LLNL Nanosecond Gated Camera

2.1.1

Generated by Doxygen 1.8.17

Chapter 1

CODE_OF_CONDUCT

title: "Code of Conduct"

A *code of conduct* defines standards for how to engage in a community. Such a file signals an inclusive environment that respects all contributions. It also outlines procedures for addressing problems between members of your project's community. The file is typically named in all-caps with underscores: `CODE_OF_CONDUCT.md`.

Our example here is adapted from the website in the Attribution section below. It contains a few blanks ______ where you fill in your project/repo/team's name and/or email address. For more information, see GitHub's advice for `adding a code of conduct to your project`.

Example Community Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism

- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the NSGCC project or its community. Examples of representing the project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of the project may be further defined and clarified by NSGCC maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at `funsten1@llnl.gov` or the LLNL GitHub Admins at `github-admin@llnl.gov` . The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project or organization's leadership.

Attribution

This Code of Conduct is adapted from the `Contributor Covenant` (`version 1.4`).

Chapter 2

CONTRIBUTING

Contributing Guidelines

nsCamera is an open source project. We welcome questions, feature requests, or bug reports at `jerhill@llnl.gov`. We do not yet have a system in place for external contribution, but please contact us if you are interested in contributing. Please also refer to our code of conduct.

Chapter 3

Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 nsCamera Namespace Reference

Namespaces

- boards
- CameraAssembler
- comms
- sensors
- utils

Variables

- list __all__ = ["CameraAssembler.py"]

7.1.1 Variable Documentation

7.1.1.1 __all__

```
list nsCamera.__all__ = ["CameraAssembler.py"]  [private]
```

Definition at line 26 of file __init__.py.

7.2 nsCamera.boards Namespace Reference

Namespaces

- LLNL_v1
- LLNL_v4

Variables

- list __all__ = ["LLNL_v1", "LLNL_v4"]

7.2.1 Variable Documentation

7.2.1.1 __all__

```
list nsCamera.boards.__all__ = ["LLNL_v1", "LLNL_v4"]  [private]
```

Definition at line 23 of file __init__.py.

7.3 nsCamera.boards.LLNL_v1 Namespace Reference

Classes

- class llnl_v1

7.4 nsCamera.boards.LLNL_v4 Namespace Reference

Classes

- class llnl_v4

7.5 nsCamera.CameraAssembler Namespace Reference

Classes

- class CameraAssembler

7.6 nsCamera.comms Namespace Reference

Namespaces

- GigE
- RS422

Variables

- list __all__ = ["RS422", "GigE"]

7.6.1 Variable Documentation

7.6.1.1 __all__

```
list nsCamera.comms.__all__ = ["RS422", "GigE"]  [private]
```

Definition at line 25 of file __init__.py.

7.7 nsCamera.comms.GigE Namespace Reference

Classes

- class GigE

7.8 nsCamera.comms.RS422 Namespace Reference

Classes

- class RS422

7.9 nsCamera.sensors Namespace Reference

Namespaces

- daedalus
- icarus
- icarus2

Variables

- list __all__ = ["icarus", "icarus2", "daedalus"]

7.9.1 Variable Documentation

7.9.1.1 __all__

```
list nsCamera.sensors.__all__ = ["icarus", "icarus2", "daedalus"]  [private]
```

Definition at line 23 of file __init__.py.

7.10 nsCamera.sensors.daedalus Namespace Reference

Classes

- class daedalus

7.11 nsCamera.sensors.icarus Namespace Reference

Classes

- class icarus

7.12 nsCamera.sensors.icarus2 Namespace Reference

Classes

- class icarus2

7.13 nsCamera.utils Namespace Reference

Namespaces

- crc16pure
- FlatField
- GenTec
- Ophir
- Packet
- Subregister

Variables

- list __all__ = ["SubRegister", "Packet", "GenTec", "Ophir", "FlatField"]

7.13.1 Variable Documentation

7.13.1.1 __all__

```
list nsCamera.utils.__all__ = ["SubRegister", "Packet", "GenTec", "Ophir", "FlatField"]  [private]
```

Definition at line 30 of file __init__.py.

7.14 nsCamera.utils.crc16pure Namespace Reference

Functions

- def _crc16 (data, crc, table)
- def crc16xmodem (data, crc=0)

Variables

- list CRC16_XMODEM_TABLE

7.14.1 Detailed Description

```
Pure python library for calculating CRC16
NOTE: modified slightly to combine Python 2 and Python 3 versions in single file
```

7.14.2 Function Documentation

7.14.2.1 _crc16()

```
def nsCamera.utils.crc16pure._crc16 (
              data,
              crc,
              table )  [private]
```

```
Calculate CRC16 using the given table.
`data`      - data for calculating CRC, must be a string
`crc`       - initial value
`table`     - table for caclulating CRC (list of 256 integers)
Return calculated value of CRC
```

Definition at line 299 of file crc16pure.py.

```
def _crc16(data, crc, table):
    """Calculate CRC16 using the given table.
    `data`      - data for calculating CRC, must be a string
    `crc`       - initial value
    `table`     - table for caclulating CRC (list of 256 integers)
    Return calculated value of CRC
    """
    for byte in data:
        if sys.version_info > (3,):
            crc = ((crc « 8) & 0xFF00) ^ table[((crc » 8) & 0xFF) ^ byte]
        else:
            crc = ((crc « 8) & 0xFF00) ^ table[((crc » 8) & 0xFF) ^ ord(byte)]

    return crc & 0xFFFF


```

7.14.2.2 crc16xmodem()

```
def nsCamera.utils.crc16pure.crc16xmodem (
              data,
              crc = 0 )
```

```
Calculate CRC-CCITT (XModem) variant of CRC16.
`data`      - data for calculating CRC, must be a string
`crc`       - initial value
Return calculated value of CRC
```

Definition at line 315 of file crc16pure.py.

```
def crc16xmodem(data, crc=0):
    """Calculate CRC-CCITT (XModem) variant of CRC16.
    `data`      - data for calculating CRC, must be a string
    `crc`       - initial value
    Return calculated value of CRC
    """
    return _crc16(data, crc, CRC16_XMODEM_TABLE)
```

7.14.3 Variable Documentation

7.14.3.1 CRC16_XMODEM_TABLE

```
list nsCamera.utils.crc16pure.CRC16_XMODEM_TABLE
```

Definition at line 39 of file crc16pure.py.

7.15 nsCamera.utils.FlatField Namespace Reference

Functions

- def getFilenames (frame="Frame 1")
- def getROIvector (imgfilename, roi)
- def tslopes (x, y)
- def generateFF (FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"], roi=[0, 0, 512, 1024], directory="", ncores=-1)
- def removeFF (filename, directory="", roi=[0, 0, 512, 1024])
- def removeFFall (directory="", FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"], roi=[0, 0, 512, 1024])

Variables

- parser = argparse.ArgumentParser()
- action
- dest
- default
- help
- nargs
- args = parser.parse_args()
- list framelist = ["Frame_" + str(frame) for frame in args.frames]
- directory

7.15.1 Function Documentation

7.15.1.1 generateFF()

```
def nsCamera.utils.FlatField.generateFF (
                FRAMES = ["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
                roi = [0, 0, 512, 1024],
                directory = "",
                ncores = -1 )
```

Definition at line 58 of file FlatField.py.

```
58 def generateFF(
59     FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
60     roi=[0, 0, 512, 1024],
61     directory="",
62     ncores=-1,
63 ):
64     # TODO: documentation
65     # use of ROI here not compatible with use of ROI in removeFF
66 
67     if directory:
68         cwd = os.getcwd()
69         newpath = os.path.join(cwd, directory)
70         os.chdir(newpath)
71     if not FRAMES:
72         print("No framelist provided, defaulting to four frames")
73         FRAMES = ["Frame_0", "Frame_1", "Frame_2", "Frame_3"]
74     for f in FRAMES:
75         files = getFilenames(frame=f)
76         imgslist = [getROIvector(fn, roi) for fn in files]  # a list of flattened images
77         imgsarray = np.vstack(imgslist)  # turn the list into an array
78         npix = np.shape(imgsarray)[1]  # total number of pixels
79         x = np.median(imgsarray, axis=1)  # median of each image used for flat fielding
80         y = []
81         for i in range(npix):
82             # each member of y represents a pixel, as a list of magnitudes over all the
83             #   images
84             y.append(imgsarray[:, i])
85         # get pixel gain and offset for flatfield ff using Thiel-Sen slopes
86         ff = []
87         ff = parallel.Parallel(n_jobs=ncores, verbose=5, pre_dispatch="2 * n_jobs")(
88             delayed(tslopes)(x, pixel) for pixel in y
89         )
90         # x is the dependent variable; here uses median of image as characteristic of
91         #   noise level
92         m, c = zip(*ff)  # separate into gain and offset
93         m = np.array(m)
94         m[m < 0.1] = 0.1  # handle outliers
95         m[m > 1000] = 1000  # handle outliers
96         m = 1.0 / m
97         m = m.reshape(roi[3] - roi[1], roi[2] - roi[0])  # turn into matrix
98         c = np.array(c).reshape(roi[3] - roi[1], roi[2] - roi[0])  # turn into matrix
99 
100         with open("px_gain_%s.txt" % f.replace("Frame_", "f"), "w+") as file:
101             np.savetxt(file, m)
102         with open("px_off_%s.txt" % f.replace("Frame_", "f"), "w+") as file:
103             np.savetxt(file, c)
104 
105 
```

7.15.1.2 getFilenames()

```
def nsCamera.utils.FlatField.getFilenames (
                frame = "Frame 1" )
```

```
get a list of tiff filenames in current working director for frame
```

Definition at line 32 of file FlatField.py.

```
32 def getFilenames(frame="Frame 1"):
33     """
34     get a list of tiff filenames in current working director for frame
35     """
36     onlyfiles = next(os.walk("./"))[2]
37     return [k for k in onlyfiles if frame in k and "tif" in k]
38 
39 
```

7.15.1.3 getROIvector()

```
def nsCamera.utils.FlatField.getROIvector (
              imgfilename,
              roi )
```

```
return a numpy row vector of version of the image
```

Definition at line 40 of file FlatField.py.

```
40 def getROIvector(imgfilename, roi):
41     """
42     return a numpy row vector of version of the image
43     """
44     img = imread(imgfilename)
45     vroi = img[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])].flatten()
46     return vroi
47 
48 
```

7.15.1.4 removeFF()

```
def nsCamera.utils.FlatField.removeFF (
              filename,
              directory = "",
              roi = [0, 0, 512, 1024] )
```

Definition at line 106 of file FlatField.py.

```
106 def removeFF(filename, directory="", roi=[0, 0, 512, 1024]):
107     if directory:
108         cwd = os.getcwd()
109         newpath = os.path.join(cwd, directory)
110         os.chdir(newpath)
111     framenum = re.search("Frame_(\d)", filename).group(1)
112     gainFilename = "px_gain_f" + framenum + ".txt"
113     gainall = np.loadtxt(gainFilename)
114     gain = gainall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
115     offFilename = "px_off_f" + framenum + ".txt"
116     offsetall = np.loadtxt(offFilename, dtype="uint32")
117     offset = offsetall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
118 
119     beforeImageall = imread(filename)
120     beforeImage = beforeImageall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
121     imageMed = np.median(beforeImage)
122 
123     flat = imageMed * gain + offset
124     flat = flat.clip(0)
125     fix = beforeImage - flat
126     clipped = fix.clip(0)
127     fixinit = clipped.astype("uint16")
128     fiximg = Image.fromarray(fixinit)
129 
130     fixFilename = filename[:-4] + "ff" + filename[-4:]
131     fiximg.save(fixFilename)
132 
```

7.15.1.5 removeFFall()

```
def nsCamera.utils.FlatField.removeFFall (
              directory = "",
              FRAMES = ["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
              roi = [0, 0, 512, 1024] )
```

Definition at line 133 of file FlatField.py.

```
def removeFFall(
    directory="",
    FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
    roi=[0, 0, 512, 1024],
):
    cwd = os.getcwd()
    if directory:
        newpath = os.path.join(cwd, directory)
    else:
        newpath = cwd
    os.chdir(newpath)
    files = next(os.walk("./"))[2]
    filelist = []
    for frame in FRAMES:
        filelist.extend([k for k in files if frame in k and "tif" in k])
    for fname in filelist:
        removeFF(fname, directory, roi)
```

7.15.1.6 tslopes()

```
def nsCamera.utils.FlatField.tslopes (
              x,
              y )
```

```
theilslopes implements a method for robust linear regression.
It computes the slope as the median of all slopes between paired values.
```

Definition at line 49 of file FlatField.py.

```
def tslopes(x, y):
    """
    theilslopes implements a method for robust linear regression.
    It computes the slope as the median of all slopes between paired values.
    """
    val = theilslopes(x, y)
    return [val[0], val[1]]
```

7.15.2 Variable Documentation

7.15.2.1 action

```
nsCamera.utils.FlatField.action
```

Definition at line 157 of file FlatField.py.

7.15.2.2 args

```
nsCamera.utils.FlatField.args = parser.parse_args()
```

Definition at line 167 of file FlatField.py.

7.15.2.3 default

```
nsCamera.utils.FlatField.default
```

Definition at line 157 of file FlatField.py.

7.15.2.4 dest

```
nsCamera.utils.FlatField.dest
```

Definition at line 157 of file FlatField.py.

7.15.2.5 directory

```
nsCamera.utils.FlatField.directory
```

Definition at line 169 of file FlatField.py.

7.15.2.6 framelist

```
nsCamera.utils.FlatField.framelist = ["Frame_" + str(frame) for frame in args.frames]
```

Definition at line 168 of file FlatField.py.

7.15.2.7 help

```
nsCamera.utils.FlatField.help
```

Definition at line 157 of file FlatField.py.

7.15.2.8 nargs

```
nsCamera.utils.FlatField.nargs
```

Definition at line 161 of file FlatField.py.

7.15.2.9 parser

```
nsCamera.utils.FlatField.parser = argparse.ArgumentParser()
```

Definition at line 155 of file FlatField.py.

7.16 nsCamera.utils.GenTec Namespace Reference

Classes

- class GenTec

Variables

- gt = GenTec()

7.16.1 Variable Documentation

7.16.1.1 gt

```
nsCamera.utils.GenTec.gt = GenTec()
```

Definition at line 92 of file GenTec.py.

7.17 nsCamera.utils.Ophir Namespace Reference

Classes

- class Ophir

7.18 nsCamera.utils.Packet Namespace Reference

Classes

- class Packet

7.19 nsCamera.utils.Subregister Namespace Reference

Classes

- class SubRegister

7.20 nsCameraExample Namespace Reference

Variables

- string BOARD = "LLNL_v4"

  (1) Initialization (REQUIRED) ###
- string COMM = "GigE"
- string SENSOR = "icarus2"
- ca = CameraAssembler(commname=COMM, boardname=BOARD, sensorname=SENSOR, verbose=4)
- timing

  (2) Timing (OPTIONAL) ##
- tune

  (3) Customization (OPTIONAL) ##

7.20.1 Variable Documentation

7.20.1.1 BOARD

```
string nsCameraExample.BOARD = "LLNL_v4"
```

(1) Initialization (REQUIRED) ###

The CameraAssembler code initializes and manages objects corresponding to the three components that comprise a particular nsCamera system. The 'verbose' flag controls the output of status messages as the code executes.

Definition at line 31 of file nsCameraExample.py.

7.20.1.2 ca

```
nsCameraExample.ca = CameraAssembler(commname=COMM, boardname=BOARD, sensorname=SENSOR, verbose=4)
```

Definition at line 42 of file nsCameraExample.py.

7.20.1.3 COMM

```
string nsCameraExample.COMM = "GigE"
```

Definition at line 34 of file nsCameraExample.py.

7.20.1.4 SENSOR

```
string nsCameraExample.SENSOR = "icarus2"
```

Definition at line 38 of file nsCameraExample.py.

7.20.1.5 timing

```
nsCameraExample.timing
```

(2) Timing (OPTIONAL) ##

The initialization phase sets the default high-speed timing parameters. Override these settings here; alternatively, switch to manual shutter control

Definition at line 63 of file nsCameraExample.py.

7.20.1.6 tune

```
nsCameraExample.tune
```

(3) Customization (OPTIONAL) ###

The initialization phase sets default operating parameters. These parameters may be overridden by explicit directives as shown here. setPotV (potname, voltage) - Voltage is float setPotV sets contents of pot 'potname' to the value corresponding to 'voltage' based on board.monVmin and board.monVmax Valid 'name' entries are listed as keys in the 'channel_lookups' dictionary in the board code

Definition at line 82 of file nsCameraExample.py.

7.21 testSuite Namespace Reference

Functions

- def testSuite (board, comm, sensor, portNum, ipAdd, interactive=True, swtrigger=True)

Variables

- parser = argparse.ArgumentParser()
- action
- dest
- default
- help
- None
- args = parser.parse_args()
- interactive
- swtrigger
- board
- comm
- sensor
- portNum
- ipAdd

7.21.1 Function Documentation

7.21.1.1 testSuite()

```
def testSuite.testSuite (
              board,
              comm,
              sensor,
              portNum,
              ipAdd,
              interactive = True,
              swtrigger = True )
```

```
Regression testing script to exercise cameraAssembler functions and camera features.
Comment out entries in 'tests' list to skip component tests

Args:
    board: board name for cameraAssembler
    comm: comm name for cameraAssembler
    sensor: sensor name for cameraAssembler
    portNum: (optional) port number
    ipAdd: (optional) ip address (e.g., '192.168.1.100')
    interactive: if False, does not wait for user input, skips some tests
    swtrigger: if True, uses software triggering, does not wait for external
      triggers
```

Definition at line 39 of file testSuite.py.

```
39 def testSuite(board, comm, sensor, portNum, ipAdd, interactive=True, swtrigger=True):
40     """
41     Regression testing script to exercise cameraAssembler functions and camera features.
42     Comment out entries in ’tests’ list to skip component tests
43 
44     Args:
45         board: board name for cameraAssembler
46         comm: comm name for cameraAssembler
47         sensor: sensor name for cameraAssembler
48         portNum: (optional) port number
49         ipAdd: (optional) ip address (e.g., ’192.168.1.100’)
50         interactive: if False, does not wait for user input, skips some tests
51         swtrigger: if True, uses software triggering, does not wait for external
52           triggers
53 
54     """
55     # Comment out any tests you wish to skip. Irrelevant tests (e.g., Manual Timing if
56     #   Daedalus sensor is attached) will be ignored
57     tests = [
58         "HST acquisition",
59         "SaveTiffs",
60         "PlotFrames",
61         "POT/DAC set & read",
62         "SW Trigger",  # perform SW trigger test when HW triggering selected
63         "Arm/Disarm",
64         "HST setting",
65         "Reinitialization",
66         "PowerSave",
67         "Timer",
68         "Register R/W",
69         "Register self-clear",
70         "Register dump",
71         "LED",
72         "Manual Timing",
73     ]
74 
75     def statusVerify(caObject, checklist):
76         errs = 0
77         for stat, flag in checklist:
78             _, check = caObject.getSubregister(stat)
79             if bool(int(check)) is not bool(flag):
80                 errs += 1
81                 print("+Error: " + stat + " is not " + str(bool(flag)))
82         if not errs:
83             print("+Status verify passed")
84 
85     errtemp = 0
86     sensorregs = {}
87     boardregs = {}
88     boardselfclear = {}
89 
90     def test_v1(ca):
91         print("\n# LLNL_v1 board-specific checks")
92         if "LED" in tests:
93             print("\nRolling LEDs")
94             ca.enableLED(0)
95             ca.enableLED(1)
96             for i in range(1, 5):
97                 for j in range(1, 9):
98                     ca.setLED(j, 1)
99                     time.sleep(0.05 * i)
100                     ca.setLED(j, 0)
101 
102         if "POT/DAC set & read" in tests:
103             time.sleep(1)
104             print("\n-Pot check-")
105             for i in [2, 3, 4, 6, 8]:
106                 potname = "POT" + str(i)
107                 monname = "MON_CH" + str(i)
108                 print("Testing " + potname)
109                 temperr = 0
110                 for j in range(7):
111                     desired = j * 0.5
112                     potobj = getattr(ca.board, potname)
113                     minvolt = potobj.resolution
114                     ca.setPotV(potname, desired, tune=True)
115                     actual = ca.getMonV(monname)
116                     # skip v=0, we expect it to be off
117                     if abs(desired - actual) > minvolt and bool(desired):
118                         print(
                            "{0:.2f} : actual = {1:.5f} ; delta = {2:.2f} mV".format(
                                (1.0 * desired), actual, 1000 * abs(actual - desired)
                            )
                        )
                        temperr = 1
                if not temperr:
                    print("+" + potname + " tunes properly")

        v1regs = OrderedDict(
            {
                "ADC_RESET": "0000001F",
                "ADC5_CONFIG_DATA": "FFFFFFFF",
                "POT_REG4_TO_1": "FFFFFFFF",
                "POT_REG8_TO_5": "FFFFFFFF",
                "POT_REG12_TO_9": "FFFFFFFF",
                "POT_REG13": "000000FF",
                "ADC5_PPER": "0FFFFFFF",
                "LED_GP": "000000FF",
                "ADC_STANDBY": "0000001F",
                "TEMP_SENSE_PPER": "0FFFFFFF",
                "SENSOR_VOLT_CTL": "00000001",
            }
        )
        v1selfclear = OrderedDict({"POT_CTL": "00000001",})

        v1restore = [
            ("ADC5_PPER", "001E8480"),
            ("ADC_RESET", "00000000"),
            ("ADC5_CONFIG_DATA", "81A883FF"),
            ("ADC_CTL", "00000010"),
        ]

        return v1regs, v1selfclear, v1restore

    def test_v4(ca):
        print("\n# LLNL_v4 board-specific checks")
        if "POT/DAC set & read" in tests:
            print("\n-DAC check-")
            for i in ["A", "B", "C", "D", "E", "F", "G", "H"]:
                dacname = "DAC" + i
                monname = "MON_CH" + i
                print("Testing " + dacname)
                temperr = 0
                for j in range(7):
                    desired = j * 0.5
                    # semi-arbitrary, need to adjust to minimize search time
                    minvolt = 0.005
                    ca.setPotV(dacname, desired, tune=True)
                    actual = ca.getMonV(monname)
                    # skip v=0, we expect it to be off
                    if abs(desired - actual) > minvolt and bool(desired):
                        print(
                            "{0:.2f} : actual = {1:.5f} ; delta = {2:.2f} mV".format(
                                (1.0 * desired), actual, 1000 * abs(actual - desired)
                            )
                        )
                        temperr = 1
                if not temperr:
                    print("+" + dacname + " tunes properly")

        v4regs = OrderedDict(
            {
                "ADC_RESET": "0000000F",
                "DAC_REG_A_AND_B": "FFFFFFFF",
                "DAC_REG_C_AND_D": "FFFFFFFF",
                "DAC_REG_E_AND_F": "FFFFFFFF",
                "DAC_REG_G_AND_H": "FFFFFFFF",
            }
        )

        v4selfclear = OrderedDict(
            {"DAC_CTL": "00000001", "SW_COARSE_CONTROL": "FFFFFFFF",}
        )

        v4restore = [
            ("ADC_PPER", "001E8480"),
            ("ADC_RESET", "00000000"),
        ]

        return v4regs, v4selfclear, v4restore

```

```
def test_icarus(ca, interactive, swtrigger):
    print("\n# Icarus sensor-specific checks")

    if "Manual Timing" in tests:
        # MANUAL TIMING & ACQUISITION
        print("\n-Testing manual shutter control-")
        ca.setManualShutters(
            timing=[
                (100, 100, 100, 100, 100, 100, 100),
                (100, 100, 100, 100, 100, 100, 100),
            ]
        )
        statusVerify(ca, [("MANSHUT_MODE", 1), ("STAT_HSTCONFIGURED", 0)])
        print(
            "The next few messages should include two error messages about "
            "invalid timing sequences "
        )
        time.sleep(1)
        ca.setManualShutters(timing=[(100, 100, 100, 100, 100, 100, 100)])
        ca.setManualShutters(
            timing=[
                (10.5, 100, 100, 100, 100, 100, 100),
                (100, 100, 100, 100, 100, 100, 100),
            ]
        )
        time.sleep(1)

        print("\n-Testing manual shutter acquisition-")
        if swtrigger:
            print("Using software trigger")
            statusVerify(
                ca,
                [
                    ("STAT_ADCSCONFIGURED", 1),
                    (ca.potsdacsconfigured, 1),
                    ("STAT_HSTCONFIGURED", 0),
                    ("MANSHUT_MODE", 1),
                ],
            )
            ca.arm("Software")
            statusVerify(ca, [("STAT_COARSE", 1), ("STAT_FINE", 1)])
        else:
            ca.arm()
            if interactive:
                ca.getEnter(
                    "> Please initiate hardware trigger, then press ENTER to "
                    "continue. <\n "
                )

        frames, datalen, data_err = ca.readoff(waitOnSRAM=True)
        print("Data length: " + str(datalen))
        if data_err:
            print("+Error in acquisition!")
        else:
            print("+No error in acquisition reported")

        if interactive:
            if "PlotFrames" in tests:
                print(
                    "Plots of the acquired frames are being displayed. Please "
                    "inspect to verify proper acquisition. "
                    "\n> Close plots to continue <"
                )
                ca.plotFrames(frames)
            if "SaveTiffs" in tests:
                ca.saveTiffs(frames, filename="msc_test")
                ca.getEnter(
                    "Tiff files of the acquired frames have been saved. Please "
                    "inspect to verify correct saves. "
                    "\n> Press ENTER to continue <\n"
                )

        else:
            if "SaveTiffs" in tests:
                ca.saveTiffs(frames, filename="msc_test")
                print("Tiffs from manual shutter control test have been saved")

        # REINITIALIZATION WITH MANUAL SHUTTERS
        if "Reinitialization" in tests and interactive:
            ca.setManualShutters(
                timing=[
                    (25, 50, 75, 100, 125, 150, 175),
                    (175, 150, 125, 100, 75, 50, 25),
                ]
            )
            ca.getEnter(
                "\n-Testing reinitialization with manual shutter control-\n> "
                "Please power-cycle the board, then press ENTER to continue <"
            )
            time.sleep(1)
            if ca.powerCheck():
                print("\n+Loss of power WAS NOT detected")
            else:
                print("\n+Loss of power WAS detected")
            time.sleep(1)
            ca.reinitialize()
            statusVerify(ca, [("STAT_TIMERCOUNTERRESET", 1)])
            ca.sensor.getManualTiming()
            if ca.sensor.getManualTiming() != [
                [25, 50, 75, 100, 125, 150, 175],
                [175, 150, 125, 100, 75, 50, 25],
            ]:
                print(
                    "+Manual timing WAS NOT restored properly after "
                    "reinitialization "
                )
            else:
                print(
                    "+Manual timing WAS restored properly after "
                    "reinitialization "
                )

    icarusregs = OrderedDict(
        {
            "VRESET_WAIT_TIME": "7FFFFFFF",
            "ICARUS_VER_SEL": "00000001",
            "MANUAL_SHUTTERS_MODE": "00000001",
            "W0_INTEGRATION": "03FFFFFF",
            "W0_INTERFRAME": "03FFFFFF",
            "W1_INTEGRATION": "03FFFFFF",
            "W1_INTERFRAME": "03FFFFFF",
            "W2_INTEGRATION": "03FFFFFF",
            "W2_INTERFRAME": "03FFFFFF",
            "W3_INTEGRATION": "03FFFFFF",
            "W0_INTEGRATION_B": "03FFFFFF",
            "W0_INTERFRAME_B": "03FFFFFF",
            "W1_INTEGRATION_B": "03FFFFFF",
            "W1_INTERFRAME_B": "03FFFFFF",
            "W2_INTEGRATION_B": "03FFFFFF",
            "W2_INTERFRAME_B": "03FFFFFF",
            "W3_INTEGRATION_B": "03FFFFFF",
        }
    )

    if ca.sensorname == "icarus":
        icarusregs["VRESET_HIGH_VALUE"] = "000000FF"

    return icarusregs

def test_daedalus(ca, interactive, swtrigger):
    print("\n# Daedalus sensor-specific checks")
    daedalusregs = {}  # TODO: add daedalus registers when available
    return daedalusregs

print("-Initial setup-")
ca = CameraAssembler(
    commname=comm,
    boardname=board,
    sensorname=sensor,
    verbose=5,
    port=portNum,
    ip=ipAdd,
)
ca.potsdacsconfigured = "STAT_DACSCONFIGURED"

statusVerify(ca, [("STAT_TIMERCOUNTERRESET", 1)])

if "PowerSave" in tests:
    print("\n-Testing PowerSave mode-")
    ca.setPowerSave(1)
    statusVerify(ca, [("POWERSAVE", 1)])
    ca.setPowerSave(0)
    statusVerify(ca, [("POWERSAVE", 0)])

# ARM/DISARM
if "Arm/Disarm" in tests:
    print("\n-Testing Arm-")
    # HST has to be set for arming to complete; tested separately later
    ca.setTiming("A", (2, 2))
    ca.setTiming("B", (2, 2))
    ca.arm()
    time.sleep(1)
    statusVerify(
        ca,
        [
            ("STAT_ADCSCONFIGURED", 1),
            (ca.potsdacsconfigured, 1),
            ("STAT_COARSE", 0),
            ("STAT_FINE", 0),
            ("STAT_ARMED", 1),
        ],
    )
    time.sleep(1)
    print("\n-Testing Disarm-")
    ca.disarm()
    statusVerify(ca, [("STAT_ARMED", 0)])
    time.sleep(1)

# HIGH SPEED TIMING & ACQUISITION
if "HST setting" in tests:
    print("\n-Testing high speed timing control-")
    ca.setTiming("A", (39, 1), 0)
    ca.setTiming("B", (1, 1), 31)
    if not ("A", 39, 1, 0) == ca.getTiming("A"):
        errtemp = 1
    if not ("B", 1, 1, 37) == ca.getTiming("B"):
        errtemp = 1
    if errtemp:
        print("Error in setting high speed timing")
        errtemp = 0

    ca.setTiming("A", (5, 2), 3)
    ca.setTiming("B", (3, 4), 1)
    if not ("A", 5, 2, 3) == ca.getTiming("A"):
        errtemp = 1
    if not ("B", 3, 4, 1) == ca.getTiming("B"):
        errtemp = 1
    if errtemp:
        print("Error in setting high speed timing")
        errtemp = 0

    ca.setTiming("B", (10, 10), 0)

    time.sleep(1)
    print(
        "The next few messages should include a warning about inter-frame timing:"
    )
    time.sleep(1)
    ca.setTiming("A", (9, 8), 1)
    print(
        "The next few messages should include a error message regarding timing "
        "sequence: "
    )
    time.sleep(1)
    ca.setTiming("A", (15, 15), 15)

    statusVerify(ca, [("STAT_HSTCONFIGURED", 1), ("MANSHUT_MODE", 0)])
    time.sleep(1)

# ca.setInterlacing(2)  # TODO: sensor-specific testing?

if "HST acquisition" in tests:
    print("\n-Testing HST acquisition-")
    ca.arm()
    time.sleep(1)

    ca.reportStatus()
    statusVerify(
        ca,
        [
            ("STAT_ADCSCONFIGURED", 1),
            (ca.potsdacsconfigured, 1),
            ("STAT_HSTCONFIGURED", 1),
            ("MANSHUT_MODE", 0),
            ("STAT_COARSE", 0),
            ("STAT_FINE", 0),
        ],
    )
    if swtrigger:
        print("Using software trigger")
        ca.arm("Software")
        statusVerify(ca, [("STAT_COARSE", 1), ("STAT_FINE", 1)])

    else:
        if interactive:
            ca.getEnter(
                "> Please initiate hardware trigger, then press ENTER to "
                "continue. <\n "
            )

    frames, datalen, data_err = ca.readoff(waitOnSRAM=True)

    margin = 1600
    print(
        "Check of dummy sensor; number of pixels exceeding margin of "
        + str(margin)
        + " from dummy sensor expected value:"
    )
    for frame in frames:
        bads, diff = ca.dummyCheck(frame, margin)
        print(bads)

    print("Data length: " + str(datalen) + " bytes")
    if data_err:
        print("+Error in acquisition!")
    else:
        print("+No error in acquisition reported")

    if interactive:
        if "PlotFrames" in tests:
            print(
                "\nPlots of the acquired frames are being displayed. Please "
                "inspect to verify proper acquisition. "
                "\n> Close plots to continue <"
            )
            ca.plotFrames(frames)
        if "SaveTiffs" in tests:
            ca.saveTiffs(frames, filename="hst_test")
            ca.getEnter(
                "Tiff files of the acquired frames have been saved. Please "
                "inspect to verify correct saves. "
                "\n> Press ENTER to continue. <\n"
            )
    else:
        if "SaveTiffs" in tests:
            ca.saveTiffs(frames, filename="hst_test")
            print("Tiffs from HST test have been saved")

    if not swtrigger and "SW Trigger" in tests:
        print("\n-Testing HST acquisition with software trigger-")
        ca.arm()
        time.sleep(1)

        ca.reportStatus()
        statusVerify(
            ca,
            [
                ("STAT_ADCSCONFIGURED", 1),
                (ca.potsdacsconfigured, 1),
                ("STAT_HSTCONFIGURED", 1),
                ("MANSHUT_MODE", 0),
                ("STAT_COARSE", 0),
                ("STAT_FINE", 0),
            ],
        )

        ca.arm("Software")
        statusVerify(ca, [("STAT_COARSE", 1), ("STAT_FINE", 1)])

        frames, datalen, data_err = ca.readoff(waitOnSRAM=True)

        margin = 1600
        print(
            "Check of dummy sensor; number of pixels exceeding margin of "
            + str(margin)
            + " from dummy sensor expected value:"
        )
        for frame in frames:
            bads, diff = ca.dummyCheck(frame, margin)
            print(bads)

        print("Data length: " + str(datalen) + " bytes")
        if data_err:
            print("+Error in acquisition!")
        else:
            print("+No error in acquisition reported")

        if interactive:
            if "PlotFrames" in tests:
                print(
                    "\nPlots of the acquired frames are being displayed. Please "
                    "inspect to verify proper acquisition. "
                    "\n> Close plots to continue <"
                )
                ca.plotFrames(frames)
            if "SaveTiffs" in tests:
                ca.saveTiffs(frames, filename="SWtrig_test")
                ca.getEnter(
                    "Tiff files of the acquired frames have been saved. Please "
                    "inspect to verify correct saves. "
                    "\n> Press ENTER to continue. <\n"
                )
        else:
            if "SaveTiffs" in tests:
                ca.saveTiffs(frames, filename="SWtrig_test")
                print("Tiffs from HST test have been saved")

# REINITIALIZATION
if "Reinitialization" in tests and interactive:
    time.sleep(1)
    ca.setTiming("A", (2, 3), 4)
    ca.setTiming("B", (5, 3), 1)
    time.sleep(1)
    ca.getEnter(
        "\n-Testing reinitialization with high speed timing-\n>Please power-cycle "
        "the board, then press ENTER to continue <"
    )
    if ca.powerCheck():
        print("\n+Loss of power WAS NOT detected")
    else:
        print("\n+Loss of power WAS detected")
    time.sleep(1)
    ca.reinitialize()
    statusVerify(ca, [("STAT_TIMERCOUNTERRESET", 1)])
    if ("A", 2, 3, 4) != ca.getTiming("A") or ("B", 5, 3, 1,) != ca.getTiming("B"):
        print("+High speed timing WAS NOT restored properly after reinitialization")
    else:
        print("+High speed timing WAS restored properly after reinitialization")

if ca.sensorname == "icarus" or ca.sensorname == "icarus2":
    sensorregs = test_icarus(ca, interactive, swtrigger)
elif ca.sensorname == "daedalus":
    sensorregs = test_daedalus(ca, interactive, swtrigger)

# MISCELLANEOUS
print("\n\n-Testing miscellaneous board features-")

if "Timer" in tests:
    print("Checking on-board timer reset")
    ca.resetTimer()
    ztime = ca.getTimer()
    if not ztime:
        print("+Timer reset check successful")
    else:
        print("+Timer reset failed, timer reads " + str(ztime))
    statusVerify(ca, [("STAT_TIMERCOUNTERRESET", 1)])

print("Temperature sensor reading: " + str(ca.getTemp()))
time.sleep(1)

if ca.boardname == "llnl_v1":
    ca.potsdacsconfigured = "STAT_POTSCONFIGURED"
    boardregs, boardselfclear, boardrestore = test_v1(ca)
elif ca.boardname == "llnl_v4":
    ca.potsdacsconfigured = "STAT_DACSCONFIGURED"
    boardregs, boardselfclear, boardrestore = test_v4(ca)

if "POT/DAC set & read" in tests:
    print("\n-VRST check-")
    for a in (0, 0.05, 0.15, 0.25, 0.5, 0.75, 1, 3, 3.5):
        ca.setPotV("VRST", voltage=a, tune=True)
        actual = ca.getMonV("VRST")
        print(
            "{0:.2f} : actual = {1:.5f} ; delta = {2:.2f} mV".format(
                (1.0 * a), actual, 1000 * abs(actual - a)
            )
        )

if "Register R/W" in tests:
    print("\n\n-Verifying register read/writes-\n")
    regchecklist = OrderedDict(
        {  # register name: writable bits
            # read-only, write-only, and self-clearing registers are skipped
            "HS_TIMING_DATA_ALO": "FFFFFFFF",
            "HS_TIMING_DATA_AHI": "000000FF",
            "HS_TIMING_DATA_BLO": "FFFFFFFF",
            "HS_TIMING_DATA_BHI": "000000FF",
            "CTRL_REG": "0000000F",
            "HST_SETTINGS": "00000003",
            "DIAG_MAX_CNT_0": "FFFF00FF",
            "DIAG_MAX_CNT_1": "FFFFFFFF",
            "TRIGGER_CTL": "00000003",
            "FPA_ROW_INITIAL": "000003FF",
            "FPA_ROW_FINAL": "000003FF",
            "FPA_FRAME_INITIAL": "00000003",
            "FPA_FRAME_FINAL": "00000003",
            "FPA_DIVCLK_EN_ADDR": "00000001",
            "FPA_OSCILLATOR_SEL_ADDR": "00000003",
            "ADC1_CONFIG_DATA": "FFFFFFFF",
            "ADC2_CONFIG_DATA": "FFFFFFFF",
            "ADC3_CONFIG_DATA": "FFFFFFFF",
            "ADC4_CONFIG_DATA": "FFFFFFFF",
            "ADC_RESET": "0000001F",
        }
    )

    regchecklist.update(sensorregs)
    regchecklist.update(boardregs)

    checkvals = ["00000000", "FFFFFFFF"]

    for reg, mask in regchecklist.items():
        temperr = 0
        for val in checkvals:
            valmasked = "{0:0=8x}".format(int(val, 16) & int(mask, 16))
            if temperr:
                continue
            if not ca.checkRegSet(reg, valmasked):
                temperr = 1
                continue
        if not temperr:
            print("+ {: <24} - R/W OK".format(reg))

    ca.submitMessages(boardrestore)

if "Register self-clear" in tests:
    time.sleep(1)
    print("\n\n-Verifying self-clearing registers-\n")

    selfclear = OrderedDict(
        {  # register name: writable bits
            "HS_TIMING_CTL": "FFFFFFFF",  # Read-write registers
            "TIMER_CTL": "FFFFFFFF",
            "ADC_CTL": "FFFFFFFF",
            "STAT_REG_SRC": "00004FFF",  # Read-only registers
            "STAT_REG2_SRC": "FFFFFFFF",
        }
    )

    selfclear.update(boardselfclear)

    for reg, mask in selfclear.items():
        ca.setRegister(reg, "FFFFFFFF")
        ca.getRegister(reg)
        time.sleep(0.1)
        _, resp = ca.getRegister(reg)
        masked = int(resp, 16) & int(mask, 16)

        if not masked:

            print("+ {: <17} - self-clear OK".format(reg))
        else:
            print(
                "+ {: <17} - self-clear FAIL: ".format(reg)
                + "0x"
                + "{0:0=8x}".format(masked)
            )

    ca.submitMessages(boardrestore)

if "Register dump" in tests:
    time.sleep(1)
    print("\n\n-Register dump-\n")
    print("\n".join(ca.dumpRegisters()))

ca.closeDevice()
time.sleep(1)
logging.info("Done")


```

7.21.2 Variable Documentation

7.21.2.1 action

`testSuite.action`

Definition at line 717 of file testSuite.py.

7.21.2.2 args

`testSuite.args = parser.parse_args()`

Definition at line 749 of file testSuite.py.

7.21.2.3 board

`testSuite.board`

Definition at line 754 of file testSuite.py.

7.21.2.4 comm

```
testSuite.comm
```

Definition at line 755 of file testSuite.py.

7.21.2.5 default

```
testSuite.default
```

Definition at line 719 of file testSuite.py.

7.21.2.6 dest

```
testSuite.dest
```

Definition at line 718 of file testSuite.py.

7.21.2.7 help

```
testSuite.help
```

Definition at line 720 of file testSuite.py.

7.21.2.8 interactive

```
testSuite.interactive
```

Definition at line 752 of file testSuite.py.

7.21.2.9 ipAdd

```
testSuite.ipAdd
```

Definition at line 758 of file testSuite.py.

7.21.2.10 None

```
testSuite.None
```

Definition at line 737 of file testSuite.py.

7.21.2.11 parser

```
testSuite.parser = argparse.ArgumentParser()
```

Definition at line 713 of file testSuite.py.

7.21.2.12 portNum

```
testSuite.portNum
```

Definition at line 757 of file testSuite.py.

7.21.2.13 sensor

```
testSuite.sensor
```

Definition at line 756 of file testSuite.py.

7.21.2.14 swtrigger

```
testSuite.swtrigger
```

Definition at line 753 of file testSuite.py.

