example, the PLC could transfer a target value for a stepheight measurement as a float value. The vision application on the camera uses this value to compare it to the measurement result and set or reset a bool PDO depending on whether the measured value is higher or lower than the target value. The PLC can then use this return value to execute further actions.

Listing 4.15: Using camera features to realize a basic vision application

```
FieldbusBoolSelector = 8        #Or any value 8..15
FieldbusBoolSource[FieldbusBoolSelector] = "UserOutput0" #Or any value UserOutput0.
↪.UserOutput7
UserOutputSelector = "UserOutput0"  #Must match with the configuration above
UserOutputValue[UserOutputSelector] = False #True enables (high) the output, False
↪disables (low) the output
FieldbusFloatDataSelector = 0   #Select the PDO which is used by the PLC to write
↪the step height information (any float PDO between 0..15)

acquisition loop:
        target_step_height = FieldbusFloatDataValue[FieldbusFloatDataSelector]
        measured_step_height = acquire_and_process()
        if (measured_step_height > target_step_height):
                UserOutputValue[UserOutputSelector] = True
        else:
                UserOutputValue[UserOutputSelector] = False
```

Listing 4.16: Transferring measurement parameters and receiving results on the PLC

```
acquisition loop:
        ApplDataFloat0 = TargetStepHeight       #Can be ApplDataFloat0..
↪ApplDataFloat15
        result = Bool8          #Can be Bool8..Bool15
        if (result == 0):
                execute action A
        else:
                execute action B
```

Note: This simplified example has no synchronisation for when the PLC should read the result. This can be achieved by using status features as shown in the *"Camera Status" example*

4.2.8 Vision Application, Result Transmission

Instead of parameterisation and evaluation on the gen4 device, measurement results can also be transmitted directly using integer or float output PDOs.

Listing 4.17: Using camera features to return measurement results

```
FieldbusFloatDataSelector = 16  #Select the PDO the results will be written to
↪(any float PDO between 16..31)

acquisition loop:
        measured_step_height = acquire_and_process()
        FieldbusFloatDataValue[FieldbusFloatDataSelector] = measured_step_height
```

Listing 4.18: Transferring measurement parameters and receiving results on the PLC

```
acquisition loop:
        result = ApplDataFloat16        #Can be ApplDataFloat16..ApplDataFloat31
```

4.3 Further Reading

The following chapter contains links to other documents and hardware ordering information.

With the Heliotis SDK (C4Utility) available on https://www.heliotis.com/support/updates/ (needs a login) further examples and documentation are installed. The installation directory is stored in the environment variable $C4UTILITY_ROOT and is by default C:\Program Files\C4Utility

4.3.1 Heliotis

ftrDesc: The documentation for the camera features can be found in $C4UTILITY_ROOT\doc\ftrDesc.pdf

ProgrammersGuide: Go to https://www.heliotis.com/support/updates/ (needs a login) and download the zip archive labeled “heliInspect H8 Programmer’s Guide”

Examples: $C4UTILITY_ROOT\examples offers examples for a variety of programming languages

Hardware ordering information: Independent of the chosen fieldbus, a heliDriver D3A is always required. Additionally only one of the available fieldbus modules needs to be added to the D3A. The choice of heliInspect (H8/H9) or heliCam (C4) devices has no influence on the fieldbus connectivity and is made independently.

device	type number TN	article number AN
heliDriver D3A	D3A.0	40 81 10
EtherCAT Module	D3MRTETH.0.0-ETHERCAT	40 81 90
PROFINET Module	D3MRTETH.0.0-PROFINET	40 81 95

4.3.2 Beckhoff

Documentation: The user manual for TwinCAT can be found on https://infosys.beckhoff.com/english.php?content=../content

Examples: An EtherCAT example project can be found in $C4UTILITY_ROOT\fieldbus\examples\Beckhoff

Device description file: The EtherCAT Slave Information file (ESI) is located in $C4UTILITY_ROOT\fieldbus\deviceDescription\Beckhoff\heliotisGen4Device_230926.xml

4.3.3 Siemens

Documentation: The TIA Portal user manual can be found on https://cache.industry.siemens.com/dl/files/542/40263542/att_829827/v1/GS_STEP7Bas105enUS.pdf

Examples: An example project for PROFINET is located in $C4UTILITY_ROOT\fieldbus\examples\Siemens

Device description file: The General Station Description file (GSD) is located in $C4UTILITY_ROOT\fieldbus\deviceDescription\Siemens\GSDML-V2.41-Heliotis AG-heliotis gen4 device-20231023.xml

4.4 Document History

Version	Date	Changes	Responsible
51.1.0.0	27-02-2024	Initial version	sb