Chapter 8

Class Documentation

8.1 nsCamera.CameraAssembler.CameraAssembler Class Reference

Public Member Functions

- def __init__ (self, boardname="llnl_v4", commname="GigE", sensorname="icarus2", verbose=4, port=None, ip=None, logfile=None, logtag=None)
- def initBoard (self)

    Aliases to other objects' methods.
- def initPots (self)
- def latchPots (self)
- def initSensor (self)
- def configADCs (self)
- def disarm (self)
- def startCapture (self, mode)
- def readSRAM (self)
- def waitForSRAM (self, timeout=None)
- def getTimer (self)
- def resetTimer (self)
- def enableLED (self, status=1)
- def setLED (self, LED=1, status=1)
- def setPowerSave (self, status=1)
- def setPPER (self, time=None)
- def getTemp (self, scale=None)
- def getPressure (self, offset=None, sensitivity=None, units=None)
- def clearStatus (self)
- def checkStatus (self)
- def checkStatus2 (self)
- def reportStatus (self)
- def reportEdgeDetects (self)
- def dumpStatus (self)
- def checkSensorVoltStat (self)
- def setTiming (self, side=None, sequence=None, delay=None)
- def setArbTiming (self, side=None, sequence=None)

- def getTiming (self, side=None, actual=None)
- def setManualShutters (self, timing=None)
- def getManualTiming (self)
- def sensorSpecific (self)
- def setInterlacing (self, ifactor=None)
- def setHighFullWell (self, flag=True)
- def setZeroDeadTime (self, flag=True)
- def setTriggerDelay (self, delayblocks=0)
- def parseReadoff (self, frames)
- def sendCMD (self, pkt)
- def arm (self, mode=None)
- def readoff (self, waitOnSRAM=None, timeout=0, fast=None)
- def writeSerial (self, cmd, timeout=None)
- def readSerial (self, size, timeout=None)
- def closeDevice (self)
- def initialize (self)

  End aliases.
- def reinitialize (self)
- def reboot (self)
- def getBoardInfo (self)
- def getRegister (self, regname)
- def setRegister (self, regname, regval)
- def resolveSubreg (self, srname)
- def getSubregister (self, subregname)
- def setSubregister (self, subregname, valstring)
- def submitMessages (self, messages, errorstring="Error")
- def getPot (self, potname, errflag=False)
- def setPot (self, potname, value=1.0, errflag=False)
- def getPotV (self, potname, errflag=False)
- def setPotV (self, potname, voltage, tune=False, accuracy=0.01, iterations=20, approach=0.75, errflag=False)
- def getMonV (self, monname, errflag=False)
- def readImgs (self, waitOnSRAM=True, mode="Hardware")
- def deInterlace (self, frames, ifactor=1)
- def saveFrames (self, frames, path=None, filename="frames", prefix=None)
- def saveTiffs (self, frames, path=None, filename="Frame", prefix=None, index=None)
- def saveNumpys (self, frames, path=None, filename="Frame", prefix=None, index=None)
- def dumpNumpy (self, datastream, path=None, filename="Dump", prefix=None)
- def plotFrames (self, frames, index=None)
- def checkCRC (self, rval)
- def checkRegSet (self, regname, teststring)
- def initPowerCheck (self)
- def powerCheck (self, delta=10)
- def dummyCheck (self, image, margin, dummyVals=None)
- def printBoardInfo (self)
- def dumpRegisters (self)
- def dumpSubregisters (self)
- def str2bytes (self, astring)
- def bytes2str (self, bytesequence)
- def str2nparray (self, valstring)
- def flatten (self, x)

- def getEnter (self, text)
- def mmReadoff (self, waitOnSRAM, variation=None)
- def setFrames (self, minframe=None, maxframe=None)
- def setRows (self, minrow=0, maxrow=None, fullsize=False)
- def generateFrames (self, data)
- def abortReadoff (self, flag=True)
- def batchAcquire (self, sets=1, trig="Hardware", path=None, filename="Frame", prefix=None, showProgress=0)
- def loadTextFrames (self, filename='frames.txt', path=None)

Public Attributes

- version
- currtime
- oldtime
- trigtime
- waited
- read
- unstringed
- parsedtime
- savetime
- cycle
- boardname
- commname
- sensorname

  For regular version.
- verbose
- port
- PY3
- platform
- FPGAVersion
- FPGANum
- FPGAboardtype
- FPGArad
- FPGAsensor
- FPGAinterfaces
- FPGAinvalid
- iplist
- packageroot
- armed
- senstiming
- sensmanual
- inittime
- padToFull
- abort
- verbmap
- logtag
- logcritbase
- logerrbase
- logwarnbase

- loginfobase
- logdebugbase
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- verblevel
- payloaderror
- sensor
- comms
- board

8.1.1 Detailed Description

```
Code to assemble correct code to manage FPGA, frame grabber, and sensor

Exposed methods:
    initialize() - initializes board registers and pots, sets up sensor
    reinitialize() - initialize board and sensors, restore last known timer settings
    reboot() - perform software reset of board and reinitialize
    getBoardInfo() - parses FPGA_NUM register to retrieve board description
    getRegister(regname) - retrieves contents of named register
    setRegister(regname, string) - sets named register to given value
    resolveSubreg(srname) - resolves alias and retrieves object associated with
      srname
    getSubregister(subregname) - return substring of register identified in board
      attribute 'subregname'
    setSubregister(subregname, valstring) - replace substring of register identified
      in board attribute 'subregname' with 'valstring'
    submitMessages(messages) - set registers or subregisters based on list of
      destination/payload tuples
    getPot(potname) - returns float (0 < value < 1) corresponding to integer stored
      in pot or monitor 'potname'
    setPot(potname, value) - 0 < value < 1; sets named pot to fixed-point number =
      'value' * (maximum pot value)
    getPotV(potname) - returns voltage setting of 'potname'
    setPotV(potname, voltage) - sets named pot to voltage
    getMonV(monname) - returns voltage read by monitor 'monname' (or monitor
      associated with given potname)
    readImgs() - calls arm() and readoff() functions
    deInterlace(frames, interlacing) - extract interlaced frames
    saveFrames(frames) - save image object as one file
    saveTiffs(frames) - save individual frames as tiffs
    saveNumpys(frames) - save individual frames as numpy data files
    dumpNumpy(datastream) - save datastream string to numpy file
    plotFrames(frames) - plot individual frames as tiffs
    checkCRC(string) - checks last four characters of string is valid CRC for rest
      of string
    checkRegSet(register, string) - test set and get register functions for named
      register
    initPowerCheck() - start timers for power continutity check
    powerCheck(delta) - check that board power has not failed
    dummyCheck(image, margin) - counts how many pixels differ from expected dummy
      sensor values by more than margin
    printBoardInfo() - print board information derived from FPGA_NUM register
    dumpRegisters() - return contents of all board registers
    dumpSubregisters() - return contents of all named board subregisters
    str2bytes(string) - convert hexadecimal string to byte string
    bytes2str(sequence) - convert byte sequence to hexadecimal string
    str2nparray(string) - convert string of hexadecimal values into uint16 array
    flatten(llist) - flattens list of lists into single list
    getEnter(text) - print text, then wait for Enter keypress
    mmReadoff(waitflag, variation) - convenience function for MicroManager plugin
    setFrames(min, max) - select subset of frames for readoff
    setRows(min, max, fullsize) - select subset of rows for readoff
    generateFrames(data) - processes data stream from board into frames
    abortReadoff() - cancel readoff in wait-for-SRAM loop
    batchAquire() - fast acquire a finite series of images
    loadTextFrames() - load data sets previously saved as text and convert to frames

Includes aliases to board- and sensor- specific functions:
    Board functions
        initBoard() - initialize default board register settings and configures ADCs
        initPots() - configure default pot settings before image acquisition
        latchPots() - latch all pot settings into sensor
        initSensor() - register sensor, set default timing settings
        configADCs() - set default ADC configuration
        startCapture() - reads ADC data into SRAM
        disarm() - take camera out of waiting-for-trigger state
        readSRAM() - trigger read from SRAM
        waitForSRAM() - puts board in wait state until data are ready in SRAM
        clearStatus() - clear contents of status registers
        checkStatus() - print contents of status register as reversed bit string
        checkStatus2() - print contents of status register 2 as reversed bit string
        reportStatus() - print report on contents of status registers
        resetTimer() - reset on-board timer
        getTimer() - read on-board timer
        enableLED(status) - enable (default) or disable (status = 0) on-board LEDs
        setLED(LED#, status) - turn LED on (default) or off (status = 0)
        setPowerSave(status) - turn powersave functionality on (default) or off
          (status = 0)
        getPressure() - read on-board pressure sensor
        getTemp() - read on-board temperature sensor
        checkStatus() - read and return status bits in status register 1
        checkStatus2() - read and return status bits in status register 2
        clearStatus() - clear status registers 1 and 2
        reportStatus() - print out human-readable board status report based on
          status registers
    Sensor functions
        checkSensorVoltStat() - checks that jumper settings match sensor selection
        setTiming(side, sequencetuple, delay) - configure high-speed timing
        setArbTiming(side, sequencelist) - configure arbitrary high-speed timing
          sequence
        getTiming(side) - returns high speed timing settings from registers
        setManualShutters() - configures manual shutter timing
        getManualTiming() - returns manual shutter settings from registers
        sensorSpecific() - returns register settings specific to implemented sensor
        setInterlacing(ifactor) - sets interlacing factor
        setHighFullWell(flag) - controls High Full Well mode
        setZeroDeadTime(flag) - controls Zero Dead Time mode
        setTriggerDelay(delayblocks) - sets trigger delay
        parseReadoff(frames) - performs sensor-specific parsing and separation of
          images
    Comms functions
        sendCMD(pkt)- sends packet object via serial port
        arm() - configures software buffers & arms camera
        readoff() - waits for data ready flag, then downloads image data
        writeSerial(cmdString)- submits string 'cmdstring' (usually string is
          preformed packet)
        readSerial(stringlength) - reads string of length 'stringlength' from serial
          port
        closeDevice() - disconnect interface and release resources
Informational class variables:
    version - nsCamera software version
    FPGAVersion - firmware version (date)
    FPGANum - firmware implementation identifier
    FPGAboardtype - FPGA self-identified board type (should match 'boardname')
    FPGArad = Boolean indicating radiation-tolerant FPGA build
    FPGAsensor = FPGA self-identified sensor family (should correspond to
      'sensorname')
    FPGAinterfaces = FPGA self-identified interfaces (list should include
      'commname')
    FPGAinvalid = invalid FPGA information in register
```

Definition at line 47 of file CameraAssembler.py.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 __init__()

```
def nsCamera.CameraAssembler.CameraAssembler.__init__ (
              self,
              boardname = "llnl_v4",
              commname = "GigE",
              sensorname = "icarus2",
              verbose = 4,
              port = None,
              ip = None,
              logfile = None,
              logtag = None )
```

```
Args:
    boardname: name of FPGA board: llnl_v1, llnl_v4
    commname: name of communication interface: rs422, gige
    sensorname: name of sensor: icarus, icarus2, daedalus
    verbose: optional, sets logging level
0: print no logging messages
1: print CRITICAL logging messages (camera will not operate, e.g.,
  unable to connect to board)
2: print ERROR logging messages (camera will not operate as directed,
  e.g., an attempt to set the timing mode has failed, but the camera
  is still operational)
3: print WARNING logging messages (camera will operate as directed, but
  perhaps not as expected, e.g., ca.setTiming('A', (9, 8), 1) may be
  programmed correctly, but the actual timing generated by the board
  will be {1} [9, 8, 9, 14, 9, 8, 9]
4: print INFO logging messages (operational messages from ordinary
  camera operation)
    port: optional integer
RS422: preselects comport for RS422, bypasses port search
GigE: preselect OrangeTree control port for GigE (ignored if ip option
  not also given)
    ip: optional string (e.g., '192.168.1.100')
GigE: bypasses network search and selects particular OrangeTree board -
  required for some operating systems
    logfile: optional string, name of file to divert console output
    errtag: suffix to add to logging labels
```

Definition at line 170 of file CameraAssembler.py.

```
170     def __init__(
171         self,
172         boardname="llnl_v4",
173         commname="GigE",
174         sensorname="icarus2",
175         verbose=4,
176         port=None,
177         ip=None,
178         logfile=None,
179         logtag=None,
180     ):
181         """
182         Args:
183             boardname: name of FPGA board: llnl_v1, llnl_v4
184             commname: name of communication interface: rs422, gige
185             sensorname: name of sensor: icarus, icarus2, daedalus
186             verbose: optional, sets logging level
187                 0: print no logging messages
188                 1: print CRITICAL logging messages (camera will not operate, e.g.,
189                   unable to connect to board)
190                 2: print ERROR logging messages (camera will not operate as directed,
191                   e.g., an attempt to set the timing mode has failed, but the camera
192                   is still operational)
193                 3: print WARNING logging messages (camera will operate as directed, but
194                   perhaps not as expected, e.g., ca.setTiming(’A’, (9, 8), 1) may be
195                   programmed correctly, but the actual timing generated by the board
196                   will be {1} [9, 8, 9, 14, 9, 8, 9]
197                 4: print INFO logging messages (operational messages from ordinary
198                   camera operation)
199             port: optional integer
200                 RS422: preselects comport for RS422, bypasses port search
201                 GigE: preselect OrangeTree control port for GigE (ignored if ip option
202                   not also given)
203             ip: optional string (e.g., ’192.168.1.100’)
204                 GigE: bypasses network search and selects particular OrangeTree board -
205                   required for some operating systems
206             logfile: optional string, name of file to divert console output
207             errtag: suffix to add to logging labels
208         """
209         self.version = "2.1.1"
210         self.currtime = 0
211         self.oldtime = 0
212         self.trigtime = []
213         self.waited = []
214         self.read = []
215         self.unstringed = []
216         self.parsedtime = []
217         self.savetime = []
218         self.cycle = []
219         self.boardname = boardname.lower()
220         if self.boardname == "llnlv1":
221             self.boardname = "llnl_v1"
222         if self.boardname == "llnlv4":
223             self.boardname = "llnl_v4"
224         self.commname = commname.lower()
225         self.sensorname = sensorname.lower()
226         self.verbose = verbose
227         self.port = port
228         self.python, self.pyth1, self.pyth2, _, _ = sys.version_info
229         self.PY3 = self.python >= 3
230         self.platform = platform.system()
231         self.arch, _ = platform.architecture()
232 
233         self.FPGAVersion = ""
234         self.FPGANum = ""
235         # FPGA information here and below populated during initialization using
236         #   getBoardInfo
237         self.FPGAboardtype = ""
238         self.FPGArad = False
239         self.FPGAsensor = ""
240         self.FPGAinterfaces = []
241 
242         # indicates invalid FPGA information in register# (0x80000001 accepted as valid)
243         self.FPGAinvalid = False
244 
245         self.iplist = None
246         self.packageroot = os.path.dirname(inspect.getfile(CameraAssembler))
247         self.armed = False
248 
249         # only one of these collections (senstiming, sensmanual) should be nonempty at
```

```
        #   any given time
        self.senstiming = {}  # preserve HST setting against possible power failure
        self.sensmanual = []  # preserve manual timing
        self.inittime = 0
        self.padToFull = False
        self.abort = False

        self.verbmap = {
            0: 99,
            1: logging.CRITICAL,
            2: logging.ERROR,
            3: logging.WARNING,
            4: logging.INFO,
            5: logging.DEBUG,
        }
        if logtag is None:
            logtag = ""
        self.logtag = logtag
        self.logcritbase = "CRITICAL" + self.logtag + ": "
        self.logerrbase = "ERROR" + self.logtag + ": "
        self.logwarnbase = "WARNING" + self.logtag + ": "
        self.loginfobase = "INFO" + self.logtag + ": "
        self.logdebugbase = "DEBUG" + self.logtag + ": "

        self.logcrit = self.logcritbase + "[CA] "
        self.logerr = self.logerrbase + "[CA] "
        self.logwarn = self.logwarnbase + "[CA] "
        self.loginfo = self.loginfobase + "[CA] "
        self.logdebug = self.logdebugbase + "[CA] "

        self.verblevel = self.verbmap.get(verbose, 5)  # defaults to 5 for invalid entry

        if logfile:
            logging.basicConfig(format="%(message)s", filename=logfile)
        else:
            logging.basicConfig(format="%(message)s")
        logging.getLogger().setLevel(self.verblevel)
        logging.getLogger("matplotlib.font_manager").disabled = True

        if ip:
            try:
                iphex = socket.inet_aton(ip)
            except socket.error:
                logging.critical(self.logcrit + "CameraAssembler: invalid IP provided")
                sys.exit(1)
            ipnum = [0, 0, 0, 0]
            for i in range(4):
                if self.PY3:
                    ipnum[i] = iphex[i]
                else:
                    ipnum[i] = int(iphex[i].encode("hex"), 16)
            self.iplist = ipnum

        self.payloaderror = False
        self.initialize()

```

8.1.3 Member Function Documentation

8.1.3.1 abortReadoff()

```
def nsCamera.CameraAssembler.CameraAssembler.abortReadoff (
                self,
                flag = True )
```

```
Simple abort command for readoff in waiting mode--does not interrupt download in
   progress. Requires external threading to function. WARNING: if not
   intercepted by active readoff command, will terminate next readoff command
   immediately at inception.
Args:
    flag: Sets passive abort flag read by readoff command
Returns:
    boolean: updated setting of flag
```

Definition at line 1937 of file CameraAssembler.py.

```
    def abortReadoff(self, flag=True):
        """
        Simple abort command for readoff in waiting mode--does not interrupt download in
           progress. Requires external threading to function. WARNING: if not
           intercepted by active readoff command, will terminate next readoff command
           immediately at inception.
        Args:
            flag: Sets passive abort flag read by readoff command
        Returns:
            boolean: updated setting of flag
        """
        self.abort = flag
        return flag

```

8.1.3.2 arm()

```
def nsCamera.CameraAssembler.CameraAssembler.arm (
              self,
              mode = None )
```

Definition at line 416 of file CameraAssembler.py.

```
    def arm(self, mode=None):
        return self.comms.arm(mode)

```

8.1.3.3 batchAcquire()

```
def nsCamera.CameraAssembler.CameraAssembler.batchAcquire (
              self,
              sets = 1,
              trig = "Hardware",
              path = None,
              filename = "Frame",
              prefix = None,
              showProgress = 0 )
```

```
Acquire a series of images as fast as possible, then process and save to disk.

Args:
    sets: Number of acquisitions to perform
    path: save path, defaults to './output'
    filename: defaults to 'frames.bin'
    prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
```

```
      DO NOT USE unless providing a varying value (a fixed prefix will cause
      overwriting)
    showProgress: if non-zero, show notice every 'showProgress' acquisitions and
      print total acquisition time

Returns:
    Time taken for acquisition (seconds)
```

Definition at line 1951 of file CameraAssembler.py.

```
    def batchAcquire(
        self,
        sets=1,
        trig="Hardware",
        path=None,
        filename="Frame",
        prefix=None,
        showProgress=0,
    ):
        """
        Acquire a series of images as fast as possible, then process and save to disk.

        Args:
            sets: Number of acquisitions to perform
            path: save path, defaults to './output'
            filename: defaults to 'frames.bin'
            prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
              DO NOT USE unless providing a varying value (a fixed prefix will cause
              overwriting)
            showProgress: if non-zero, show notice every 'showProgress' acquisitions and
              print total acquisition time

        Returns:
            Time taken for acquisition (seconds)
        """
        datalist = ["0"] * sets
        timelist = [datetime.now()] * sets
        logging.info(
            self.loginfo
            + "batchAcquire: temporarily disabling warning and information "
            "logging "
        )
        logging.getLogger().setLevel(self.verbmap.get(2))
        beforeread = time.time()
        for i in range(sets):
            if showProgress and not (i + 1) % showProgress:
                print(self.loginfo + "batchAcquire: Acquiring set " + str(i + 1))
            self.arm(trig)
            data, datalen, data_err = self.readoff(fast=True)
            datalist[i] = data
            timelist[i] = datetime.now()
        afterread = time.time()
        if showProgress:
            print(
                self.loginfo
                + "batchAcquire: "
                + str(afterread - beforeread)
                + " seconds for "
                + str(sets)
                + " sets"
            )
        setnum = 0
        if path is None:
            path = os.path.join(os.getcwd(), "output")
        for (imset, imtime) in zip(datalist, timelist):
            setnum = setnum + 1
            if showProgress and not setnum % showProgress:
                print(self.loginfo + "batchAcquire: Saving set " + str(setnum))
            parsed = self.generateFrames(imset)
            if prefix is None:
                setprefix = imtime.strftime("%y%m%d-%H%M%S%f")[:-2] + "_"
            else:
                setprefix = prefix
            self.saveTiffs(parsed, path, filename, prefix=setprefix)
        logging.getLogger().setLevel(self.verblevel)
        logging.info(self.loginfo + "batchAcquire: re-enabling logging")
        return afterread - beforeread

```

8.1.3.4 bytes2str()

```
def nsCamera.CameraAssembler.CameraAssembler.bytes2str (
              self,
              bytesequence )
```

```
Python-version-agnostic converter of bytes to hexadecimal strings

Args:
    bytesequence: sequence of bytes as string (Py2) or bytes (Py3)

Returns:
    hexadecimal string representation of 'bytes' without '0x'
```

Definition at line 1686 of file CameraAssembler.py.

```
    def bytes2str(self, bytesequence):
        """
        Python-version-agnostic converter of bytes to hexadecimal strings

        Args:
            bytesequence: sequence of bytes as string (Py2) or bytes (Py3)

        Returns:
            hexadecimal string representation of 'bytes' without '0x'
        """
        estring = binascii.b2a_hex(bytesequence)
        if self.PY3:
            estring = str(estring)[2:-1]
        return estring

```

8.1.3.5 checkCRC()

```
def nsCamera.CameraAssembler.CameraAssembler.checkCRC (
              self,
              rval )
```

```
Calculate CRC for rval[:-4] and compare with expected CRC in rval[-4:]

Args:
    rval: hexadecimal string

Returns:
    boolean, True if CRCs match
```

Definition at line 1472 of file CameraAssembler.py.

```
    def checkCRC(self, rval):
        """
        Calculate CRC for rval[:-4] and compare with expected CRC in rval[-4:]

        Args:
            rval: hexadecimal string

        Returns:
            boolean, True if CRCs match
        """
        data_crc = int(rval[-4:], base=16)
        CRC_calc = crc16pure.crc16xmodem(self.str2bytes(rval[:-4]))
        return CRC_calc == data_crc

```

8.1.3.6 checkRegSet()

```
def nsCamera.CameraAssembler.CameraAssembler.checkRegSet (
                self,
                regname,
                teststring )
```

```
Quick check to confirm that data read from register matches data write

Args:
    regname: register to test
    teststring: value to assign to register, as hexadecimal string without ’0x’

Returns:
    boolean, True if read and write values match
```

Definition at line 1486 of file CameraAssembler.py.

```
1486     def checkRegSet(self, regname, teststring):
1487         """
1488         Quick check to confirm that data read from register matches data write
1489 
1490         Args:
1491             regname: register to test
1492             teststring: value to assign to register, as hexadecimal string without ’0x’
1493 
1494         Returns:
1495             boolean, True if read and write values match
1496         """
1497         self.setRegister(regname, teststring)
1498         # tell board to send data; wait to clear before interrogating register contents
1499         if regname == "SRAM_CTL":
1500             time.sleep(2)
1501             if self.commname == "rs422":
1502                 logging.info(
1503                     self.loginfo + "skipping ’SRAM_CTL’ register check for RS422"
1504                 )
1505                 return True
1506         else:
1507             time.sleep(0.1)
1508         temp = self.getRegister(regname)
1509         resp = temp[1].upper()
1510         if resp != teststring.upper():
1511             logging.error(
1512                 self.logerr
1513                 + "checkRegSet failure: "
1514                 + regname
1515                 + " ; set: "
1516                 + teststring
1517                 + " ; read: "
1518                 + resp
1519             )
1520             return False
1521         return True
1522 
```

8.1.3.7 checkSensorVoltStat()

```
def nsCamera.CameraAssembler.CameraAssembler.checkSensorVoltStat (
                self )
```

Definition at line 377 of file CameraAssembler.py.

```
377     def checkSensorVoltStat(self):
378         return self.sensor.checkSensorVoltStat()
379 
```

8.1.3.8 checkStatus()

```
def nsCamera.CameraAssembler.CameraAssembler.checkStatus (
              self )
```

Definition at line 362 of file CameraAssembler.py.

```
362     def checkStatus(self):
363         return self.board.checkStatus()
364
```

8.1.3.9 checkStatus2()

```
def nsCamera.CameraAssembler.CameraAssembler.checkStatus2 (
              self )
```

Definition at line 365 of file CameraAssembler.py.

```
365     def checkStatus2(self):
366         return self.board.checkStatus2()
367
```

8.1.3.10 clearStatus()

```
def nsCamera.CameraAssembler.CameraAssembler.clearStatus (
              self )
```

Definition at line 359 of file CameraAssembler.py.

```
359     def clearStatus(self):
360         return self.board.clearStatus()
361
```

8.1.3.11 closeDevice()

```
def nsCamera.CameraAssembler.CameraAssembler.closeDevice (
              self )
```

Definition at line 428 of file CameraAssembler.py.

```
428     def closeDevice(self):
429         return self.comms.closeDevice()
430
```

8.1.3.12 configADCs()

```
def nsCamera.CameraAssembler.CameraAssembler.configADCs (
              self )
```

Definition at line 320 of file CameraAssembler.py.

```
320     def configADCs(self):
321         return self.board.configADCs()
322
```

8.1.3.13 deInterlace()

```
def nsCamera.CameraAssembler.CameraAssembler.deInterlace (
              self,
              frames,
              ifactor = 1 )
```

```
Extracts interlaced frames. If interlacing does not evenly divide the height,
  remainder lines will be dropped
Args:
    frames: list of full-sized frames
    ifactor: interlacing factor; number of interlaced lines (generates
      ifactor + 1 images per frame)

Returns: list of deinterlaced frames
```

Definition at line 1183 of file CameraAssembler.py.

```
1183     def deInterlace(self, frames, ifactor=1):
1184         """
1185         Extracts interlaced frames. If interlacing does not evenly divide the height,
1186           remainder lines will be dropped
1187         Args:
1188             frames: list of full-sized frames
1189             ifactor: interlacing factor; number of interlaced lines (generates
1190               ifactor + 1 images per frame)
1191
1192         Returns: list of deinterlaced frames
1193         """
1194         if ifactor == 0:  # don’t do anything
1195             return frames
1196         warntrimmed = False
1197         if self.padToFull:
1198             newheight = self.sensor.maxheight // (ifactor + 1)
1199             if newheight != (self.sensor.maxheight / (ifactor + 1)):
1200                 warntrimmed = True
1201         else:
1202             newheight = self.sensor.height // (ifactor + 1)
1203             if newheight != (self.sensor.height / (ifactor + 1)):
1204                 warntrimmed = True
1205
1206         if warntrimmed:
1207             logging.warning(
1208                 self.logwarn + "deInterlace: interlacing setting requires dropping of "
1209                 "lines to maintain consistent frame sizes "
1210             )
1211         delaced = []
1212         for frame in frames:
1213             for sub in range(ifactor + 1):
1214                 current = np.zeros((newheight, self.sensor.width), dtype=int)
1215                 for line in range(newheight):
1216                     current[line] = frame[(ifactor + 1) * line + sub]
1217                 delaced.append(current)
1218         return delaced
1219
```

8.1.3.14 disarm()

```
def nsCamera.CameraAssembler.CameraAssembler.disarm (
              self )
```

Definition at line 323 of file CameraAssembler.py.

```
323     def disarm(self):
324         return self.board.disarm()
325
```

8.1.3.15 dummyCheck()

```
def nsCamera.CameraAssembler.CameraAssembler.dummyCheck (
              self,
              image,
              margin,
              dummyVals = None )
```

```
Compare image with 'canonical' dummy sensor image (actual values estimated)

Args:
    image: numpy array containing frame image
    margin: maxmimum allowed error for sensor
    dummyVals: condensed array of expected dummy sensor image values

Returns:
    tuple, (number of pixels exceeding difference margin, numpy array containing
      image subtracted from expected dummy image)
```

Definition at line 1553 of file CameraAssembler.py.

```
1553     def dummyCheck(self, image, margin, dummyVals=None):
1554         """
1555         Compare image with 'canonical' dummy sensor image (actual values estimated)
1556
1557         Args:
1558             image: numpy array containing frame image
1559             margin: maxmimum allowed error for sensor
1560             dummyVals: condensed array of expected dummy sensor image values
1561
1562         Returns:
1563             tuple, (number of pixels exceeding difference margin, numpy array containing
1564               image subtracted from expected dummy image)
1565         """
1566         if dummyVals is None:
1567             dummyVals = self.board.dummySensorVals
1568         stripe0 = []
1569         stripe1 = []
1570         for i in range(16):
1571             stripe0.append([dummyVals[0][i]] * 32)
1572             stripe1.append([dummyVals[1][i]] * 32)
1573         stripet = [val for sublist in stripe0 for val in sublist]
1574         stripeb = [val for sublist in stripe1 for val in sublist]
1575         testVals = [stripet] * 512 + [stripeb] * 512
1576         testimage = np.array(testVals)
1577         if image.size == testimage.size:
1578             image.shape = (self.sensor.height, self.sensor.width)
1579             diff = testimage - image
1580             diffabs = [abs(i) for sublist in diff for i in sublist]
1581             bads = sum(1 for i in diffabs if i > margin)
1582             return bads, diff
1583         else:
1584             logging.error(
1585                 self.logerr + "dummyCheck: Image size does not match dummy image; "
1586                 "returning zero, actual testimage "
1587             )
1588             return 0, testimage
1589
```

8.1.3.16 dumpNumpy()

```
def nsCamera.CameraAssembler.CameraAssembler.dumpNumpy (
              self,
              datastream,
              path = None,
              filename = "Dump",
              prefix = None )
```

```
Datastream is converted directly to numpy array and saved to disk. No attempt
  to parse headers or separate into individual frames is made.

Args:
    datastream: string to be saved
    path: save path, defaults to './output'
    filename: defaults to 'Dump'
    prefix: prepended to 'filename', defaults to time/date
      (e.g. '160830-124704_')

Returns:
    Error string
```

Definition at line 1389 of file CameraAssembler.py.

```
    def dumpNumpy(
        self, datastream, path=None, filename="Dump", prefix=None,
    ):
        """
        Datastream is converted directly to numpy array and saved to disk. No attempt
          to parse headers or separate into individual frames is made.

        Args:
            datastream: string to be saved
            path: save path, defaults to './output'
            filename: defaults to 'Dump'
            prefix: prepended to 'filename', defaults to time/date
              (e.g. '160830-124704_')

        Returns:
            Error string
        """
        logging.info(self.loginfo + "dumpNumpy")
        err = ""
        if path is None:
            path = os.path.join(os.getcwd(), "output")
        if prefix is None:
            prefix = time.strftime("%y%m%d-%H%M%S_", time.localtime())
        if not os.path.exists(path):
            os.makedirs(path)
        npdata = self.str2nparray(datastream)
        try:
            nppath = os.path.join(path, prefix + filename + ".npy")
            np.save(nppath, npdata)
        except:
            err = self.logerr + "dumpNumpy: unable to save data stream"
            logging.error(err)
        return err

```

8.1.3.17 dumpRegisters()

```
def nsCamera.CameraAssembler.CameraAssembler.dumpRegisters (
              self )
```

```
List contents of all registers in board.registers. WARNING: some status flags
  will reset when read.
DEPRECATED: use dumpStatus() instead

Returns:
    Sorted list: [register name (register address) : register contents as
      hexadecimal string without '0x']
```

Definition at line 1621 of file CameraAssembler.py.

```
    def dumpRegisters(self):
        """
        List contents of all registers in board.registers. WARNING: some status flags
          will reset when read.
        DEPRECATED: use dumpStatus() instead

        Returns:
            Sorted list: [register name (register address) : register contents as
              hexadecimal string without '0x']
        """
        dump = {}
        for key in self.board.registers.keys():
            err, rval = self.getRegister(key)
            dump[key] = rval
        reglistmax = int(max(self.board.registers.values()), 16)
        dumplist = [0] * (reglistmax + 1)
        for k, v in dump.items():
            regnum = self.board.registers[k]
            dumplist[int(regnum, 16)] = (
                "(" + regnum + ") {0:<24} {1}".format(k, v.upper())
            )
        reglist = [a for a in dumplist if a]
        return reglist
```

8.1.3.18 dumpStatus()

```
def nsCamera.CameraAssembler.CameraAssembler.dumpStatus (
              self )
```

Definition at line 374 of file CameraAssembler.py.

```
    def dumpStatus(self):
        return self.board.dumpStatus()
```

8.1.3.19 dumpSubregisters()

```
def nsCamera.CameraAssembler.CameraAssembler.dumpSubregisters (
              self )
```

```
List contents of all subregisters in board.channel_lookups and
  board.monitor_lookups.
WARNING: some registers will reset when read- only the first subregister from
  such a register will return the correct value, the remainder will return zeros

DEPRECATED: use dumpStatus() instead

Returns:
    dictionary  {subregister name : subregister contents as binary string
      without initial '0b'}
```

Definition at line 1645 of file CameraAssembler.py.

```
    def dumpSubregisters(self):
        """
        List contents of all subregisters in board.channel_lookups and
          board.monitor_lookups.
        WARNING: some registers will reset when read- only the first subregister from
          such a register will return the correct value, the remainder will return zeros

        DEPRECATED: use dumpStatus() instead

        Returns:
            dictionary  {subregister name : subregister contents as binary string
              without initial '0b'}
        """
        dump = {}
        for sub in self.board.subreglist:
            key = sub.name
            err, resp = self.getSubregister(key)
            if err:
                logging.warning(
                    self.logwarn + "dumpSubregisters: unable to read subregister " + key
                )
            val = hex(int(resp, 2))
            dump[key] = val
        return dump
```

8.1.3.20 enableLED()

```
def nsCamera.CameraAssembler.CameraAssembler.enableLED (
                self,
                status = 1 )
```

Definition at line 341 of file CameraAssembler.py.

```
    def enableLED(self, status=1):
        return self.board.enableLED(status)
```

8.1.3.21 flatten()

```
def nsCamera.CameraAssembler.CameraAssembler.flatten (
                self,
                x )
```

```
Flatten list of lists into single list
```

Definition at line 1719 of file CameraAssembler.py.

```
    def flatten(self, x):
        """
        Flatten list of lists into single list
        """
        if isinstance(x, collections.Iterable):
            return [a for i in x for a in self.flatten(i)]
        else:
            return [x]
```

8.1.3.22 generateFrames()

```
def nsCamera.CameraAssembler.CameraAssembler.generateFrames (
              self,
              data )
```

```
Processes data stream from board into frames and applies sensor-specific
  parsing. Generates padded data for fullsize option of setRows.

Args:
    data: stream from board.

Returns: list of parsed frames
```

Definition at line 1901 of file CameraAssembler.py.

```
1901     def generateFrames(self, data):
1902         """
1903         Processes data stream from board into frames and applies sensor-specific
1904           parsing. Generates padded data for fullsize option of setRows.
1905 
1906         Args:
1907             data: stream from board.
1908 
1909         Returns: list of parsed frames
1910         """
1911         allframes = self.str2nparray(data)
1912         # self.oldtime = self.currtime
1913         # self.currtime = time.time()
1914         # self.unstringed.append(self.currtime - self.oldtime)
1915         frames = [0] * self.sensor.nframes
1916         framesize = self.sensor.width * self.sensor.height
1917         if self.padToFull:
1918             toprows = self.sensor.firstrow
1919             botrows = (self.sensor.maxheight - 1) - self.sensor.lastrow
1920             for n in range(self.sensor.nframes):
1921                 padtop = np.zeros(toprows * self.sensor.maxwidth, dtype=int)
1922                 padbot = np.zeros(botrows * self.sensor.maxwidth, dtype=int)
1923                 thisframe = np.concatenate(
1924                     (padtop, allframes[n * framesize : (n + 1) * framesize], padbot)
1925                 )
1926                 frames[n] = thisframe
1927         else:
1928             for n in range(self.sensor.nframes):
1929                 frames[n] = allframes[n * framesize : (n + 1) * framesize]
1930         self.clearStatus()
1931         parsed = self.parseReadoff(frames)
1932         # self.oldtime = self.currtime
1933         # self.currtime = time.time()
1934         # self.parsedtime.append(self.currtime - self.oldtime)
1935         return parsed
1936 
```

8.1.3.23 getBoardInfo()

```
def nsCamera.CameraAssembler.CameraAssembler.getBoardInfo (
              self )
```

```
Get board info from FPGA_NUM register. Returns error flag if register contents
  are invalid and tuple (board version number, rad tolerance flag, sensor name)

Returns:
    tuple (errorFlag, (board version, rad tolerance flag, sensor name))
```

Definition at line 570 of file CameraAssembler.py.

```
570     def getBoardInfo(self):
571         """
572         Get board info from FPGA_NUM register. Returns error flag if register contents
573           are invalid and tuple (board version number, rad tolerance flag, sensor name)
574 
575         Returns:
576             tuple (errorFlag, (board version, rad tolerance flag, sensor name))
577         """
578         invalidFPGANum = False
579         interfaces = []
580 
581         if int(self.FPGANum[0], 16) & 8:
582             if self.FPGANum[1] == "1":
583                 boardtype = "LLNLv1"
584             elif self.FPGANum[1] == "4":
585                 boardtype = "LLNLv4"
586             else:
587                 boardtype = "LLNLv?"
588                 invalidFPGANum = True
589         else:
590             boardtype = "SNLrevC"
591             logging.warning(
592                 self.logwarn + "FPGA self-identifies as SNLrevC, which is not "
593                 "supported by this software "
594             )
595             invalidFPGANum = True
596         self.FPGAboardtype = boardtype
597 
598         if int(self.FPGANum[6], 16) & 1:
599             rad = True
600         else:
601             rad = False
602         self.FPGArad = rad
603 
604         if self.FPGANum[7] == "1":
605             sensor = "Icarus"
606         elif self.FPGANum[7] == "2":
607             sensor = "Daedalus"
608         elif self.FPGANum[7] == "3":
609             sensor = "Horus"
610         else:
611             sensor = "Undefined"
612             invalidFPGANum = True
613         self.FPGAsensor = sensor
614 
615         if int(self.FPGANum[5], 16) & 1:
616             interfaces.append("RS422")
617         if int(self.FPGANum[5], 16) & 2:
618             interfaces.append("GigE")
619         self.FPGAinterfaces = interfaces
620 
621         if invalidFPGANum:
622             if self.FPGANum == "80000001":
623                 invalidFPGANum = False
624             else:
625                 logging.warning(self.logwarn + "FPGA self-identification is invalid")
626         self.FPGAinvalid = invalidFPGANum
627 
628         return invalidFPGANum, (boardtype, rad, sensor)
629 
```

8.1.3.24 getEnter()

```
def nsCamera.CameraAssembler.CameraAssembler.getEnter (
                self,
                text )
```

```
Wait for enter key to be pressed.

Args:
    text: message asking for keypress
```

Definition at line 1728 of file CameraAssembler.py.

```
    def getEnter(self, text):
        """
        Wait for enter key to be pressed.

        Args:
            text: message asking for keypress
        """
        if self.PY3:
            input(text)
        else:
            raw_input(text)

```

8.1.3.25 getManualTiming()

```
def nsCamera.CameraAssembler.CameraAssembler.getManualTiming (
              self )
```

Definition at line 392 of file CameraAssembler.py.

```
    def getManualTiming(self):
        return self.sensor.getManualTiming()

```

8.1.3.26 getMonV()

```
def nsCamera.CameraAssembler.CameraAssembler.getMonV (
              self,
              monname,
              errflag = False )
```

```
Reads voltage from monitor named or that associated with the pot named 'monname'

Args:
    monname: name of pot or monitor, e.g., VRST or MON_CH2 found in
      board.subreg_aliases or defined in board.subregisters
    errflag: if True, return tuple with error string

Returns:
    if errflag:
tuple: (error string, float value of voltage measured by monitor)
    else:
float value of voltage measured by monitor
```

Definition at line 1122 of file CameraAssembler.py.

```
    def getMonV(self, monname, errflag=False):
        """
        Reads voltage from monitor named or that associated with the pot named 'monname'

        Args:
            monname: name of pot or monitor, e.g., VRST or MON_CH2 found in
              board.subreg_aliases or defined in board.subregisters
            errflag: if True, return tuple with error string

        Returns:
            if errflag:
                tuple: (error string, float value of voltage measured by monitor)
            else:
                float value of voltage measured by monitor
        """
        monname = monname.upper()
        if monname in self.board.subreg_aliases:
            monname = self.board.subreg_aliases[monname].upper()
        # else:
        for key, value in self.board.monitor_controls.items():
            if value == monname:
                monname = key
        if monname not in self.board.monitor_controls:
            if monname in self.board.subreglist:
                pass  # no change necessary
            else:
                err = (
                    self.logerr + "getMonV: invalid lookup " + monname + ", returning 0"
                )
                logging.error(err)
                if errflag:
                    return err, 0
                return 0
        err, monval = self.getPot(monname, errflag=True)
        if err:
            logging.error(
                self.logerr + "getMonV: unable to read monitor value for " + monname
            )
        if self.board.ADC5_bipolar:
            if monval >= 0.5:
                monval -= 1  # handle negative measurements (two's complement)
            if errflag:
                return err, 2 * self.board.ADC5_mult * monval * self.board.VREF
            return 2 * self.board.ADC5_mult * monval * self.board.VREF
        else:
            if errflag:
                return err, self.board.ADC5_mult * monval * self.board.VREF
            return self.board.ADC5_mult * monval * self.board.VREF

```

8.1.3.27 getPot()

```
def nsCamera.CameraAssembler.CameraAssembler.getPot (
              self,
              potname,
              errflag = False )
```

```
Retrieves value of pot or ADC monitor subregister, scaled to [0,1).

Args:
    potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
      board.subreg_aliases or defined in board.subregisters
    errflag: if True, return tuple with error string

Returns:
    if errflag:
tuple: (error string, float value of subregister, scaled to [0,1) )
    else:
float value of subregister, scaled to [0,1)
```

Definition at line 816 of file CameraAssembler.py.

```
    def getPot(self, potname, errflag=False):
        """
        Retrieves value of pot or ADC monitor subregister, scaled to [0,1).

        Args:
            potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
              board.subreg_aliases or defined in board.subregisters
            errflag: if True, return tuple with error string

        Returns:
            if errflag:
                tuple: (error string, float value of subregister, scaled to [0,1) )
            else:
                float value of subregister, scaled to [0,1)
        """
        potname, potobj, _ = self.resolveSubreg(potname)
        if not potobj:
            err = (
                self.logerr + "getPot: invalid lookup: " + potname + ’ , returning "0" ’
            )
            logging.error(err)
            if errflag:
                return err, "0"
            return "0"
        err, b_pot_value = self.getSubregister(potname)
        if err:
            logging.warning(
                self.logerr + "getPot: unable to read subregister " + potname
            )
        # convert binary string back to decimal
        f_reg_value = 1.0 * int(b_pot_value, 2)
        value = (f_reg_value - potobj.min) / (potobj.max - potobj.min)
        if errflag:
            return err, value
        return value

```

8.1.3.28 getPotV()

```
def nsCamera.CameraAssembler.CameraAssembler.getPotV (
                self,
                potname,
                errflag = False )
```

```
Reads voltage _setting_ (not actual voltage) of specified pot

Args:
    potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
      board.subreg_aliases or defined in board.subregisters
    errflag: if True, return tuple with error string

Returns:
    if errflag:
tuple: (error string, float value of pot voltage)
    else:
float value of pot voltage
```

Definition at line 921 of file CameraAssembler.py.

```
    def getPotV(self, potname, errflag=False):
        """
        Reads voltage _setting_ (not actual voltage) of specified pot

        Args:
            potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
              board.subreg_aliases or defined in board.subregisters
            errflag: if True, return tuple with error string

        Returns:
            if errflag:
                tuple: (error string, float value of pot voltage)
            else:
                float value of pot voltage
        """
        potname, potobj, _ = self.resolveSubreg(potname)
        if not potobj:
            err = (
                self.logerr
                + "getPotV: invalid lookup: "
                + potname
                + ’ , returning "0" ’
            )
            logging.error(err)
            if errflag:
                return err, "0"
            return "0"
        err, val = self.getPot(potname, errflag=True)
        if err:
            logging.error(self.logerr + "getPotV: unable to read pot " + potname)
        minV = potobj.minV
        maxV = potobj.maxV
        if errflag:
            return err, val * (maxV - minV)
        return val * (maxV - minV)

```

8.1.3.29 getPressure()

```
def nsCamera.CameraAssembler.CameraAssembler.getPressure (
                self,
                offset = None,
                sensitivity = None,
                units = None )
```

Definition at line 356 of file CameraAssembler.py.

```
    def getPressure(self, offset=None, sensitivity=None, units=None):
        return self.board.getPressure(offset, sensitivity, units)

```

8.1.3.30 getRegister()

```
def nsCamera.CameraAssembler.CameraAssembler.getRegister (
                self,
                regname )
```

```
Retrieves contents of named register as hexadecimal string without ’0x’

Args:
    regname: name of register as given in ICD

Returns:
    tuple: (error string, register contents as hexadecimal string without ’0x’)
```

Definition at line 630 of file CameraAssembler.py.

```
    def getRegister(self, regname):
        """
        Retrieves contents of named register as hexadecimal string without ’0x’

        Args:
            regname: name of register as given in ICD

        Returns:
            tuple: (error string, register contents as hexadecimal string without '0x')
        """
        regname = regname.upper()
        if regname not in self.board.registers:
            err = (
                self.logerr + "invalid register name: " + regname + " ; returning zeros"
            )
            logging.error(err)
            return err, "00000000"
        sendpkt = Packet(cmd="1", addr=self.board.registers[regname])
        err, rval = self.comms.sendCMD(sendpkt)
        if err:
            logging.error(self.logerr + "getRegister " + regname + " " + err)
        return err, rval[8:16]

```

8.1.3.31 getSubregister()

```
def nsCamera.CameraAssembler.CameraAssembler.getSubregister (
                self,
                subregname )
```

```
Returns substring of register identified in board attribute 'subregname'

Args:
    subregname: listed in board.subreg_aliases or defined in board.subregisters

Returns:
    tuple: (error string, contents of subregister as binary string without '0b')
```

Definition at line 697 of file CameraAssembler.py.

```
    def getSubregister(self, subregname):
        """
        Returns substring of register identified in board attribute 'subregname'

        Args:
            subregname: listed in board.subreg_aliases or defined in board.subregisters

        Returns:
            tuple: (error string, contents of subregister as binary string without '0b')
        """
        subregname, subregobj, _ = self.resolveSubreg(subregname)
        if not subregobj:
            err = (
                self.logerr
                + "getSubregister: invalid lookup: "
                + subregname
                + ' , returning "0" string '
            )
            logging.error(err)
            return err, "".zfill(8)
        err, resp = self.getRegister(subregobj.register)
        if err:
            logging.error(
                self.logerr
                + "getSubregister: unable to retrieve register setting: "
                + subregname
                + ' , returning "0" string'
            )
            return err, "".zfill(8)
        hex_str = "0x" + resp  # this should be a hexadecimalstring
        b_reg_value = "{0:0=32b}".format(int(hex_str, 16))  # convert to binary string
        # list indexing is reversed from bit string; the last bit of the string is at
        #   index 0 in the list (thus bit 0 is at index 0)
        startindex = 31 - subregobj.start_bit
        return "", b_reg_value[startindex : startindex + subregobj.width]

```

8.1.3.32 getTemp()

```
def nsCamera.CameraAssembler.CameraAssembler.getTemp (
              self,
              scale = None )
```

Definition at line 353 of file CameraAssembler.py.

```
353     def getTemp(self, scale=None):
354         return self.board.getTemp(scale)
355
```

8.1.3.33 getTimer()

```
def nsCamera.CameraAssembler.CameraAssembler.getTimer (
              self )
```

Definition at line 335 of file CameraAssembler.py.

```
335     def getTimer(self):
336         return self.board.getTimer()
337
```

8.1.3.34 getTiming()

```
def nsCamera.CameraAssembler.CameraAssembler.getTiming (
              self,
              side = None,
              actual = None )
```

Definition at line 386 of file CameraAssembler.py.

```
386     def getTiming(self, side=None, actual=None):
387         return self.sensor.getTiming(side, actual)
388
```

8.1.3.35 initBoard()

```
def nsCamera.CameraAssembler.CameraAssembler.initBoard (
              self )
```

Aliases to other objects' methods.

Definition at line 308 of file CameraAssembler.py.

```
308     def initBoard(self):
309         return self.board.initBoard()
310
```

8.1.3.36 initialize()

```
def nsCamera.CameraAssembler.CameraAssembler.initialize (
                self )
```

End aliases.

```
Initialize board registers and set pots
```

Definition at line 433 of file CameraAssembler.py.

```
433     def initialize(self):
434         """
435         Initialize board registers and set pots
436         """
437 
438 
440 
441         # get sensor
442         if self.sensorname == "icarus":
443             import nsCamera.sensors.icarus as snsr
444         elif self.sensorname == "icarus2":
445             import nsCamera.sensors.icarus2 as snsr
446         elif self.sensorname == "daedalus":
447             import nsCamera.sensors.daedalus as snsr
448         else:  # catch-all for added sensors to attempt object encapsulation
449             sensormodname = ".sensors." + self.sensorname
450             try:
451                 sensormod = importlib.import_module(sensormodname, "nsCamera")
452             except ImportError:
453                 logging.critical(self.logcrit + "invalid sensor name")
454                 sys.exit(1)
455             snsr = getattr(sensormod, self.sensorname)
456         self.sensor = snsr(self)
457 
458         # kill existing connections (for reinitialize)
459         if hasattr(self, "comms"):
460             self.closeDevice()
461 
462         # get communications interface
463         if self.commname == "rs422":
464             import nsCamera.comms.RS422 as comms
465         elif self.commname == "gige":
466             import nsCamera.comms.GigE as comms
467         else:
468             commsmodname = ".comms." + self.commname
469             try:
470                 commsmod = importlib.import_module(commsmodname, "nsCamera")
471             except ImportError:
472                 logging.critical(self.logcrit + "invalid comms name")
473                 sys.exit(1)
474             comms = getattr(commsmod, self.commname)
475         self.comms = comms(self)
476 
477         # get board
478         if self.boardname == "llnl_v1":
479             import nsCamera.boards.LLNL_v1 as brd
480 
481             self.board = brd.llnl_v1(self)
482         elif self.boardname == "llnl_v4":
483             import nsCamera.boards.LLNL_v4 as brd
484 
485             self.board = brd.llnl_v4(self)
486         else:
487             boardmodname = ".board." + self.boardname
488             try:
489                 boardmod = importlib.import_module(boardmodname, "nsCamera")
490             except ImportError:
491                 logging.critical(self.logcrit + "invalid board name")
492                 sys.exit(1)
493             boardobj = getattr(boardmod, self.boardname)
494             self.board = boardobj(self)
495 
496 
497         # ###############
```

```
        # # For cython version
        #
        # # get sensor
        # if self.sensorname == "icarus":
        #     import nsCamera.sensors.icarus as snsr
        #     self.sensor = snsr.icarus(self)
        # elif self.sensorname == "icarus2":
        #     import nsCamera.sensors.icarus2 as snsr
        #     self.sensor = snsr.icarus2(self)
        # elif self.sensorname == "daedalus":
        #     import nsCamera.sensors.daedalus as snsr
        #     self.sensor = snsr.daedalus(self)
        #
        # # kill existing connections (for reinitialize)
        # if hasattr(self, "comms"):
        #     self.closeDevice()
        #
        # # get communications interface
        # if self.commname == "rs422":
        #     import nsCamera.comms.RS422 as comms
        #     self.comms = comms.RS422(self)
        # elif self.commname == "gige":
        #     import nsCamera.comms.GigE as comms
        #     self.comms = comms.GigE(self)
        #
        # # get board
        # if self.boardname == "llnl_v1":
        #     import nsCamera.boards.LLNL_v1 as brd
        #     self.board = brd.llnl_v1(self)
        # elif self.boardname == "llnl_v4":
        #     import nsCamera.boards.LLNL_v4 as brd
        #     self.board = brd.llnl_v4(self)
        # ###############

        err, rval = self.getRegister("FPGA_NUM")
        if err or rval == "":
            err, rval = self.getRegister("FPGA_NUM")
            if err or rval == "":
                logging.critical(
                    self.logcrit + "Initialization failed: unable to communicate with "
                    "board. "
                )
            sys.exit(1)

        self.initBoard()
        self.initPots()
        self.initSensor()
        self.initPowerCheck()
        self.getBoardInfo()
        self.printBoardInfo()

```

8.1.3.37 initPots()

```
def nsCamera.CameraAssembler.CameraAssembler.initPots (
                self )
```

Definition at line 311 of file CameraAssembler.py.

```
    def initPots(self):
        return self.board.initPots()

```

8.1.3.38 initPowerCheck()

```
def nsCamera.CameraAssembler.CameraAssembler.initPowerCheck (
              self )
```

```
Reset software and board timers for monitoring power status
```

Definition at line 1523 of file CameraAssembler.py.

```
1523     def initPowerCheck(self):
1524         """
1525         Reset software and board timers for monitoring power status
1526         """
1527         self.inittime = time.time()
1528         logging.info(self.loginfo + "resetting timer for power check function")
1529         self.resetTimer()
1530
```

8.1.3.39 initSensor()

```
def nsCamera.CameraAssembler.CameraAssembler.initSensor (
              self )
```

Definition at line 317 of file CameraAssembler.py.

```
317     def initSensor(self):
318         return self.board.initSensor()
319
```

8.1.3.40 latchPots()

```
def nsCamera.CameraAssembler.CameraAssembler.latchPots (
              self )
```

Definition at line 314 of file CameraAssembler.py.

```
314     def latchPots(self):
315         return self.board.latchPots()
316
```

8.1.3.41 loadTextFrames()

```
def nsCamera.CameraAssembler.CameraAssembler.loadTextFrames (
              self,
              filename = 'frames.txt',
              path = None )
```

```
Load a image set previously saved as text and convert to frames. NOTE: to work
  properly, the cameraAssembler object must have the same geometry and sensor
  tyoe that was used to create the text file

Args:
    filename: name of textfile to load
    path: path to file, if not the current working directory

       Returns: list of parsed frames
```

Definition at line 2019 of file CameraAssembler.py.

```
    def loadTextFrames(self, filename='frames.txt', path=None):
        """
        Load a image set previously saved as text and convert to frames. NOTE: to work
          properly, the cameraAssembler object must have the same geometry and sensor
          tyoe that was used to create the text file

        Args:
            filename: name of textfile to load
            path: path to file, if not the current working directory

       Returns: list of parsed frames
        """
        if path is None:
            path = os.path.join(os.getcwd())
        textfile = os.path.join(path, filename)

        try:
            f = open(textfile, "r")
            s = f.read()
            frames = self.generateFrames(s)
            return frames
        except OSError as err:
            print("OS error: {0}".format(err))
        except ValueError:
            print("Could not convert data to an integer.")
        except:
            print("Unexpected error:", sys.exc_info()[0])


"""
Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
LLNL-CODE-838080

This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
(DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
See license for disclaimers, notice of U.S. Government Rights and license terms and
conditions.
"""
```

8.1.3.42 mmReadoff()

```
def nsCamera.CameraAssembler.CameraAssembler.mmReadoff (
              self,
              waitOnSRAM,
              variation = None )
```

```
Convenience function for parsing frames for use by MicroManager plugin
Args:
    waitOnSRAM: readoff wait flag
    variation: format of frames generated from readoff
default - return first frame only
"LastFrame" - return last frame only
"Average" - provide average of frames as single frame
"Landscape" - stitch frames together horizontally into single wide frame

Returns:
    ndarray - single image frame
```

Definition at line 1740 of file CameraAssembler.py.

```
    def mmReadoff(self, waitOnSRAM, variation=None):
        """
        Convenience function for parsing frames for use by MicroManager plugin
        Args:
            waitOnSRAM: readoff wait flag
            variation: format of frames generated from readoff
                default - return first frame only
                "LastFrame" - return last frame only
                "Average" - provide average of frames as single frame
                "Landscape" - stitch frames together horizontally into single wide frame

        Returns:
            ndarray - single image frame
        """
        frames, datalen, data_err = self.readoff(waitOnSRAM)
        if variation == "LastFrame":
            return frames[self.sensor.nframes - 1]
        elif variation == "Average":
            return np.sum(frames, axis=0) // self.sensor.nframes
        elif variation == "Landscape":
            shaped = [np.reshape(frame, (1024, 512)) for frame in frames]
            return np.concatenate(shaped, axis=1)
        else:
            return frames[0]

```

8.1.3.43 parseReadoff()

```
def nsCamera.CameraAssembler.CameraAssembler.parseReadoff (
              self,
              frames )
```

Definition at line 410 of file CameraAssembler.py.

```
    def parseReadoff(self, frames):
        return self.sensor.parseReadoff(frames)

```

8.1.3.44 plotFrames()

```
def nsCamera.CameraAssembler.CameraAssembler.plotFrames (
              self,
              frames,
              index = None )
```

```
Plot frame or list of frames as individual graphs.

Args:
    frames: numpy array or list of numpy arrays
    index: number to start frame numbering

Returns:
    Error string
```

Definition at line 1423 of file CameraAssembler.py.

```
    def plotFrames(self, frames, index=None):
        """
        Plot frame or list of frames as individual graphs.

        Args:
            frames: numpy array or list of numpy arrays
            index: number to start frame numbering

        Returns:
            Error string
        """
        logging.info(self.loginfo + "plotFrames")
        err = ""
        if index is None:
            nframe = self.sensor.firstframe
        else:
            nframe = index

        if type(frames) is not list:
            frames = [frames]

        # if this is a text string from fast readoff, do the numpy conversion now
        if isinstance(frames[0], str):
            frames = self.generateFrames(frames)

        framestemp = np.copy(frames)
        for frame in framestemp:
            try:
                if self.padToFull:
                    frame.shape = (
                        self.sensor.maxheight // (self.sensor.interlacing + 1),
                        self.sensor.maxwidth,
                    )
                else:
                    frame.shape = (
                        self.sensor.height // (self.sensor.interlacing + 1),
                        self.sensor.width,
                    )
            except:
                err = self.logerr + "plotFrames: unable to plot frame"
                logging.error(err)
                continue
            plt.imshow(frame, cmap="gray")
            name = "Frame %d" % nframe
            plt.title(name)
            plt.show()
            nframe += 1
        return err
```

8.1.3.45 powerCheck()

```
def nsCamera.CameraAssembler.CameraAssembler.powerCheck (
              self,
              delta = 10 )
```

```
Check to see if board power has persisted since powerCheck was last initialized.
  Compares time elapsed since initialization against board’s timer. If
  difference is greater than ’delta,’ flag as False (power has likely failed)

Args:
    delta: difference in seconds permitted between software and board timers

Returns:
    boolean, ’True’ means timer difference is less than ’delta’ parameter;
     ’False’ indicates power failure
```

Definition at line 1531 of file CameraAssembler.py.

```
1531     def powerCheck(self, delta=10):
1532         """
1533         Check to see if board power has persisted since powerCheck was last initialized.
1534           Compares time elapsed since initialization against board’s timer. If
1535           difference is greater than ’delta,’ flag as False (power has likely failed)
1536 
1537         Args:
1538             delta: difference in seconds permitted between software and board timers
1539 
1540         Returns:
1541             boolean, ’True’ means timer difference is less than ’delta’ parameter;
1542                      ’False’ indicates power failure
1543         """
1544         elapsed = time.time() - self.inittime
1545         difference = abs(elapsed - self.getTimer())
1546         if difference > delta:
1547             logging.warning(
1548                 self.logwarn + "powerCheck function has failed; may indicate current "
1549                 "or recent power failure "
1550             )
1551         return difference < delta
1552 
```

8.1.3.46 printBoardInfo()

```
def nsCamera.CameraAssembler.CameraAssembler.printBoardInfo (
                self )
```

Definition at line 1590 of file CameraAssembler.py.

```
1590     def printBoardInfo(self):
1591         logging.info(
1592             self.loginfo
1593             + "Python version: "
1594             + str(self.python)
1595             + "."
1596             + str(self.pyth1)
1597             + "."
1598             + str(self.pyth2)
1599         )
1600         logging.info(self.loginfo + "nsCamera software version: " + self.version)
1601         logging.info(self.loginfo + "FPGA firmware version: " + self.FPGAVersion)
1602         logging.info(self.loginfo + "FPGA implementation: " + self.FPGANum)
1603         if self.FPGAinvalid:
1604             logging.info(self.loginfo + "FPGA information unavailable")
1605         else:
1606             logging.info(self.loginfo + "Board type: " + self.FPGAboardtype)
1607             logging.info(self.loginfo + "Rad-Tolerant: " + str(self.FPGArad))
1608             logging.info(self.loginfo + "Sensor family: " + self.FPGAsensor)
1609             logging.info(
1610                 self.loginfo + "Available interfaces: " + ", ".join(self.FPGAinterfaces)
1611             )
1612         if self.commname == "gige":
1613             ci = self.comms.CardInfoP.contents
1614             ip = ".".join(str(e) for e in [b for b in ci.IPAddr])
1615             logging.info(
1616                 self.loginfo + "GigE connected to " + ip + ":" + str(self.port)
```

```
            )
        elif self.commname == "rs422":
            logging.info(self.loginfo + "RS422 connected to " + self.comms._port)

```

8.1.3.47 readImgs()

```
def nsCamera.CameraAssembler.CameraAssembler.readImgs (
                self,
                waitOnSRAM = True,
                mode = "Hardware" )
```

```
Combines arm() and readoff() functions

Returns:
    tuple (list of numpy arrays, length of downloaded payload, payload error
      flag) returned by readoff
```

Definition at line 1171 of file CameraAssembler.py.

```
    def readImgs(self, waitOnSRAM=True, mode="Hardware"):
        """
        Combines arm() and readoff() functions

        Returns:
            tuple (list of numpy arrays, length of downloaded payload, payload error
              flag) returned by readoff
        """
        logging.info(self.loginfo + "readImgs")
        self.arm(mode)
        return self.readoff(waitOnSRAM)

```

8.1.3.48 readoff()

```
def nsCamera.CameraAssembler.CameraAssembler.readoff (
                self,
                waitOnSRAM = None,
                timeout = 0,
                fast = None )
```

Definition at line 419 of file CameraAssembler.py.

```
    def readoff(self, waitOnSRAM=None, timeout=0, fast=None):
        return self.comms.readoff(waitOnSRAM, timeout, fast)

```

8.1.3.49 readSerial()

```
def nsCamera.CameraAssembler.CameraAssembler.readSerial (
              self,
              size,
              timeout = None )
```

Definition at line 425 of file CameraAssembler.py.

```
425     def readSerial(self, size, timeout=None):
426         return self.comms.readSerial(size, timeout)
427
```

8.1.3.50 readSRAM()

```
def nsCamera.CameraAssembler.CameraAssembler.readSRAM (
              self )
```

Definition at line 329 of file CameraAssembler.py.

```
329     def readSRAM(self):
330         return self.board.readSRAM()
331
```

8.1.3.51 reboot()

```
def nsCamera.CameraAssembler.CameraAssembler.reboot (
              self )
```

```
Perform soft reboot on board and reinitialize
```

Definition at line 563 of file CameraAssembler.py.

```
563     def reboot(self):
564         """
565         Perform soft reboot on board and reinitialize
566         """
567         self.board.softReboot()
568         self.reinitialize()
569
```

8.1.3.52 reinitialize()

```
def nsCamera.CameraAssembler.CameraAssembler.reinitialize (
              self )
```

```
Reinitialize board registers and pots, reinitialize sensor timing (if
  previously set)
```

Definition at line 549 of file CameraAssembler.py.

```
    def reinitialize(self):
        """
        Reinitialize board registers and pots, reinitialize sensor timing (if
          previously set)
        """
        logging.info(self.loginfo + "reinitializing")
        self.initialize()

        for side in self.senstiming:
            self.setTiming(side, self.senstiming[side][0], self.senstiming[side][1])

        if self.sensmanual:  # should be mutually exclusive with anything in senstiming
            self.setManualShutters(self.sensmanual)

```

8.1.3.53 reportEdgeDetects()

```
def nsCamera.CameraAssembler.CameraAssembler.reportEdgeDetects (
              self )
```

Definition at line 371 of file CameraAssembler.py.

```
    def reportEdgeDetects(self):
        return self.board.reportEdgeDetects()

```

8.1.3.54 reportStatus()

```
def nsCamera.CameraAssembler.CameraAssembler.reportStatus (
              self )
```

Definition at line 368 of file CameraAssembler.py.

```
    def reportStatus(self):
        return self.board.reportStatus()

```

8.1.3.55 resetTimer()

```
def nsCamera.CameraAssembler.CameraAssembler.resetTimer (
              self )
```

Definition at line 338 of file CameraAssembler.py.

```
338     def resetTimer(self):
339         return self.board.resetTimer()
340
```

8.1.3.56 resolveSubreg()

```
def nsCamera.CameraAssembler.CameraAssembler.resolveSubreg (
              self,
              srname )
```

```
Resolves subregister name or alias, returns object associated with subregister
  and flag indicating writability

Args:
    srname: name or alias of subregister

Returns:
    tuple(subregister name string, associated object, writable flag)
```

Definition at line 675 of file CameraAssembler.py.

```
675     def resolveSubreg(self, srname):
676         """
677         Resolves subregister name or alias, returns object associated with subregister
678           and flag indicating writability
679
680         Args:
681             srname: name or alias of subregister
682
683         Returns:
684             tuple(subregister name string, associated object, writable flag)
685         """
686         writable = False
687         srname = srname.upper()
688         if srname in self.board.subreg_aliases:
689             srname = self.board.subreg_aliases[srname].upper()
690         if srname in self.board.subreglist:
691             srobj = getattr(self.board, srname)
692             writable = getattr(self.board, srname).writable
693         else:
694             srobj = None
695         return srname, srobj, writable
696
```

8.1.3.57 saveFrames()

```
def nsCamera.CameraAssembler.CameraAssembler.saveFrames (
              self,
              frames,
              path = None,
              filename = "frames",
              prefix = None )
```

```
Save list of numpy arrays to disk. If passed an unprocessed text string, convert
  to numpy before saving. Use ’prefix=""’ for no prefix

Args:
    frames: numpy array or list of numpy arrays OR text string
    path: save path, defaults to ’./output’
    filename: defaults to ’frames.bin’
    prefix: prepended to filename, defaults to time/date (e.g. ’160830-124704_’)

Returns:
    Error string
```

Definition at line 1220 of file CameraAssembler.py.

```
1220     def saveFrames(
1221         self, frames, path=None, filename="frames", prefix=None,
1222     ):
1223         """
1224         Save list of numpy arrays to disk. If passed an unprocessed text string, convert
1225           to numpy before saving. Use ’prefix=""’ for no prefix
1226
1227         Args:
1228             frames: numpy array or list of numpy arrays OR text string
1229             path: save path, defaults to ’./output’
1230             filename: defaults to ’frames.bin’
1231             prefix: prepended to filename, defaults to time/date (e.g. ’160830-124704_’)
1232
1233         Returns:
1234             Error string
1235         """
1236         logging.info(self.loginfo + "saveFrames")
1237         err = ""
1238         if path is None:
1239             path = os.path.join(os.getcwd(), "output")
1240         if prefix is None:
1241             prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
1242         if not os.path.exists(path):
1243             os.makedirs(path)
1244
1245         if isinstance(frames[0], str):
1246             filename = filename + ".txt"
1247             savefile = open(os.path.join(path, prefix + filename), "w+")
1248             savefile.write(frames)
1249
1250         else:
1251             filename = filename + ".bin"
1252             stacked = np.stack(frames)
1253             try:
1254                 stacked = stacked.reshape(
1255                     (
1256                         self.sensor.nframes,
1257                         self.sensor.height // (self.sensor.interlacing + 1),
1258                         self.sensor.width,
1259                     )
1260                 )
1261             except Exception as e:
1262                 err = self.logerr + "saveFrames: unable to save frames: " + str(e)
1263                 logging.error(err)
1264
1265             stacked.tofile(os.path.join(path, prefix + filename))
1266         return err
1267
```

8.1.3.58 saveNumpys()

```
def nsCamera.CameraAssembler.CameraAssembler.saveNumpys (
              self,
              frames,
              path = None,
              filename = "Frame",
              prefix = None,
              index = None )
```

```
Save numpy array or list of numpy arrays to disk as individual numpy data files,
  with frame number appended to filename.

Args:
    frames: numpy array or list of numpy arrays or single numpy array
    path: save path, defaults to './output'
    filename: defaults to 'Frame' followed by frame number
    prefix: prepended to 'filename', defaults to time/date
      (e.g. '160830-124704_')
    index: number to start frame numbering

Returns:
    Error string
```

Definition at line 1329 of file CameraAssembler.py.

```
def saveNumpys(
    self, frames, path=None, filename="Frame", prefix=None, index=None,
):
    """
    Save numpy array or list of numpy arrays to disk as individual numpy data files,
      with frame number appended to filename.

    Args:
        frames: numpy array or list of numpy arrays or single numpy array
        path: save path, defaults to './output'
        filename: defaults to 'Frame' followed by frame number
        prefix: prepended to 'filename', defaults to time/date
          (e.g. '160830-124704_')
        index: number to start frame numbering

    Returns:
        Error string
    """
    logging.info(self.loginfo + "saveNumpys")
    err = ""
    if path is None:
        path = os.path.join(os.getcwd(), "output")
    if prefix is None:
        prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
    if not os.path.exists(path):
        os.makedirs(path)
    if index is None:
        nframe = self.sensor.firstframe
    else:
        nframe = index
    if type(frames) is not list:
        frames = [frames]

    # if this is a text string from fast readoff, do the numpy conversion now
    if isinstance(frames[0], str):
        frames = self.generateFrames(frames)

    framestemp = np.copy(frames)
    for frame in framestemp:
        try:
            if self.padToFull:
                frame.shape = (
                    self.sensor.maxheight // (self.sensor.interlacing + 1),
                    self.sensor.maxwidth,
                )
            else:
                frame.shape = (
                    self.sensor.height // (self.sensor.interlacing + 1),
                    self.sensor.width,
                )
            namenum = filename + "_%d" % nframe
            nppath = os.path.join(path, prefix + namenum + ".npy")
            np.save(nppath, frame)
            nframe += 1
        except:
            err = self.logerr + "saveNumpys: unable to save arrays"
            logging.error(err)
            continue
    return err

```

8.1.3.59 saveTiffs()

```
def nsCamera.CameraAssembler.CameraAssembler.saveTiffs (
              self,
              frames,
              path = None,
              filename = "Frame",
              prefix = None,
              index = None )
```

```
Save numpy array or list of numpy arrays or single array to disk as individual
  tiffs, with frame number appended to filename.

Args:
    frames: numpy array or list of numpy arrays
    path: save path, defaults to './output'
    filename: defaults to 'Frame' followed by frame number
    prefix: prepended to 'filename', defaults to time/date
      (e.g. '160830-124704_')
    index: number to start frame numbering

Returns:
    Error string
```

Definition at line 1268 of file CameraAssembler.py.

```
1268     def saveTiffs(
1269         self, frames, path=None, filename="Frame", prefix=None, index=None,
1270     ):
1271         """
1272         Save numpy array or list of numpy arrays or single array to disk as individual
1273           tiffs, with frame number appended to filename.
1274 
1275         Args:
1276             frames: numpy array or list of numpy arrays
1277             path: save path, defaults to './output'
1278             filename: defaults to 'Frame' followed by frame number
1279             prefix: prepended to 'filename', defaults to time/date
1280               (e.g. '160830-124704_')
1281             index: number to start frame numbering
1282 
1283         Returns:
1284             Error string
1285         """
1286         logging.info(self.loginfo + "saveTiffs")
1287         err = ""
1288         if path is None:
1289             path = os.path.join(os.getcwd(), "output")
1290         if prefix is None:
```

```
            prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
        if not os.path.exists(path):
            os.makedirs(path)
        if index is None:
            nframe = self.sensor.firstframe
        else:
            nframe = index

        if type(frames) is not list:
            frames = [frames]
        # if this is a text string from fast readoff, do the numpy conversion now
        if isinstance(frames[0], str):
            frames = self.generateFrames(frames)

        framestemp = np.copy(frames)
        for frame in framestemp:
            try:
                if self.padToFull:
                    frame.shape = (
                        self.sensor.maxheight // (self.sensor.interlacing + 1),
                        self.sensor.maxwidth,
                    )
                else:
                    frame.shape = (
                        self.sensor.height // (self.sensor.interlacing + 1),
                        self.sensor.width,
                    )
                frameimg = Image.fromarray(frame)
                namenum = filename + "_%d" % nframe
                tifpath = os.path.join(path, prefix + namenum + ".tif")
                frameimg.save(tifpath)
                nframe += 1
            except:
                err = self.logerr + "saveTiffs: unable to save images"
                logging.error(err)
                continue
        return err

```

8.1.3.60 sendCMD()

```
def nsCamera.CameraAssembler.CameraAssembler.sendCMD (
                self,
                pkt )
```

Definition at line 413 of file CameraAssembler.py.

```
413     def sendCMD(self, pkt):
414         return self.comms.sendCMD(pkt)
415
```

8.1.3.61 sensorSpecific()

```
def nsCamera.CameraAssembler.CameraAssembler.sensorSpecific (
                self )
```

Definition at line 395 of file CameraAssembler.py.

```
395     def sensorSpecific(self):
396         return self.sensor.sensorSpecific()
397
```

8.1.3.62 setArbTiming()

```
def nsCamera.CameraAssembler.CameraAssembler.setArbTiming (
              self,
              side = None,
              sequence = None )
```

Definition at line 383 of file CameraAssembler.py.

```
    def setArbTiming(self, side=None, sequence=None):
        return self.sensor.setArbTiming(side, sequence)

```

8.1.3.63 setFrames()

```
def nsCamera.CameraAssembler.CameraAssembler.setFrames (
              self,
              minframe = None,
              maxframe = None )
```

```
Sets bounds on frames returned by board, inclusive (e.g., 0,3 returns four
frames). If called without parameters, resets to full set of frames.

Args:
    minframe: first frame to read from board
    maxframe: last frame to read from board

Returns:
    Error string
```

Definition at line 1765 of file CameraAssembler.py.

```
    def setFrames(self, minframe=None, maxframe=None):
        """
        Sets bounds on frames returned by board, inclusive (e.g., 0,3 returns four
        frames). If called without parameters, resets to full set of frames.

        Args:
            minframe: first frame to read from board
            maxframe: last frame to read from board

        Returns:
            Error string
        """
        if minframe is None:
            minframe = self.sensor.minframe
        if maxframe is None:
            maxframe = self.sensor.maxframe
        if (
            not isinstance(minframe, int)
            or minframe < self.sensor.minframe
            or minframe > maxframe
            or not isinstance(maxframe, int)
            or maxframe > self.sensor.maxframe
        ):
            err = (
                self.logerr + "setFrames: invalid frame limits submitted. Frame "
                "selection remains unchanged. "
            )
            logging.error(err)
            return err

        initframe = hex(minframe)[2:].zfill(8)
        finframe = hex(maxframe)[2:].zfill(8)
        err1, _ = self.setRegister("FPA_FRAME_INITIAL", initframe)
        err2, _ = self.setRegister("FPA_FRAME_FINAL", finframe)
        self.sensor.firstframe = minframe
        self.sensor.lastframe = maxframe
        self.sensor.nframes = maxframe - minframe + 1
        self.comms.payloadsize = (
            self.sensor.width
            * self.sensor.height
            * self.sensor.nframes
            * self.sensor.bytesperpixel
        )
        plural = ""
        if self.sensor.nframes > 1:
            plural = "s"
        logging.info(
            self.loginfo
            + "Readoff set to "
            + str(self.sensor.nframes)
            + " frame"
            + plural
            + " ("
            + str(minframe)
            + ", "
            + str(maxframe)
            + ")"
        )
        err = err1 + err2
        if err:
            logging.error(
                self.logerr + "setFrames may not have functioned properly: " + err
            )
        return err

```

8.1.3.64 setHighFullWell()

```
def nsCamera.CameraAssembler.CameraAssembler.setHighFullWell (
                self,
                flag = True )
```

Definition at line 401 of file CameraAssembler.py.

```
    def setHighFullWell(self, flag=True):
        return self.sensor.setHighFullWell(flag)

```

8.1.3.65 setInterlacing()

```
def nsCamera.CameraAssembler.CameraAssembler.setInterlacing (
                self,
                ifactor = None )
```

Definition at line 398 of file CameraAssembler.py.

```
    def setInterlacing(self, ifactor=None):
        return self.sensor.setInterlacing(ifactor)

```

8.1.3.66 setLED()

```
def nsCamera.CameraAssembler.CameraAssembler.setLED (
              self,
              LED = 1,
              status = 1 )
```

Definition at line 344 of file CameraAssembler.py.

```
344     def setLED(self, LED=1, status=1):
345         return self.board.setLED(LED, status)
346
```

8.1.3.67 setManualShutters()

```
def nsCamera.CameraAssembler.CameraAssembler.setManualShutters (
              self,
              timing = None )
```

Definition at line 389 of file CameraAssembler.py.

```
389     def setManualShutters(self, timing=None):
390         return self.sensor.setManualShutters(timing)
391
```

8.1.3.68 setPot()

```
def nsCamera.CameraAssembler.CameraAssembler.setPot (
              self,
              potname,
              value = 1.0,
              errflag = False )
```

```
Sets value of pot to value, normalized so that  ’1.0’ corresponds with the
  fixed point maximum value of pot.

Args:
    potname: common name of pot, e.g., VRST found in board.subreg_aliases or
      defined in board.subregisters
    value: float between 0 and 1
    errflag: if True, return tuple with error string

Returns:
    if errflag:
tuple: (error string, response packet as string)
    else:
response packet as string
```

Definition at line 852 of file CameraAssembler.py.

```
    def setPot(self, potname, value=1.0, errflag=False):
        """
        Sets value of pot to value, normalized so that  ’1.0’ corresponds with the
          fixed point maximum value of pot.

        Args:
            potname: common name of pot, e.g., VRST found in board.subreg_aliases or
              defined in board.subregisters
            value: float between 0 and 1
            errflag: if True, return tuple with error string

        Returns:
            if errflag:
                tuple: (error string, response packet as string)
            else:
                response packet as string
        """
        if value < 0:
            value = 0.0
        if value > 1:
            value = 1.0

        potname, potobj, writable = self.resolveSubreg(potname)
        if not potobj:
            err = (
                self.logerr + "setPot: invalid lookup: " + potname + ’ , returning "0" ’
            )
            logging.error(err)
            if errflag:
                return err, "0"
            return "0"
        if not writable:
            err = self.logerr + "setPot: not a writable subregister: " + potname
            logging.error(err)
            if errflag:
                return err, "0"
            return 0
        setpoint = int(round(value * potobj.max_value))
        setpointpadded = "{num:{fill}{width}b}".format(
            num=setpoint, fill="0", width=potobj.width
        )
        err, rval = self.setSubregister(potname, setpointpadded)
        if err:
            logging.error(
                self.logerr
                + "setPot: unable to confirm setting of subregister: "
                + potname
            )
        ident = potname[3:]
        if ident[0].isdigit():  # numbered pot scheme
            potnumlatch = int(ident) * 2 + 1
            potnumlatchstring = "{num:{fill}{width}x}".format(
                num=potnumlatch, fill="0", width=8
            )
            err1, resp = self.setRegister("POT_CTL", potnumlatchstring)
        else:  # alphabetical DAC scheme
            ident = ident.upper()  # expects single character, e.g. ’A’ from ’DACA’
            identnum = ord(ident) - ord("A")  # DACA -> 0
            potnumlatch = int(identnum) * 2 + 1
            potnumlatchstring = "{num:{fill}{width}x}".format(
                num=potnumlatch, fill="0", width=8
            )
            err1, resp = self.setRegister("DAC_CTL", potnumlatchstring)
        if err1:
            logging.error(self.logerr + "setPot: unable to latch register")
        if errflag:
            return err + err1, rval
        return rval

```

8.1.3.69 setPotV()

```
def nsCamera.CameraAssembler.CameraAssembler.setPotV (
                self,
```

potname,
voltage,
tune = False,
accuracy = 0.01,
iterations = 20,
approach = 0.75,
errflag = False)

```
Sets pot to specified voltage. If tune=True, uses monitor to adjust pot to
  correct voltage. Tuning will attempt to tune to closest LSB on pot; if
  'accuracy' > LSB resolution, will only complain if tuning is unable to get
  the voltage within 'accuracy'

Args:
    potname: common name of pot, e.g., VRST found in board.subreg_aliases or
      defined in board.subregisters
    voltage: voltage bound by pot max and min (set in board object)
    tune: if True, iterate with monitor to correct voltage
    accuracy: acceptable error in volts (if None, attempts to find closest
      possible pot setting and warns if last iteration does not reduce error
      below the resolution of the pot)
    iterations: number of iteration attempts
    approach: approximation parameter (>1 may cause overshoot)
    errflag: if True, return tuple with error string

Returns:
    if errflag:
tuple: (error string, response string)
    else:
response string
```

Definition at line 957 of file CameraAssembler.py.

```
    def setPotV(
        self,
        potname,
        voltage,
        tune=False,
        accuracy=0.01,
        iterations=20,
        approach=0.75,
        errflag=False,
    ):
        """
        Sets pot to specified voltage. If tune=True, uses monitor to adjust pot to
          correct voltage. Tuning will attempt to tune to closest LSB on pot; if
          'accuracy' > LSB resolution, will only complain if tuning is unable to get
          the voltage within 'accuracy'

        Args:
            potname: common name of pot, e.g., VRST found in board.subreg_aliases or
              defined in board.subregisters
            voltage: voltage bound by pot max and min (set in board object)
            tune: if True, iterate with monitor to correct voltage
            accuracy: acceptable error in volts (if None, attempts to find closest
              possible pot setting and warns if last iteration does not reduce error
              below the resolution of the pot)
            iterations: number of iteration attempts
            approach: approximation parameter (>1 may cause overshoot)
            errflag: if True, return tuple with error string

        Returns:
            if errflag:
                tuple: (error string, response string)
            else:
                response string
        """
        potname, potobj, writable = self.resolveSubreg(potname)
        if not potobj:
            err = (
                self.logerr
```

```
               + "setPotV: invalid lookup: "
               + potname
               + ' , returning "0" '
           )
           logging.error(err)
            if errflag:
                return err, "0"
            return "0"
        if not writable:
            err = self.logerr + "setPotV: not a writable subregister: " + potname
            logging.error(err)
            if errflag:
                return err, "0"
            return "0"
        if voltage < potobj.minV:
            voltage = potobj.minV
        if voltage > potobj.maxV:
            voltage = potobj.maxV
        setting = (voltage - potobj.minV) / (potobj.maxV - potobj.minV)
        err, rval = self.setPot(potname, setting, errflag=True)
        time.sleep(0.1)
        if tune:
            if potname not in self.board.monitor_controls.values():
                err = (
                    self.logerr
                    + "setPotV: pot '"
                    + potname
                    + "' does not have a corresponding monitor"
                )
                logging.error(err)
                if errflag:
                    return err, rval
                return rval
            self.setPot(potname, 0.65)
            time.sleep(0.2)
            err1, mon65 = self.getMonV(potname, errflag=True)
            self.setPot(potname, 0.35)
            time.sleep(0.2)
            err2, mon35 = self.getMonV(potname, errflag=True)
            # theoretical voltage range assuming linearity
            potrange = (mon65 - mon35) / 0.3
            stepsize = potrange / (potobj.max_value + 1)
            err += err1 + err2
            if err or potrange < 1:
                err += " ERROR: [CA] setPotV: unable to tune pot " + potname
                if potrange < 1:  # potrange should be on the order of 3.3 or 5 volts
                    err += "; monitor shows insufficient change with pot variation"
                logging.error(err)
                if errflag:
                    return err, rval
                return rval
            potzero = 0.35 - (mon35 / potrange)
            potone = 1.65 - (mon65 / potrange)
            if potzero < 0:
                potzero = 0
            if potone > 1:
                potone = 1

            if accuracy > stepsize:
                mindiff = accuracy
            else:
                mindiff = stepsize
            setting = potzero + (voltage / potone)
            self.setPot(potname, setting)
            lastdiff = 0
            smalladjust = 0
            err3 = ""
            for _ in range(iterations):
                err3i, measured = self.getMonV(potname, errflag=True)
                if err3i:
                    err3 = err3 + err3i + " "
                diff = voltage - measured
                if abs(diff - lastdiff) < stepsize / 2:
                    if (
                        smalladjust > 12
                    ):  # magic number for now; if it doesn't converge after several
                        #   tries, it never will, usually because the setting is pinned
                        #   to 0 or 1 and adjust can't change it
                        logging.warning(
                            self.logwarn
                            + "setPotV: Tuning converged too slowly: pot "
                            + potname
                            + " set to "
                            + str(voltage)
                            + "V, monitor returns "
                            + str(measured)
                            + "V"
                        )
                        if errflag:
                            return "", rval
                        return rval
                    smalladjust += 1
                if not int(2 * diff / stepsize):
                    if errflag:
                        return "", rval
                    return rval
                adjust = approach * (diff / potrange)
                setting += adjust
                if setting > 1:
                    setting = 1
                elif setting < 0:
                    setting = 0
                err1, rval = self.setPot(potname, setting, True)
                lastdiff = diff
                time.sleep(0.2)
            err4, measured = self.getMonV(potname, errflag=True)
            diff = voltage - measured
            # code will try to get to within one stepsize, but will only complain if it
            #   doesn’t get within mindiff
            if int(diff / mindiff):
                logging.warning(
                    self.logwarn
                    + "setPotV: pot "
                    + potname
                    + " set to "
                    + str(voltage)
                    + "V, monitor returns "
                    + str(measured)
                    + "V"
                )
            err += err1 + err2 + err3 + err4
        if err:
            logging.error(self.logerr + "setPotV: errors occurred: " + err)
        if errflag:
            return err, rval
        return rval

```

8.1.3.70 setPowerSave()

```
def nsCamera.CameraAssembler.CameraAssembler.setPowerSave (
                self,
                status = 1 )
```

Definition at line 347 of file CameraAssembler.py.

```
    def setPowerSave(self, status=1):
        return self.board.setPowerSave(status)

```

8.1.3.71 setPPER()

```
def nsCamera.CameraAssembler.CameraAssembler.setPPER (
                self,
                time = None )
```

Definition at line 350 of file CameraAssembler.py.

```
    def setPPER(self, time=None):
        return self.board.setPPER(time)

```

8.1.3.72 setRegister()

```
def nsCamera.CameraAssembler.CameraAssembler.setRegister (
              self,
              regname,
              regval )
```

```
Sets named register to given value as hexadecimal string without '0x'

Args:
    regname: name of register as given in ICD
    regval: value to assign to register, as hexadecimal string without '0x'

Returns:
    tuple: (error string, response string)
```

Definition at line 653 of file CameraAssembler.py.

```
    def setRegister(self, regname, regval):
        """
        Sets named register to given value as hexadecimal string without '0x'

        Args:
            regname: name of register as given in ICD
            regval: value to assign to register, as hexadecimal string without '0x'

        Returns:
            tuple: (error string, response string)
        """
        regname = regname.upper()
        if regname not in self.board.registers:
            err = self.logerr + "Invalid register name: " + regname
            logging.error(err)
            return err, "00000000"
        pkt = Packet(addr=self.board.registers[regname], data=regval)
        err, rval = self.comms.sendCMD(pkt)
        if err:
            logging.error(self.logerr + "setRegister " + regname + ": " + err)
        return err, rval

```

8.1.3.73 setRows()

```
def nsCamera.CameraAssembler.CameraAssembler.setRows (
              self,
              minrow = 0,
              maxrow = None,
              fullsize = False )
```

```
Sets bounds on rows returned by board, inclusive (e.g., 0,1023 returns all 1024
  rows). If called without parameters, resets to full image size.

Args:
    minrow: first row to return from board
    maxrow: last row to return from board
    fullsize: if True, generate full size frames, padding collected rows with
    zeroes as necessary
```

Definition at line 1830 of file CameraAssembler.py.

```
1830     def setRows(self, minrow=0, maxrow=None, fullsize=False):
1831         """
1832         Sets bounds on rows returned by board, inclusive (e.g., 0,1023 returns all 1024
1833           rows). If called without parameters, resets to full image size.
1834 
1835         Args:
1836             minrow: first row to return from board
1837             maxrow: last row to return from board
1838             fullsize: if True, generate full size frames, padding collected rows with
1839             zeroes as necessary
1840         """
1841         err = ""
1842         if maxrow is None:
1843             maxrow = self.sensor.maxheight - 1
1844         if (
1845             not isinstance(minrow, int)
1846             or minrow < 0
1847             or minrow > maxrow
1848             or not isinstance(maxrow, int)
1849             or maxrow >= self.sensor.maxheight
1850         ):
1851             err = (
1852                 self.logerr + "setRows: invalid row arguments submitted. Frame size "
1853                 "remains unchanged. "
1854             )
1855             logging.error(err)
1856             return err
1857 
1858         initrow = hex(minrow)[2:].zfill(8)
1859         finrow = hex(maxrow)[2:].zfill(8)
1860         err1, _ = self.setRegister("FPA_ROW_INITIAL", initrow)
1861         err2, _ = self.setRegister("FPA_ROW_FINAL", finrow)
1862         self.sensor.firstrow = minrow
1863         self.sensor.lastrow = maxrow
1864         self.sensor.height = maxrow - minrow + 1
1865         self.comms.payloadsize = (
1866             self.sensor.width
1867             * self.sensor.height
1868             * self.sensor.nframes
1869             * self.sensor.bytesperpixel
1870         )
1871 
1872         if self.commname == "rs422":
1873             self.comms._datatimeout = (
1874                 (1.0 * self.sensor.height / self.sensor.maxheight)
1875                 * 5e7
1876                 * self.sensor.nframes
1877                 / self.comms._baud
1878             )
1879 
1880         if fullsize:
1881             self.padToFull = True
1882         else:
1883             self.padToFull = False
1884         logging.info(
1885             self.loginfo
1886             + "Readoff set to "
1887             + str(self.sensor.height)
1888             + " rows ("
1889             + str(minrow)
1890             + ", "
1891             + str(maxrow)
1892             + ")"
1893         )
1894         err = err1 + err2
1895         if err:
1896             logging.error(
1897                 self.logerr + "setRows may not have functioned properly: " + err
```

```
            )
        return err

```

8.1.3.74 setSubregister()

```
def nsCamera.CameraAssembler.CameraAssembler.setSubregister (
              self,
              subregname,
              valstring )
```

```
Sets substring of register identified in board attribute 'subregname' to
  valstring if subregister is writable

Args:
    subregname: listed in board.subreg_aliases or defined in board.subregisters
    valstring: binary string without '0b'

Returns:
    tuple: (error, packet response string) from setRegister
```

Definition at line 733 of file CameraAssembler.py.

```
    def setSubregister(self, subregname, valstring):
        """
        Sets substring of register identified in board attribute 'subregname' to
          valstring if subregister is writable

        Args:
            subregname: listed in board.subreg_aliases or defined in board.subregisters
            valstring: binary string without '0b'

        Returns:
            tuple: (error, packet response string) from setRegister
        """
        subregname, subregobj, writable = self.resolveSubreg(subregname)
        if not subregobj:
            err = self.logerr + "setSubregister: invalid lookup: " + subregname
            logging.error(err)
            return err, "0"
        if not writable:
            err = (
                self.logerr
                + "setSubregister: not a writable subregister: "
                + subregname
            )
            logging.error(err)
            return err, "0"
        if len(str(valstring)) > subregobj.width:
            err = self.logerr + "setSubregister: replacement string is too long"
            logging.error(err)
            return err, "0"
        # read current value of register data
        err, resp = self.getRegister(subregobj.register)
        if err:
            logging.error(
                self.logerr + "setSubregister: unable to retrieve register setting; "
                "setting of " + subregname + " likely failed)"
            )
            return err, "0"
        hex_str = "0x" + resp
        b_reg_value = "{0:0=32b}".format(int(hex_str, 16))  # convert to binary
        # list indexing is reversed from bit string; the last bit of the string is at
        #   index 0 in the list (thus bit 0 is at index 0)
        startindex = 31 - subregobj.start_bit
        valstringpadded = str(valstring).zfill(subregobj.width)
        fullreg = list(b_reg_value)
```

```
777         fullreg[startindex : startindex + subregobj.width] = valstringpadded
778         # convert binary string back to hexadecimal string for writing
779         new_reg_value = "".join(fullreg)
780         h_reg_value = "{num:{fill}{width}x}".format(
781             num=int(new_reg_value, 2), fill="0", width=8
782         )
783         return self.setRegister(subregobj.register, h_reg_value)
784
```

8.1.3.75 setTiming()

```
def nsCamera.CameraAssembler.CameraAssembler.setTiming (
                self,
                side = None,
                sequence = None,
                delay = None )
```

Definition at line 380 of file CameraAssembler.py.

```
380     def setTiming(self, side=None, sequence=None, delay=None):
381         return self.sensor.setTiming(side, sequence, delay)
382
```

8.1.3.76 setTriggerDelay()

```
def nsCamera.CameraAssembler.CameraAssembler.setTriggerDelay (
                self,
                delayblocks = 0 )
```

Definition at line 407 of file CameraAssembler.py.

```
407     def setTriggerDelay(self, delayblocks=0):
408         return self.sensor.setTriggerDelay(delayblocks)
409
```

8.1.3.77 setZeroDeadTime()

```
def nsCamera.CameraAssembler.CameraAssembler.setZeroDeadTime (
                self,
                flag = True )
```

Definition at line 404 of file CameraAssembler.py.

```
404     def setZeroDeadTime(self, flag=True):
405         return self.sensor.setZeroDeadTime(flag)
406
```

8.1.3.78 startCapture()

```
def nsCamera.CameraAssembler.CameraAssembler.startCapture (
              self,
              mode )
```

Definition at line 326 of file CameraAssembler.py.

```
326     def startCapture(self, mode):
327         return self.board.startCapture(mode)
328
```

8.1.3.79 str2bytes()

```
def nsCamera.CameraAssembler.CameraAssembler.str2bytes (
              self,
              astring )
```

```
Python-version-agnostic converter of hexadecimal strings to bytes

Args:
    astring: hexadecimal string without '0x'

Returns:
    byte string equivalent to input string
```

Definition at line 1670 of file CameraAssembler.py.

```
1670     def str2bytes(self, astring):
1671         """
1672         Python-version-agnostic converter of hexadecimal strings to bytes
1673
1674         Args:
1675             astring: hexadecimal string without '0x'
1676
1677         Returns:
1678             byte string equivalent to input string
1679         """
1680         if self.PY3:
1681             dbytes = binascii.a2b_hex(astring)
1682         else:
1683             dbytes = astring.decode("hex")
1684         return dbytes
1685
```

8.1.3.80 str2nparray()

```
def nsCamera.CameraAssembler.CameraAssembler.str2nparray (
              self,
              valstring )
```

```
Convert string into array of uint16s

Args:
    valstring: string of hexadecimal characters

Returns:
    numpy array of uint16
```

Definition at line 1701 of file CameraAssembler.py.

```
    def str2nparray(self, valstring):
        """
        Convert string into array of uint16s

        Args:
            valstring: string of hexadecimal characters

        Returns:
            numpy array of uint16
        """
        stringlen = len(valstring)
        arraylen = int(stringlen / 4)
        outarray = np.empty(int(arraylen), dtype="uint16")

        for i in range(0, arraylen):
            outarray[i] = int(valstring[4 * i : 4 * i + 4], 16)
        return outarray
```

8.1.3.81 submitMessages()

```
def nsCamera.CameraAssembler.CameraAssembler.submitMessages (
                self,
                messages,
                errorstring = "Error" )
```

```
Serially set multiple register / subregister values

Args:
    messages: list of tuples (register name, hexadecimal string without ’0x’)
      and/or (subregister name, binary string without ’0b’)
    errorstring: error message to print in case of failure

Returns:
    tuple (accumulated error string, response string of final message)
```

Definition at line 785 of file CameraAssembler.py.

```
    def submitMessages(self, messages, errorstring="Error"):
        """
        Serially set multiple register / subregister values

        Args:
            messages: list of tuples (register name, hexadecimal string without ’0x’)
              and/or (subregister name, binary string without ’0b’)
            errorstring: error message to print in case of failure

        Returns:
            tuple (accumulated error string, response string of final message)
        """
        errs = ""
        err = ""
        rval = ""
        for m in messages:
            if m[0].upper() in self.board.registers:
                err, rval = self.setRegister(m[0].upper(), m[1])
            elif m[0].upper() in self.board.subreglist:
                err, rval = self.setSubregister(m[0].upper(), m[1])
            else:
                err = (
                    self.logerr
                    + "submitMessages: Invalid register/subregister: "
                    + errorstring
                    + m[0]
                )
                logging.error(err)
            errs = errs + err
        return err, rval
```

8.1.3.82 waitForSRAM()

```
def nsCamera.CameraAssembler.CameraAssembler.waitForSRAM (
              self,
              timeout = None )
```

Definition at line 332 of file CameraAssembler.py.

```
332     def waitForSRAM(self, timeout=None):
333         return self.board.waitForSRAM(timeout)
334
```

8.1.3.83 writeSerial()

```
def nsCamera.CameraAssembler.CameraAssembler.writeSerial (
              self,
              cmd,
              timeout = None )
```

Definition at line 422 of file CameraAssembler.py.

```
422     def writeSerial(self, cmd, timeout=None):
423         return self.comms.writeSerial(cmd, timeout)
424
```

8.1.4 Member Data Documentation

8.1.4.1 abort

```
nsCamera.CameraAssembler.CameraAssembler.abort
```

Definition at line 245 of file CameraAssembler.py.

8.1.4.2 armed

```
nsCamera.CameraAssembler.CameraAssembler.armed
```

Definition at line 237 of file CameraAssembler.py.

8.1.4.3 board

```
nsCamera.CameraAssembler.CameraAssembler.board
```

Definition at line 481 of file CameraAssembler.py.

8.1.4.4 boardname

```
nsCamera.CameraAssembler.CameraAssembler.boardname
```

Definition at line 209 of file CameraAssembler.py.

8.1.4.5 commname

```
nsCamera.CameraAssembler.CameraAssembler.commname
```

Definition at line 214 of file CameraAssembler.py.

8.1.4.6 comms

```
nsCamera.CameraAssembler.CameraAssembler.comms
```

Definition at line 475 of file CameraAssembler.py.

8.1.4.7 currtime

```
nsCamera.CameraAssembler.CameraAssembler.currtime
```

Definition at line 200 of file CameraAssembler.py.

8.1.4.8 cycle

```
nsCamera.CameraAssembler.CameraAssembler.cycle
```

Definition at line 208 of file CameraAssembler.py.

8.1.4.9 FPGAboardtype

```
nsCamera.CameraAssembler.CameraAssembler.FPGAboardtype
```

Definition at line 227 of file CameraAssembler.py.

8.1.4.10 FPGAinterfaces

```
nsCamera.CameraAssembler.CameraAssembler.FPGAinterfaces
```

Definition at line 230 of file CameraAssembler.py.

8.1.4.11 FPGAinvalid

```
nsCamera.CameraAssembler.CameraAssembler.FPGAinvalid
```

Definition at line 233 of file CameraAssembler.py.

8.1.4.12 FPGANum

```
nsCamera.CameraAssembler.CameraAssembler.FPGANum
```

Definition at line 224 of file CameraAssembler.py.

8.1.4.13 FPGArad

```
nsCamera.CameraAssembler.CameraAssembler.FPGArad
```

Definition at line 228 of file CameraAssembler.py.

8.1.4.14 FPGAsensor

```
nsCamera.CameraAssembler.CameraAssembler.FPGAsensor
```

Definition at line 229 of file CameraAssembler.py.

8.1.4.15 FPGAVersion

```
nsCamera.CameraAssembler.CameraAssembler.FPGAVersion
```

Definition at line 223 of file CameraAssembler.py.

8.1.4.16 inittime

```
nsCamera.CameraAssembler.CameraAssembler.inittime
```

Definition at line 243 of file CameraAssembler.py.

8.1.4.17 iplist

```
nsCamera.CameraAssembler.CameraAssembler.iplist
```

Definition at line 235 of file CameraAssembler.py.

8.1.4.18 logcrit

```
nsCamera.CameraAssembler.CameraAssembler.logcrit
```

Definition at line 264 of file CameraAssembler.py.

8.1.4.19 logcritbase

```
nsCamera.CameraAssembler.CameraAssembler.logcritbase
```

Definition at line 258 of file CameraAssembler.py.

8.1.4.20 logdebug

```
nsCamera.CameraAssembler.CameraAssembler.logdebug
```

Definition at line 268 of file CameraAssembler.py.

8.1.4.21 logdebugbase

```
nsCamera.CameraAssembler.CameraAssembler.logdebugbase
```

Definition at line 262 of file CameraAssembler.py.

8.1.4.22 logerr

```
nsCamera.CameraAssembler.CameraAssembler.logerr
```

Definition at line 265 of file CameraAssembler.py.

8.1.4.23 logerrbase

```
nsCamera.CameraAssembler.CameraAssembler.logerrbase
```

Definition at line 259 of file CameraAssembler.py.

8.1.4.24 loginfo

```
nsCamera.CameraAssembler.CameraAssembler.loginfo
```

Definition at line 267 of file CameraAssembler.py.

8.1.4.25 loginfobase

```
nsCamera.CameraAssembler.CameraAssembler.loginfobase
```

Definition at line 261 of file CameraAssembler.py.

8.1.4.26 logtag

```
nsCamera.CameraAssembler.CameraAssembler.logtag
```

Definition at line 257 of file CameraAssembler.py.

8.1.4.27 logwarn

```
nsCamera.CameraAssembler.CameraAssembler.logwarn
```

Definition at line 266 of file CameraAssembler.py.

8.1.4.28 logwarnbase

```
nsCamera.CameraAssembler.CameraAssembler.logwarnbase
```

Definition at line 260 of file CameraAssembler.py.

8.1.4.29 oldtime

```
nsCamera.CameraAssembler.CameraAssembler.oldtime
```

Definition at line 201 of file CameraAssembler.py.

8.1.4.30 packageroot

```
nsCamera.CameraAssembler.CameraAssembler.packageroot
```

Definition at line 236 of file CameraAssembler.py.

8.1.4.31 padToFull

```
nsCamera.CameraAssembler.CameraAssembler.padToFull
```

Definition at line 244 of file CameraAssembler.py.

8.1.4.32 parsedtime

```
nsCamera.CameraAssembler.CameraAssembler.parsedtime
```

Definition at line 206 of file CameraAssembler.py.

8.1.4.33 payloaderror

```
nsCamera.CameraAssembler.CameraAssembler.payloaderror
```

Definition at line 293 of file CameraAssembler.py.

8.1.4.34 platform

```
nsCamera.CameraAssembler.CameraAssembler.platform
```

Definition at line 220 of file CameraAssembler.py.

8.1.4.35 port

```
nsCamera.CameraAssembler.CameraAssembler.port
```

Definition at line 217 of file CameraAssembler.py.

8.1.4.36 PY3

```
nsCamera.CameraAssembler.CameraAssembler.PY3
```

Definition at line 219 of file CameraAssembler.py.

8.1.4.37 read

```
nsCamera.CameraAssembler.CameraAssembler.read
```

Definition at line 204 of file CameraAssembler.py.

8.1.4.38 savetime

```
nsCamera.CameraAssembler.CameraAssembler.savetime
```

Definition at line 207 of file CameraAssembler.py.

8.1.4.39 sensmanual

```
nsCamera.CameraAssembler.CameraAssembler.sensmanual
```

Definition at line 242 of file CameraAssembler.py.

8.1.4.40 sensor

```
nsCamera.CameraAssembler.CameraAssembler.sensor
```

Definition at line 456 of file CameraAssembler.py.

8.1.4.41 sensorname

```
nsCamera.CameraAssembler.CameraAssembler.sensorname
```

For regular version.

Definition at line 215 of file CameraAssembler.py.

8.1.4.42 senstiming

```
nsCamera.CameraAssembler.CameraAssembler.senstiming
```

Definition at line 241 of file CameraAssembler.py.

8.1.4.43 trigtime

```
nsCamera.CameraAssembler.CameraAssembler.trigtime
```

Definition at line 202 of file CameraAssembler.py.

8.1.4.44 unstringed

```
nsCamera.CameraAssembler.CameraAssembler.unstringed
```

Definition at line 205 of file CameraAssembler.py.

8.1.4.45 verblevel

```
nsCamera.CameraAssembler.CameraAssembler.verblevel
```

Definition at line 270 of file CameraAssembler.py.

8.1.4.46 verbmap

```
nsCamera.CameraAssembler.CameraAssembler.verbmap
```

Definition at line 247 of file CameraAssembler.py.

8.1.4.47 verbose

```
nsCamera.CameraAssembler.CameraAssembler.verbose
```

Definition at line 216 of file CameraAssembler.py.

8.1.4.48 version

```
nsCamera.CameraAssembler.CameraAssembler.version
```

Definition at line 199 of file CameraAssembler.py.

8.1.4.49 waited

```
nsCamera.CameraAssembler.CameraAssembler.waited
```

Definition at line 203 of file CameraAssembler.py.

The documentation for this class was generated from the following file:

- nsCamera/CameraAssembler.py

8.2 nsCamera.sensors.daedalus.daedalus Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def checkSensorVoltStat (self)
- def sensorSpecific (self)
- def setInterlacing (self, ifactor)
- def setHighFullWell (self, flag)
- def setZeroDeadTime (self, flag)
- def setTriggerDelay (self, delayblocks)
- def setTiming (self, side, sequence, delay)
- def setArbTiming (self, side, sequence)
- def getTiming (self, side, actual)
- def setManualShutters (self, timing)
- def getManualTiming (self)
- def parseReadoff (self, frames)
- def reportStatusSensor (self, statusbits)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- minframe
- maxframe
- firstframe
- lastframe
- nframes
- maxwidth
- maxheight
- firstrow
- lastrow
- width
- height
- bytesperpixel
- fpganumID
- interlacing
- ZDT
- HFW
- sens_registers
- sens_subregisters

8.2.1 Detailed Description

Definition at line 26 of file daedalus.py.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 __init__()

```
def nsCamera.sensors.daedalus.daedalus.__init__ (
                self,
                camassem )
```

Definition at line 27 of file daedalus.py.

```
27     def __init__(self, camassem):
28         self.ca = camassem
29         self.logcrit = self.ca.logcritbase + "[Daedalus] "
30         self.logerr = self.ca.logerrbase + "[Daedalus] "
31         self.logwarn = self.ca.logwarnbase + "[Daedalus] "
32         self.loginfo = self.ca.loginfobase + "[Daedalus] "
33         self.logdebug = self.ca.logdebugbase + "[Daedalus] "
34         logging.info(self.loginfo + "initializing sensor object")
35 
36         self.minframe = 0
37         self.maxframe = 2
38         self.firstframe = self.minframe
39         self.lastframe = self.maxframe
40         self.nframes = self.maxframe - self.minframe + 1
41         self.maxwidth = 512
42         self.maxheight = 1024
43         self.firstrow = 0
44         self.lastrow = self.maxheight - 1
45         self.width = self.maxwidth
46         self.height = self.maxheight
47         self.bytesperpixel = 2
48         self.fpganumID = "2"  # last nybble of FPGA_NUM
49         self.interlacing = 0
50         self.ZDT = False
51         self.HFW = False
52 
53         self.sens_registers = OrderedDict(
54             {
55                 "HST_READBACK_A_LO": "018",
56                 "HST_READBACK_A_HI": "019",
57                 "HST_READBACK_B_LO": "01A",
58                 "HST_READBACK_B_HI": "01B",
59                 "HSTALLWEN_WAIT_TIME": "03F",
60                 "FRAME_ORDER_SEL": "04B",
61                 "HST_TRIGGER_DELAY_DATA_LO": "120",
62                 "HST_TRIGGER_DELAY_DATA_HI": "121",
63                 "HST_PHI_DELAY_DATA_LO": "122",
64                 "HST_PHI_DELAY_DATA_HI": "123",
65                 "HST_TRIG_DELAY_READBACK_LO": "125",
66                 "HST_TRIG_DELAY_READBACK_HI": "126",
67                 "HST_PHI_DELAY_READBACK_LO": "127",
68                 "HST_PHI_DELAY_READBACK_HI": "128",
69                 "HST_COUNT_TRIG": "130",
70                 "HST_DELAY_EN": "131",
71                 "HST_TEST_PHI_EN": "132",
72                 "RSL_HFW_MODE_EN": "133",
73                 "RSL_ZDT_MODE_R_EN": "135",
74                 "RSL_ZDT_MODE_L_EN": "136",
75                 "BGTRIMA": "137",
76                 "BGTRIMB": "138",
77                 "COLUMN_TEST_EN": "139",
78                 "RSL_CONFIG_DATA_R0": "140",
79                 "RSL_CONFIG_DATA_R1": "141",
80                 "RSL_CONFIG_DATA_R2": "142",
81                 "RSL_CONFIG_DATA_R3": "143",
82                 "RSL_CONFIG_DATA_R4": "144",
83                 "RSL_CONFIG_DATA_R5": "145",
84                 "RSL_CONFIG_DATA_R6": "146",
85                 "RSL_CONFIG_DATA_R7": "147",
86                 "RSL_CONFIG_DATA_R8": "148",
87                 "RSL_CONFIG_DATA_R9": "149",
                "RSL_CONFIG_DATA_R10": "14A",
                "RSL_CONFIG_DATA_R11": "14B",
                "RSL_CONFIG_DATA_R12": "14C",
                "RSL_CONFIG_DATA_R13": "14D",
                "RSL_CONFIG_DATA_R14": "14E",
                "RSL_CONFIG_DATA_R15": "14F",
                "RSL_CONFIG_DATA_R16": "150",
                "RSL_CONFIG_DATA_R17": "151",
                "RSL_CONFIG_DATA_R18": "152",
                "RSL_CONFIG_DATA_R19": "153",
                "RSL_CONFIG_DATA_R20": "154",
                "RSL_CONFIG_DATA_R21": "155",
                "RSL_CONFIG_DATA_R22": "156",
                "RSL_CONFIG_DATA_R23": "157",
                "RSL_CONFIG_DATA_R24": "158",
                "RSL_CONFIG_DATA_R25": "159",
                "RSL_CONFIG_DATA_R26": "15A",
                "RSL_CONFIG_DATA_R27": "15B",
                "RSL_CONFIG_DATA_R28": "15C",
                "RSL_CONFIG_DATA_R29": "15D",
                "RSL_CONFIG_DATA_R30": "15E",
                "RSL_CONFIG_DATA_R31": "15F",
                "RSL_CONFIG_DATA_L0": "160",
                "RSL_CONFIG_DATA_L1": "161",
                "RSL_CONFIG_DATA_L2": "162",
                "RSL_CONFIG_DATA_L3": "163",
                "RSL_CONFIG_DATA_L4": "164",
                "RSL_CONFIG_DATA_L5": "165",
                "RSL_CONFIG_DATA_L6": "166",
                "RSL_CONFIG_DATA_L7": "167",
                "RSL_CONFIG_DATA_L8": "168",
                "RSL_CONFIG_DATA_L9": "169",
                "RSL_CONFIG_DATA_L10": "16A",
                "RSL_CONFIG_DATA_L11": "16B",
                "RSL_CONFIG_DATA_L12": "16C",
                "RSL_CONFIG_DATA_L13": "16D",
                "RSL_CONFIG_DATA_L14": "16E",
                "RSL_CONFIG_DATA_L15": "16F",
                "RSL_CONFIG_DATA_L16": "170",
                "RSL_CONFIG_DATA_L17": "171",
                "RSL_CONFIG_DATA_L18": "172",
                "RSL_CONFIG_DATA_L19": "173",
                "RSL_CONFIG_DATA_L20": "174",
                "RSL_CONFIG_DATA_L21": "175",
                "RSL_CONFIG_DATA_L22": "176",
                "RSL_CONFIG_DATA_L23": "177",
                "RSL_CONFIG_DATA_L24": "178",
                "RSL_CONFIG_DATA_L25": "179",
                "RSL_CONFIG_DATA_L26": "17A",
                "RSL_CONFIG_DATA_L27": "17B",
                "RSL_CONFIG_DATA_L28": "17C",
                "RSL_CONFIG_DATA_L29": "17D",
                "RSL_CONFIG_DATA_L30": "17E",
                "RSL_CONFIG_DATA_L31": "17F",
                "RSL_READ_BACK_R0": "180",
                "RSL_READ_BACK_R1": "181",
                "RSL_READ_BACK_R2": "182",
                "RSL_READ_BACK_R3": "183",
                "RSL_READ_BACK_R4": "184",
                "RSL_READ_BACK_R5": "185",
                "RSL_READ_BACK_R6": "186",
                "RSL_READ_BACK_R7": "187",
                "RSL_READ_BACK_R8": "188",
                "RSL_READ_BACK_R9": "189",
                "RSL_READ_BACK_R10": "18A",
                "RSL_READ_BACK_R11": "18B",
                "RSL_READ_BACK_R12": "18C",
                "RSL_READ_BACK_R13": "18D",
                "RSL_READ_BACK_R14": "18E",
                "RSL_READ_BACK_R15": "18F",
                "RSL_READ_BACK_R16": "190",
                "RSL_READ_BACK_R17": "191",
                "RSL_READ_BACK_R18": "192",
                "RSL_READ_BACK_R19": "193",
                "RSL_READ_BACK_R20": "194",
                "RSL_READ_BACK_R21": "195",
                "RSL_READ_BACK_R22": "196",
                "RSL_READ_BACK_R23": "197",
                "RSL_READ_BACK_R24": "198",
                "RSL_READ_BACK_R25": "199",
                "RSL_READ_BACK_R26": "19A",
                "RSL_READ_BACK_R27": "19B",
                "RSL_READ_BACK_R28": "19C",
                "RSL_READ_BACK_R29": "19D",
                "RSL_READ_BACK_R30": "19E",
                "RSL_READ_BACK_R31": "19F",
                "RSL_READ_BACK_L0": "1A0",
                "RSL_READ_BACK_L1": "1A1",
                "RSL_READ_BACK_L2": "1A2",
                "RSL_READ_BACK_L3": "1A3",
                "RSL_READ_BACK_L4": "1A4",
                "RSL_READ_BACK_L5": "1A5",
                "RSL_READ_BACK_L6": "1A6",
                "RSL_READ_BACK_L7": "1A7",
                "RSL_READ_BACK_L8": "1A8",
                "RSL_READ_BACK_L9": "1A9",
                "RSL_READ_BACK_L10": "1AA",
                "RSL_READ_BACK_L11": "1AB",
                "RSL_READ_BACK_L12": "1AC",
                "RSL_READ_BACK_L13": "1AD",
                "RSL_READ_BACK_L14": "1AE",
                "RSL_READ_BACK_L15": "1AF",
                "RSL_READ_BACK_L16": "1B0",
                "RSL_READ_BACK_L17": "1B1",
                "RSL_READ_BACK_L18": "1B2",
                "RSL_READ_BACK_L19": "1B3",
                "RSL_READ_BACK_L20": "1B4",
                "RSL_READ_BACK_L21": "1B5",
                "RSL_READ_BACK_L22": "1B6",
                "RSL_READ_BACK_L23": "1B7",
                "RSL_READ_BACK_L24": "1B8",
                "RSL_READ_BACK_L25": "1B9",
                "RSL_READ_BACK_L26": "1BA",
                "RSL_READ_BACK_L27": "1BB",
                "RSL_READ_BACK_L28": "1BC",
                "RSL_READ_BACK_L29": "1BD",
                "RSL_READ_BACK_L30": "1BE",
                "RSL_READ_BACK_L31": "1BF",
            }
        )

        self.sens_subregisters = [
            ("STAT_RSLROWOUTL", "STAT_REG", 3, 1, False),
            ("STAT_RSLROWOUTR", "STAT_REG", 4, 1, False),
            ("STAT_RSLNALLWENR", "STAT_REG", 12, 1, False),
            ("STAT_RSLNALLWENL", "STAT_REG", 15, 1, False),
            ("STAT_CONFIGHSTDONE", "STAT_REG", 16, 1, False),
            ("SLOWREADOFF_0", "CTRL_REG", 4, 1, True),
            ("SLOWREADOFF_1", "CTRL_REG", 5, 1, True),
            ("HFW", "RSL_HFW_MODE_EN", 0, 1, True),
            ("ZDT_R", "RSL_ZDT_MODE_R_EN", 0, 1, True),
            ("ZDT_L", "RSL_ZDT_MODE_L_EN", 0, 1, True),
        ]

```

8.2.3 Member Function Documentation

8.2.3.1 checkSensorVoltStat()

```
def nsCamera.sensors.daedalus.daedalus.checkSensorVoltStat (
              self )
```

```
Checks register tied to sensor select jumpers to confirm match with sensor
  object

Returns:
    boolean, True if jumpers select for Daedalus sensor
```

Definition at line 222 of file daedalus.py.

```
    def checkSensorVoltStat(self):
        """
        Checks register tied to sensor select jumpers to confirm match with sensor
          object

        Returns:
            boolean, True if jumpers select for Daedalus sensor
        """
        err, status = self.ca.getSubregister("DAEDALUS_DET")
        if err:
            logging.error(self.logerr + "unable to confirm sensor status")
            return False
        if not int(status):
            logging.error(self.logerr + "Daedalus sensor not detected")
            return False
        return True
```

8.2.3.2 getManualTiming()

```
def nsCamera.sensors.daedalus.daedalus.getManualTiming (
              self )
```

```
Dummy function; feature is not implemented on Daedalus

Returns:
    list of 2 dummy lists
```

Definition at line 766 of file daedalus.py.

```
    def getManualTiming(self):
        """
        Dummy function; feature is not implemented on Daedalus

        Returns:
            list of 2 dummy lists
        """
        logging.warning(
            self.logwarn + "manual shutter control is not implemented in the "
            "Daedalus sensor "
        )
        return [[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
```

8.2.3.3 getTiming()

```
def nsCamera.sensors.daedalus.daedalus.getTiming (
              self,
              side,
              actual )
actual = True: returns actual high speed intervals that will be generated by the
  FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2, open3]
actual = False: Returns high speed timing settings as set by setTiming. Assumes
  that timing was set via the setTiming method--it will not accurately report
  arbitrary timings set by direct register sets or manual shutter control

Args:
    side: Hemisphere 'A' or 'B'
    actual: False: return HST settings
    True: calculate and return actual HST behavior

Returns:
    actual= False: tuple   (hemisphere label,
                    'open shutter' in ns,
                    'closed shutter' in ns,
                    initial delay in ns)
    True: list of times [delay, open0, closed0, open1, closed1, open2,
      closed2, open3]
```

Definition at line 685 of file daedalus.py.

```
685     def getTiming(self, side, actual):
686         """
687         actual = True: returns actual high speed intervals that will be generated by the
688           FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2, open3]
689         actual = False: Returns high speed timing settings as set by setTiming. Assumes
690           that timing was set via the setTiming method--it will not accurately report
691           arbitrary timings set by direct register sets or manual shutter control
692 
693         Args:
694             side: Hemisphere 'A' or 'B'
695             actual: False: return HST settings
696                     True: calculate and return actual HST behavior
697 
698         Returns:
699             actual= False: tuple   (hemisphere label,
700                                     'open shutter' in ns,
701                                     'closed shutter' in ns,
702                                     initial delay in ns)
703                     True: list of times [delay, open0, closed0, open1, closed1, open2,
704                       closed2, open3]
705         """
706         if side is None:
707             side = "A"
708 
709         logging.info(self.loginfo + "get timing, side " + side.upper())
710         if side.upper() == "A":
711             lowreg = "HS_TIMING_DATA_ALO"
712             highreg = "HS_TIMING_DATA_AHI"
713         elif side.upper() == "B":
714             lowreg = "HS_TIMING_DATA_BLO"
715             highreg = "HS_TIMING_DATA_BHI"
716         else:
717             logging.error(
718                 self.logerr
719                 + "Invalid sensor side: "
720                 + side
721                 + "; timing settings unchanged"
722             )
723             return "", 0, 0, 0
724         err, lowpart = self.ca.getRegister(lowreg)
725         err1, highpart = self.ca.getRegister(highreg)
726         if err or err1:
727             logging.error(
728                 self.logerr + "Unable to retrieve timing setting (getTiming), "
729                 "returning zeroes "
730             )
731             return side.upper(), 0, 0, 0
732         full40hex = highpart[-2:] + lowpart.zfill(8)
733         full40bin = "{0:0=40b}".format(int(full40hex, 16))
734         if actual:
735             full160 = 4 * full40bin
736             gblist = [[k, len(list(g))] for k, g in itertools.groupby(full160)]
737             times = [int(x[1]) for x in gblist[:-7:-1]]
738             times[0] = times[0] - 1
739             return times
740         else:
```

```
741             gblist = [[k, len(list(g))] for k, g in itertools.groupby(full40bin)]
742             delay = gblist[-1][1] - 1
743             timeon = gblist[-2][1]
744             if len(gblist) == 2:  # 39,1 corner case
745                 timeoff = 1
746             elif len(gblist) == 3:  # sequence fits only once
747                 timeoff = 40 - timeon
748             else:
749                 timeoff = gblist[-3][1]
750             return side.upper(), timeon, timeoff, delay
751
```

8.2.3.4 parseReadoff()

```
def nsCamera.sensors.daedalus.daedalus.parseReadoff (
                self,
                frames )
```

```
Parses frames from board into images
Args:
    frames: data sets returned from board
Returns:
    list of frames reordered and deinterlaced
```

Definition at line 779 of file daedalus.py.

```
779     def parseReadoff(self, frames):
780         """
781         Parses frames from board into images
782         Args:
783             frames: data sets returned from board
784         Returns:
785             list of frames reordered and deinterlaced
786         """
787         w = self.width
788         if self.ca.padToFull:
789             rows = self.maxheight
790         else:
791             rows = self.lastrow - self.firstrow + 1
792         parsed = []
793         for frame in frames:
794             current = np.zeros((rows, w), dtype=int)
795             mapped = np.zeros((rows, w), dtype=int)
796             frame = frame.reshape(rows, w)
797
798             for entry in range(int(w / 2)):
799                 col = 32 * (entry % 8) + entry // 8  # lookup from daedlookup.xls
800                 for row in range(rows):
801                     current[row][col] = frame[row][2 * entry]
802                     current[row][col + 256] = frame[row][2 * entry + 1]
803
804             for row in range(rows):
805                 mapped[row][0:32] = current[row][320:352]
806                 mapped[row][32:64] = current[row][352:384]
807                 mapped[row][64:96] = current[row][192:224]
808                 mapped[row][96:128] = current[row][160:192]
809                 mapped[row][128:160] = current[row][256:288]
810                 mapped[row][160:192] = current[row][288:320]
811                 mapped[row][192:224] = current[row][416:448]
812                 mapped[row][224:256] = current[row][32:64]
813                 mapped[row][256:288] = current[row][128:160]
814                 mapped[row][288:320] = current[row][224:256]
815                 mapped[row][320:352] = current[row][384:416]
816                 mapped[row][352:384] = current[row][448:480]
817                 mapped[row][384:416] = current[row][480:512]
818                 mapped[row][416:448] = current[row][0:32]
819                 mapped[row][448:480] = current[row][64:96]
820                 mapped[row][480:512] = current[row][96:128]
821             parsed.append(mapped)
```

```

        images = self.ca.deInterlace(parsed, self.interlacing)
        flatimages = [x.flatten() for x in images]
        return flatimages

```

8.2.3.5 reportStatusSensor()

```
def nsCamera.sensors.daedalus.daedalus.reportStatusSensor (
             self,
             statusbits )
```

```
Print status messages from sensor-specific bits of status register or object
  status flags

Args:
    statusbits: result of checkStatus()
```

Definition at line 827 of file daedalus.py.

```
    def reportStatusSensor(self, statusbits):
        """
        Print status messages from sensor-specific bits of status register or object
          status flags

        Args:
            statusbits: result of checkStatus()
        """
        if int(statusbits[3]):
            logging.info(self.loginfo + "RSLROWINL detected")
        if int(statusbits[4]):
            logging.info(self.loginfo + "RSLROWINR detected")
        if int(statusbits[12]):
            logging.info(self.loginfo + "RSLNALLWENR detected")
        if int(statusbits[15]):
            logging.info(self.loginfo + "RSLNALLWENL detected")
        if int(statusbits[16]):
            logging.info(self.loginfo + "CONFIGHSTDONE detected")
        if self.HFW:
            logging.info(self.loginfo + "High Full Well mode active")
        if self.ZDT:
            logging.info(self.loginfo + "Zero Dead Time mode active")


"""
Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
LLNL-CODE-838080

This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
(DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
See license for disclaimers, notice of U.S. Government Rights and license terms and
conditions.
"""
```

8.2.3.6 sensorSpecific()

```
def nsCamera.sensors.daedalus.daedalus.sensorSpecific (
             self )
```

```
Returns:
    list of tuples, (Sensor-specific register, default setting)
```

Definition at line 239 of file daedalus.py.

```
239     def sensorSpecific(self):
240         """
241         Returns:
242             list of tuples, (Sensor-specific register, default setting)
243         """
244         return [
245             ("FPA_FRAME_INITIAL", "00000000"),
246             ("FPA_FRAME_FINAL", "00000002"),
247             ("FPA_ROW_INITIAL", "00000000"),
248             ("FPA_ROW_FINAL", "000003FF"),
249             ("HS_TIMING_DATA_ALO", "00006666"),  # 0db6 = 2-1; 6666 = 2-2
250             ("HS_TIMING_DATA_AHI", "00000000"),
251             ("HS_TIMING_DATA_BLO", "00006666"),
252             ("HS_TIMING_DATA_BHI", "00000000"),
253             ("FRAME_ORDER_SEL", "00000000"),
254             ("RSL_HFW_MODE_EN", "00000000"),
255             ("RSL_ZDT_MODE_R_EN", "00000000"),
256             ("RSL_ZDT_MODE_L_EN", "00000000"),
257             ("RSL_CONFIG_DATA_R0", "00000000"),
258             ("RSL_CONFIG_DATA_R1", "00000000"),
259             ("RSL_CONFIG_DATA_R2", "00000000"),
260             ("RSL_CONFIG_DATA_R3", "00000000"),
261             ("RSL_CONFIG_DATA_R4", "00000000"),
262             ("RSL_CONFIG_DATA_R5", "00000000"),
263             ("RSL_CONFIG_DATA_R6", "00000000"),
264             ("RSL_CONFIG_DATA_R7", "00000000"),
265             ("RSL_CONFIG_DATA_R8", "00000000"),
266             ("RSL_CONFIG_DATA_R9", "00000000"),
267             ("RSL_CONFIG_DATA_R10", "00000000"),
268             ("RSL_CONFIG_DATA_R11", "00000000"),
269             ("RSL_CONFIG_DATA_R12", "00000000"),
270             ("RSL_CONFIG_DATA_R13", "00000000"),
271             ("RSL_CONFIG_DATA_R14", "00000000"),
272             ("RSL_CONFIG_DATA_R15", "00000000"),
273             ("RSL_CONFIG_DATA_R16", "00000000"),
274             ("RSL_CONFIG_DATA_R17", "00000000"),
275             ("RSL_CONFIG_DATA_R18", "00000000"),
276             ("RSL_CONFIG_DATA_R19", "00000000"),
277             ("RSL_CONFIG_DATA_R20", "00000000"),
278             ("RSL_CONFIG_DATA_R21", "00000000"),
279             ("RSL_CONFIG_DATA_R22", "00000000"),
280             ("RSL_CONFIG_DATA_R23", "00000000"),
281             ("RSL_CONFIG_DATA_R24", "00000000"),
282             ("RSL_CONFIG_DATA_R25", "00000000"),
283             ("RSL_CONFIG_DATA_R26", "00000000"),
284             ("RSL_CONFIG_DATA_R27", "00000000"),
285             ("RSL_CONFIG_DATA_R28", "00000000"),
286             ("RSL_CONFIG_DATA_R29", "00000000"),
287             ("RSL_CONFIG_DATA_R30", "00000000"),
288             ("RSL_CONFIG_DATA_R31", "00000000"),
289             ("RSL_CONFIG_DATA_L0", "00000000"),
290             ("RSL_CONFIG_DATA_L1", "00000000"),
291             ("RSL_CONFIG_DATA_L2", "00000000"),
292             ("RSL_CONFIG_DATA_L3", "00000000"),
293             ("RSL_CONFIG_DATA_L4", "00000000"),
294             ("RSL_CONFIG_DATA_L5", "00000000"),
295             ("RSL_CONFIG_DATA_L6", "00000000"),
296             ("RSL_CONFIG_DATA_L7", "00000000"),
297             ("RSL_CONFIG_DATA_L8", "00000000"),
298             ("RSL_CONFIG_DATA_L9", "00000000"),
299             ("RSL_CONFIG_DATA_L10", "00000000"),
300             ("RSL_CONFIG_DATA_L11", "00000000"),
301             ("RSL_CONFIG_DATA_L12", "00000000"),
302             ("RSL_CONFIG_DATA_L13", "00000000"),
303             ("RSL_CONFIG_DATA_L14", "00000000"),
304             ("RSL_CONFIG_DATA_L15", "00000000"),
305             ("RSL_CONFIG_DATA_L16", "00000000"),
306             ("RSL_CONFIG_DATA_L17", "00000000"),
307             ("RSL_CONFIG_DATA_L18", "00000000"),
308             ("RSL_CONFIG_DATA_L19", "00000000"),
309             ("RSL_CONFIG_DATA_L20", "00000000"),
310             ("RSL_CONFIG_DATA_L21", "00000000"),
311             ("RSL_CONFIG_DATA_L22", "00000000"),
312             ("RSL_CONFIG_DATA_L23", "00000000"),
313             ("RSL_CONFIG_DATA_L24", "00000000"),
            ("RSL_CONFIG_DATA_L25", "00000000"),
            ("RSL_CONFIG_DATA_L26", "00000000"),
            ("RSL_CONFIG_DATA_L27", "00000000"),
            ("RSL_CONFIG_DATA_L28", "00000000"),
            ("RSL_CONFIG_DATA_L29", "00000000"),
            ("RSL_CONFIG_DATA_L30", "00000000"),
            ("RSL_CONFIG_DATA_L31", "00000000"),
            ("HST_TRIGGER_DELAY_DATA_LO", "00000000"),
            ("HST_TRIGGER_DELAY_DATA_HI", "00000000"),
            ("HST_PHI_DELAY_DATA_LO", "00000000"),
            ("HST_PHI_DELAY_DATA_HI", "00000000"),
            ("SLOWREADOFF_0", "0"),
            ("SLOWREADOFF_1", "0"),
        ]

```

8.2.3.7 setArbTiming()

```
def nsCamera.sensors.daedalus.daedalus.setArbTiming (
                self,
                side,
                sequence )
```

```
Args:
    side: Hemisphere 'A' or 'B'
    sequence: list of arbitrary timing intervals, beginning with initial delay.
      The conventional timing (5,2) with delay = 3 would be represented by
      [3,5,2,5,2,5,2,5].
    *WARNING* arbitrary timings will not be restored after a board power cycle

Returns:
    tuple (error string, 10-character hexadecimal representation of timing
      sequence)
```

Definition at line 612 of file daedalus.py.

```
612     def setArbTiming(self, side, sequence):
613         """
614         Args:
615             side: Hemisphere 'A' or 'B'
616             sequence: list of arbitrary timing intervals, beginning with initial delay.
617               The conventional timing (5,2) with delay = 3 would be represented by
618               [3,5,2,5,2,5,2,5].
619             *WARNING* arbitrary timings will not be restored after a board power cycle
620 
621         Returns:
622             tuple (error string, 10-character hexadecimal representation of timing
623               sequence)
624         """
625         if side is None:
626             side = "A"
627         if sequence is None:
628             sequence = [0, 3, 2, 3, 2, 3, 2, 3]
629 
630         logging.info(
631             self.loginfo + "HST side " + side.upper() + " (arbitrary): " + str(sequence)
632         )
633         if side.upper() == "A":
634             lowreg = "HS_TIMING_DATA_ALO"
635             highreg = "HS_TIMING_DATA_AHI"
636         elif side.upper() == "B":
637             lowreg = "HS_TIMING_DATA_BLO"
638             highreg = "HS_TIMING_DATA_BHI"
639         else:
640             err = (
641                 self.logerr
642                 + "Invalid sensor side: "
```

```
                + side
                + "; timing settings unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        full40 = [0] * 40
        bitlist = []
        flag = 0  # similar to setTiming, but starts with delay
        sequence = sequence[: (2 * self.nframes)]
        for a in sequence:
            add = [flag] * a
            bitlist += add
            if flag:
                flag = 0
            else:
                flag = 1
        reversedlist = bitlist[39::-1]
        full40[-(len(reversedlist) + 1) : -1] = reversedlist
        full40bin = "".join(str(x) for x in full40)
        full40hex = "%x" % int(full40bin, 2)
        highpart = full40hex[-10:-8].zfill(8)
        lowpart = full40hex[-8:].zfill(8)
        self.ca.setRegister(lowreg, lowpart)
        self.ca.setRegister(highreg, highpart)
        self.ca.setRegister("HS_TIMING_CTL", "00000001")
        # deactivates manual shutter mode if previously engaged
        self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
        actual = self.getTiming(side, actual=True)
        if actual != sequence:
            logging.warning(
                self.logwarn + "Due to sequence length, actual timing sequence "
                "for side "
                + side
                + " will be "
                + "{"
                + str(actual[0])
                + "}"
                + " "
                + str(actual[1 : 2 * self.nframes])
            )
        return actual

```

8.2.3.8 setHighFullWell()

```
def nsCamera.sensors.daedalus.daedalus.setHighFullWell (
                self,
                flag )
```

```
Activates High Full Well mode. All frames are acquired simultaneously. Zero Dead
  Time mode and interlacing will be automatically deactivated. NOTE: after
  deactivating HFW, the board remains in uninterlaced mode (interlacing = 0)

Args:
    flag: True to activate HFW mode, False to deactivate

Returns:
    Error message
```

Definition at line 393 of file daedalus.py.

```
393     def setHighFullWell(self, flag):
394         """
395         Activates High Full Well mode. All frames are acquired simultaneously. Zero Dead
396           Time mode and interlacing will be automatically deactivated. NOTE: after
397           deactivating HFW, the board remains in uninterlaced mode (interlacing = 0)
398 
399         Args:
```

```
400             flag: True to activate HFW mode, False to deactivate
401 
402         Returns:
403             Error message
404         """
405         err0 = ""
406         if flag:
407             if self.ZDT:
408                 logging.warning(
409                     self.logwarn + "ZDT mode will be disengaged because of HFW "
410                     "setting "
411                 )
412                 err0 = self.setZeroDeadTime(False)
413             err1, _ = self.ca.setSubregister("HFW", "1")
414             self.HFW = True
415             logging.info(self.loginfo + "High Full Well mode active")
416         else:
417             self.HFW = False
418             err1, _ = self.ca.setSubregister("HFW", "0")
419             self.setInterlacing(0)
420             logging.info(self.loginfo + "High Full Well mode inactivate")
421         err = err0 + err1
422         if err:
423             logging.error(self.logerr + "HFW option may not be set correctly ")
424         return err
425 
```

8.2.3.9 setInterlacing()

```
def nsCamera.sensors.daedalus.daedalus.setInterlacing (
                self,
                ifactor )
```

```
Sets interlacing factor. NOTE: if called directly when HFW or ZDT mode is
active, this will disengage those modes automatically.

Args:
    ifactor: number of interlaced lines (generates ifactor + 1 images per frame)
      defaults to 0 (no interlacing)

Returns:
    integer: active interlacing factor (unchanged if error)
```

Definition at line 329 of file daedalus.py.

```
329     def setInterlacing(self, ifactor):
330         """
331         Sets interlacing factor. NOTE: if called directly when HFW or ZDT mode is
332         active, this will disengage those modes automatically.
333 
334         Args:
335             ifactor: number of interlaced lines (generates ifactor + 1 images per frame)
336               defaults to 0 (no interlacing)
337 
338         Returns:
339             integer: active interlacing factor (unchanged if error)
340         """
341         if ifactor is None:
342             ifactor = 0
343         if (
344             not isinstance(ifactor, int)
345             or ifactor < 0
346             or ifactor > (self.maxheight - 1)
347         ):
348             err = (
349                 self.logerr + "invalid interlacing factor submitted. "
350                 "Interlacing remains unchanged. "
351             )
```

```
            logging.error(err)
            return self.interlacing
        if self.HFW:
            logging.warning(
                self.logwarn + "HFW mode will be disengaged because of new "
                "interlacing setting "
            )
            self.setHighFullWell(False)
        if self.ZDT:
            logging.warning(
                self.logwarn + "ZDT mode will be disengaged because of new "
                "interlacing setting "
            )
            self.setZeroDeadTime(False)
        if ifactor == 0:
            bitscheme = self.maxheight * [0]
        else:
            pattern = [0] + ifactor * [1]
            reps = 1 + self.maxheight // (ifactor + 1)
            bitscheme = (reps * pattern)[0 : self.maxheight]
        err = ""
        for a in range(32):
            rname = "RSL_CONFIG_DATA_R" + str(a)
            lname = "RSL_CONFIG_DATA_L" + str(a)
            regbits = bitscheme[32 * a : 32 * (a + 1)]
            # generated pattern is reverse order from placement in register (element 0
            #   of the list is the LSB of the register)
            bitsrev = regbits[::-1]
            s = [str(i) for i in bitsrev]
            b = "".join(s)  # assemble as binary number for processing
            hexval = "%x" % int(b, 2)
            val = hexval.zfill(8)
            err0, _ = self.ca.setRegister(rname, val)
            err1, _ = self.ca.setRegister(lname, val)
            err = err + err0 + err1
        if err:
            logging.error(self.logerr + "interlacing may not be set correctly: " + err)
        logging.info(self.loginfo + "Interlacing factor set to " + str(ifactor))
        self.interlacing = ifactor
        return self.interlacing
```

8.2.3.10 setManualShutters()

```
def nsCamera.sensors.daedalus.daedalus.setManualShutters (
                self,
                timing )
```

```
Dummy function; feature is not implemented on Daedalus

Returns:
    tuple (error string, dummy response string from final message)
```

Definition at line 752 of file daedalus.py.

```
752     def setManualShutters(self, timing):
753         """
754         Dummy function; feature is not implemented on Daedalus
755 
756         Returns:
757             tuple (error string, dummy response string from final message)
758         """
759         err = (
760             self.logerr + "manual shutter control is not implemented in the "
761             "Daedalus sensor "
762         )
763         logging.error(err)
764         return err, "00000000"
765 
```

8.2.3.11 setTiming()

```
def nsCamera.sensors.daedalus.daedalus.setTiming (
              self,
              side,
              sequence,
              delay )
```

```
Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
  fit into the 40-bit register space, then the actual timings may differ from
  those requested. If the timing sequence fits only once into register space
  (i.e., for a single frame, open + closed > 19 ns ), then actual timing will be
  (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
  actually result in (35,5) timing

Args:
    side: Hemisphere 'A' or 'B'
    sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
    delay: initial delay in ns

Returns:
    tuple (error string, 10-character hexadecimal representation of timing
      sequence)
```

Definition at line 493 of file daedalus.py.

```
    def setTiming(self, side, sequence, delay):
        """
        Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
          fit into the 40-bit register space, then the actual timings may differ from
          those requested. If the timing sequence fits only once into register space
          (i.e., for a single frame, open + closed > 19 ns ), then actual timing will be
          (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
          actually result in (35,5) timing

        Args:
            side: Hemisphere 'A' or 'B'
            sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
            delay: initial delay in ns

        Returns:
            tuple (error string, 10-character hexadecimal representation of timing
              sequence)
        """
        if side is None:
            side = "A"
        if sequence is None:
            sequence = (3, 2)
        if delay is None:
            delay = 0

        if len(sequence) != 2:
            err = (
                self.logerr
                + "Invalid sequence setting for side: "
                + side
                + "; timing settings are unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        logging.info(
            self.loginfo
            + "HST side "
            + side.upper()
            + ": "
            + str(sequence)
            + "; delay = "
            + str(delay)
        )
        if side.upper() == "A":
            lowreg = "HS_TIMING_DATA_ALO"
            highreg = "HS_TIMING_DATA_AHI"
        elif side.upper() == "B":
            lowreg = "HS_TIMING_DATA_BLO"
            highreg = "HS_TIMING_DATA_BHI"
        else:
            err = (
                self.logerr
                + "Invalid sensor side: "
                + side
                + "; timing settings unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        if (sequence[0] + sequence[1]) + delay > 40:
            err = (
                self.logerr + "Timing sequence is too long to be implemented; "
                "timing settings unchanged "
            )
            logging.error(err)
            return err, "0000000000"

        self.ca.senstiming[side.upper()] = (sequence, delay)
        self.ca.sensmanual = []  # clear manual settings from ca

        full40 = [0] * 40
        bitlist = []
        flag = 1
        sequence = sequence[:2]
        for a in sequence:
            add = [flag] * a
            bitlist += add
            if flag:
                flag = 0
            else:
                flag = 1
        # automatically truncates sequence to 39 characters
        reversedlist = bitlist[39::-1]
        repeats = (40 - delay) // len(reversedlist)
        if repeats > self.nframes:
            repeats = self.nframes
        # Pattern from sequence repeated to fit inside 40 bits up to a maximum of
        #   'nframes' times
        repeated = reversedlist * repeats
        if (len(repeated) + delay + 1) < 40 and repeats == self.nframes:
            # add 'stop' bit for ZDT mode if full sequence is less than the full 40 bits
            repeated = [1] + repeated
        full40[-(len(repeated) + delay + 1) : -(delay + 1)] = repeated
        full40bin = "".join(str(x) for x in full40)
        full40hex = "%x" % int(full40bin, 2)
        highpart = full40hex[-10:-8].zfill(8)
        lowpart = full40hex[-8:].zfill(8)
        err0, _ = self.ca.setRegister(lowreg, lowpart)
        err1, _ = self.ca.setRegister(highreg, highpart)
        err2, _ = self.ca.setRegister("HS_TIMING_CTL", "00000001")
        err = err0 + err1 + err2
        if err:
            logging.error(self.logerr + "Timing may not have been set correctly")
        if repeats < self.nframes:
            actual = self.getTiming(side, actual=True)
            expected = [delay] + 3 * list(sequence) + [sequence[0]]
            if actual != expected:
                logging.warning(
                    self.logwarn + "Warning: Due to sequence length, actual "
                    "timing sequence for side "
                    + side
                    + " will be "
                    + "{"
                    + str(actual[0])
                    + "}"
                    + " "
                    + str(actual[1 : 2 * self.nframes])
                )
        return err, full40hex

```

8.2.3.12 setTriggerDelay()

```
def nsCamera.sensors.daedalus.daedalus.setTriggerDelay (
              self,
              delayblocks )
```

```
NOTE: THIS IS BASED ON AN UNCERTAIN INTERPRETATION OF THE HDD

Args:
    delayblocks: number of 150 ps blocks to delay trigger (maximum of 38?)
```

Definition at line 468 of file daedalus.py.

```
468     def setTriggerDelay(self, delayblocks):
469         """
470         NOTE: THIS IS BASED ON AN UNCERTAIN INTERPRETATION OF THE HDD
471 
472         Args:
473             delayblocks: number of 150 ps blocks to delay trigger (maximum of 38?)
474         """
475         if not isinstance(delayblocks, int) or delayblocks < 0 or delayblocks > 38:
476             err = (
477                 self.logerr + "invalid trigger delay submitted. Delay remains "
478                 "unchanged. "
479             )
480             logging.error(err)
481             return err
482         delayseq = (38 - delayblocks) * [0] + delayblocks * [1] + [0, 1]
483         seqstr = "".join(str(x) for x in delayseq)
484         seqhex = "%x" % int(seqstr, 2)
485         highpart = seqhex[-10:-8].zfill(8)
486         lowpart = seqhex[-8:].zfill(8)
487         err0, _ = self.ca.setRegister("HST_TRIGGER_DELAY_DATA_LO", lowpart)
488         err1, _ = self.ca.setRegister("HST_TRIGGER_DELAY_DATA_HI", highpart)
489         err2, _ = self.ca.setRegister("HS_TIMING_CTL", "00000001")
490         delayed = delayblocks * 0.15
491         logging.info(self.loginfo + "Trigger delay = " + str(delayed) + " ns")
492 
```

8.2.3.13 setZeroDeadTime()

```
def nsCamera.sensors.daedalus.daedalus.setZeroDeadTime (
              self,
              flag )
```

```
Activates Zero Dead Time mode. Even rows follow the assigned HST schedule; odd
  rows are acquired while the 'shutter' for the even rows are closed. High Full
  Well mode and interlacing will be automatically deactivated.
NOTE: after deactivating ZDT, the board reverts to uninterlaced mode
  (interlacing = 0)

Args:
    flag: True to activate ZDT mode, False to deactivate

Returns:
    Error message
```

Definition at line 426 of file daedalus.py.

```
426     def setZeroDeadTime(self, flag):
427         """
428         Activates Zero Dead Time mode. Even rows follow the assigned HST schedule; odd
429           rows are acquired while the 'shutter' for the even rows are closed. High Full
430           Well mode and interlacing will be automatically deactivated.
431         NOTE: after deactivating ZDT, the board reverts to uninterlaced mode
432           (interlacing = 0)
433 
434         Args:
435             flag: True to activate ZDT mode, False to deactivate
436 
437         Returns:
438             Error message
439         """
440         err0 = ""
441         if flag:
442             if self.HFW:
443                 logging.warning(
444                     self.logwarn + "HFW mode will be disengaged because of ZDT "
445                     "setting "
446                 )
447                 err0 = self.setHighFullWell(False)
448             err1, _ = self.ca.setSubregister("ZDT_R", "1")
449             err2, _ = self.ca.setSubregister("ZDT_L", "1")
450             self.ZDT = False  # preclude ZDT deactivation message
451             self.setInterlacing(0)
452             self.interlacing = 1
453             self.ZDT = True
454             logging.info(
455                 self.loginfo + "Zero Dead Time mode active; actual interlacing = 1"
456             )
457         else:
458             self.ZDT = False
459             err1, _ = self.ca.setSubregister("ZDT_R", "0")
460             err2, _ = self.ca.setSubregister("ZDT_L", "0")
461             self.setInterlacing(0)
462             logging.info(self.loginfo + "Zero Dead Time mode inactivate")
463         err = err0 + err1 + err2
464         if err:
465             logging.error(self.logerr + "ZDT option may not be set correctly ")
466         return err
467 
```

8.2.4 Member Data Documentation

8.2.4.1 bytesperpixel

`nsCamera.sensors.daedalus.daedalus.bytesperpixel`

Definition at line 47 of file daedalus.py.

8.2.4.2 ca

`nsCamera.sensors.daedalus.daedalus.ca`

Definition at line 28 of file daedalus.py.

8.2.4.3 firstframe

```
nsCamera.sensors.daedalus.daedalus.firstframe
```

Definition at line 38 of file daedalus.py.

8.2.4.4 firstrow

```
nsCamera.sensors.daedalus.daedalus.firstrow
```

Definition at line 43 of file daedalus.py.

8.2.4.5 fpganumID

```
nsCamera.sensors.daedalus.daedalus.fpganumID
```

Definition at line 48 of file daedalus.py.

8.2.4.6 height

```
nsCamera.sensors.daedalus.daedalus.height
```

Definition at line 46 of file daedalus.py.

8.2.4.7 HFW

```
nsCamera.sensors.daedalus.daedalus.HFW
```

Definition at line 51 of file daedalus.py.

8.2.4.8 interlacing

```
nsCamera.sensors.daedalus.daedalus.interlacing
```

Definition at line 49 of file daedalus.py.

8.2.4.9 lastframe

```
nsCamera.sensors.daedalus.daedalus.lastframe
```

Definition at line 39 of file daedalus.py.

8.2.4.10 lastrow

```
nsCamera.sensors.daedalus.daedalus.lastrow
```

Definition at line 44 of file daedalus.py.

8.2.4.11 logcrit

```
nsCamera.sensors.daedalus.daedalus.logcrit
```

Definition at line 29 of file daedalus.py.

8.2.4.12 logdebug

```
nsCamera.sensors.daedalus.daedalus.logdebug
```

Definition at line 33 of file daedalus.py.

8.2.4.13 logerr

```
nsCamera.sensors.daedalus.daedalus.logerr
```

Definition at line 30 of file daedalus.py.

8.2.4.14 loginfo

```
nsCamera.sensors.daedalus.daedalus.loginfo
```

Definition at line 32 of file daedalus.py.

8.2.4.15 logwarn

```
nsCamera.sensors.daedalus.daedalus.logwarn
```

Definition at line 31 of file daedalus.py.

8.2.4.16 maxframe

```
nsCamera.sensors.daedalus.daedalus.maxframe
```

Definition at line 37 of file daedalus.py.

8.2.4.17 maxheight

```
nsCamera.sensors.daedalus.daedalus.maxheight
```

Definition at line 42 of file daedalus.py.

8.2.4.18 maxwidth

```
nsCamera.sensors.daedalus.daedalus.maxwidth
```

Definition at line 41 of file daedalus.py.

8.2.4.19 minframe

```
nsCamera.sensors.daedalus.daedalus.minframe
```

Definition at line 36 of file daedalus.py.

8.2.4.20 nframes

```
nsCamera.sensors.daedalus.daedalus.nframes
```

Definition at line 40 of file daedalus.py.

8.2.4.21 sens_registers

```
nsCamera.sensors.daedalus.daedalus.sens_registers
```

Definition at line 53 of file daedalus.py.

8.2.4.22 sens_subregisters

```
nsCamera.sensors.daedalus.daedalus.sens_subregisters
```

Definition at line 209 of file daedalus.py.

8.2.4.23 width

```
nsCamera.sensors.daedalus.daedalus.width
```

Definition at line 45 of file daedalus.py.

8.2.4.24 ZDT

```
nsCamera.sensors.daedalus.daedalus.ZDT
```

Definition at line 50 of file daedalus.py.

The documentation for this class was generated from the following file:

- nsCamera/sensors/daedalus.py

8.3 nsCamera.utils.GenTec.GenTec Class Reference

Public Member Functions

- def __init__ (self)
- def closeDevice (self)
- def sendSerial (self, ser, message, sleep=0.3)
- def ready (self)
- def GenTecReadTest (self)

Public Attributes

- serial

8.3.1 Detailed Description

Definition at line 25 of file GenTec.py.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 __init__()

```
def nsCamera.utils.GenTec.GenTec.__init__ (
                self )
```

Definition at line 26 of file GenTec.py.

```
    def __init__(self):
        self.serial = None
        ports = list(serial.tools.list_ports.comports())
        for p, desc, add in ports:
            try:
                ser = serial.Serial(
                    p,
                    115200,
                    parity=serial.PARITY_NONE,
                    timeout=0.01,
                    write_timeout=0.01,
                )
                resp = self.sendSerial(ser, "*VER")
                if "Maestro" in resp:
                    self.serial = ser
                    break
                # print (desc, add, resp)   # uncomment to see available ports
            except Exception as e:
                print(e)
        if not self.serial:
            print("Unable to contact a GenTec Device")

```

8.3.3 Member Function Documentation

8.3.3.1 closeDevice()

```
def nsCamera.utils.GenTec.GenTec.closeDevice (
                self )
```

Definition at line 48 of file GenTec.py.

```
    def closeDevice(self):
        self.serial.close()

```

8.3.3.2 GenTecReadTest()

```
def nsCamera.utils.GenTec.GenTec.GenTecReadTest (
              self )
```

Definition at line 60 of file GenTec.py.

```
60     def GenTecReadTest(self):
61         print(self.sendSerial(self.serial, "*VER"))
62         print("Press ctrl-c to stop read")
63         try:
64             while 1:
65                 time.sleep(1)
66                 if not "Not" in self.sendSerial(
67                     self.serial, "*NVU"
68                 ):  # skip when response is 'Not Available'
69                     print(self.sendSerial(self.serial, "*CVU"))
70         except KeyboardInterrupt:
71             print("\n --GenTecTest terminated--")
72             # self.serial.close()
73 
74 
75 """Command list, with response in []
76 "*SCS03" - set display range to index 03 (see manual p61 for indices) []
77 "*STL18.0" - set internal trigger level to 18% []
78 "*GTL" - get internal trigger level [2.0\r\n]
79 "*GMD" - get index of current display mode (see manual p65) [0\r\n]
80 "*CVU" - get current device reading [0.012\r\n]
81 "*NVU" - check if new data available [text response]
82 "*PWC01550" - set wavelength (interpolate for non-standard) to 1550nm (five digits) []
83 "*GWL" - get wavelength setting [1064\r\n]
84 "*VER" - get device info [MAESTRO Version 1.00.18\r\n]
85 "*STS" - query status [extended list, see p72]
86 "*ST2" - extended query status [extended list, see p74]
87 
88 see p58 for parsing joulemeters in binary
89 """
90 
```

8.3.3.3 ready()

```
def nsCamera.utils.GenTec.GenTec.ready (
              self )
```

Definition at line 57 of file GenTec.py.

```
57     def ready(self):
58         self.sendSerial(self.serial, "*CVU")  # should clear NVU in prep for new data
59 
```

8.3.3.4 sendSerial()

```
def nsCamera.utils.GenTec.GenTec.sendSerial (
              self,
              ser,
              message,
              sleep = 0.3 )
```

Definition at line 51 of file GenTec.py.

```
51     def sendSerial(self, ser, message, sleep=0.3):
52         ser.write(message)
53         time.sleep(sleep)
54         avail = ser.in_waiting
55         return ser.read(avail)
56 
```

8.3.4 Member Data Documentation

8.3.4.1 serial

```
nsCamera.utils.GenTec.GenTec.serial
```

Definition at line 27 of file GenTec.py.

The documentation for this class was generated from the following file:

- nsCamera/utils/GenTec.py

8.4 nsCamera.comms.GigE.GigE Class Reference

Classes

- class ZESTETM1_CARD_INFO

Public Member Functions

- def __init__ (self, camassem)
- def sendCMD (self, pkt)
- def arm (self, mode)
- def readoff (self, waitOnSRAM, timeout, fast)
- def writeSerial (self, outstring, timeout=None)
- def readSerial (self, size, timeout=None)
- def openDevice (self)
- def closeDevice (self)
- def getCardIP (self)
- def getCardInfo (self)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- mode
- writeTimeout
- readTimeout
- payloadsize
- skipError
- dport
- closecard
- CardInfo
- CardInfoP
- ZCountCards
- ZOpenConnection
- ZWriteData
- ZReadData
- Connection

Private Attributes

- _zest

8.4.1 Detailed Description

```
Code to manage Gigabit Ethernet connection to board. Each GigE object manages a
  single OT card; to use multiple cards, instantiate multiple cameraAssembler
  objects, each specifying the unique IPs of the corresponding OT card.

Note: Orange Tree card must be configured before use. See the README for details

Exposed methods:
    arm() - puts camera into wait state for external trigger
    readoff() - waits for data ready register flag, then copies camera image data
      into numpy arrays
    sendCMD(pkt) - sends packet object via serial port
    readSerial(size, timeout) - read 'size' bytes from connection
    writeSerial(outstring) - submits string 'outstring' over connection
    closeDevice() - close connections and free resources
    getCardIP() - returns IP address of OT card
    getCardInfo() - prints report of details of OT card and connection
```

Definition at line 26 of file GigE.py.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 __init__()

```
def nsCamera.comms.GigE.GigE.__init__ (
                self,
                camassem )
```

```
Args:
    camassem: parent cameraAssembler object
```

Definition at line 46 of file GigE.py.

```
46     def __init__(self, camassem):
47         """
48         Args:
49             camassem: parent cameraAssembler object
50         """
51         self.ca = camassem
52         self.logcrit = self.ca.logcritbase + "[GigE] "
53         self.logerr = self.ca.logerrbase + "[GigE] "
54         self.logwarn = self.ca.logwarnbase + "[GigE] "
55         self.loginfo = self.ca.loginfobase + "[GigE] "
56         self.logdebug = self.ca.logdebugbase + "[GigE] "
57         logging.info(self.loginfo + "initializing comms object")
58         self.mode = 1
59         self.writeTimeout = 10000
60         self.readTimeout = 10000
61         self.payloadsize = (
62             self.ca.sensor.width
63             * self.ca.sensor.height
64             * self.ca.sensor.nframes
65             * self.ca.sensor.bytesperpixel
66         )
67         self.skipError = False
68 
69         if self.ca.port:
70             if isinstance(self.ca.port, int) and 0 < self.ca.port < 65536:
71                 self.dport = self.ca.port
72             else:
73                 logging.error(
74                     self.logerr + "GigE: invalid port number supplied, defaulting to "
75                     "20482 "
76                 )
77                 self.dport = 20482
78         else:
79             self.dport = 20482  # default
80 
81         self.ca.port = self.dport
82 
83         if self.ca.arch == "64bit":
84             arch = "64"
85         else:
86             arch = "32"
87 
88         if self.ca.platform == "Windows":
89             lib_name = "ZestETM1.dll"
90         elif self.ca.platform == "Linux" or self.ca.platform == "Darwin":
91             lib_name = "libZestETM1.so"
92         else:
93             logging.warning(
94                 self.logwarn + "System does not self-identify as Linux, Windows, "
95                 "or Mac. Assuming posix-style libraries "
96             )
97             lib_name = "libZestETM1.so"
98 
99         self.closecard = False
100 
101         libpath = os.path.join(self.ca.packageroot, "comms", "ZestETM1", arch, lib_name)
102         self._zest = C.CDLL(libpath)
103 
104         self.CardInfo = self.ZESTETM1_CARD_INFO()
105         self.CardInfoP = C.pointer(self.CardInfo)
106 
107         # functions
108         self.ZCountCards = self._zest.ZestETM1CountCards
109         self.ZCountCards.argtypes = [
            C.POINTER(C.c_ulong),
            C.POINTER(C.POINTER(self.ZESTETM1_CARD_INFO)),
            C.c_int,
        ]

        self.ZOpenConnection = self._zest.ZestETM1OpenConnection
        self.ZOpenConnection.argtypes = [
            C.POINTER(self.ZESTETM1_CARD_INFO),
            C.c_int,
            C.c_ushort,
            C.c_ushort,
            C.POINTER(C.c_void_p),
        ]

        self.ZWriteData = self._zest.ZestETM1WriteData
        self.ZWriteData.argtypes = [
            C.c_void_p,
            C.c_void_p,
            C.c_ulong,
            C.POINTER(C.c_ulong),
            C.c_ulong,
        ]

        self.ZReadData = self._zest.ZestETM1ReadData
        self.ZReadData.argtypes = [
            C.c_void_p,
            C.c_void_p,
            C.c_ulong,
            C.POINTER(C.c_ulong),
            C.c_ulong,
        ]

        self.Connection = C.c_void_p()
        self.openDevice()

```

8.4.3 Member Function Documentation

8.4.3.1 arm()

```
def nsCamera.comms.GigE.GigE.arm (
              self,
              mode )
```

```
Puts camera into wait state for trigger. Mode determines source; arm() in
  CameraAssembler defaults to 'Hardware'

Args:
    mode:   'Software' activates software triggering, disables hardware trigger
    'Hardware  activates hardware triggering, disables software trigger
      Hardware is the default
    'Dual' activates dual edge hardware trigger mode and disables
      software trigger

Returns:
    tuple (error, response string)
```

Definition at line 184 of file GigE.py.

```
184     def arm(self, mode):
185         """
186         Puts camera into wait state for trigger. Mode determines source; arm() in
187           CameraAssembler defaults to 'Hardware'
```

```

        Args:
            mode:   ’Software’ activates software triggering, disables hardware trigger
                    ’Hardware  activates hardware triggering, disables software trigger
                      Hardware is the default
                    ’Dual’ activates dual edge hardware trigger mode and disables
                      software trigger

        Returns:
            tuple (error, response string)
        """
        if not mode:
            mode = "Hardware"
        logging.info(self.loginfo + "arm")
        self.ca.clearStatus()
        self.ca.latchPots()
        err, resp = self.ca.startCapture(mode)
        if err:
            logging.error(self.logerr + "unable to arm camera")
        else:
            self.ca.armed = True
            self.skipError = True
        return err, resp

```

8.4.3.2 closeDevice()

```
def nsCamera.comms.GigE.GigE.closeDevice (
              self )
```

```
Close connection to Orange Tree card and free resources
```

Definition at line 372 of file GigE.py.

```
372     def closeDevice(self):
373         """
374         Close connection to Orange Tree card and free resources
375         """
376         self._zest.ZestETM1CloseConnection(self.Connection)
377         if self.closecard:
378             try:
379                 self._zest.ZestETM1FreeCards(self.CardInfoP)
380             except:
381                 logging.error(self.logerr + "Error reported in OT card closure")
382         self._zest.ZestETM1Close()
383
```

8.4.3.3 getCardInfo()

```
def nsCamera.comms.GigE.GigE.getCardInfo (
              self )
```

```
Prints status message with information returned by OT card
```

Definition at line 392 of file GigE.py.

```
392     def getCardInfo(self):
393         """
394         Prints status message with information returned by OT card
395         """
396         ci = self.CardInfoP.contents
397         print("GigE Card Status:")
398         print("-------------")
399         print("IP: " + ".".join(str(e) for e in [b for b in ci.IPAddr]))
400         print("ControlPort: " + str(ci.ControlPort))
401         print("Timeout: " + str(ci.Timeout))
402         print("HTTPPort: " + str(ci.HTTPPort))
403         print("MACAddr: " + ".".join(format(e, "02X") for e in [b for b in ci.MACAddr]))
404         print("SubNet: " + ".".join(str(e) for e in [b for b in ci.SubNet]))
405         print("Gateway: " + ".".join(str(e) for e in [b for b in ci.Gateway]))
406         print("SerialNumber: " + str(ci.SerialNumber))
407         print("FirmwareVersion: " + str(ci.FirmwareVersion))
408         print("HardwareVersion: " + str(ci.HardwareVersion))
409         print("-------------")
410
```

8.4.3.4 getCardIP()

```
def nsCamera.comms.GigE.GigE.getCardIP (
              self )
```

```
Query IP address of OT card

Returns: address of OT card as list of bytes
```

Definition at line 384 of file GigE.py.

```
384     def getCardIP(self):
385         """
386         Query IP address of OT card
387
388         Returns: address of OT card as list of bytes
389         """
390         return self.CardInfo.IPAddr
391
```

8.4.3.5 openDevice()

```
def nsCamera.comms.GigE.GigE.openDevice (
              self )
```

```
Find Orange Tree card and open a connection; if ip is supplied as parameter for
  the CameraAssembler, bypass network search and connect directly to indicated
  IP address
```

Definition at line 307 of file GigE.py.

```
    def openDevice(self):
        """
        Find Orange Tree card and open a connection; if ip is supplied as parameter for
          the CameraAssembler, bypass network search and connect directly to indicated
          IP address
        """
        err = self._zest.ZestETM1Init()
        if err:
            logging.critical(self.logcrit + "ZestETM1Init failure")
            sys.exit(1)
        logging.info(self.loginfo + "searching for Orange Tree cards")
        NumCards = C.c_ulong(0)

        if self.ca.iplist:
            ubyte4 = C.c_ubyte * 4
            self.CardInfo.IPAddr = ubyte4(*self.ca.iplist)
            self.CardInfo.ControlPort = C.c_ushort(self.dport)
            self.CardInfo.Timeout = C.c_ulong(self.writeTimeout)
            self.closecard = False
        else:
            err = self.ZCountCards(C.byref(NumCards), C.byref(self.CardInfoP), 2000)
            self.closecard = True
            if err:
                logging.critical(self.logcrit + "CountCards failure")
                sys.exit(1)
            if NumCards.value == 0:
                self.ZCountCards(C.byref(NumCards), C.byref(self.CardInfoP), 3000)
                # try again with longer wait (e.g., after powerup)
                if NumCards.value == 0:
                    logging.info(self.loginfo + "trying to connect again, please wait")
                    self.ZCountCards(C.byref(NumCards), C.byref(self.CardInfoP), 5000)
                    if NumCards.value == 0:
                        logging.info(self.loginfo + "still trying to connect...")
                        self.ZCountCards(
                            C.byref(NumCards), C.byref(self.CardInfoP), 6000
                        )
                        if NumCards.value == 0:
                            self.ZCountCards(
                                C.byref(NumCards), C.byref(self.CardInfoP), 7000
                            )
                            if NumCards.value == 0:
                                self.ZCountCards(
                                    C.byref(NumCards), C.byref(self.CardInfoP), 7000
                                )
                                if NumCards.value == 0:
                                    logging.critical(
                                        self.logcrit + "no Orange Tree cards found"
                                    )
                                    sys.exit(1)
            else:
                logging.info(
                    self.loginfo
                    + ""
                    + str(NumCards.value)
                    + " Orange Tree card(s) found"
                )  # TODO: add check for GigE bit in board description
        err = self.ZOpenConnection(
            self.CardInfoP, 0, self.dport, 0, C.byref(self.Connection)
        )
        if err:
            logging.critical(
                self.logcrit + "OpenConnection failure, error #" + str(err)
            )
            sys.exit(1)

```

8.4.3.6 readoff()

```
def nsCamera.comms.GigE.GigE.readoff (
               self,
               waitOnSRAM,
```

timeout,

fast)

```
Copies image data from board into numpy arrays. The FPGA returns a packet
  without the CRC suffix

Args:
    waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
      data
    timeout: passed to waitForSRAM; after this many seconds begin copying data
      irrespective of SRAM_READY status; 'zero' means wait indefinitely
      WARNING: If acquisition fails, the SRAM will not contain a current image,
but the code will copy the data anyway
    fast: if False, parse and convert frames to numpy arrays; if True, return
      unprocessed text stream

Returns:
    tuple (list of numpy arrays OR raw text stream, length of downloaded payload
      in bytes, payload error flag) since CRC check is handled by TCP/IP,
      payload error flag is always False for GigE
```

Definition at line 212 of file GigE.py.

```
    def readoff(self, waitOnSRAM, timeout, fast):
        """
        Copies image data from board into numpy arrays. The FPGA returns a packet
          without the CRC suffix

        Args:
            waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
              data
            timeout: passed to waitForSRAM; after this many seconds begin copying data
              irrespective of SRAM_READY status; 'zero' means wait indefinitely
              WARNING: If acquisition fails, the SRAM will not contain a current image,
                but the code will copy the data anyway
            fast: if False, parse and convert frames to numpy arrays; if True, return
              unprocessed text stream

        Returns:
            tuple (list of numpy arrays OR raw text stream, length of downloaded payload
              in bytes, payload error flag) since CRC check is handled by TCP/IP,
              payload error flag is always False for GigE
        """
        logging.info(self.loginfo + "readoff")

        # Wait for data to be ready on board
        # Skip wait only if explicitly tagged 'False' ('None' defaults to True)
        if not waitOnSRAM==False:
            self.ca.waitForSRAM(timeout)
        self.skipError = False
        self.ca.oldtime = self.ca.currtime
        self.ca.currtime = time.time()
        self.ca.waited.append(self.ca.currtime - self.ca.oldtime)
        err, rval = self.ca.readSRAM()
        if err:
            logging.error(self.logerr + "Error detected in readSRAM")
        self.ca.oldtime = self.ca.currtime
        self.ca.currtime = time.time()
        self.ca.read.append(self.ca.currtime - self.ca.oldtime)
        # extract the data. Remove header; the FPGA returns a packet without the CRC
        #   suffix
        data = rval[32:]
        if fast:
            return data, len(data) // 2, bool(err)
        else:
            parsed = self.ca.generateFrames(data)
            return parsed, len(data) // 2, bool(err)

```

8.4.3.7 readSerial()

```
def nsCamera.comms.GigE.GigE.readSerial (
              self,
              size,
              timeout = None )
```

```
Read bytes from the serial port. Does not verify packets.

Args:
   size: number of bytes to read
   timeout: serial timeout in sec (defaults to self.readTimeout)

Returns:
   tuple (error string, string read from serial port)
```

Definition at line 281 of file GigE.py.

```
281     def readSerial(self, size, timeout=None):
282         """
283         Read bytes from the serial port. Does not verify packets.
284 
285         Args:
286            size: number of bytes to read
287            timeout: serial timeout in sec (defaults to self.readTimeout)
288 
289         Returns:
290            tuple (error string, string read from serial port)
291         """
292         if not timeout:
293             timeout = self.readTimeout
294         inbuff = C.create_string_buffer(size + 1)
295         inbuffp = C.pointer(inbuff)
296         readlen = C.c_ulong(0)
297         err = self.ZReadData(self.Connection, inbuffp, size, C.byref(readlen), timeout)
298         if err:
299             if self.skipError:
300                 self.skipError = False
301             else:
302                 logging.error(self.logerr + "readSerial error #" + str(err))
303             # 32768 = socket error, 32776 = timeout, see comms/ZestETM1/ZestETM1.h line
304             #   77 et seq.
305         return self.ca.bytes2str(inbuff.raw)[:-2]
306 
```

8.4.3.8 sendCMD()

```
def nsCamera.comms.GigE.GigE.sendCMD (
              self,
              pkt )
```

```
Submit packet and verify response packet
Packet communications with FPGA omit CRC suffix, so adds fake CRC bytes to
  response

Args:
    pkt: Packet object

Returns:
    tuple (error, response string)
```

Definition at line 145 of file GigE.py.

```
    def sendCMD(self, pkt):
        """
        Submit packet and verify response packet
        Packet communications with FPGA omit CRC suffix, so adds fake CRC bytes to
          response

        Args:
            pkt: Packet object

        Returns:
            tuple (error, response string)
        """
        pktStr = pkt.pktStr()[0:16]
        err = ""
        self.ca.writeSerial(pktStr)
        if (
            hasattr(self.ca, "board")
            and pktStr[4] == "0"
            and pktStr[5:8] == self.ca.board.registers["SRAM_CTL"]
        ):
            bufsize = self.payloadsize + 16
            resptext = self.readSerial(bufsize)
            if len(resptext) < bufsize + 16:
                err += (
                    self.logerr + "sendCMD- packet too small, payload may be incomplete"
                )
                logging.error(err)
        else:
            # add fake CRC to maintain consistency with other comms
            resp = self.readSerial(8)
            if len(resp) < 8:
                err += self.logerr + "sendCMD- response too small, returning zeros"
                resptext = "00000000000000000000"
                logging.error(err)
            else:
                resptext = resp + "0000"

        return err, resptext

```

8.4.3.9 writeSerial()

```
def nsCamera.comms.GigE.GigE.writeSerial (
                self,
                outstring,
                timeout = None )
```

```
Transmit string to board
Args:
    outstring: string to write
    timeout: serial timeout in sec (defaults to self.writeTimeout)

Returns:
    integer number of bytes written
```

Definition at line 257 of file GigE.py.

```
    def writeSerial(self, outstring, timeout=None):
        """
        Transmit string to board
        Args:
            outstring: string to write
            timeout: serial timeout in sec (defaults to self.writeTimeout)

        Returns:
            integer number of bytes written
        """
        if not timeout:
            timeout = self.writeTimeout
        outstring = self.ca.str2bytes(outstring)
        outbuff = C.create_string_buffer(outstring)
        outbuffp = C.pointer(outbuff)
        outbufflen = len(outstring)
        writelen = C.c_ulong(0)
        err = self.ZWriteData(
            self.Connection, outbuffp, outbufflen, C.byref(writelen), timeout
        )
        if err:
            logging.error(self.logerr + "writeSerial error #" + str(err))
        return writelen

```

8.4.4 Member Data Documentation

8.4.4.1 _zest

`nsCamera.comms.GigE.GigE._zest [private]`

Definition at line 102 of file GigE.py.

8.4.4.2 ca

`nsCamera.comms.GigE.GigE.ca`

Definition at line 51 of file GigE.py.

8.4.4.3 CardInfo

`nsCamera.comms.GigE.GigE.CardInfo`

Definition at line 104 of file GigE.py.

8.4.4.4 CardInfoP

`nsCamera.comms.GigE.GigE.CardInfoP`

Definition at line 105 of file GigE.py.

8.4.4.5 closecard

`nsCamera.comms.GigE.GigE.closecard`

Definition at line 99 of file GigE.py.

8.4.4.6 Connection

`nsCamera.comms.GigE.GigE.Connection`

Definition at line 142 of file GigE.py.

8.4.4.7 dport

`nsCamera.comms.GigE.GigE.dport`

Definition at line 71 of file GigE.py.

8.4.4.8 logcrit

`nsCamera.comms.GigE.GigE.logcrit`

Definition at line 52 of file GigE.py.

8.4.4.9 logdebug

`nsCamera.comms.GigE.GigE.logdebug`

Definition at line 56 of file GigE.py.

8.4.4.10 logerr

`nsCamera.comms.GigE.GigE.logerr`

Definition at line 53 of file GigE.py.

8.4.4.11 loginfo

`nsCamera.comms.GigE.GigE.loginfo`

Definition at line 55 of file GigE.py.

8.4.4.12 logwarn

`nsCamera.comms.GigE.GigE.logwarn`

Definition at line 54 of file GigE.py.

8.4.4.13 mode

`nsCamera.comms.GigE.GigE.mode`

Definition at line 58 of file GigE.py.

8.4.4.14 payloadsize

`nsCamera.comms.GigE.GigE.payloadsize`

Definition at line 61 of file GigE.py.

8.4.4.15 readTimeout

`nsCamera.comms.GigE.GigE.readTimeout`

Definition at line 60 of file GigE.py.

8.4.4.16 skipError

`nsCamera.comms.GigE.GigE.skipError`

Definition at line 67 of file GigE.py.

8.4.4.17 writeTimeout

```
nsCamera.comms.GigE.GigE.writeTimeout
```

Definition at line 59 of file GigE.py.

8.4.4.18 ZCountCards

```
nsCamera.comms.GigE.GigE.ZCountCards
```

Definition at line 108 of file GigE.py.

8.4.4.19 ZOpenConnection

```
nsCamera.comms.GigE.GigE.ZOpenConnection
```

Definition at line 115 of file GigE.py.

8.4.4.20 ZReadData

```
nsCamera.comms.GigE.GigE.ZReadData
```

Definition at line 133 of file GigE.py.

8.4.4.21 ZWriteData

```
nsCamera.comms.GigE.GigE.ZWriteData
```

Definition at line 124 of file GigE.py.

The documentation for this class was generated from the following file:

- nsCamera/comms/GigE.py

8.5 nsCamera.sensors.icarus.icarus Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def checkSensorVoltStat (self)
- def sensorSpecific (self)
- def setInterlacing (self, ifactor)
- def setHighFullWell (self, flag)
- def setZeroDeadTime (self, flag)
- def setTriggerDelay (self, delayblocks)
- def setTiming (self, side, sequence, delay)
- def setArbTiming (self, side, sequence)
- def getTiming (self, side, actual)
- def setManualShutters (self, timing)
- def getManualTiming (self)
- def parseReadoff (self, frames)
- def reportStatusSensor (self, statusbits)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- minframe
- maxframe
- firstframe
- lastframe
- nframes
- maxwidth
- maxheight
- firstrow
- lastrow
- width
- height
- bytesperpixel
- icarustype
- fpganumID
- interlacing
- sens_registers
- sens_subregisters

8.5.1 Detailed Description

Definition at line 26 of file icarus.py.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 __init__()

```
def nsCamera.sensors.icarus.icarus.__init__ (
                self,
                camassem )
```

Definition at line 27 of file icarus.py.

```
27     def __init__(self, camassem):
28         self.ca = camassem
29         self.logcrit = self.ca.logcritbase + "[Icarus] "
30         self.logerr = self.ca.logerrbase + "[Icarus] "
31         self.logwarn = self.ca.logwarnbase + "[Icarus] "
32         self.loginfo = self.ca.loginfobase + "[Icarus] "
33         self.logdebug = self.ca.logdebugbase + "[Icarus] "
34         logging.info(self.loginfo + "initializing sensor object")
35 
36         self.minframe = 1
37         self.maxframe = 2
38         self.firstframe = self.minframe
39         self.lastframe = self.maxframe
40         # WARNING: the camera will always 'acquire' four frames, but will only generate
41         #   images for the middle two; HST and manual shutters will manage all four
42         #   frames
43         self.nframes = self.maxframe - self.minframe + 1
44         self.maxwidth = 512
45         self.maxheight = 1024
46         self.firstrow = 0
47         self.lastrow = self.maxheight - 1
48         self.width = self.maxwidth
49         self.height = self.maxheight
50         self.bytesperpixel = 2
51         self.icarustype = 1  # 2-frame version
52         self.fpganumID = "1"  # last nybble of FPGA_NUM
53         self.interlacing = 0
54 
55         self.sens_registers = OrderedDict(
56             {
57                 "VRESET_WAIT_TIME": "03E",
58                 "ICARUS_VER_SEL": "041",
59                 "VRESET_HIGH_VALUE": "04A",
60                 "MISC_SENSOR_CTL": "04C",
61                 "MANUAL_SHUTTERS_MODE": "050",
62                 "W0_INTEGRATION": "051",
63                 "W0_INTERFRAME": "052",
64                 "W1_INTEGRATION": "053",
65                 "W1_INTERFRAME": "054",
66                 "W2_INTEGRATION": "055",
67                 "W2_INTERFRAME": "056",
68                 "W3_INTEGRATION": "057",
69                 "W0_INTEGRATION_B": "058",
70                 "W0_INTERFRAME_B": "059",
71                 "W1_INTEGRATION_B": "05A",
72                 "W1_INTERFRAME_B": "05B",
73                 "W2_INTEGRATION_B": "05C",
74                 "W2_INTERFRAME_B": "05D",
75                 "W3_INTEGRATION_B": "05E",
76                 "TIME_ROW_DCD": "05F",
77             }
78         )
79 
80         self.sens_subregisters = [
81             ("MANSHUT_MODE", "MANUAL_SHUTTERS_MODE", 0, 1, True),
82             ("STAT_W3TOPLEDGE1", "STAT_REG", 3, 1, False),
83             ("STAT_W3TOPREDGE1", "STAT_REG", 4, 1, False),
84             ("STAT_HST_ALL_W_EN_DETECTED", "STAT_REG", 12, 1, False),
85             ("REVREAD", "CTRL_REG", 4, 1, True),
86             ("PDBIAS_LOW", "CTRL_REG", 6, 1, True),
87             ("ROWDCD_CTL", "CTRL_REG", 7, 1, True),
            ("PDBIAS_UNREADY", "STAT_REG2", 5, 1, False),
            ("ACCUMULATION_CTL", "MISC_SENSOR_CTL", 0, 1, True),
            ("HST_TST_ANRST_EN", "MISC_SENSOR_CTL", 1, 1, True),
            ("HST_TST_BNRST_EN", "MISC_SENSOR_CTL", 2, 1, True),
            ("HST_TST_ANRST_IN", "MISC_SENSOR_CTL", 3, 1, True),
            ("HST_TST_BNRST_IN", "MISC_SENSOR_CTL", 4, 1, True),
            ("HST_PXL_RST_EN", "MISC_SENSOR_CTL", 5, 1, True),
            ("HST_CONT_MODE", "MISC_SENSOR_CTL", 6, 1, True),
            ("COL_DCD_EN", "MISC_SENSOR_CTL", 7, 1, True),
            ("COL_READOUT_EN", "MISC_SENSOR_CTL", 8, 1, True),
        ]

        if self.ca.boardname == "llnl_v1":
            self.sens_subregisters.append(
                ("VRESET_HIGH", "VRESET_HIGH_VALUE", 7, 8, True)
            )
        else:
            self.sens_subregisters.append(
                ("VRESET_HIGH", "VRESET_HIGH_VALUE", 15, 16, True)
            )

```

8.5.3 Member Function Documentation

8.5.3.1 checkSensorVoltStat()

def nsCamera.sensors.icarus.icarus.checkSensorVoltStat (
self)

```
Checks register tied to sensor select jumpers to confirm match with sensor
  object

Returns:
    boolean, True if jumpers select for Icarus sensor
```

Definition at line 109 of file icarus.py.

```
    def checkSensorVoltStat(self):
        """
        Checks register tied to sensor select jumpers to confirm match with sensor
          object

        Returns:
            boolean, True if jumpers select for Icarus sensor
        """
        err, status = self.ca.getSubregister("ICARUS_DET")
        if err:
            logging.error(self.logerr + "unable to confirm sensor status")
            return False
        if not int(status):
            logging.error(self.logerr + "Icarus sensor not detected")
            return False
        return True

```

8.5.3.2 getManualTiming()

```
def nsCamera.sensors.icarus.icarus.getManualTiming (
              self )
```

```
Read off manual shutter settings
Returns:
    list of 2 lists of timing from A and B sides, respectively
```

Definition at line 538 of file icarus.py.

```
    def getManualTiming(self):
        """
        Read off manual shutter settings
        Returns:
            list of 2 lists of timing from A and B sides, respectively
        """
        aside = []
        bside = []
        for reg in [
            "W0_INTEGRATION",
            "W0_INTERFRAME",
            "W1_INTEGRATION",
            "W1_INTERFRAME",
            "W2_INTEGRATION",
            "W2_INTERFRAME",
            "W3_INTEGRATION",
        ]:
            _, reghex = self.ca.getRegister(reg)
            aside.append(25 * int(reghex, 16))
        for reg in [
            "W0_INTEGRATION_B",
            "W0_INTERFRAME_B",
            "W1_INTEGRATION_B",
            "W1_INTERFRAME_B",
            "W2_INTEGRATION_B",
            "W2_INTERFRAME_B",
            "W3_INTEGRATION_B",
        ]:
            _, reghex = self.ca.getRegister(reg)
            bside.append(25 * int(reghex, 16))
        return [aside, bside]

```

8.5.3.3 getTiming()

```
def nsCamera.sensors.icarus.icarus.getTiming (
              self,
              side,
              actual )
```

```
actual = True: returns actual high speed intervals that will be generated by the
  FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2, open3]
actual = False: Returns high speed timing settings as set by setTiming. Assumes
  that timing was set via the setTiming method--it will not accurately report
  arbitrary timings set by direct register sets or manual shutter control.

Args:
    side: Hemisphere 'A' or 'B'
    actual: False: return HST settings
    True: calculate and return actual HST behavior

Returns:
    actual= False: tuple   (hemisphere label,
                    'open shutter' in ns,
                    'closed shutter' in ns,
                    initial delay in ns)
    True: list of relevant times [delay, open1, closed1, open2]
```

Definition at line 414 of file icarus.py.

```
    def getTiming(self, side, actual):
        """
        actual = True: returns actual high speed intervals that will be generated by the
          FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2, open3]
        actual = False: Returns high speed timing settings as set by setTiming. Assumes
          that timing was set via the setTiming method--it will not accurately report
          arbitrary timings set by direct register sets or manual shutter control.

        Args:
            side: Hemisphere ’A’ or ’B’
            actual: False: return HST settings
                    True: calculate and return actual HST behavior

        Returns:
            actual= False: tuple   (hemisphere label,
                                    ’open shutter’ in ns,
                                    ’closed shutter’ in ns,
                                    initial delay in ns)
                    True: list of relevant times [delay, open1, closed1, open2]
        """
        if side is None:
            side = "A"

        logging.info(self.loginfo + "get timing, side " + side.upper())
        if side.upper() == "A":
            lowreg = "HS_TIMING_DATA_ALO"
            highreg = "HS_TIMING_DATA_AHI"
        elif side.upper() == "B":
            lowreg = "HS_TIMING_DATA_BLO"
            highreg = "HS_TIMING_DATA_BHI"
        else:
            logging.error(
                self.logerr
                + "Invalid sensor side: "
                + side
                + "; timing settings unchanged"
            )
            return "", 0, 0, 0
        err, lowpart = self.ca.getRegister(lowreg)
        err1, highpart = self.ca.getRegister(highreg)
        if err or err1:
            logging.error(
                self.logerr + "Unable to retrieve timing setting (getTiming), "
                "returning zeroes "
            )
            return side.upper(), 0, 0, 0
        full40hex = highpart[-2:] + lowpart.zfill(8)
        full40bin = "{0:0=40b}".format(int(full40hex, 16))
        if actual:
            full160 = 4 * full40bin
            gblist = [[k, len(list(g))] for k, g in itertools.groupby(full160)]
            times = [int(x[1]) for x in gblist[:-9:-1]]
            times[0] = times[0] - 1
            # get timing for frames 1 and 2, keep delay as offset
            times12 = [times[0]] + times[3:6]
            return times12
        else:
            gblist = [[k, len(list(g))] for k, g in itertools.groupby(full40bin)]
            delay = gblist[-1][1] - 1
            timeon = gblist[-2][1]
            if len(gblist) == 2:  # 39,1 corner case
                timeoff = 1
            elif len(gblist) == 3:  # sequence fits only once
                timeoff = 40 - timeon
            else:
                timeoff = gblist[-3][1]
            return side.upper(), timeon, timeoff, delay

```

8.5.3.4 parseReadoff()

```
def nsCamera.sensors.icarus.icarus.parseReadoff (
               self,
               frames )
```

```
Dummy function; unnecessary for Icarus sensor
```

Definition at line 570 of file icarus.py.

```
570     def parseReadoff(self, frames):
571         """
572         Dummy function; unnecessary for Icarus sensor
573         """
574         return frames
575
```

8.5.3.5 reportStatusSensor()

```
def nsCamera.sensors.icarus.icarus.reportStatusSensor (
              self,
              statusbits )
```

```
Print status messages from sensor-specific bits of status register

Args:
    statusbits: result of checkStatus()
```

Definition at line 576 of file icarus.py.

```
576     def reportStatusSensor(self, statusbits):
577         """
578         Print status messages from sensor-specific bits of status register
579
580         Args:
581             statusbits: result of checkStatus()
582         """
583         if int(statusbits[3]):
584             logging.info(self.loginfo + "W3_Top_L_Edge1 detected")
585         if int(statusbits[4]):
586             logging.info(self.loginfo + "W3_Top_R_Edge1 detected")
587         if int(statusbits[12]):
588             logging.info(self.loginfo + "HST_All_W_En detected")
589
590
591 """
592 Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
593 LLNL-CODE-838080
594
595 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
596 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
597 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
598 See license for disclaimers, notice of U.S. Government Rights and license terms and
599 conditions.
600 """
```

8.5.3.6 sensorSpecific()

```
def nsCamera.sensors.icarus.icarus.sensorSpecific (
              self )
```

```
Returns:
    list of tuples, (Sensor-specific register, default setting)
```

Definition at line 126 of file icarus.py.

```
    def sensorSpecific(self):
        """
        Returns:
            list of tuples, (Sensor-specific register, default setting)
        """
        icarussettings = [
            ("ICARUS_VER_SEL", "00000001"),
            ("FPA_FRAME_INITIAL", "00000001"),
            ("FPA_FRAME_FINAL", "00000002"),
            ("FPA_ROW_INITIAL", "00000000"),
            ("FPA_ROW_FINAL", "000003FF"),
            ("VRESET_WAIT_TIME", "000927C0"),
            ("HS_TIMING_DATA_BHI", "00000000"),
            ("HS_TIMING_DATA_BLO", "00006666"),  # 0db6 = 2-1; 6666 = 2-2
            ("HS_TIMING_DATA_AHI", "00000000"),
            ("HS_TIMING_DATA_ALO", "00006666"),
        ]
        if self.ca.boardname == "llnl_v1":
            icarussettings.append(
                ("VRESET_HIGH_VALUE", "000000D5")  # 3.3 V (FF = 3.96)
            )
        else:
            icarussettings.append(("VRESET_HIGH_VALUE", "0000FFFF"))

        return icarussettings

```

8.5.3.7 setArbTiming()

```
def nsCamera.sensors.icarus.icarus.setArbTiming (
              self,
              side,
              sequence )
```

```
Set arbitrary high-speed timing sequence. NOTE: Icarus sensors generally cannot
  use 1 ns timing, so should be at least 2 ns for frame 2 wnd 3 open and their
  interframe
Args:
    side: Hemisphere 'A' or 'B'
    sequence: list of arbitrary timing intervals, beginning with initial delay.
      The conventional timing (5,2) with delay = 3 would be represented by
      [3,5,2,5,2,5,2,5]. NOTE: although Icarus only images the middle two
      frames, timing must be provided for all four frames; to implement frame 1
      open for X, shutter closed for Y, and frame 2 open for Z, use the
      seequence [0,1,1,X,Y,Z,1,1]
    *WARNING* arbitrary timings will not be restored after a board power cycle

Returns:
    list: Actual timing results
```

Definition at line 323 of file icarus.py.

```
    def setArbTiming(self, side, sequence):
        """
        Set arbitrary high-speed timing sequence. NOTE: Icarus sensors generally cannot
          use 1 ns timing, so should be at least 2 ns for frame 2 wnd 3 open and their
          interframe
        Args:
            side: Hemisphere 'A' or 'B'
            sequence: list of arbitrary timing intervals, beginning with initial delay.
              The conventional timing (5,2) with delay = 3 would be represented by
              [3,5,2,5,2,5,2,5]. NOTE: although Icarus only images the middle two
              frames, timing must be provided for all four frames; to implement frame 1
              open for X, shutter closed for Y, and frame 2 open for Z, use the
              seequence [0,1,1,X,Y,Z,1,1]
            *WARNING* arbitrary timings will not be restored after a board power cycle

        Returns:
            list: Actual timing results
        """
        if side is None:
            side = "A"
        if sequence is None:
            sequence = [0, 3, 2, 3, 2, 3, 2, 3]

        logging.info(
            self.loginfo + "HST side " + side.upper() + " (arbitrary): " + str(sequence)
        )
        if side.upper() == "A":
            lowreg = "HS_TIMING_DATA_ALO"
            highreg = "HS_TIMING_DATA_AHI"
        elif side.upper() == "B":
            lowreg = "HS_TIMING_DATA_BLO"
            highreg = "HS_TIMING_DATA_BHI"
        else:
            err = (
                self.logerr
                + "Invalid sensor side: "
                + side
                + "; timing settings unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        full40 = [0] * 40
        bitlist = []
        flag = 0  # similar to setTiming, but starts with delay
        sequence = sequence[:8]  # need all 4 frames to work properly
        for a in sequence:
            add = [flag] * a
            bitlist += add
            if flag:
                flag = 0
            else:
                flag = 1
        reversedlist = bitlist[39::-1]
        full40[-(len(reversedlist) + 1) : -1] = reversedlist
        full40bin = "".join(str(x) for x in full40)
        full40hex = "%x" % int(full40bin, 2)
        highpart = full40hex[-10:-8].zfill(8)
        lowpart = full40hex[-8:].zfill(8)
        self.ca.setRegister(lowreg, lowpart)
        self.ca.setRegister(highreg, highpart)
        self.ca.setRegister("HS_TIMING_CTL", "00000001")
        # deactivates manual shutter mode if previously engaged
        self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
        actual = self.getTiming(side, actual=True)
        f0delay = sequence[1] + sequence[2]
        if actual != sequence[:1] + sequence[3:6]:
            logging.warning(
                self.logwarn + "Due to sequence length and use of the Icarus model "
                "1 sensor, the actual timing sequence for side "
                + side
                + " will be "
                + "{"
                + str(actual[0] + f0delay)
                + "}"
                + " "
                + str(actual[1 : 2 * self.nframes])
            )
        else:
            logging.warning(
                self.logwarn + "Due to use of the Icarus model 1 sensor, the actual"
                " timing sequence for side "
                + side
                + " will be "
                + "{"
                + str(actual[0] + f0delay)
                + "}"
                + " "
                + str(actual[1 : 2 * self.nframes])
            )
        return actual

```

8.5.3.8 setHighFullWell()

```
def nsCamera.sensors.icarus.icarus.setHighFullWell (
              self,
              flag )
```

```
Dummy function; feature is not implemented on Icarus
```

Definition at line 165 of file icarus.py.

```
165     def setHighFullWell(self, flag):
166         """
167         Dummy function; feature is not implemented on Icarus
168         """
169         if flag:
170             logging.warning(
171                 self.logwarn + "HighFullWell mode is not supported by the Icarus "
172                 "sensor. "
173             )
174
```

8.5.3.9 setInterlacing()

```
def nsCamera.sensors.icarus.icarus.setInterlacing (
              self,
              ifactor )
```

```
Dummy function; feature is not implemented on Icarus2

Returns:
    integer 1
```

Definition at line 152 of file icarus.py.

```
152     def setInterlacing(self, ifactor):
153         """
154         Dummy function; feature is not implemented on Icarus2
155
156         Returns:
157             integer 1
158         """
159         if ifactor:
160             logging.warning(
161                 self.logwarn + "Interlacing is not supported by the Icarus sensor."
162             )
163         return 1
164
```

8.5.3.10 setManualShutters()

```
def nsCamera.sensors.icarus.icarus.setManualShutters (
              self,
              timing )
```

```
Manual shutter timing, seven intervals for each side of the imager given in
  nanoseconds, e.g., [(100,50,100,50,100,50,100),(100,50,100,50,100,50,100)]

The timing list is flattened before processing; the suggested tuple structure is
  just for clarity (first tuple is A, second is B) and is optional.

The actual timing is rounded down to nearest multiple of 25 ns. (Each
  count = 25 ns. e.g., 140 ns rounds down to a count of ’5’ which corresponds
  to 125 ns))

Args:
    timing: 14-element list (substructure optional) in nanoseconds

Returns:
    tuple (error string, response string from final message)
```

Definition at line 482 of file icarus.py.

```
482     def setManualShutters(self, timing):
483         """
484         Manual shutter timing, seven intervals for each side of the imager given in
485           nanoseconds, e.g., [(100,50,100,50,100,50,100),(100,50,100,50,100,50,100)]
486 
487         The timing list is flattened before processing; the suggested tuple structure is
488           just for clarity (first tuple is A, second is B) and is optional.
489 
490         The actual timing is rounded down to nearest multiple of 25 ns. (Each
491           count = 25 ns. e.g., 140 ns rounds down to a count of ’5’ which corresponds
492           to 125 ns))
493 
494         Args:
495             timing: 14-element list (substructure optional) in nanoseconds
496 
497         Returns:
498             tuple (error string, response string from final message)
499 
500         """
501         if timing is None:
502             timing = [
503                 (100, 50, 100, 50, 100, 50, 100),
504                 (100, 50, 100, 50, 100, 50, 100),
505             ]
506 
507         logging.info(self.loginfo + "Manual shutter sequence: " + str(timing))
508         flattened = self.ca.flatten(timing)
509         if len(flattened) != 14 or not all(type(x) is int for x in flattened):
510             err = self.logerr + "Invalid manual shutter timing list: " + str(timing)
511             logging.error(err + "; timing settings unchanged")
512             return err, "00000000"
513 
514         timecounts = [a // 25 for a in flattened]
515         self.ca.sensmanual = timing
516         self.ca.senstiming = {}  # clear HST settings from ca object
517 
518         control_messages = [
519             ("W0_INTEGRATION", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
520             ("W0_INTERFRAME", "{0:#0{1}x}".format(timecounts[1], 10)[2:10]),
521             ("W1_INTEGRATION", "{0:#0{1}x}".format(timecounts[2], 10)[2:10]),
522             ("W1_INTERFRAME", "{0:#0{1}x}".format(timecounts[3], 10)[2:10]),
523             ("W2_INTEGRATION", "{0:#0{1}x}".format(timecounts[4], 10)[2:10]),
524             ("W2_INTERFRAME", "{0:#0{1}x}".format(timecounts[5], 10)[2:10]),
525             ("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[6], 10)[2:10]),
526             ("W0_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[7], 10)[2:10]),
527             ("W0_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[8], 10)[2:10]),
528             ("W1_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[9], 10)[2:10]),
529             ("W1_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[10], 10)[2:10]),
530             ("W2_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[11], 10)[2:10]),
            ("W2_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[12], 10)[2:10]),
            ("W3_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[13], 10)[2:10]),
            ("HS_TIMING_CTL", "00000000"),
            ("MANUAL_SHUTTERS_MODE", "00000001"),
        ]
        return self.ca.submitMessages(control_messages, " setManualShutters: ")

```

8.5.3.11 setTiming()

```
def nsCamera.sensors.icarus.icarus.setTiming (
                self,
                side,
                sequence,
                delay )
```

```
Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
  fit into the 40-bit register space, then the actual timings may differ from
  those requested. If the timing sequence fits only once into register space
  (i.e., for a single frame, open + closed > 19 ns ), then actual timing will be
  (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
  actually result in (35,5) timing
NOTE: Icarus sensors generally cannot use 1 ns timing, so all values (besides the
  delay) should be at least 2 ns
Args:
    side: Hemisphere 'A' or 'B'
    sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
    delay: initial delay in ns

Returns:
    tuple (error string, 10-character hexadecimal representation of timing
sequence)
```

Definition at line 195 of file icarus.py.

```
    def setTiming(self, side, sequence, delay):
        """
        Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
          fit into the 40-bit register space, then the actual timings may differ from
          those requested. If the timing sequence fits only once into register space
          (i.e., for a single frame, open + closed > 19 ns ), then actual timing will be
          (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
          actually result in (35,5) timing
        NOTE: Icarus sensors generally cannot use 1 ns timing, so all values (besides the
          delay) should be at least 2 ns
        Args:
            side: Hemisphere 'A' or 'B'
            sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
            delay: initial delay in ns

        Returns:
            tuple (error string, 10-character hexadecimal representation of timing
                sequence)
        """
        if side is None:
            side = "A"
        if sequence is None:
            sequence = (3, 2)
        if delay is None:
            delay = 0

        if len(sequence) != 2:
            err = (
                self.logerr
                + "Invalid sequence setting for side: "
                + side
                + "; timing settings are unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        logging.info(
            self.loginfo
            + "HST side "
            + side.upper()
            + ": "
            + str(sequence)
            + "; delay = "
            + str(delay)
        )
        if side.upper() == "A":
            lowreg = "HS_TIMING_DATA_ALO"
            highreg = "HS_TIMING_DATA_AHI"
        elif side.upper() == "B":
            lowreg = "HS_TIMING_DATA_BLO"
            highreg = "HS_TIMING_DATA_BHI"
        else:
            err = (
                self.logerr
                + "Invalid sensor side: "
                + side
                + "; timing settings unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        if (sequence[0] + sequence[1]) + delay > 40:
            err = (
                self.logerr + "Timing sequence is too long to be implemented; "
                "timing settings unchanged "
            )
            logging.error(err)
            return err, "0000000000"

        self.ca.senstiming[side.upper()] = (sequence, delay)
        self.ca.sensmanual = []  # clear manual settings from ca

        full40 = [0] * 40
        bitlist = []
        flag = 1
        sequence = sequence[:2]
        for a in sequence:
            add = [flag] * a
            bitlist += add
            if flag:
                flag = 0
            else:
                flag = 1
            # automatically truncates sequence to 39 characters
            reversedlist = bitlist[39::-1]
        repeats = (40 - delay) // len(reversedlist)
        # all four frames must be managed, even though only two are acquired
        if repeats > 4:
            repeats = 4
        # Pattern from sequence repeated to fit inside 40 bits up to a maximum of
        #   ’nframes’ times
        repeated = reversedlist * repeats
        full40[-(len(repeated) + delay + 1) : -(delay + 1)] = repeated
        full40bin = "".join(str(x) for x in full40)
        full40hex = "%x" % int(full40bin, 2)
        highpart = full40hex[-10:-8].zfill(8)
        lowpart = full40hex[-8:].zfill(8)
        self.ca.setRegister(lowreg, lowpart)
        self.ca.setRegister(highreg, highpart)
        self.ca.setRegister("HS_TIMING_CTL", "00000001")
        # deactivates manual shutter mode if previously engaged
        self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
        f0delay = sequence[0] + sequence[1]
        if repeats < 4:
            actual = self.getTiming(side, actual=True)
            expected = [delay] + list(sequence) + [sequence[0]]
            if actual != expected:
                logging.warning(
                    self.logwarn + "Due to sequence length and use of the Icarus "
                    "model 1 sensor, the actual timing sequence for side "
                    + side
                    + " will be "
                    + "{"
                    + str(actual[0] + f0delay)
```

```
307                     + "}"
308                     + " "
309                     + str(actual[1 : 2 * self.nframes])
310                 )
311         else:
312             logging.warning(
313                 self.logwarn + "Due to use of the Icarus model 1 sensor, the"
314                 " initial delay for side "
315                 + side
316                 + " will actually be "
317                 + str(delay + f0delay)
318                 + " nanoseconds"
319             )
320 
321         return "", full40hex
322 
```

8.5.3.12 setTriggerDelay()

```
def nsCamera.sensors.icarus.icarus.setTriggerDelay (
                self,
                delayblocks )
```

```
Dummy function; feature is not implemented on Icarus
```

Definition at line 185 of file icarus.py.

```
185     def setTriggerDelay(self, delayblocks):
186         """
187         Dummy function; feature is not implemented on Icarus
188         """
189         if delayblocks:
190             logging.warning(
191                 self.logwarn + "Trigger Delay mode is not supported by the Icarus "
192                 "sensor. "
193             )
194 
```

8.5.3.13 setZeroDeadTime()

```
def nsCamera.sensors.icarus.icarus.setZeroDeadTime (
                self,
                flag )
```

```
Dummy function; feature is not implemented on Icarus
```

Definition at line 175 of file icarus.py.

```
175     def setZeroDeadTime(self, flag):
176         """
177         Dummy function; feature is not implemented on Icarus
178         """
179         if flag:
180             logging.warning(
181                 self.logwarn + "ZeroDeadTime mode is not supported by the Icarus "
182                 "sensor. "
183             )
184 
```

8.5.4 Member Data Documentation

8.5.4.1 bytesperpixel

```
nsCamera.sensors.icarus.icarus.bytesperpixel
```

Definition at line 50 of file icarus.py.

8.5.4.2 ca

```
nsCamera.sensors.icarus.icarus.ca
```

Definition at line 28 of file icarus.py.

8.5.4.3 firstframe

```
nsCamera.sensors.icarus.icarus.firstframe
```

Definition at line 38 of file icarus.py.

8.5.4.4 firstrow

```
nsCamera.sensors.icarus.icarus.firstrow
```

Definition at line 46 of file icarus.py.

8.5.4.5 fpganumID

```
nsCamera.sensors.icarus.icarus.fpganumID
```

Definition at line 52 of file icarus.py.

8.5.4.6 height

```
nsCamera.sensors.icarus.icarus.height
```

Definition at line 49 of file icarus.py.

8.5.4.7 icarustype

```
nsCamera.sensors.icarus.icarus.icarustype
```

Definition at line 51 of file icarus.py.

8.5.4.8 interlacing

```
nsCamera.sensors.icarus.icarus.interlacing
```

Definition at line 53 of file icarus.py.

8.5.4.9 lastframe

```
nsCamera.sensors.icarus.icarus.lastframe
```

Definition at line 39 of file icarus.py.

8.5.4.10 lastrow

```
nsCamera.sensors.icarus.icarus.lastrow
```

Definition at line 47 of file icarus.py.

8.5.4.11 logcrit

```
nsCamera.sensors.icarus.icarus.logcrit
```

Definition at line 29 of file icarus.py.

8.5.4.12 logdebug

```
nsCamera.sensors.icarus.icarus.logdebug
```

Definition at line 33 of file icarus.py.

8.5.4.13 logerr

```
nsCamera.sensors.icarus.icarus.logerr
```

Definition at line 30 of file icarus.py.

8.5.4.14 loginfo

```
nsCamera.sensors.icarus.icarus.loginfo
```

Definition at line 32 of file icarus.py.

8.5.4.15 logwarn

```
nsCamera.sensors.icarus.icarus.logwarn
```

Definition at line 31 of file icarus.py.

8.5.4.16 maxframe

```
nsCamera.sensors.icarus.icarus.maxframe
```

Definition at line 37 of file icarus.py.

8.5.4.17 maxheight

```
nsCamera.sensors.icarus.icarus.maxheight
```

Definition at line 45 of file icarus.py.

8.5.4.18 maxwidth

```
nsCamera.sensors.icarus.icarus.maxwidth
```

Definition at line 44 of file icarus.py.

8.5.4.19 minframe

```
nsCamera.sensors.icarus.icarus.minframe
```

Definition at line 36 of file icarus.py.

8.5.4.20 nframes

```
nsCamera.sensors.icarus.icarus.nframes
```

Definition at line 43 of file icarus.py.

8.5.4.21 sens_registers

```
nsCamera.sensors.icarus.icarus.sens_registers
```

Definition at line 55 of file icarus.py.

8.5.4.22 sens_subregisters

```
nsCamera.sensors.icarus.icarus.sens_subregisters
```

Definition at line 80 of file icarus.py.

8.5.4.23 width

```
nsCamera.sensors.icarus.icarus.width
```

Definition at line 48 of file icarus.py.

The documentation for this class was generated from the following file:

- nsCamera/sensors/icarus.py

8.6 nsCamera.sensors.icarus2.icarus2 Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def checkSensorVoltStat (self)
- def sensorSpecific (self)
- def setInterlacing (self, ifactor)
- def setHighFullWell (self, flag)
- def setZeroDeadTime (self, flag)
- def setTriggerDelay (self, delayblocks)
- def setTiming (self, side, sequence, delay)
- def setArbTiming (self, side, sequence)
- def getTiming (self, side, actual)
- def setManualShutters (self, timing)
- def getManualTiming (self)
- def parseReadoff (self, frames)
- def reportStatusSensor (self, statusbits)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- minframe
- maxframe
- firstframe
- lastframe
- nframes
- maxwidth
- maxheight
- firstrow
- lastrow
- width
- height
- bytesperpixel
- icarustype
- fpganumID
- interlacing
- sens_registers
- sens_subregisters

8.6.1 Detailed Description

Definition at line 24 of file icarus2.py.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 __init__()

```
def nsCamera.sensors.icarus2.icarus2.__init__ (
               self,
               camassem )
```

Definition at line 25 of file icarus2.py.

```
    def __init__(self, camassem):
        self.ca = camassem
        self.logcrit = self.ca.logcritbase + "[Icarus2] "
        self.logerr = self.ca.logerrbase + "[Icarus2] "
        self.logwarn = self.ca.logwarnbase + "[Icarus2] "
        self.loginfo = self.ca.loginfobase + "[Icarus2] "
        self.logdebug = self.ca.logdebugbase + "[Icarus2] "
        logging.info(self.loginfo + "initializing sensor object")
        self.minframe = 0
        self.maxframe = 3
        self.firstframe = self.minframe
        self.lastframe = self.maxframe
        self.nframes = self.maxframe - self.minframe + 1
        self.maxwidth = 512
        self.maxheight = 1024
        self.firstrow = 0
        self.lastrow = self.maxheight - 1
        self.width = self.maxwidth
        self.height = self.maxheight
        self.bytesperpixel = 2
        self.icarustype = 0  # 4-frame version
        self.fpganumID = "1"  # last nybble of FPGA_NUM
        self.interlacing = 0

        self.sens_registers = OrderedDict(
            {
                "VRESET_WAIT_TIME": "03E",
                "ICARUS_VER_SEL": "041",
                "MISC_SENSOR_CTL": "04C",
                "MANUAL_SHUTTERS_MODE": "050",
                "W0_INTEGRATION": "051",
                "W0_INTERFRAME": "052",
                "W1_INTEGRATION": "053",
                "W1_INTERFRAME": "054",
                "W2_INTEGRATION": "055",
                "W2_INTERFRAME": "056",
                "W3_INTEGRATION": "057",
                "W0_INTEGRATION_B": "058",
                "W0_INTERFRAME_B": "059",
                "W1_INTEGRATION_B": "05A",
                "W1_INTERFRAME_B": "05B",
                "W2_INTEGRATION_B": "05C",
                "W2_INTERFRAME_B": "05D",
                "W3_INTEGRATION_B": "05E",
                "TIME_ROW_DCD": "05F",
            }
        )

        self.sens_subregisters = [
            ("MANSHUT_MODE", "MANUAL_SHUTTERS_MODE", 0, 1, True),
            ("STAT_W3TOPLEDGE1", "STAT_REG", 3, 1, False),
            ("STAT_W3TOPREDGE1", "STAT_REG", 4, 1, False),
            ("STAT_HST_ALL_W_EN_DETECTED", "STAT_REG", 12, 1, False),
            ("REVREAD", "CTRL_REG", 4, 1, True),
            ("PDBIAS_UNREADY", "STAT_REG2", 5, 1, False),
            ("PDBIAS_LOW", "CTRL_REG", 6, 1, True),
            ("ROWDCD_CTL", "CTRL_REG", 7, 1, True),
            ("ACCUMULATION_CTL", "MISC_SENSOR_CTL", 0, 1, True),
            ("HST_TST_ANRST_EN", "MISC_SENSOR_CTL", 1, 1, True),
            ("HST_TST_BNRST_EN", "MISC_SENSOR_CTL", 2, 1, True),
            ("HST_TST_ANRST_IN", "MISC_SENSOR_CTL", 3, 1, True),
86               ("HST_TST_BNRST_IN", "MISC_SENSOR_CTL", 4, 1, True),
87               ("HST_PXL_RST_EN", "MISC_SENSOR_CTL", 5, 1, True),
88               ("HST_CONT_MODE", "MISC_SENSOR_CTL", 6, 1, True),
89               ("COL_DCD_EN", "MISC_SENSOR_CTL", 7, 1, True),
90               ("COL_READOUT_EN", "MISC_SENSOR_CTL", 8, 1, True),
91           ]
92
```

8.6.3 Member Function Documentation

8.6.3.1 checkSensorVoltStat()

```
def nsCamera.sensors.icarus2.icarus2.checkSensorVoltStat (
              self )
```

```
Checks register tied to sensor select jumpers to confirm match with sensor
object

Returns:
    boolean, True if jumpers select for Icarus sensor
```

Definition at line 93 of file icarus2.py.

```
93      def checkSensorVoltStat(self):
94          """
95          Checks register tied to sensor select jumpers to confirm match with sensor
96          object
97
98          Returns:
99              boolean, True if jumpers select for Icarus sensor
100         """
101         err, status = self.ca.getSubregister("ICARUS_DET")
102         if err:
103             logging.error(self.logerr + "unable to confirm sensor status")
104             return False
105         if not int(status):
106             logging.error(self.logerr + "Icarus sensor not detected")
107             return False
108         return True
109
```

8.6.3.2 getManualTiming()

```
def nsCamera.sensors.icarus2.icarus2.getManualTiming (
              self )
```

```
Read off manual shutter settings
Returns:
    list of 2 lists of timing from A and B sides, respectively
```

Definition at line 485 of file icarus2.py.

```
    def getManualTiming(self):
        """
        Read off manual shutter settings
        Returns:
            list of 2 lists of timing from A and B sides, respectively
        """
        aside = []
        bside = []
        for reg in [
            "W0_INTEGRATION",
            "W0_INTERFRAME",
            "W1_INTEGRATION",
            "W1_INTERFRAME",
            "W2_INTEGRATION",
            "W2_INTERFRAME",
            "W3_INTEGRATION",
        ]:
            _, reghex = self.ca.getRegister(reg)
            aside.append(25 * int(reghex, 16))
        for reg in [
            "W0_INTEGRATION_B",
            "W0_INTERFRAME_B",
            "W1_INTEGRATION_B",
            "W1_INTERFRAME_B",
            "W2_INTEGRATION_B",
            "W2_INTERFRAME_B",
            "W3_INTEGRATION_B",
        ]:
            _, reghex = self.ca.getRegister(reg)
            bside.append(25 * int(reghex, 16))
        return [aside, bside]

```

8.6.3.3 getTiming()

```
def nsCamera.sensors.icarus2.icarus2.getTiming (
              self,
              side,
              actual )
```

```
actual = True: returns actual high speed intervals that will be generated by the
    FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2,
    open3]
 False: Returns high speed timing settings as set by setTiming. Assumes
    that timing was set via the setTiming method--it will not accurately
    report arbitrary timings set by direct register sets or manual
    shutter control


Args:
    side: Hemisphere 'A' or 'B'
    actual: False: return HST settings
    True: calculate and return actual HST behavior

Returns:
    actual= False: tuple   (hemisphere label,
                    'open shutter' in ns,
                    'closed shutter' in ns,
                    initial delay in ns)
    True: list of times [delay, open0, closed0, open1, closed1, open2,
      closed2, open3]
```

Definition at line 363 of file icarus2.py.

```
    def getTiming(self, side, actual):
        """
        actual = True: returns actual high speed intervals that will be generated by the
                    FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2,
                    open3]
                 False: Returns high speed timing settings as set by setTiming. Assumes
                    that timing was set via the setTiming method--it will not accurately
                    report arbitrary timings set by direct register sets or manual
                    shutter control


        Args:
            side: Hemisphere ’A’ or ’B’
            actual: False: return HST settings
                    True: calculate and return actual HST behavior

        Returns:
            actual= False: tuple   (hemisphere label,
                                    ’open shutter’ in ns,
                                    ’closed shutter’ in ns,
                                    initial delay in ns)
                    True: list of times [delay, open0, closed0, open1, closed1, open2,
                      closed2, open3]
        """
        if side is None:
            side = "A"

        logging.info(self.loginfo + "get timing, side " + side.upper())
        if side.upper() == "A":
            lowreg = "HS_TIMING_DATA_ALO"
            highreg = "HS_TIMING_DATA_AHI"
        elif side.upper() == "B":
            lowreg = "HS_TIMING_DATA_BLO"
            highreg = "HS_TIMING_DATA_BHI"
        else:
            logging.error(
                self.logerr
                + "Invalid sensor side: "
                + side
                + "; timing settings unchanged"
            )
            return "", 0, 0, 0
        err, lowpart = self.ca.getRegister(lowreg)
        err1, highpart = self.ca.getRegister(highreg)
        if err or err1:
            logging.error(
                self.logerr + "Unable to retrieve timing setting (getTiming), "
                "returning zeroes "
            )
            return side.upper(), 0, 0, 0
        full40hex = highpart[-2:] + lowpart.zfill(8)
        full40bin = "{0:0=40b}".format(int(full40hex, 16))
        if actual:
            full160 = 4 * full40bin
            gblist = [[k, len(list(g))] for k, g in itertools.groupby(full160)]
            times = [int(x[1]) for x in gblist[:-9:-1]]
            times[0] = times[0] - 1
            return times
        else:
            gblist = [[k, len(list(g))] for k, g in itertools.groupby(full40bin)]
            delay = gblist[-1][1] - 1
            timeon = gblist[-2][1]
            if len(gblist) < 4:  # sequence fits only once
                timeoff = 40 - timeon
            else:
                timeoff = gblist[-3][1]
            return side.upper(), timeon, timeoff, delay

```

8.6.3.4 parseReadoff()

```
def nsCamera.sensors.icarus2.icarus2.parseReadoff (
               self,
               frames )
```

```
Dummy function; unnecessary for Icarus2 sensor
```

Definition at line 517 of file icarus2.py.

```
517     def parseReadoff(self, frames):
518         """
519         Dummy function; unnecessary for Icarus2 sensor
520         """
521         return frames
522 
```

8.6.3.5 reportStatusSensor()

```
def nsCamera.sensors.icarus2.icarus2.reportStatusSensor (
              self,
              statusbits )
```

```
Print status messages from sensor-specific bits of status register

Args:
    statusbits: result of checkStatus()
```

Definition at line 523 of file icarus2.py.

```
523     def reportStatusSensor(self, statusbits):
524         """
525         Print status messages from sensor-specific bits of status register
526 
527         Args:
528             statusbits: result of checkStatus()
529         """
530         if int(statusbits[3]):
531             logging.info(self.loginfo + "W3_Top_L_Edge1 detected")
532         if int(statusbits[4]):
533             logging.info(self.loginfo + "W3_Top_R_Edge1 detected")
534         if int(statusbits[12]):
535             logging.info(self.loginfo + "HST_All_W_En detected")
536 
537 
538 """
539 Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
540 LLNL-CODE-838080
541 
542 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
543 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
544 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
545 See license for disclaimers, notice of U.S. Government Rights and license terms and
546 conditions.
547 """
```

8.6.3.6 sensorSpecific()

```
def nsCamera.sensors.icarus2.icarus2.sensorSpecific (
              self )
```

```
Returns:
    list of tuples, (Sensor-specific register, default setting)
```

Definition at line 110 of file icarus2.py.

```
110     def sensorSpecific(self):
111         """
112         Returns:
113             list of tuples, (Sensor-specific register, default setting)
114         """
115         return [
116             ("ICARUS_VER_SEL", "00000000"),
117             ("FPA_FRAME_INITIAL", "00000000"),
118             ("FPA_FRAME_FINAL", "00000003"),
119             ("FPA_ROW_INITIAL", "00000000"),
120             ("FPA_ROW_FINAL", "000003FF"),
121             ("HS_TIMING_DATA_BHI", "00000000"),
122             ("HS_TIMING_DATA_BLO", "00006666"),  # 0db6 = 2-1; 6666 = 2-2
123             ("HS_TIMING_DATA_AHI", "00000000"),
124             ("HS_TIMING_DATA_ALO", "00006666"),
125         ]
126
```

8.6.3.7 setArbTiming()

```
def nsCamera.sensors.icarus2.icarus2.setArbTiming (
                self,
                side,
                sequence )
```

```
Set arbitrary high-speed timing sequence. NOTE: Icarus sensors generally cannot
  use 1 ns timing, so all values (besides the delay) should be at least 2 ns
Args:
    side: Hemisphere 'A' or 'B'
    sequence: list of arbitrary timing intervals, beginning with initial delay.
      The conventional timing (5,2) with delay = 3 would be represented by
      [3,5,2,5,2,5,2,5].
    *WARNING* arbitrary timings will not be restored after a board power cycle

Returns:
    tuple (error string, 10-character hexadecimal representation of timing
      sequence)
```

Definition at line 287 of file icarus2.py.

```
287     def setArbTiming(self, side, sequence):
288         """
289         Set arbitrary high-speed timing sequence. NOTE: Icarus sensors generally cannot
290           use 1 ns timing, so all values (besides the delay) should be at least 2 ns
291         Args:
292             side: Hemisphere 'A' or 'B'
293             sequence: list of arbitrary timing intervals, beginning with initial delay.
294               The conventional timing (5,2) with delay = 3 would be represented by
295               [3,5,2,5,2,5,2,5].
296             *WARNING* arbitrary timings will not be restored after a board power cycle
297
298         Returns:
299             tuple (error string, 10-character hexadecimal representation of timing
300               sequence)
301         """
302         if side is None:
303             side = "A"
304         if sequence is None:
305             sequence = [0, 3, 2, 3, 2, 3, 2, 3]
306
307         logging.info(
308             self.loginfo + "HST side " + side.upper() + " (arbitrary): " + str(sequence)
309         )
310         if side.upper() == "A":
311             lowreg = "HS_TIMING_DATA_ALO"
312             highreg = "HS_TIMING_DATA_AHI"
```

```
        elif side.upper() == "B":
            lowreg = "HS_TIMING_DATA_BLO"
            highreg = "HS_TIMING_DATA_BHI"
        else:
            err = (
                self.logerr
                + "Invalid sensor side: "
                + side
                + "; timing settings unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        # TODO; restore arbitrary timing after power cycle?
        full40 = [0] * 40
        bitlist = []
        flag = 0  # similar to setTiming, but starts with delay
        sequence = sequence[: (2 * self.nframes)]
        for a in sequence:
            add = [flag] * a
            bitlist += add
            if flag:
                flag = 0
            else:
                flag = 1
        reversedlist = bitlist[39::-1]
        full40[-(len(reversedlist) + 1) : -1] = reversedlist
        full40bin = "".join(str(x) for x in full40)
        full40hex = "%x" % int(full40bin, 2)
        highpart = full40hex[-10:-8].zfill(8)
        lowpart = full40hex[-8:].zfill(8)
        self.ca.setRegister(lowreg, lowpart)
        self.ca.setRegister(highreg, highpart)
        self.ca.setRegister("HS_TIMING_CTL", "00000001")
        # deactivates manual shutter mode if previously engaged
        self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
        actual = self.getTiming(side, actual=True)
        if actual != sequence:
            logging.warning(
                self.logwarn + "Due to sequence length, actual timing sequence "
                "for side "
                + side
                + " will be "
                + "{"
                + str(actual[0])
                + "}"
                + " "
                + str(actual[1 : 2 * self.nframes])
            )
        return actual

```

8.6.3.8 setHighFullWell()

```
def nsCamera.sensors.icarus2.icarus2.setHighFullWell (
                self,
                flag )
```

Dummy function; feature is not implemented on Icarus2

Definition at line 140 of file icarus2.py.

```
    def setHighFullWell(self, flag):
        """
        Dummy function; feature is not implemented on Icarus2
        """
        if flag:
            logging.warning(
                self.logwarn + "HighFullWell mode is not supported by the Icarus2 "
                "sensor. "
            )

```

8.6.3.9 setInterlacing()

```
def nsCamera.sensors.icarus2.icarus2.setInterlacing (
              self,
              ifactor )
```

```
Dummy function; feature is not implemented on Icarus2

Returns:
    integer 1
```

Definition at line 127 of file icarus2.py.

```
    def setInterlacing(self, ifactor):
        """
        Dummy function; feature is not implemented on Icarus2

        Returns:
            integer 1
        """
        if ifactor:
            logging.warning(
                self.logwarn + "Interlacing is not supported by the Icarus2 sensor."
            )
        return 1

```

8.6.3.10 setManualShutters()

```
def nsCamera.sensors.icarus2.icarus2.setManualShutters (
              self,
              timing )
```

```
Manual shutter timing, seven intervals for each side of the imager given in
  nanoseconds, e.g., [(100,50,100,50,100,50,100),(100,50,100,50,100,50,100)]

The timing list is flattened before processing; the suggested tuple structure is
  just for clarity (first tuple is A, second is B) and is optional.

The actual timing is rounded down to nearest multiple of 25 ns. (Each
  count = 25 ns. e.g., 140 ns rounds down to a count of ’5’ which corresponds
  to 125 ns))

Args:
    timing: 14-element list (substructure optional) in nanoseconds

Returns:
    tuple (error string, response string from final message)
```

Definition at line 431 of file icarus2.py.

```
    def setManualShutters(self, timing):
        """
        Manual shutter timing, seven intervals for each side of the imager given in
          nanoseconds, e.g., [(100,50,100,50,100,50,100),(100,50,100,50,100,50,100)]

        The timing list is flattened before processing; the suggested tuple structure is
          just for clarity (first tuple is A, second is B) and is optional.

        The actual timing is rounded down to nearest multiple of 25 ns. (Each
          count = 25 ns. e.g., 140 ns rounds down to a count of ’5’ which corresponds
          to 125 ns))

        Args:
            timing: 14-element list (substructure optional) in nanoseconds

        Returns:
            tuple (error string, response string from final message)
        """
        if timing is None:
            timing = [
                (100, 50, 100, 50, 100, 50, 100),
                (100, 50, 100, 50, 100, 50, 100),
            ]
        logging.info(self.loginfo + "Manual shutter sequence: " + str(timing))
        flattened = self.ca.flatten(timing)
        if len(flattened) != 14 or not all(type(x) is int for x in flattened):
            err = self.logerr + "Invalid manual shutter timing list: " + str(timing)
            logging.error(err + "; timing settings unchanged")
            return err, "00000000"

        timecounts = [a // 25 for a in flattened]
        self.ca.sensmanual = timing
        self.ca.senstiming = {}  # clear HST settings from ca object

        control_messages = [
            ("W0_INTEGRATION", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
            ("W0_INTERFRAME", "{0:#0{1}x}".format(timecounts[1], 10)[2:10]),
            ("W1_INTEGRATION", "{0:#0{1}x}".format(timecounts[2], 10)[2:10]),
            ("W1_INTERFRAME", "{0:#0{1}x}".format(timecounts[3], 10)[2:10]),
            ("W2_INTEGRATION", "{0:#0{1}x}".format(timecounts[4], 10)[2:10]),
            ("W2_INTERFRAME", "{0:#0{1}x}".format(timecounts[5], 10)[2:10]),
            ("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[6], 10)[2:10]),
            ("W0_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[7], 10)[2:10]),
            ("W0_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[8], 10)[2:10]),
            ("W1_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[9], 10)[2:10]),
            ("W1_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[10], 10)[2:10]),
            ("W2_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[11], 10)[2:10]),
            ("W2_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[12], 10)[2:10]),
            ("W3_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[13], 10)[2:10]),
            ("HS_TIMING_CTL", "00000000"),
            ("MANUAL_SHUTTERS_MODE", "00000001"),
        ]
        return self.ca.submitMessages(control_messages, " setManualShutters: ")

```

8.6.3.11 setTiming()

```
def nsCamera.sensors.icarus2.icarus2.setTiming (
                self,
                side,
                sequence,
                delay )
```

```
Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
  fit into the 40-bit register space, then the actual timings may differ from
  those requested. If the timing sequence fits only once into register space
  (i.e., for a single frame, open + closed > 19 ns), then actual timing will be
  (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
  actually result in (35,5) timing.
NOTE: Icarus sensors generally cannot use 1 ns timing, so all values (besides
  the delay) should be at least 2 ns

Args:
    side: Hemisphere 'A' or 'B'
    sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
    delay: initial delay in ns

Returns:
    tuple (error string, 10-character hexadecimal representation of timing
      sequence)
```

Definition at line 170 of file icarus2.py.

```
    def setTiming(self, side, sequence, delay):
        """
        Sets timing registers based on ’sequence.’ WARNING: if entire sequence does not
          fit into the 40-bit register space, then the actual timings may differ from
          those requested. If the timing sequence fits only once into register space
          (i.e., for a single frame, open + closed > 19 ns), then actual timing will be
          (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
          actually result in (35,5) timing.
        NOTE: Icarus sensors generally cannot use 1 ns timing, so all values (besides
          the delay) should be at least 2 ns

        Args:
            side: Hemisphere ’A’ or ’B’
            sequence: two-element tuple of timing durations in ns, e.g., ’(5,2)’
            delay: initial delay in ns

        Returns:
            tuple (error string, 10-character hexadecimal representation of timing
              sequence)
        """
        if side is None:
            side = "A"
        if sequence is None:
            sequence = (3, 2)
        if delay is None:
            delay = 0

        if len(sequence) != 2:
            err = (
                self.logerr
                + "Invalid sequence setting for side: "
                + side
                + "; timing settings are unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        logging.info(
            self.loginfo
            + "HST side "
            + side.upper()
            + ": "
            + str(sequence)
            + "; delay = "
            + str(delay)
        )
        if side.upper() == "A":
            lowreg = "HS_TIMING_DATA_ALO"
            highreg = "HS_TIMING_DATA_AHI"
        elif side.upper() == "B":
            lowreg = "HS_TIMING_DATA_BLO"
            highreg = "HS_TIMING_DATA_BHI"
        else:
            err = (
                self.logerr
                + "Invalid sensor side: "
                + side
                + "; timing settings unchanged"
            )
            logging.error(err)
            return err, "0000000000"
        if (sequence[0] + sequence[1]) + delay > 40:
            err = (
                self.logerr + "Timing sequence is too long to be implemented; "
                "timing settings unchanged "
            )
            logging.error(err)
            return err, "0000000000"

        self.ca.senstiming[side.upper()] = (sequence, delay)
        self.ca.sensmanual = []  # clear manual settings from ca

        full40 = [0] * 40
        bitlist = []
        flag = 1
        sequence = sequence[:2]
        for a in sequence:
            add = [flag] * a
            bitlist += add
            if flag:
                flag = 0
            else:
                flag = 1
        # automatically truncates sequence to 39 characters
        reversedlist = bitlist[39::-1]
        repeats = (40 - delay) // len(reversedlist)
        if repeats > self.nframes:
            repeats = self.nframes
        # Pattern from sequence repeated to fit inside 40 bits up to a maximum of
        #   ’nframes’ times
        repeated = reversedlist * repeats
        full40[-(len(repeated) + delay + 1) : -(delay + 1)] = repeated
        full40bin = "".join(str(x) for x in full40)
        full40hex = "%x" % int(full40bin, 2)
        highpart = full40hex[-10:-8].zfill(8)
        lowpart = full40hex[-8:].zfill(8)
        self.ca.setRegister(lowreg, lowpart)
        self.ca.setRegister(highreg, highpart)
        self.ca.setRegister("HS_TIMING_CTL", "00000001")
        # deactivates manual shutter mode if previously engaged
        self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
        if repeats < self.nframes:
            actual = self.getTiming(side, actual=True)
            expected = [delay] + 3 * list(sequence) + [sequence[0]]
            if actual != expected:
                logging.warning(
                    self.logwarn + "Due to sequence length, actual timing "
                    "sequence for side "
                    + side
                    + " will be "
                    + "{"
                    + str(actual[0])
                    + "}"
                    + " "
                    + str(actual[1 : 2 * self.nframes])
                )
        return "", full40hex

```

8.6.3.12 setTriggerDelay()

```
def nsCamera.sensors.icarus2.icarus2.setTriggerDelay (
                self,
                delayblocks )
```

Dummy function; feature is not implemented on Icarus2

Definition at line 160 of file icarus2.py.

```
    def setTriggerDelay(self, delayblocks):
        """
        Dummy function; feature is not implemented on Icarus2
        """
        if delayblocks:
            logging.warning(
                self.logwarn + "Trigger Delay is not supported by the Icarus2 "
                "sensor. "
            )

```

8.6.3.13 setZeroDeadTime()

```
def nsCamera.sensors.icarus2.icarus2.setZeroDeadTime (
              self,
              flag )
```

```
Dummy function; feature is not implemented on Icarus2
```

Definition at line 150 of file icarus2.py.

```
    def setZeroDeadTime(self, flag):
        """
        Dummy function; feature is not implemented on Icarus2
        """
        if flag:
            logging.warning(
                self.logwarn + "ZeroDeadTime mode is not supported by the Icarus2 "
                "sensor. "
            )

```

8.6.4 Member Data Documentation

8.6.4.1 bytesperpixel

```
nsCamera.sensors.icarus2.icarus2.bytesperpixel
```

Definition at line 44 of file icarus2.py.

8.6.4.2 ca

```
nsCamera.sensors.icarus2.icarus2.ca
```

Definition at line 26 of file icarus2.py.

8.6.4.3 firstframe

```
nsCamera.sensors.icarus2.icarus2.firstframe
```

Definition at line 35 of file icarus2.py.

8.6.4.4 firstrow

```
nsCamera.sensors.icarus2.icarus2.firstrow
```

Definition at line 40 of file icarus2.py.

8.6.4.5 fpganumID

```
nsCamera.sensors.icarus2.icarus2.fpganumID
```

Definition at line 46 of file icarus2.py.

8.6.4.6 height

```
nsCamera.sensors.icarus2.icarus2.height
```

Definition at line 43 of file icarus2.py.

8.6.4.7 icarustype

```
nsCamera.sensors.icarus2.icarus2.icarustype
```

Definition at line 45 of file icarus2.py.

8.6.4.8 interlacing

```
nsCamera.sensors.icarus2.icarus2.interlacing
```

Definition at line 47 of file icarus2.py.

8.6.4.9 lastframe

```
nsCamera.sensors.icarus2.icarus2.lastframe
```

Definition at line 36 of file icarus2.py.

8.6.4.10 lastrow

```
nsCamera.sensors.icarus2.icarus2.lastrow
```

Definition at line 41 of file icarus2.py.

8.6.4.11 logcrit

```
nsCamera.sensors.icarus2.icarus2.logcrit
```

Definition at line 27 of file icarus2.py.

8.6.4.12 logdebug

```
nsCamera.sensors.icarus2.icarus2.logdebug
```

Definition at line 31 of file icarus2.py.

8.6.4.13 logerr

```
nsCamera.sensors.icarus2.icarus2.logerr
```

Definition at line 28 of file icarus2.py.

8.6.4.14 loginfo

```
nsCamera.sensors.icarus2.icarus2.loginfo
```

Definition at line 30 of file icarus2.py.

8.6.4.15 logwarn

```
nsCamera.sensors.icarus2.icarus2.logwarn
```

Definition at line 29 of file icarus2.py.

8.6.4.16 maxframe

```
nsCamera.sensors.icarus2.icarus2.maxframe
```

Definition at line 34 of file icarus2.py.

8.6.4.17 maxheight

```
nsCamera.sensors.icarus2.icarus2.maxheight
```

Definition at line 39 of file icarus2.py.

8.6.4.18 maxwidth

```
nsCamera.sensors.icarus2.icarus2.maxwidth
```

Definition at line 38 of file icarus2.py.

8.6.4.19 minframe

```
nsCamera.sensors.icarus2.icarus2.minframe
```

Definition at line 33 of file icarus2.py.

8.6.4.20 nframes

```
nsCamera.sensors.icarus2.icarus2.nframes
```

Definition at line 37 of file icarus2.py.

8.6.4.21 sens_registers

```
nsCamera.sensors.icarus2.icarus2.sens_registers
```

Definition at line 49 of file icarus2.py.

8.6.4.22 sens_subregisters

```
nsCamera.sensors.icarus2.icarus2.sens_subregisters
```

Definition at line 73 of file icarus2.py.

8.6.4.23 width

```
nsCamera.sensors.icarus2.icarus2.width
```

Definition at line 42 of file icarus2.py.

The documentation for this class was generated from the following file:

- nsCamera/sensors/icarus2.py

8.7 nsCamera.boards.LLNL_v1.llnl_v1 Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def initBoard (self)
- def initPots (self)
- def latchPots (self)
- def initSensor (self)
- def configADCs (self)
- def softReboot (self)
- def disarm (self)
- def startCapture (self, mode="Hardware")
- def readSRAM (self)
- def waitForSRAM (self, timeout)
- def getTimer (self)
- def resetTimer (self)
- def enableLED (self, status)
- def setLED (self, LED, status)
- def setPowerSave (self, status)
- def setPPER (self, time)
- def getTemp (self, scale)
- def getPressure (self, offset, sensitivity, units)
- def clearStatus (self)
- def checkStatus (self)
- def checkStatus2 (self)
- def reportStatus (self)
- def reportEdgeDetects (self)
- def dumpStatus (self)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- VREF
- ADC5_mult
- ADC5_bipolar
- rs422_baud
- rs422_cmd_wait
- subreg_aliases
- monitor_controls
- subreglist

Static Public Attributes

- registers
- list subregisters
- list dummySensorVals

8.7.1 Detailed Description

```
Livermore LLNL v1.0 board

Compatible communication protocols: RS422, GigE
Compatible sensors: icarus, icarus2, daedalus
```

Definition at line 29 of file LLNL_v1.py.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 __init__()

```
def nsCamera.boards.LLNL_v1.llnl_v1.__init__ (
                self,
                camassem )
```

Definition at line 233 of file LLNL_v1.py.

```
233     def __init__(self, camassem):
234         self.ca = camassem
235         self.logcrit = self.ca.logcritbase + "[LLNL_v1] "
236         self.logerr = self.ca.logerrbase + "[LLNL_v1] "
237         self.logwarn = self.ca.logwarnbase + "[LLNL_v1] "
238         self.loginfo = self.ca.loginfobase + "[LLNL_v1] "
239         self.logdebug = self.ca.logdebugbase + "[LLNL_v1] "
240         logging.info(self.loginfo + "initializing board object")
241         self.VREF = 2.5  # default
242         self.ADC5_mult = 2  # monmax = 2 * VREF
243         # False => monitor range runs 0 to monmax, True => +/- monmax
244         self.ADC5_bipolar = True
245 
246         self.rs422_baud = 921600
247         self.rs422_cmd_wait = 0.3
248 
249         fpgaNum_pkt = Packet(cmd="1", addr=self.registers["FPGA_NUM"])
250         fpgaRev_pkt = Packet(cmd="1", addr=self.registers["FPGA_REV"])
251 
252         _, _ = self.ca.sendCMD(fpgaNum_pkt)  # dummy duplicate call
253         err, rval = self.ca.sendCMD(fpgaNum_pkt)
254         self.ca.FPGANum = rval[8:16]
255 
256         err, rval = self.ca.sendCMD(fpgaRev_pkt)
257         self.ca.FPGAVersion = rval[8:16]
258 
259         # map channels to signal names for abstraction at the camera assembler level;
260         #   each requires a corresponding entry in ’subregisters’
261         if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
262             self.subreg_aliases = OrderedDict(
263                 {
264                     "COL_BOT_IBIAS_IN": "POT1",
265                     "HST_A_PDELAY": "POT2",
266                     "HST_B_NDELAY": "POT3",
267                     "HST_RO_IBIAS": "POT4",
268                     "HST_OSC_VREF_IN": "POT5",
269                     "HST_B_PDELAY": "POT6",
270                     "HST_OSC_CTL": "POT7",
271                     "HST_A_NDELAY": "POT8",
272                     "COL_TOP_IBIAS_IN": "POT9",
273                     "HST_OSC_R_BIAS": "POT10",
274                     "VAB": "POT11",
275                     "HST_RO_NC_IBIAS": "POT12",
276                     "VRST": "POT13",
277                     "MON_HST_A_PDELAY": "MON_CH2",
278                     "MON_HST_B_NDELAY": "MON_CH3",
279                     "MON_HST_RO_IBIAS": "MON_CH4",
280                     "MON_HST_OSC_VREF_IN": "MON_CH5",
281                     "MON_HST_B_PDELAY": "MON_CH6",
282                     "MON_HST_OSC_CTL": "MON_CH7",
283                     "MON_HST_A_NDELAY": "MON_CH8",
284                 }
285             )
286             # Read-only; identifies controls corresponding to monitors
287             self.monitor_controls = OrderedDict(
288                 {
289                     "MON_CH2": "POT2",
290                     "MON_CH3": "POT3",
291                     "MON_CH4": "POT4",
292                     "MON_CH5": "POT5",
293                     "MON_CH6": "POT6",
294                     "MON_CH7": "POT7",
295                     "MON_CH8": "POT8",
296                     # Note: VRST is not measured across the pot; it will read a voltage
297                     #   approximately 1 Volt lower than pot13’s actual output
298                     "MON_VRST": "POT13",
299                 }
300             )
301         else:  # Daedalus
302             self.subreg_aliases = OrderedDict(
                {
                    "HST_OSC_CTL": "POT4",
                    "HST_RO_NC_IBIAS": "POT5",
                    "HST_OSC_VREF_IN": "POT6",
                    "VAB": "POT11",
                    "MON_TSENSEOUT": "MON_CH2",
                    "MON_BGREF": "MON_CH3",
                    "MON_HST_OSC_CTL": "MON_CH4",
                    "MON_HST_RO_NC_IBIAS": "MON_CH5",
                    "MON_HST_OSC_VREF_IN": "MON_CH6",
                    "MON_COL_TST_IN": "MON_CH7",
                    "MON_HST_OSC_PBIAS_PAD": "MON_CH8",
                }
            )
            # Read-only; identifies controls corresponding to monitors
            self.monitor_controls = OrderedDict(
                {
                    "MON_CH4": "POT4",
                    "MON_CH5": "POT5",
                    "MON_CH6": "POT6",
                    # Note: VRST is not measured across the pot; it will read a voltage
                    #   lower than pot13’s actual output
                    "MON_VRST": "POT13",
                }
            )

        self.subreglist = []
        for s in self.subregisters:
            self.subreglist.append(s[0].upper())
            sr = SubRegister(
                self,
                name=s[0].upper(),
                register=s[1].upper(),
                start_bit=s[2],
                width=s[3],
                writable=s[4],
            )
            setattr(self, s[0].upper(), sr)

        # set voltage ranges for all pots
        for n in range(1, 13):
            potname = "POT" + str(n)
            potobj = getattr(self, potname)
            potobj.minV = 0
            potobj.maxV = 3.3
            # resolution is approximately .0129 V / LSB
            potobj.resolution = (1.0 * potobj.maxV - potobj.minV) / potobj.max_value
        self.POT13.minV = 0
        self.POT13.maxV = 3.96
        # POT13 resolution is approximately .0155 V / LSB
        self.POT13.resolution = (
            1.0 * self.POT13.maxV - self.POT13.minV
        ) / self.POT13.max_value

```

8.7.3 Member Function Documentation

8.7.3.1 checkStatus()

```
def nsCamera.boards.LLNL_v1.llnl_v1.checkStatus (
              self )
```

```
Check status register, convert to reverse-order bit stream (i.e., bit 0 is
  statusbits[0])

Returns:
    bit string (no ’0b’) in reversed order
```

Definition at line 773 of file LLNL_v1.py.

```
    def checkStatus(self):
        """
        Check status register, convert to reverse-order bit stream (i.e., bit 0 is
          statusbits[0])

        Returns:
            bit string (no ’0b’) in reversed order
        """
        err, rval = self.ca.getRegister("STAT_REG")
        if not rval:
            logging.error(
                self.logerr + "Unable to check status register (zeroes returned)"
            )
            rval = "0"
        rvalbits = bin(int(rval, 16))[2:].zfill(32)
        statusbits = rvalbits[::-1]
        return statusbits  # TODO: add error handling

```

8.7.3.2 checkStatus2()

```
def nsCamera.boards.LLNL_v1.llnl_v1.checkStatus2 (
              self )
```

```
Check second status register, convert to reverse-order bit stream (i.e., bit 0
  is statusbits[0])

Returns: bit string (no ’0b’) in reversed order
```

Definition at line 791 of file LLNL_v1.py.

```
    def checkStatus2(self):
        """
        Check second status register, convert to reverse-order bit stream (i.e., bit 0
          is statusbits[0])

        Returns: bit string (no ’0b’) in reversed order
        """
        err, rval = self.ca.getRegister("STAT_REG2")
        if not rval:
            logging.error(
                self.logerr + "Unable to check status register 2 (zeroes returned)"
            )
            rval = "0"
        rvalbits = bin(int(rval, 16))[2:].zfill(5)
        statusbits = rvalbits[::-1]
        return statusbits  # TODO: add error handling

```

8.7.3.3 clearStatus()

```
def nsCamera.boards.LLNL_v1.llnl_v1.clearStatus (
              self )
```

```
Check status registers to clear them

Returns:
    error string
```

Definition at line 759 of file LLNL_v1.py.

```
759     def clearStatus(self):
760         """
761         Check status registers to clear them
762 
763         Returns:
764             error string
765         """
766         err1, rval = self.ca.getRegister("STAT_REG_SRC")
767         err2, rval = self.ca.getRegister("STAT_REG2_SRC")
768         err = err1 + err2
769         if err:
770             logging.error(self.logerr + "clearStatus failed")
771         return err
772 
```

8.7.3.4 configADCs()

```
def nsCamera.boards.LLNL_v1.llnl_v1.configADCs (
               self )
```

```
Sets default ADC configuration (does not latch settings)

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 479 of file LLNL_v1.py.

```
479     def configADCs(self):
480         """
481         Sets default ADC configuration (does not latch settings)
482 
483         Returns:
484             tuple (error string, response string) from final control message
485         """
486         logging.info(self.loginfo + "configADCs")
487 
488         control_messages = [
489             # just in case ADC_RESET was set (pull all ADCs out # of reset)
490             (
491                 "ADC_RESET",
492                 "00000000",
493             ),
494             # workaround for uncertain behavior after previous readoff
495             (
496                 "ADC1_CONFIG_DATA",
497                 "FFFFFFFF",
498             ),
499             ("ADC2_CONFIG_DATA", "FFFFFFFF"),
500             ("ADC3_CONFIG_DATA", "FFFFFFFF"),
501             ("ADC4_CONFIG_DATA", "FFFFFFFF"),
502             ("ADC_CTL", "FFFFFFFF"),
503             ("ADC1_CONFIG_DATA", "81A801FF"),  # ext Vref 1.25V
504             ("ADC2_CONFIG_DATA", "81A801FF"),  # ext Vref 1.25V
505             ("ADC3_CONFIG_DATA", "81A801FF"),  # ext Vref 1.25V
506             ("ADC4_CONFIG_DATA", "81A801FF"),  # ext Vref 1.25V
507             ("ADC5_CONFIG_DATA", "81A883FF"),  # int Vref 2.50V
508         ]
509         return self.ca.submitMessages(control_messages, " configADCs: ")
510 
```

8.7.3.5 disarm()

```
def nsCamera.boards.LLNL_v1.llnl_v1.disarm (
              self )
```

```
Takes camera out of trigger wait state. Has no effect if camera is not already
  in wait state.

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 523 of file LLNL_v1.py.

```
523     def disarm(self):
524         """
525         Takes camera out of trigger wait state. Has no effect if camera is not already
526           in wait state.
527 
528         Returns:
529             tuple (error string, response string) from final control message
530         """
531         logging.info(self.loginfo + "disarm")
532         self.ca.clearStatus()
533         self.ca.armed = False
534         control_messages = [
535             ("HW_TRIG_EN", "0"),
536             ("DUAL_EDGE_TRIG_EN", "0"),
537             ("SW_TRIG_EN", "0"),
538         ]
539         return self.ca.submitMessages(control_messages, " disarm: ")
540 
```

8.7.3.6 dumpStatus()

```
def nsCamera.boards.LLNL_v1.llnl_v1.dumpStatus (
              self )
```

```
Create dictionary of status values, DAC settings, monitor values, and register
  values

WARNING: the behavior of self-resetting subregisters may be difficult to predict
  and may generate contradictory results

Returns:
    dictionary of system diagnostic values
```

Definition at line 866 of file LLNL_v1.py.

```
866     def dumpStatus(self):
867         """
868         Create dictionary of status values, DAC settings, monitor values, and register
869           values
870 
871         WARNING: the behavior of self-resetting subregisters may be difficult to predict
872           and may generate contradictory results
873 
874         Returns:
875             dictionary of system diagnostic values
876         """
877         statusbits = self.checkStatus()
878         statusbits2 = self.checkStatus2()
879         temp = self.ca.getTemp()
880 
```

```
        statDict = OrderedDict(
            {
                "Temperature reading": "{0:1.2f}".format(temp) + " C",
                "Sensor read complete": str(statusbits[0]),
                "Coarse trigger detected": str(statusbits[1]),
                "Fine trigger detected": str(statusbits[2]),
                "Sensor readout in progress": str(statusbits[5]),
                "Sensor readout complete": str(statusbits[6]),
                "SRAM readout started": str(statusbits[7]),
                "SRAM readout complete": str(statusbits[8]),
                "High-speed timing configured": str(statusbits[9]),
                "All ADCs configured": str(statusbits[10]),
                "All pots configured": str(statusbits[11]),
                "HST_All_W_En detected": str(statusbits[12]),
                "Timer has reset": str(statusbits[13]),
                "Camera is Armed": str(statusbits[14]),
                "FPA_IF_TO": str(statusbits2[0]),
                "SRAM_RO_TO": str(statusbits2[1]),
                "PixelRd Timeout Error": str(statusbits2[2]),
                "UART_TX_TO_RST": str(statusbits2[3]),
                "UART_RX_TO_RST": str(statusbits2[4]),
            }
        )

        DACDict = OrderedDict()
        MonDict = OrderedDict()
        for entry in self.subreg_aliases:
            if self.subreg_aliases[entry][0] == "P":
                val = str(round(self.ca.getPotV(entry), 3)) + " V"
                DACDict["POT_" + entry] = val
            else:
                val = str(round(self.ca.getMonV(entry), 3)) + " V"
                MonDict[entry] = val

        regDict = OrderedDict()
        for key in self.registers.keys():
            err, rval = self.ca.getRegister(key)
            regDict[key] = rval

        dumpDict = OrderedDict()
        for x in [statDict, MonDict, POTDict, regDict]:
            dumpDict.update(x)
        return dumpDict


"""
Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
LLNL-CODE-838080

This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
(DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
See license for disclaimers, notice of U.S. Government Rights and license terms and
conditions.
"""
```

8.7.3.7 enableLED()

```
def nsCamera.boards.LLNL_v1.llnl_v1.enableLED (
              self,
              status )
```

```
Enable/disable on-board LEDs

Args:
    status: 0 for disabled, 1 for enabled

Returns:
    tuple: (error string, response string from subregister set)
```

Definition at line 655 of file LLNL_v1.py.

```
    def enableLED(self, status):
        """
        Enable/disable on-board LEDs

        Args:
            status: 0 for disabled, 1 for enabled

        Returns:
            tuple: (error string, response string from subregister set)
        """
        if status:
            status = 1
        return self.ca.setSubregister("LED_EN", str(status))

```

8.7.3.8 getPressure()

```
def nsCamera.boards.LLNL_v1.llnl_v1.getPressure (
              self,
              offset,
              sensitivity,
              units )
```

```
Read pressure sensor

Currently unimplemented

Returns:
    0 as float
```

Definition at line 745 of file LLNL_v1.py.

```
    def getPressure(self, offset, sensitivity, units):
        """
        Read pressure sensor

        Currently unimplemented

        Returns:
            0 as float
        """
        logging.warning(
            "WARNING: [LLNL_v1] 'getPressure' is not implemented on the LLNLv1 board"
        )
        return 0.0

```

8.7.3.9 getTemp()

```
def nsCamera.boards.LLNL_v1.llnl_v1.getTemp (
              self,
              scale )
```

```
Read temperature sensor
Args:
    scale: temperature scale to report (defaults to C, options are F and K)
Returns:
    temperature as float on given scale
```

Definition at line 720 of file LLNL_v1.py.

```
    def getTemp(self, scale):
        """
        Read temperature sensor
        Args:
            scale: temperature scale to report (defaults to C, options are F and K)
        Returns:
            temperature as float on given scale
        """
        err, rval = self.ca.getRegister("TEMP_SENSE_DATA")
        if err:
            logging.error(
                self.logerr + "unable to retrieve temperature information ("
                'getTemp), returning "0" '
            )
            return 0.0

        ctemp = int(rval[-3:], 16) / 16.0
        if scale == "K":
            temp = ctemp + 273.15
        elif scale == "F":
            temp = 1.8 * ctemp + 32
        else:
            temp = ctemp
        return temp

```

8.7.3.10 getTimer()

```
def nsCamera.boards.LLNL_v1.llnl_v1.getTimer (
              self )
```

```
Read value of on-board timer

Returns:
    timer value as integer
```

Definition at line 628 of file LLNL_v1.py.

```
    def getTimer(self):
        """
        Read value of on-board timer

        Returns:
            timer value as integer
        """
        err, rval = self.ca.getRegister("TIMER_VALUE")
        if err:
            logging.error(
                self.logerr + "unable to retrieve timer information (getTimer), "
                'returning "0" '
            )
            return 0
        return int(rval, 16)

```

8.7.3.11 initBoard()

```
def nsCamera.boards.LLNL_v1.llnl_v1.initBoard (
              self )
```

```
Register and reset board, set up firmware for sensor

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 357 of file LLNL_v1.py.

```
357     def initBoard(self):
358         """
359         Register and reset board, set up firmware for sensor
360 
361         Returns:
362             tuple (error string, response string) from final control message
363         """
364         logging.info(self.loginfo + "initBoard")
365         control_messages = [("LED_EN", "1")]
366 
367         self.clearStatus()
368         self.configADCs()
369 
370         err, resp = self.ca.getSubregister("ADC5_VREF3")
371         if err:
372             logging.error(self.logerr + "unable to read ’ADC5_VREF3’")
373         if int(resp, 2):  # check to see if Vref is 3 or 2.5 volts
374             vrefmax = 3.0
375         else:
376             vrefmax = 2.5
377         err, resp = self.ca.getSubregister("ADC5_VREF")
378         if err:
379             logging.error(self.logerr + "unable to read ’ADC5_VREF’")
380         self.VREF = vrefmax * int(resp, 2) / 1024.0
381         err, multmask = self.ca.getSubregister("ADC5_MULT")
382         if err:
383             logging.error(self.logerr + "unable to read ’ADC5_MULT’")
384         if multmask[0] and multmask[1] and multmask[3] and multmask[5]:
385             self.ADC5_mult = 2
386         elif not (multmask[0] or multmask[1] or multmask[3] or multmask[5]):
387             self.ADC5_mult = 4
388         else:
389             logging.error(self.logerr + "inconsistent mode settings on ADC5")
390         return self.ca.submitMessages(control_messages, " initBoard: ")
391 
```

8.7.3.12 initPots()

```
def nsCamera.boards.LLNL_v1.llnl_v1.initPots (
              self )
```

```
Configure default pot settings before image acquisition

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 392 of file LLNL_v1.py.

```
392     def initPots(self):
393         """
394         Configure default pot settings before image acquisition
395 
396         Returns:
397             tuple (error string, response string) from final control message
398         """
399         logging.info(self.loginfo + "initPots")
400         if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
401             err0, _ = self.ca.setPot("HST_A_PDELAY", 0, errflag=True)
402             err1, _ = self.ca.setPotV("HST_B_NDELAY", 3.3, errflag=True)
403             err2, _ = self.ca.setPotV("HST_RO_IBIAS", 2.5, tune=True, errflag=True)
404             err3, _ = self.ca.setPotV("HST_OSC_VREF_IN", 2.9, tune=True, errflag=True)
405             err4, _ = self.ca.setPot("HST_B_PDELAY", 0, errflag=True)
406             err5, _ = self.ca.setPotV("HST_OSC_CTL", 1.45, tune=True, errflag=True)
407             err6, _ = self.ca.setPotV("HST_A_NDELAY", 3.3, errflag=True)
408             err7, _ = self.ca.setPotV("VAB", 0.5, errflag=True)
409             err8, _ = self.ca.setPotV("HST_RO_NC_IBIAS", 2.5, errflag=True)
410             err9, _ = self.ca.setPotV("VRST", 0.3, tune=True, errflag=True)
411             err = err0 + err1 + err2 + err3 + err4 + err5 + err6 + err7 + err8 + err9
412         else:  # Daedalus
413             err0, _ = self.ca.setPotV("HST_OSC_CTL", 1.0, tune=True, errflag=True)
414             err1, _ = self.ca.setPotV("HST_RO_NC_IBIAS", 1.0, errflag=True)
415             err2, _ = self.ca.setPotV("HST_OSC_VREF_IN", 1.0, tune=True, errflag=True)
416             err3, _ = self.ca.setPotV("VAB", 0.5, errflag=True)
417             err = err0 + err1 + err2 + err3
418         return err, ""
419 
```

8.7.3.13 initSensor()

```
def nsCamera.boards.LLNL_v1.llnl_v1.initSensor (
              self )
```

```
Register sensor, set default timing settings

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 446 of file LLNL_v1.py.

```
446     def initSensor(self):
447         """
448         Register sensor, set default timing settings
449 
450         Returns:
451             tuple (error string, response string) from final control message
452         """
453         logging.info(self.loginfo + "initSensor")
454         if self.ca.FPGANum[7] is not self.ca.sensor.fpganumID:
455             logging.error(
456                 self.logerr + "unable to confirm sensor compatibility with FPGA"
457             )
458         self.registers.update(self.ca.sensor.sens_registers)
459         self.subregisters.extend(self.ca.sensor.sens_subregisters)
460         for s in self.ca.sensor.sens_subregisters:
461             sr = SubRegister(
462                 self,
463                 name=s[0].upper(),
464                 register=s[1].upper(),
465                 start_bit=s[2],
466                 width=s[3],
467                 writable=s[4],
468             )
469             setattr(self, s[0].upper(), sr)
470             self.subreglist.append(s[0])
471         self.ca.checkSensorVoltStat()
472         control_messages = self.ca.sensorSpecific() + [
473            # ring w/caps=01, relax=00, ring w/o caps = 02
474            ("FPA_OSCILLATOR_SEL_ADDR", "00000000"),
475            ("FPA_DIVCLK_EN_ADDR", "00000001"),
476        ]
477        return self.ca.submitMessages(control_messages, " initSensor: ")
478
```

8.7.3.14 latchPots()

```
def nsCamera.boards.LLNL_v1.llnl_v1.latchPots (
                self )
```

```
Latch pot settings into sensor

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 420 of file LLNL_v1.py.

```
420    def latchPots(self):
421        """
422        Latch pot settings into sensor
423
424        Returns:
425            tuple (error string, response string) from final control message
426        """
427        logging.info(self.loginfo + "latchPots")
428
429        control_messages = [
430            ("POT_CTL", "00000003"),  # latches register settings for pot 1
431            ("POT_CTL", "00000005"),
432            ("POT_CTL", "00000007"),
433            ("POT_CTL", "00000009"),
434            ("POT_CTL", "0000000B"),
435            ("POT_CTL", "0000000D"),
436            ("POT_CTL", "0000000F"),
437            ("POT_CTL", "00000011"),
438            ("POT_CTL", "00000013"),
439            ("POT_CTL", "00000015"),
440            ("POT_CTL", "00000017"),
441            ("POT_CTL", "00000019"),
442            ("POT_CTL", "0000001B"),
443        ]
444        return self.ca.submitMessages(control_messages, " latchPots: ")
445
```

8.7.3.15 readSRAM()

```
def nsCamera.boards.LLNL_v1.llnl_v1.readSRAM (
                self )
```

```
Start readoff of SRAM

Returns:
    tuple (error string, response string from register set)
```

Definition at line 582 of file LLNL_v1.py.

```
    def readSRAM(self):
        """
        Start readoff of SRAM

        Returns:
            tuple (error string, response string from register set)
        """
        logging.info(self.loginfo + "readSRAM")
        control_messages = [("READ_SRAM", "1")]
        return self.ca.submitMessages(control_messages, " readSRAM: ")

```

8.7.3.16 reportEdgeDetects()

```
def nsCamera.boards.LLNL_v1.llnl_v1.reportEdgeDetects (
                self )
```

Unimplemented

Definition at line 857 of file LLNL_v1.py.

```
    def reportEdgeDetects(self):
        """
        Unimplemented
        """
        logging.warning(
            self.logwarn + "’reportEdgeDetects’ is not implemented on the LLNLv1 "
            "board "
        )

```

8.7.3.17 reportStatus()

```
def nsCamera.boards.LLNL_v1.llnl_v1.reportStatus (
                self )
```

Check contents of status register, print relevant messages

Definition at line 808 of file LLNL_v1.py.

```
    def reportStatus(self):
        """
        Check contents of status register, print relevant messages
        """
        statusbits = self.checkStatus()
        statusbits2 = self.checkStatus2()
        logging.info(self.loginfo + "Status report:")
        if int(statusbits[0]):
            logging.info(self.loginfo + "Sensor read complete")
        if int(statusbits[1]):
            logging.info(self.loginfo + "Coarse trigger detected")
        if int(statusbits[2]):
            logging.info(self.loginfo + "Fine trigger detected")
        if int(statusbits[5]):
            logging.info(self.loginfo + "Sensor readout in progress")
        if int(statusbits[6]):
            logging.info(self.loginfo + "Sensor readout complete")
        if int(statusbits[7]):
```

```
826             logging.info(self.loginfo + "SRAM readout started")
827         if int(statusbits[8]):
828             logging.info(self.loginfo + "SRAM readout complete")
829         if int(statusbits[9]):
830             logging.info(self.loginfo + "High-speed timing configured")
831         if int(statusbits[10]):
832             logging.info(self.loginfo + "All ADCs configured")
833         if int(statusbits[11]):
834             logging.info(self.loginfo + "All pots configured")
835         if int(statusbits[13]):
836             logging.info(self.loginfo + "Timer has reset")
837         if int(statusbits[14]):
838             logging.info(self.loginfo + "Camera is Armed")
839         self.ca.sensor.reportStatusSensor(statusbits)
840         temp = int(statusbits[27:15:-1], 2) / 16.0
841         logging.info(
842             self.loginfo + "Temperature reading: " + "{0:1.2f}".format(temp) + " C"
843         )
844         press = int(statusbits[:27:-1], 2)
845         logging.info(self.loginfo + "Pressure reading: " + "{0:1.2f}".format(press))
846         if int(statusbits2[0]):
847             logging.info(self.loginfo + "FPA_IF_TO")
848         if int(statusbits2[1]):
849             logging.info(self.loginfo + "INFO: [LLNL_v1] SRAM_RO_TO")
850         if int(statusbits2[2]):
851             logging.info(self.loginfo + "PixelRd Timeout Error")
852         if int(statusbits2[3]):
853             logging.info(self.loginfo + "UART_TX_TO_RST")
854         if int(statusbits2[4]):
855             logging.info(self.loginfo + "UART_RX_TO_RST")
856
```

8.7.3.18 resetTimer()

```
def nsCamera.boards.LLNL_v1.llnl_v1.resetTimer (
              self )
```

```
Reset on-board timer

Returns:
    tuple (error string, response string from register set)
```

Definition at line 644 of file LLNL_v1.py.

```
644     def resetTimer(self):
645         """
646         Reset on-board timer
647
648         Returns:
649             tuple (error string, response string from register set)
650         """
651         logging.info(self.loginfo + "resetTimer")
652         control_messages = [("RESET_TIMER", "1"), ("RESET_TIMER", "0")]
653         return self.ca.submitMessages(control_messages, " resetTimer: ")
654
```

8.7.3.19 setLED()

```
def nsCamera.boards.LLNL_v1.llnl_v1.setLED (
              self,
              LED,
              status )
```

```
Illuminate on-board LED

Args:
    LED: LED number (1-8)
    status: 0 is off, 1 is on

Returns:
    tuple: (error string, response string from subregister set)
```

Definition at line 669 of file LLNL_v1.py.

```
    def setLED(self, LED, status):
        """
        Illuminate on-board LED

        Args:
            LED: LED number (1-8)
            status: 0 is off, 1 is on

        Returns:
            tuple: (error string, response string from subregister set)
        """
        key = "LED" + str(LED)
        return self.ca.setSubregister(key, str(status))

```

8.7.3.20 setPowerSave()

```
def nsCamera.boards.LLNL_v1.llnl_v1.setPowerSave (
              self,
              status )
```

```
Select powersave option

Args:
    status: setting for powersave option (1 is enabled)

Returns:
    tuple (error string, response string from subregister set)
```

Definition at line 683 of file LLNL_v1.py.

```
    def setPowerSave(self, status):
        """
        Select powersave option

        Args:
            status: setting for powersave option (1 is enabled)

        Returns:
            tuple (error string, response string from subregister set)
        """
        if status:
            status = 1
        return self.ca.setSubregister("POWERSAVE", str(status))

```

8.7.3.21 setPPER()

```
def nsCamera.boards.LLNL_v1.llnl_v1.setPPER (
              self,
              time )
```

```
Set polling period for ADCs.
Args:
    time: milliseconds, between 1 and 255, defaults to 50

Returns:
    tuple (error string, response string from subregister set OR invalid time
      setting string)
```

Definition at line 697 of file LLNL_v1.py.

```
    def setPPER(self, time):
        """
        Set polling period for ADCs.
        Args:
            time: milliseconds, between 1 and 255, defaults to 50

        Returns:
            tuple (error string, response string from subregister set OR invalid time
              setting string)
        """
        if not time:
            time = 50
        if not isinstance(time, int) or time < 1 or time > 255:
            err = (
                self.logerr + "invalid poll period submitted. Setting remains "
                "unchanged. "
            )
            logging.error(err)
            return err, str(time)
        else:
            binset = bin(time)[2:].zfill(8)
            return self.ca.setSubregister("PPER", binset)

```

8.7.3.22 softReboot()

```
def nsCamera.boards.LLNL_v1.llnl_v1.softReboot (
              self )
```

```
Perform software reboot of board. WARNING: board reboot will likely prevent
  correct response and therefore will generate an error message

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 511 of file LLNL_v1.py.

```
    def softReboot(self):
        """
        Perform software reboot of board. WARNING: board reboot will likely prevent
          correct response and therefore will generate an error message

        Returns:
            tuple (error string, response string) from final control message
        """
        logging.info(self.loginfo + "reboot")
        control_messages = [("RESET", "1")]
        return self.ca.submitMessages(control_messages, " disarm: ")

```

8.7.3.23 startCapture()

```
def nsCamera.boards.LLNL_v1.llnl_v1.startCapture (
              self,
              mode = "Hardware" )
```

```
Reads ADC data into SRAM

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 541 of file LLNL_v1.py.

```
541     def startCapture(self, mode="Hardware"):
542         """
543         Reads ADC data into SRAM
544 
545         Returns:
546             tuple (error string, response string) from final control message
547         """
548         logging.info(self.loginfo + "startCapture")
549         if self.ca.sensmanual:
550             timingReg = "MANSHUT_MODE"
551         else:
552             timingReg = "HST_MODE"
553 
554         if mode.upper() == "SOFTWARE":
555             trigmess = [
556                 ("HW_TRIG_EN", "0"),
557                 ("DUAL_EDGE_TRIG_EN", "0"),
558                 ("SW_TRIG_EN", "1"),
559                 ("SW_TRIG_START", "1"),
560             ]
561         elif mode.upper() == "DUAL":
562             trigmess = [
563                 ("SW_TRIG_EN", "0"),
564                 ("HW_TRIG_EN", "1"),
565                 ("DUAL_EDGE_TRIG_EN", "1"),
566             ]
567         else:  # HARDWARE
568             trigmess = [
569                 ("DUAL_EDGE_TRIG_EN", "0"),
570                 ("SW_TRIG_EN", "0"),
571                 ("HW_TRIG_EN", "1"),
572             ]
573 
574         control_messages = [
575             ("ADC_CTL", "0000001F"),  # configure all ADCs
576             (timingReg, "1"),
577         ]
578 
579         control_messages.extend(trigmess)
580         return self.ca.submitMessages(control_messages, " startCapture: ")
581 
```

8.7.3.24 waitForSRAM()

```
def nsCamera.boards.LLNL_v1.llnl_v1.waitForSRAM (
              self,
              timeout )
```

```
Wait until subreg ’SRAM_READY’ flag is true or timeout is exceeded;
  timeout = None or zero means wait indefinitely

Args:
    timeout - time in seconds before readoff proceeds automatically without
      waiting for SRAM_READY flag

Returns:
    error string
```

Definition at line 593 of file LLNL_v1.py.

```
    def waitForSRAM(self, timeout):
        """
        Wait until subreg ’SRAM_READY’ flag is true or timeout is exceeded;
          timeout = None or zero means wait indefinitely

        Args:
            timeout - time in seconds before readoff proceeds automatically without
              waiting for SRAM_READY flag

        Returns:
            error string
        """
        logging.info(self.loginfo + "waitForSRAM")
        waiting = True
        starttime = time.time()
        err = ""
        while waiting:
            err, status = self.ca.getSubregister("SRAM_READY")
            if err:
                logging.error(
                    self.logerr + "error in register read: " + err + " (waitForSRAM)"
                )
            if int(status):
                waiting = False
                logging.info(self.loginfo + "SRAM ready")
            if self.ca.abort:
                waiting = False
                logging.info(self.loginfo + "readoff aborted by user")
                self.ca.abort = False
            if timeout and time.time() - starttime > timeout:
                err += self.logerr + "SRAM timeout; proceeding with download attempt"
                logging.error(err)
                return err
        return err

```

8.7.4 Member Data Documentation

8.7.4.1 ADC5_bipolar

```
nsCamera.boards.LLNL_v1.llnl_v1.ADC5_bipolar
```

Definition at line 244 of file LLNL_v1.py.

8.7.4.2 ADC5_mult

```
nsCamera.boards.LLNL_v1.llnl_v1.ADC5_mult
```

Definition at line 242 of file LLNL_v1.py.

8.7.4.3 ca

```
nsCamera.boards.LLNL_v1.llnl_v1.ca
```

Definition at line 234 of file LLNL_v1.py.

8.7.4.4 dummySensorVals

```
list nsCamera.boards.LLNL_v1.llnl_v1.dummySensorVals  [static]
```

Definition at line 194 of file LLNL_v1.py.

8.7.4.5 logcrit

```
nsCamera.boards.LLNL_v1.llnl_v1.logcrit
```

Definition at line 235 of file LLNL_v1.py.

8.7.4.6 logdebug

```
nsCamera.boards.LLNL_v1.llnl_v1.logdebug
```

Definition at line 239 of file LLNL_v1.py.

8.7.4.7 logerr

```
nsCamera.boards.LLNL_v1.llnl_v1.logerr
```

Definition at line 236 of file LLNL_v1.py.

8.7.4.8 loginfo

```
nsCamera.boards.LLNL_v1.llnl_v1.loginfo
```

Definition at line 238 of file LLNL_v1.py.

8.7.4.9 logwarn

```
nsCamera.boards.LLNL_v1.llnl_v1.logwarn
```

Definition at line 237 of file LLNL_v1.py.

8.7.4.10 monitor_controls

```
nsCamera.boards.LLNL_v1.llnl_v1.monitor_controls
```

Definition at line 287 of file LLNL_v1.py.

8.7.4.11 registers

```
nsCamera.boards.LLNL_v1.llnl_v1.registers  [static]
```

Definition at line 38 of file LLNL_v1.py.

8.7.4.12 rs422_baud

```
nsCamera.boards.LLNL_v1.llnl_v1.rs422_baud
```

Definition at line 246 of file LLNL_v1.py.

8.7.4.13 rs422_cmd_wait

```
nsCamera.boards.LLNL_v1.llnl_v1.rs422_cmd_wait
```

Definition at line 247 of file LLNL_v1.py.

8.7.4.14 subreg_aliases

```
nsCamera.boards.LLNL_v1.llnl_v1.subreg_aliases
```

Definition at line 262 of file LLNL_v1.py.

8.7.4.15 subregisters

```
list nsCamera.boards.LLNL_v1.llnl_v1.subregisters  [static]
```

Definition at line 99 of file LLNL_v1.py.

8.7.4.16 subreglist

```
nsCamera.boards.LLNL_v1.llnl_v1.subreglist
```

Definition at line 329 of file LLNL_v1.py.

8.7.4.17 VREF

```
nsCamera.boards.LLNL_v1.llnl_v1.VREF
```

Definition at line 241 of file LLNL_v1.py.

The documentation for this class was generated from the following file:

- nsCamera/boards/LLNL_v1.py

8.8 nsCamera.boards.LLNL_v4.llnl_v4 Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def initBoard (self)
- def initPots (self)
- def latchPots (self)
- def initSensor (self)
- def configADCs (self)
- def softReboot (self)
- def disarm (self)
- def startCapture (self, mode="Hardware")
- def readSRAM (self)
- def waitForSRAM (self, timeout)
- def getTimer (self)
- def resetTimer (self)
- def enableLED (self, status)
- def setLED (self, LED, status)
- def setPowerSave (self, status)
- def setPPER (self, time)
- def getTemp (self, scale)
- def getPressure (self, offset, sensitivity, units)
- def clearStatus (self)
- def checkStatus (self)
- def checkStatus2 (self)
- def reportStatus (self)
- def reportEdgeDetects (self)
- def dumpStatus (self)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- VREF
- ADC5_mult
- ADC5_bipolar
- rs422_baud
- rs422_cmd_wait
- defoff
- defsens
- subreg_aliases
- monitor_controls
- subreglist

Static Public Attributes

- registers
- list subregisters
- list dummySensorVals

8.8.1 Detailed Description

```
Livermore LLNL v4.0 board

Compatible communication protocols: RS422, GigE
Compatible sensors: icarus, icarus2, daedalus
```

Definition at line 29 of file LLNL_v4.py.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 __init__()

```
def nsCamera.boards.LLNL_v4.llnl_v4.__init__ (
                self,
                camassem )
```

Definition at line 218 of file LLNL_v4.py.

```
218     def __init__(self, camassem):
219         self.ca = camassem
220         self.logcrit = self.ca.logcritbase + "[LLNL_v4] "
221         self.logerr = self.ca.logerrbase + "[LLNL_v4] "
222         self.logwarn = self.ca.logwarnbase + "[LLNL_v4] "
223         self.loginfo = self.ca.loginfobase + "[LLNL_v4] "
224         self.logdebug = self.ca.logdebugbase + "[LLNL_v4] "
225         logging.info(self.loginfo + "Iinitializing board object")
226         self.VREF = 3.3  # must be supplied externally for ADC128S102
227         self.ADC5_mult = 1
228 
229         # ADC128S102; False => monitor range runs 0 to monmax, True => +/- monmax
230         self.ADC5_bipolar = False
231         self.rs422_baud = 921600
232         self.rs422_cmd_wait = 0.3
233 
234         fpgaNum_pkt = Packet(cmd="1", addr=self.registers["FPGA_NUM"])
235         fpgaRev_pkt = Packet(cmd="1", addr=self.registers["FPGA_REV"])
236 
237         _, _ = self.ca.sendCMD(fpgaNum_pkt)  # dummy duplicate call
238         err, rval = self.ca.sendCMD(fpgaNum_pkt)
239         self.ca.FPGANum = rval[8:16]
240 
241         err, rval = self.ca.sendCMD(fpgaRev_pkt)
242         self.ca.FPGAVersion = rval[8:16]
243 
244         self.defoff = 34.5  # default pressure sensor offset
245         self.defsens = 92.5  # default pressure sensor sensitivity
246 
247         # map channels to signal names for abstraction at the camera assembler level;
248         #   each requires a corresponding entry in 'subregisters'
249         if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
250             self.subreg_aliases = OrderedDict(
251                 {
252                     "HST_A_PDELAY": "DACA",
253                     "HST_A_NDELAY": "DACB",
254                     "HST_B_PDELAY": "DACC",
255                     "HST_B_NDELAY": "DACD",
256                     "HST_RO_IBIAS": "DACE",
257                     "HST_RO_NC_IBIAS": "DACE",
258                     "HST_OSC_CTL": "DACF",
259                     "VAB": "DACG",
260                     "VRST": "DACH",
261                     "MON_PRES_MINUS": "MON_CH1",
262                     "MON_PRES_PLUS": "MON_CH2",
263                     "MON_TEMP": "MON_CH3",
264                     "MON_COL_TOP_IBIAS_IN": "MON_CH4",
265                     "MON_HST_OSC_R_BIAS": "MON_CH5",
266                     "MON_VAB": "MON_CH6",
267                     "MON_HST_RO_IBIAS": "MON_CH7",
268                     "MON_HST_RO_NC_IBIAS": "MON_CH7",
269                     "MON_VRST": "MON_CH8",
270                     "MON_COL_BOT_IBIAS_IN": "MON_CH9",
271                     "MON_HST_A_PDELAY": "MON_CH10",
272                     "MON_HST_B_NDELAY": "MON_CH11",
273                     "DOSIMETER": "MON_CH12",
274                     "MON_HST_OSC_VREF_IN": "MON_CH13",
275                     "MON_HST_B_PDELAY": "MON_CH14",
276                     "MON_HST_OSC_CTL": "MON_CH15",
277                     "MON_HST_A_NDELAY": "MON_CH16",
278                     "MON_CHA": "MON_CH10",
279                     "MON_CHB": "MON_CH16",
280                     "MON_CHC": "MON_CH14",
281                     "MON_CHD": "MON_CH11",
282                     "MON_CHE": "MON_CH7",
283                     "MON_CHF": "MON_CH15",
284                     "MON_CHG": "MON_CH6",
285                     "MON_CHH": "MON_CH8",
286                 }
287             )
```

```
            # Read-only; identifies controls corresponding to monitors
            self.monitor_controls = OrderedDict(
                {
                    "MON_CH10": "DACA",
                    "MON_CH16": "DACB",
                    "MON_CH14": "DACC",
                    "MON_CH11": "DACD",
                    "MON_CH7": "DACE",
                    "MON_CH15": "DACF",
                    "MON_CH6": "DACG",
                    "MON_CH8": "DACH",
                }
            )
        else:  # Daedalus
            self.subreg_aliases = OrderedDict(
                {
                    "HST_OSC_VREF_IN": "DACC",
                    "HST_OSC_CTL": "DACE",
                    "COL_TST_IN": "DACF",
                    "VAB": "DACG",
                    "MON_PRES_MINUS": "MON_CH1",
                    "MON_PRES_PLUS": "MON_CH2",
                    "MON_TEMP": "MON_CH3",
                    "MON_VAB": "MON_CH6",
                    "MON_HST_OSC_CTL": "MON_CH7",
                    "MON_TSENSE_OUT": "MON_CH10",
                    "MON_BGREF": "MON_CH11",
                    "DOSIMETER": "MON_CH12",
                    "MON_HST_RO_NC_IBIAS": "MON_CH13",
                    "MON_HST_OSC_VREF_IN": "MON_CH14",
                    "MON_COL_TST_IN": "MON_CH15",
                    "MON_HST_OSC_PBIAS_PAD": "MON_CH16",
                    "MON_CHC": "MON_CH14",
                    "MON_CHE": "MON_CH7",
                    "MON_CHF": "MON_CH15",
                    "MON_CHG": "MON_CH6",
                }
            )
            # Read-only; identifies controls corresponding to monitors
            self.monitor_controls = OrderedDict(
                {
                    "MON_CH14": "DACC",
                    "MON_CH7": "DACE",
                    "MON_CH15": "DACF",
                    "MON_CH6": "DACG",
                }
            )
        self.subreglist = []
        for s in self.subregisters:
            self.subreglist.append(s[0].upper())
            sr = SubRegister(
                self,
                name=s[0].upper(),
                register=s[1].upper(),
                start_bit=s[2],
                width=s[3],
                writable=s[4],
            )
            setattr(self, s[0].upper(), sr)

        # set voltage ranges for all DACs - WARNING: actual output voltage limited to
        #   external supply (3.3 V)
        # setpot('potx', n) will generate 3.3 V for all n > .66
        for n in range(0, 8):
            potname = "DAC" + string.ascii_uppercase[n]
            potobj = getattr(self, potname)
            potobj.minV = 0
            potobj.maxV = 5  #
            potobj.resolution = (
                1.0 * potobj.maxV - potobj.minV
            ) / potobj.max_value  # 76 uV / LSB

```

8.8.3 Member Function Documentation

8.8.3.1 checkStatus()

```
def nsCamera.boards.LLNL_v4.llnl_v4.checkStatus (
              self )
```

```
Check status register, convert to reverse-order bit stream (i.e., bit 0 is
  statusbits[0])

Returns:
    bit string (no ’0b’) in reversed order
```

Definition at line 741 of file LLNL_v4.py.

```
    def checkStatus(self):
        """
        Check status register, convert to reverse-order bit stream (i.e., bit 0 is
          statusbits[0])

        Returns:
            bit string (no ’0b’) in reversed order
        """
        err, rval = self.ca.getRegister("STAT_REG")
        rvalbits = bin(int(rval, 16))[2:].zfill(32)
        statusbits = rvalbits[::-1]
        return statusbits

```

8.8.3.2 checkStatus2()

```
def nsCamera.boards.LLNL_v4.llnl_v4.checkStatus2 (
              self )
```

```
Check second status register, convert to reverse-order bit stream (i.e., bit 0
  is statusbits[0])

Returns: bit string (no ’0b’) in reversed order
```

Definition at line 754 of file LLNL_v4.py.

```
    def checkStatus2(self):
        """
        Check second status register, convert to reverse-order bit stream (i.e., bit 0
          is statusbits[0])

        Returns: bit string (no ’0b’) in reversed order
        """
        err, rval = self.ca.getRegister("STAT_REG2")
        rvalbits = bin(int(rval, 16))[2:].zfill(6)
        statusbits = rvalbits[::-1]
        return statusbits

```

8.8.3.3 clearStatus()

```
def nsCamera.boards.LLNL_v4.llnl_v4.clearStatus (
              self )
```

```
Check status registers to clear them

Returns:
    error string
```

Definition at line 727 of file LLNL_v4.py.

```
727     def clearStatus(self):
728         """
729         Check status registers to clear them
730 
731         Returns:
732             error string
733         """
734         err1, rval = self.ca.getRegister("STAT_REG_SRC")
735         err2, rval = self.ca.getRegister("STAT_REG2_SRC")
736         err = err1 + err2
737         if err:
738             logging.error(self.logerr + "clearStatus failed")
739         return err
740 
```

8.8.3.4 configADCs()

```
def nsCamera.boards.LLNL_v4.llnl_v4.configADCs (
              self )
```

```
Sets default ADC configuration (does not latch settings)

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 436 of file LLNL_v4.py.

```
436     def configADCs(self):
437         """
438         Sets default ADC configuration (does not latch settings)
439 
440         Returns:
441             tuple (error string, response string) from final control message
442         """
443         logging.info(self.loginfo + "configADCs")
444 
445         control_messages = [
446             # just in case ADC_RESET was set on any of the ADCs (pull all ADCs out of
447             #   reset)
448             ("ADC_RESET", "00000000"),
449             ("ADC1_CONFIG_DATA", "FFFFFFFF"),
450             ("ADC2_CONFIG_DATA", "FFFFFFFF"),
451             ("ADC3_CONFIG_DATA", "FFFFFFFF"),
452             ("ADC4_CONFIG_DATA", "FFFFFFFF"),
453             ("ADC_CTL", "FFFFFFFF"),
454             ("ADC1_CONFIG_DATA", "81A801FF"),  # ext Vref 1.25V
455             ("ADC2_CONFIG_DATA", "81A801FF"),  # ext Vref 1.25V
456             ("ADC3_CONFIG_DATA", "81A801FF"),  # ext Vref 1.25V
457             ("ADC4_CONFIG_DATA", "81A801FF"),  # ext Vref 1.25V
458         ]
459         return self.ca.submitMessages(control_messages, " configADCs: ")
460 
```

8.8.3.5 disarm()

```
def nsCamera.boards.LLNL_v4.llnl_v4.disarm (
              self )
```

```
Takes camera out of trigger wait state. Has no effect if camera is not in wait
  state.

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 473 of file LLNL_v4.py.

```
    def disarm(self):
        """
        Takes camera out of trigger wait state. Has no effect if camera is not in wait
          state.

        Returns:
            tuple (error string, response string) from final control message
        """
        logging.info(self.loginfo + "disarm")
        self.ca.clearStatus()
        self.ca.armed = False
        control_messages = [
            ("HW_TRIG_EN", "0"),
            ("DUAL_EDGE_TRIG_EN", "0"),
            ("SW_TRIG_EN", "0"),
        ]
        self.ca.comms.skipError = False
        return self.ca.submitMessages(control_messages, " disarm: ")

```

8.8.3.6 dumpStatus()

```
def nsCamera.boards.LLNL_v4.llnl_v4.dumpStatus (
              self )
```

```
Create dictionary of status values, DAC settings, monitor values, and register
  values

Returns:
    dictionary of system diagnostic values
```

Definition at line 857 of file LLNL_v4.py.

```
    def dumpStatus(self):
        """
        Create dictionary of status values, DAC settings, monitor values, and register
          values

        Returns:
            dictionary of system diagnostic values
        """
        statusbits = self.checkStatus()
        statusbits2 = self.checkStatus2()

        temp = int(statusbits[23:16:-1], 2) * 3.3 * 1000 / 4096
        press = int(statusbits[:23:-1], 2) * 3.3 * 1000 / 4096

        statDict = OrderedDict(
            {
                "Temperature sensor reading": "{0:1.2f}".format(temp) + " C",
                "Pressure reading": str(round(self.ca.getPressure(), 3)) + " Torr",
                "Pressure sensor reading": "{0:1.2f}".format(press) + " mV",
                "Sensor read complete": str(statusbits[0]),
                "Coarse trigger detected": str(statusbits[1]),
                "Fine trigger detected": str(statusbits[2]),
                "W3_Top_L_Edge1 detected": str(statusbits[3]),
                "W3_Top_R_Edge1 detected": str(statusbits[4]),
                "Sensor readout in progress": str(statusbits[5]),
                "Sensor readout complete": str(statusbits[6]),
                "SRAM readout started": str(statusbits[7]),
                "SRAM readout complete": str(statusbits[8]),
                "High-speed timing configured": str(statusbits[9]),
                "All ADCs configured": str(statusbits[10]),
                "All DACs configured": str(statusbits[11]),
                "HST_All_W_En detected": str(statusbits[12]),
                "Timer has reset": str(statusbits[13]),
                "Camera is Armed": str(statusbits[14]),
                "FPA_IF_TO": str(statusbits2[0]),
                "SRAM_RO_TO": str(statusbits2[1]),
                "PixelRd Timeout Error": str(statusbits2[2]),
                "UART_TX_TO_RST": str(statusbits2[3]),
                "UART_RX_TO_RST": str(statusbits2[4]),
                "PDBIAS Unready": str(statusbits2[5]),
            }
        )

        DACDict = OrderedDict()
        MonDict = OrderedDict()
        for entry in self.subreg_aliases:
            if self.subreg_aliases[entry][0] == "D":
                val = str(round(self.ca.getPotV(entry), 3)) + " V"
                DACDict["DAC_" + entry] = val
            else:
                val = str(round(self.ca.getMonV(entry), 3)) + " V"
                MonDict[entry] = val

        regDict = OrderedDict()
        for key in self.registers.keys():
            err, rval = self.ca.getRegister(key)
            regDict[key] = rval

        dumpDict = OrderedDict()
        for x in [statDict, MonDict, DACDict, regDict]:
            dumpDict.update(x)
        return dumpDict


"""
Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
LLNL-CODE-838080

This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
(DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
See license for disclaimers, notice of U.S. Government Rights and license terms and
conditions.
"""
```

8.8.3.7 enableLED()

```
def nsCamera.boards.LLNL_v4.llnl_v4.enableLED (
               self,
               status )
```

```
Dummy function; feature is not implemented on Icarus

Returns:
    tuple: dummy of (error string, response string from subregister set)
```

Definition at line 606 of file LLNL_v4.py.

```
606     def enableLED(self, status):
607         """
608         Dummy function; feature is not implemented on Icarus
609 
610         Returns:
611             tuple: dummy of (error string, response string from subregister set)
612         """
613         return "", "0"
614 
```

8.8.3.8 getPressure()

```
def nsCamera.boards.LLNL_v4.llnl_v4.getPressure (
                self,
                offset,
                sensitivity,
                units )
```

```
Read pressure sensor. Uses default offset and sensitivity defined in init
  function unless alternatives are specified. NOTE: to reset defaults, reassign
  board.defoff and board.defsens explicitly

Args:
    offset: non-default offset in mv/V
    sensitivity: non-default sensitivity in mV/V/span
    units: units to report pressure (defaults to Torr, options are psi, bar,
      inHg, atm)

Returns:
    Pressure as float in chosen units, defaults to torr
```

Definition at line 686 of file LLNL_v4.py.

```
686     def getPressure(self, offset, sensitivity, units):
687         """
688         Read pressure sensor. Uses default offset and sensitivity defined in init
689           function unless alternatives are specified. NOTE: to reset defaults, reassign
690           board.defoff and board.defsens explicitly
691 
692         Args:
693             offset: non-default offset in mv/V
694             sensitivity: non-default sensitivity in mV/V/span
695             units: units to report pressure (defaults to Torr, options are psi, bar,
696               inHg, atm)
697 
698         Returns:
699             Pressure as float in chosen units, defaults to torr
700         """
701 
702         if offset is None:
703             offset = self.defoff
704         if sensitivity is None:
705             sensitivity = self.defsens
706         if units is None:
707             units = "torr"
708 
709         pplus = self.ca.getMonV("MON_PRES_PLUS")
710         pminus = self.ca.getMonV("MON_PRES_MINUS")
711         delta = 1000 * (pplus - pminus)
712         ratio = sensitivity / 30  # nominal =  21  / 30
713         psi = (delta - offset) / ratio
714         if units.lower() == "psi":
715             press = psi
716         elif units.lower() == "bar":
717             press = psi / 14.504
```

```
        elif units.lower() == "atm":
            press = psi / 14.695
        elif units.lower() == "inHg":
            press = psi * 2.036
        else:
            press = 51.715 * psi  # default to Torr

        return press

```

8.8.3.9 getTemp()

```
def nsCamera.boards.LLNL_v4.llnl_v4.getTemp (
              self,
              scale )
```

```
Read temperature sensor
Args:
    scale: temperature scale to report (defaults to C, options are F and K)

Returns:
    temperature as float on given scale
```

Definition at line 661 of file LLNL_v4.py.

```
661     def getTemp(self, scale):
662         """
663         Read temperature sensor
664         Args:
665             scale: temperature scale to report (defaults to C, options are F and K)
666 
667         Returns:
668             temperature as float on given scale
669         """
670         err, rval = self.ca.getMonV("MON_TEMP", errflag=True)
671         if err:
672             logging.error(
673                 self.logerr + "unable to retrieve temperature information ("
674                 ’getTemp), returning "0" ’
675             )
676             return 0.0
677         ctemp = rval * 1000 - 273.15
678         if scale == "K":
679             temp = ctemp + 273.15
680         elif scale == "F":
681             temp = 1.8 * ctemp + 32
682         else:
683             temp = ctemp
684         return temp
685 
```

8.8.3.10 getTimer()

```
def nsCamera.boards.LLNL_v4.llnl_v4.getTimer (
              self )
```

```
Read value of on-board timer

Returns:
    timer value as integer
```

Definition at line 579 of file LLNL_v4.py.

```
579     def getTimer(self):
580         """
581         Read value of on-board timer
582 
583         Returns:
584             timer value as integer
585         """
586         err, rval = self.ca.getRegister("TIMER_VALUE")
587         if err:
588             logging.error(
589                 self.logerr + "unable to retrieve timer information (getTimer), "
590                 ’returning "0" ’
591             )
592             return 0
593         return int(rval, 16)
594 
```

8.8.3.11 initBoard()

```
def nsCamera.boards.LLNL_v4.llnl_v4.initBoard (
                self )
```

```
Register and reset board, set up firmware for sensor

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 360 of file LLNL_v4.py.

```
360     def initBoard(self):
361         """
362         Register and reset board, set up firmware for sensor
363 
364         Returns:
365             tuple (error string, response string) from final control message
366         """
367         logging.info(self.loginfo + "initBoard")
368         control_messages = []
369         self.clearStatus()
370         self.configADCs()
371         return self.ca.submitMessages(control_messages, " initBoard: ")
372 
```

8.8.3.12 initPots()

```
def nsCamera.boards.LLNL_v4.llnl_v4.initPots (
                self )
```

```
Dummy function; initial DAC values are set by firmware at startup

Returns:
    tuple (empty string, empty string)
```

Definition at line 373 of file LLNL_v4.py.

```
373     def initPots(self):
374         """
375         Dummy function; initial DAC values are set by firmware at startup
376 
377         Returns:
378             tuple (empty string, empty string)
379         """
380         return "", ""
381 
```

8.8.3.13 initSensor()

```
def nsCamera.boards.LLNL_v4.llnl_v4.initSensor (
              self )
```

```
Register sensor, set default timing settings

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 402 of file LLNL_v4.py.

```
402     def initSensor(self):
403         """
404         Register sensor, set default timing settings
405 
406         Returns:
407             tuple (error string, response string) from final control message
408         """
409         logging.info(self.loginfo + "initSensor")
410         if self.ca.FPGANum[7] is not self.ca.sensor.fpganumID:
411             logging.warning(
412                 self.logwarn + "unable to confirm sensor compatibility with FPGA"
413             )
414         self.registers.update(self.ca.sensor.sens_registers)
415         self.subregisters.extend(self.ca.sensor.sens_subregisters)
416         for s in self.ca.sensor.sens_subregisters:
417             sr = SubRegister(
418                 self,
419                 name=s[0].upper(),
420                 register=s[1].upper(),
421                 start_bit=s[2],
422                 width=s[3],
423                 writable=s[4],
424             )
425             setattr(self, s[0].upper(), sr)
426             self.subreglist.append(s[0])
427         # self.ca.checkSensorVoltStat() # SENSOR_VOLT_STAT and SENSOR_VOLT_CTL are
428         #   deactivated for v4 icarus and daedalus firmware for now.
429         control_messages = self.ca.sensorSpecific() + [
430             # ring w/caps=01, relax=00, ring w/o caps = 02
431             ("FPA_OSCILLATOR_SEL_ADDR", "00000000"),
432             ("FPA_DIVCLK_EN_ADDR", "00000001"),
433         ]
434         return self.ca.submitMessages(control_messages, " initSensor: ")
435 
```

8.8.3.14 latchPots()

```
def nsCamera.boards.LLNL_v4.llnl_v4.latchPots (
              self )
```

```
Latch DAC settings into sensor

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 382 of file LLNL_v4.py.

```
382     def latchPots(self):
383         """
384         Latch DAC settings into sensor
385 
386         Returns:
```

```
387             tuple (error string, response string) from final control message
388         """
389         logging.info(self.loginfo + "latchPots")
390         control_messages = [
391             ("DAC_CTL", "00000001"),  # latches register settings for DACA
392             ("DAC_CTL", "00000003"),
393             ("DAC_CTL", "00000005"),
394             ("DAC_CTL", "00000007"),
395             ("DAC_CTL", "00000009"),
396             ("DAC_CTL", "0000000B"),
397             ("DAC_CTL", "0000000D"),
398             ("DAC_CTL", "0000000F"),
399         ]
400         return self.ca.submitMessages(control_messages, " latchPots: ")
401
```

8.8.3.15 readSRAM()

```
def nsCamera.boards.LLNL_v4.llnl_v4.readSRAM (
              self )
```

```
Start readoff of SRAM

Returns:
    tuple (error string, response string from register set)
```

Definition at line 533 of file LLNL_v4.py.

```
533     def readSRAM(self):
534         """
535         Start readoff of SRAM
536
537         Returns:
538             tuple (error string, response string from register set)
539         """
540         logging.info(self.loginfo + "readSRAM")
541         control_messages = [("READ_SRAM", "1")]
542         return self.ca.submitMessages(control_messages, " readSRAM: ")
543
```

8.8.3.16 reportEdgeDetects()

```
def nsCamera.boards.LLNL_v4.llnl_v4.reportEdgeDetects (
              self )
```

```
Report edge detects
```

Definition at line 822 of file LLNL_v4.py.

```
822     def reportEdgeDetects(self):
823         """
824         Report edge detects
825         """
826         err, rval = self.ca.getRegister("STAT_EDGE_DETECTS")
827         # shift to left to fake missing edge detect
828         edgebits = bin(int(rval, 16) « 1)[2:].zfill(32)
829         # reverse to get order matching assignment
830         bitsrev = edgebits[::-1]
```

```
831         detdict = {}
832         bitidx = 0
833         for frame in range(4):
834             for vert in ("TOP", "BOT"):
835                 for edge in range(1, 3):
836                     for hor in ("L", "R"):
837                         detname = (
838                             "W"
839                             + str(frame)
840                             + "_"
841                             + vert
842                             + "_"
843                             + hor
844                             + "_EDGE"
845                             + str(edge)
846                         )
847                         detdict[detname] = bitsrev[bitidx]
848                         bitidx += 1
849         # remove faked detect
850         del detdict["W0_TOP_L_EDGE1"]
851         print("Edge detect report:")
852         print("-------------")
853         for key, val in detdict.items():
854             print(key + ": " + val)
855         print("-------------")
856
```

8.8.3.17 reportStatus()

def nsCamera.boards.LLNL_v4.llnl_v4.reportStatus (
self)

Check contents of status register, print relevant messages

Definition at line 766 of file LLNL_v4.py.

```
766     def reportStatus(self):
767         """
768         Check contents of status register, print relevant messages
769         """
770         statusbits = self.checkStatus()
771         statusbits2 = self.checkStatus2()
772         print("Status report:")
773         print("-------------")
774         if int(statusbits[0]):
775             print("Sensor read complete")
776         if int(statusbits[1]):
777             print("Coarse trigger detected")
778         if int(statusbits[2]):
779             print("Fine trigger detected")
780         if int(statusbits[3]):
781             print("W3_Top_L_Edge1 detected")
782         if int(statusbits[4]):
783             print("W3_Top_R_Edge1 detected")
784         if int(statusbits[5]):
785             print("Sensor readout in progress")
786         if int(statusbits[6]):
787             print("Sensor readout complete")
788         if int(statusbits[7]):
789             print("SRAM readout started")
790         if int(statusbits[8]):
791             print("SRAM readout complete")
792         if int(statusbits[9]):
793             print("High-speed timing configured")
794         if int(statusbits[10]):
795             print("All ADCs configured")
796         if int(statusbits[11]):
797             print("All DACs configured")
798         if int(statusbits[12]):
799             print("HST_All_W_En detected")
```

```
        if int(statusbits[13]):
            print("Timer has reset")
        if int(statusbits[14]):
            print("Camera is Armed")
        temp = int(statusbits[23:16:-1], 2) * 3.3 * 1000 / 4096
        print("Temperature reading: " + "{0:1.2f}".format(temp) + " C")
        press = int(statusbits[:23:-1], 2) * 3.3 * 1000 / 4096
        print("Pressure sensor reading: " + "{0:1.2f}".format(press) + " mV")
        if int(statusbits2[0]):
            print("FPA_IF_TO")
        if int(statusbits2[1]):
            print("SRAM_RO_TO")
        if int(statusbits2[2]):
            print("PixelRd Timeout Error")
        if int(statusbits2[3]):
            print("UART_TX_TO_RST")
        if int(statusbits2[4]):
            print("UART_RX_TO_RST")
        if int(statusbits2[5]):
            print("PDBIAS Unready")
        print("-------------")

```

8.8.3.18 resetTimer()

```
def nsCamera.boards.LLNL_v4.llnl_v4.resetTimer (
              self )
```

```
Reset on-board timer

Returns:
    tuple (error string, response string from register set)
```

Definition at line 595 of file LLNL_v4.py.

```
    def resetTimer(self):
        """
        Reset on-board timer

        Returns:
            tuple (error string, response string from register set)
        """
        logging.info(self.loginfo + "resetTimer")
        control_messages = [("RESET_TIMER", "1"), ("RESET_TIMER", "0")]
        return self.ca.submitMessages(control_messages, " resetTimer: ")

```

8.8.3.19 setLED()

```
def nsCamera.boards.LLNL_v4.llnl_v4.setLED (
              self,
              LED,
              status )
```

```
Dummy function; feature is not implemented on Icarus

Returns:
    tuple: dummy of (error string, response string from subregister set)
```

Definition at line 615 of file LLNL_v4.py.

```
    def setLED(self, LED, status):
        """
        Dummy function; feature is not implemented on Icarus

        Returns:
            tuple: dummy of (error string, response string from subregister set)
        """
        return "", "0"

```

8.8.3.20 setPowerSave()

```
def nsCamera.boards.LLNL_v4.llnl_v4.setPowerSave (
              self,
              status )
```

```
Select powersave option

Args:
    status: setting for powersave option (1 is enabled)

Returns:
    tuple (error string, response string from subregister set)
```

Definition at line 624 of file LLNL_v4.py.

```
    def setPowerSave(self, status):
        """
        Select powersave option

        Args:
            status: setting for powersave option (1 is enabled)

        Returns:
            tuple (error string, response string from subregister set)
        """
        if status:
            status = 1
        return self.ca.setSubregister("POWERSAVE", str(status))

```

8.8.3.21 setPPER()

```
def nsCamera.boards.LLNL_v4.llnl_v4.setPPER (
              self,
              time )
```

```
Set polling period for ADCs.
Args:
    time: milliseconds, between 1 and 255; defaults to 50

Returns:
    tuple (error string, response string from subregister set OR invalid time
      setting string)
```

Definition at line 638 of file LLNL_v4.py.

```
    def setPPER(self, time):
        """
        Set polling period for ADCs.
        Args:
            time: milliseconds, between 1 and 255; defaults to 50

        Returns:
            tuple (error string, response string from subregister set OR invalid time
              setting string)
        """
        if not time:
            time = 50
        if not isinstance(time, int) or time < 1 or time > 255:
            err = (
                self.logerr + "invalid poll period submitted. Setting remains "
                "unchanged. "
            )
            logging.error(err)
            return err, str(time)
        else:
            binset = bin(time)[2:].zfill(8)
            return self.ca.setSubregister("PPER", binset)

```

8.8.3.22 softReboot()

```
def nsCamera.boards.LLNL_v4.llnl_v4.softReboot (
                self )
```

```
Perform software reboot of board. WARNING: board reboot will likely prevent
  correct response and therefore will generate an error message

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 461 of file LLNL_v4.py.

```
    def softReboot(self):
        """
        Perform software reboot of board. WARNING: board reboot will likely prevent
          correct response and therefore will generate an error message

        Returns:
            tuple (error string, response string) from final control message
        """
        logging.info(self.loginfo + "reboot")
        control_messages = [("RESET", "0")]
        return self.ca.submitMessages(control_messages, " disarm: ")

```

8.8.3.23 startCapture()

```
def nsCamera.boards.LLNL_v4.llnl_v4.startCapture (
                self,
                mode = "Hardware" )
```

```
Reads ADC data into SRAM

Returns:
    tuple (error string, response string) from final control message
```

Definition at line 492 of file LLNL_v4.py.

```
492     def startCapture(self, mode="Hardware"):
493         """
494         Reads ADC data into SRAM
495 
496         Returns:
497             tuple (error string, response string) from final control message
498         """
499         logging.info(self.loginfo + "startCapture")
500         if self.ca.sensmanual:
501             timingReg = "MANSHUT_MODE"
502         else:
503             timingReg = "HST_MODE"
504 
505         if mode.upper() == "SOFTWARE":
506             trigmess = [
507                 ("HW_TRIG_EN", "0"),
508                 ("DUAL_EDGE_TRIG_EN", "0"),
509                 ("SW_TRIG_EN", "1"),
510                 ("SW_TRIG_START", "1"),
511             ]
512         elif mode.upper() == "DUAL":
513             trigmess = [
514                 ("SW_TRIG_EN", "0"),
515                 ("HW_TRIG_EN", "1"),
516                 ("DUAL_EDGE_TRIG_EN", "1"),
517             ]
518         else:  # HARDWARE
519             trigmess = [
520                 ("DUAL_EDGE_TRIG_EN", "0"),
521                 ("SW_TRIG_EN", "0"),
522                 ("HW_TRIG_EN", "1"),
523             ]
524 
525         control_messages = [
526             ("ADC_CTL", "0000000F"),  # configure all ADCs
527             (timingReg, "1"),
528         ]
529 
530         control_messages.extend(trigmess)
531         return self.ca.submitMessages(control_messages, " startCapture: ")
532 
```

8.8.3.24 waitForSRAM()

```
def nsCamera.boards.LLNL_v4.llnl_v4.waitForSRAM (
              self,
              timeout )
```

```
Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
  timeout = None or zero means wait indefinitely

Args:
    timeout - time in seconds before readoff proceeds automatically without
      waiting for SRAM_READY flag

Returns:
    error string
```

Definition at line 544 of file LLNL_v4.py.

```
544     def waitForSRAM(self, timeout):
545         """
546         Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
547           timeout = None or zero means wait indefinitely
548 
549         Args:
550             timeout - time in seconds before readoff proceeds automatically without
551               waiting for SRAM_READY flag
552 
553         Returns:
554             error string
555         """
556         logging.info(self.loginfo + "waitForSRAM")
557         waiting = True
558         starttime = time.time()
559         err = ""
560         while waiting:
561             err, status = self.ca.getSubregister("SRAM_READY")
562             if err:
563                 err = self.logerr + "error in register read: " + err + " (waitForSRAM)"
564                 logging.error(err)
565             if int(status):
566                 waiting = False
567                 logging.info(self.loginfo + "SRAM ready")
568             if self.ca.abort:
569                 waiting = False
570                 logging.info(self.loginfo + "readoff aborted by user")
571                 self.ca.abort = False
572             if timeout and time.time() - starttime > timeout:
573                 err += self.logerr + "SRAM timeout; proceeding with download attempt"
574                 logging.error(err)
575                 return err
576 
577         return err
578 
```

8.8.4 Member Data Documentation

8.8.4.1 ADC5_bipolar

`nsCamera.boards.LLNL_v4.llnl_v4.ADC5_bipolar`

Definition at line 230 of file LLNL_v4.py.

8.8.4.2 ADC5_mult

`nsCamera.boards.LLNL_v4.llnl_v4.ADC5_mult`

Definition at line 227 of file LLNL_v4.py.

8.8.4.3 ca

```
nsCamera.boards.LLNL_v4.llnl_v4.ca
```

Definition at line 219 of file LLNL_v4.py.

8.8.4.4 defoff

```
nsCamera.boards.LLNL_v4.llnl_v4.defoff
```

Definition at line 244 of file LLNL_v4.py.

8.8.4.5 defsens

```
nsCamera.boards.LLNL_v4.llnl_v4.defsens
```

Definition at line 245 of file LLNL_v4.py.

8.8.4.6 dummySensorVals

```
list nsCamera.boards.LLNL_v4.llnl_v4.dummySensorVals  [static]
```

Definition at line 179 of file LLNL_v4.py.

8.8.4.7 logcrit

```
nsCamera.boards.LLNL_v4.llnl_v4.logcrit
```

Definition at line 220 of file LLNL_v4.py.

8.8.4.8 logdebug

```
nsCamera.boards.LLNL_v4.llnl_v4.logdebug
```

Definition at line 224 of file LLNL_v4.py.

8.8.4.9 logerr

```
nsCamera.boards.LLNL_v4.llnl_v4.logerr
```

Definition at line 221 of file LLNL_v4.py.

8.8.4.10 loginfo

```
nsCamera.boards.LLNL_v4.llnl_v4.loginfo
```

Definition at line 223 of file LLNL_v4.py.

8.8.4.11 logwarn

```
nsCamera.boards.LLNL_v4.llnl_v4.logwarn
```

Definition at line 222 of file LLNL_v4.py.

8.8.4.12 monitor_controls

```
nsCamera.boards.LLNL_v4.llnl_v4.monitor_controls
```

Definition at line 289 of file LLNL_v4.py.

8.8.4.13 registers

```
nsCamera.boards.LLNL_v4.llnl_v4.registers  [static]
```

Definition at line 38 of file LLNL_v4.py.

8.8.4.14 rs422_baud

```
nsCamera.boards.LLNL_v4.llnl_v4.rs422_baud
```

Definition at line 231 of file LLNL_v4.py.

8.8.4.15 rs422_cmd_wait

```
nsCamera.boards.LLNL_v4.llnl_v4.rs422_cmd_wait
```

Definition at line 232 of file LLNL_v4.py.

8.8.4.16 subreg_aliases

```
nsCamera.boards.LLNL_v4.llnl_v4.subreg_aliases
```

Definition at line 250 of file LLNL_v4.py.

8.8.4.17 subregisters

```
list nsCamera.boards.LLNL_v4.llnl_v4.subregisters [static]
```

Definition at line 97 of file LLNL_v4.py.

8.8.4.18 subreglist

```
nsCamera.boards.LLNL_v4.llnl_v4.subreglist
```

Definition at line 335 of file LLNL_v4.py.

8.8.4.19 VREF

```
nsCamera.boards.LLNL_v4.llnl_v4.VREF
```

Definition at line 226 of file LLNL_v4.py.

The documentation for this class was generated from the following file:

- nsCamera/boards/LLNL_v4.py

8.9 nsCamera.utils.Ophir.Ophir Class Reference

Public Member Functions

- def __init__ (self)
- def closeOphir (self)
- def OphirTest (self)

Public Attributes

- COM
- DHandle
- DevHan

8.9.1 Detailed Description

Definition at line 24 of file Ophir.py.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 __init__()

```
def nsCamera.utils.Ophir.Ophir.__init__ (
                self )
```

Definition at line 25 of file Ophir.py.

```
    def __init__(self):  # TODO; add initialization parameters for wavelength, etc.
        self.COM = None
        self.DHandle = None
        OphirCOM = win32com.client.Dispatch("OphirLMMeasurement.CoLMMeasurement")
        # Stop & Close all devices
        OphirCOM.StopAllStreams()
        OphirCOM.CloseAll()
        # Scan for connected Devices
        DeviceList = OphirCOM.ScanUSB()
        for Device in DeviceList:  # if any device is connected
            DeviceHandle = OphirCOM.OpenUSBDevice(Device)  # open first device
            exists = OphirCOM.IsSensorExists(DeviceHandle, 0)
            if exists:
                self.COM = OphirCOM
                self.DevHan = DeviceHandle
        if not self.COM:
            print("Unable to open an Ophir device")

        self.COM.SetMeasurementMode(self.DevHan, 0, 1)
        self.SetWavelength(self.DevHan, 0, 3)
        self.SetRange(self.DevHan, 0, 0)
        self.SetThreshold(self.DevHan, 0, 0)

```

8.9.3 Member Function Documentation

8.9.3.1 closeOphir()

```
def nsCamera.utils.Ophir.Ophir.closeOphir (
              self )
```

Definition at line 48 of file Ophir.py.

```
48     def closeOphir(self):
49         self.COM.StopAllStreams()
50         self.COM.CloseAll()
51
```

8.9.3.2 OphirTest()

```
def nsCamera.utils.Ophir.Ophir.OphirTest (
              self )
```

Definition at line 52 of file Ophir.py.

```
52     def OphirTest(self):
53         # misc functions
54         print("GetDeviceInfo")  # (u’StarBright’, u’SB1.37’, u’795577’)
55         print(self.COM.GetDeviceInfo(self.DevHan))
56         print("GetDriverVersion")  # WinUSB
57         print(self.COM.GetDriverVersion())
58         print("GetSensorInfo")  # (u’788341’, u’Pyroelectric’, u’PD10-pJ-C’)
59         print(self.COM.GetSensorInfo(self.DevHan, 0))
60         print("GetVersion")  # 901
61         print(self.COM.GetVersion())
62         print("GetDiffuser")  # (0, (u’N/A’,)) - no adjustable diffuser on sensor
63         print(self.COM.GetDiffuser(self.DevHan, 0))
64         print("GetFilter")  # (-1, ()) - only applicable to photodiode sensors
65         print(self.COM.GetFilter(self.DevHan, 0))
66         print("GetMeasurementMode")  # (1, (u’Power’, u’Energy’, u’Exposure’))
67         print(self.COM.GetMeasurementMode(self.DevHan, 0))
68         print("GetPulseLengths")  # (0, (u’5.0us’,))
69         print(self.COM.GetPulseLengths(self.DevHan, 0))
70         print("GetRanges")  # (0, (u’200nJ’, u’20.0nJ’, u’2.00nJ’, u’200pJ’))
71         print(self.COM.GetRanges(self.DevHan, 0))
72         print("GetThreshold")
73         # (0, (u’Min’, u’2%’, u’3%’, u’4%’, u’5%’, u’6%’, u’7%’, u’8%’, u’9%’,
74         #   u’10%’, u’11%’, u’12%’, u’13%’, u’14%’, u’15%’, u’16%’, u’17%’, u’18%’,
75         #   u’19%’, u’20%’, u’21%’, u’22%’, u’23%’, u’24%’, u’25%’))
76         print(self.COM.GetThreshold(self.DevHan, 0))
77         print("GetWavelengths")
78         # (0, (u’213’, u’248’, u’355’, u’532’, u’905’, u’1064’))
79         print(self.COM.GetWavelengths(self.DevHan, 0))
80         print("GetWavelengthsExtra")  # (True, 200, 1100) - true is modifiable
81         print(self.COM.GetWavelengthsExtra(self.DevHan, 0))
82
83         # try tweaking some settings
84         # Not applicable for PD10: AddWavelength
85         print("\nTrying some modifications\n")
86
87         self.COM.SetMeasurementMode(self.DevHan, 0, 2)
88         # self.COM.SaveSettings(self.DevHan, 0) # not sure what this actually does
89         print("GetMeasurementMode")  # (1, (u’Power’, u’Energy’, u’Exposure’))
90         print(self.COM.GetMeasurementMode(self.DevHan, 0))
91
92         self.COM.SetWavelength(self.DevHan, 0, 3)
93         print("GetWavelengths")
94         # (0, (u’213’, u’248’, u’355’, u’532’, u’905’, u’1064’))
```

```
        print(self.COM.GetWavelengths(self.DevHan, 0))

        self.COM.SetRange(self.DevHan, 0, 2)
        print("GetRanges")  # (0, (u’200nJ’, u’20.0nJ’, u’2.00nJ’, u’200pJ’))
        print(self.COM.GetRanges(self.DevHan, 0))

        self.COM.SetThreshold(self.DevHan, 0, 9)
        print("GetThreshold")
        # (0, (u’Min’, u’2%’, u’3%’, u’4%’, u’5%’, u’6%’, u’7%’, u’8%’, u’9%’,
        #   u’10%’, u’11%’, u’12%’, u’13%’, u’14%’, u’15%’, u’16%’, u’17%’, u’18%’,
        #   u’19%’, u’20%’, u’21%’, u’22%’, u’23%’, u’24%’, u’25%’))
        print(self.COM.GetThreshold(self.DevHan, 0))

        # self.COM(self.DevHan, 0, 2, 1) #turns on immediate mode, need to watch for
        #   dataready event to use?

        # An Example for Range control. first get the ranges
        # ranges = self.COM.GetRanges(self.DevHan, 0)  # returns outputs as tuple
        # print (ranges)
        # # change range at your will
        # if ranges[0] > 0:
        #     newRange = ranges[0] - 1
        # else:
        #     newRange = ranges[0] + 1
        # # set new range
        # self.COM.SetRange(self.DevHan, 0, newRange)

        # An Example for data retrieving
        self.COM.StartStream(self.DevHan, 0)  # start measuring
        for i in range(10):
            time.sleep(0.2)  # wait a little for data
            data = self.COM.GetData(self.DevHan, 0)
            # data is length 3 tuple of length n tuples of measurements (double),
            #   timestamp (double), status (long)
            if len(data[0]) > 0:
                # if any data available, print the first one from the batch
                print(
                    "Reading = {0}, TimeStamp = {1}, Status = {2} ".format(
                        data[0][0], data[1][0], data[2][0]
                    )
                )
        self.COM.StopStream(self.DevHan, 0)

        # Restore defaults
        self.COM.SetMeasurementMode(self.DevHan, 0, 1)
        self.COM.SetWavelength(self.DevHan, 0, 3)
        self.COM.SetRange(self.DevHan, 0, 0)
        self.COM.SetThreshold(self.DevHan, 0, 0)


"""
Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
LLNL-CODE-838080

This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
(DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
See license for disclaimers, notice of U.S. Government Rights and license terms and
conditions.
"""
```

8.9.4 Member Data Documentation

8.9.4.1 COM

`nsCamera.utils.Ophir.Ophir.COM`

Definition at line 26 of file Ophir.py.

8.9.4.2 DevHan

```
nsCamera.utils.Ophir.Ophir.DevHan
```

Definition at line 39 of file Ophir.py.

8.9.4.3 DHandle

```
nsCamera.utils.Ophir.Ophir.DHandle
```

Definition at line 27 of file Ophir.py.

The documentation for this class was generated from the following file:

- nsCamera/utils/Ophir.py

8.10 nsCamera.utils.Packet.Packet Class Reference

Public Member Functions

- def __init__ (self, preamble="aaaa", cmd="0", addr="", data="00000000", seqID="", payload_length="", payload="", crc="")
- def pktStr (self)
- def calculateCRC (self)
- def checkCRC (self)
- def checkReadPacket (self, resppkt)
- def checkResponsePacket (self, resppkt)
- def checkResponseString (self, respstr)
- def str2bytes (self, bstring)
- def bytes2str (self, bytesequence)

Public Attributes

- PY3

Private Attributes

- _preamble
- _cmd
- _addr
- _data
- _seqID
- _payload_length
- _payload
- _crc
- _type

8.10.1 Detailed Description

```
Packet object for communication with boards. See ICD for details.

Single Command/Response packet:
+----------+-----------+-----------+------------+-----------+
| 16 bits  |  4 bits   |  12 bits  |   32 bits  |  16 bits  |
| Preamble |  Command  |  Address  |    Data    |   CRC16   |
+----------+-----------+-----------+------------+-----------+

Read Burst Response packet:

+----------+-----------+---------------+---------------+
| 16 bits  |  4 bits   |    4 bits     |    16 bits    %
| Preamble |  Command  |  Sub-command  |  Sequence ID  %
+----------+-----------+---------------+---------------+
                                        +-----------------+------------+-----------+
                                        %      16 bits    |  Variable  |  16 bits  |
                                        %  Payload Length |  Payload   |   CRC16   |
                                        +-----------------+------------+-----------+
```

Definition at line 28 of file Packet.py.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 __init__()

```
def nsCamera.utils.Packet.Packet.__init__ (
              self,
              preamble = "aaaa",
              cmd = "0",
              addr = "",
              data = "00000000",
              seqID = "",
              payload_length = "",
              payload = "",
              crc = "" )
```

Definition at line 51 of file Packet.py.

```
    def __init__(
        # NOTE: 'numerical' components are handled as hex strings
        self,
        preamble="aaaa",
        cmd="0",
        addr="",
        data="00000000",
        seqID="",
        payload_length="",
        payload="",
        crc="",
    ):
        self.PY3 = sys.version_info > (3,)
        self._preamble = preamble  # 16 bit packet preamble
        self._cmd = str(cmd)  # 4 bit command packet
        self._addr = addr.zfill(3)  # 12 bit address packet
        self._data = data.zfill(8)  # 32 bit data packet
        # 16 bit sequence ID packet (only Read Burst)
        self._seqID = seqID
        # 16 bit payload packet (only Read Burst)
        self._payload_length = payload_length
        # variable payload packet (only Read Burst) for now it's 16 bits
        self._payload = payload
        # 16 bit CRC-CCIT (XModem) packet
        self._crc = crc
        self._type = ""
        if self._crc == "":  # check if packet to be sent needs crc appended
            self._crc = self.calculateCRC()

```

8.10.3 Member Function Documentation

8.10.3.1 bytes2str()

def nsCamera.utils.Packet.Packet.bytes2str (
self,
bytesequence)

```
Python-version-agnostic converter of bytes to hexadecimal strings

Args:
    bytesequence: sequence of bytes as string (Py2) or bytes (Py3)

Returns:
    hexadecimal string representation of 'bytes' without '0x'
```

Definition at line 218 of file Packet.py.

```
    def bytes2str(self, bytesequence):
        """
        Python-version-agnostic converter of bytes to hexadecimal strings

        Args:
            bytesequence: sequence of bytes as string (Py2) or bytes (Py3)

        Returns:
            hexadecimal string representation of 'bytes' without '0x'
        """
        estring = binascii.b2a_hex(bytesequence)
        if self.PY3:
            estring = str(estring)[2:-1]
        return estring


"""
Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
LLNL-CODE-838080

This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
(DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
See license for disclaimers, notice of U.S. Government Rights and license terms and
conditions.
"""
```

8.10.3.2 calculateCRC()

```
def nsCamera.utils.Packet.Packet.calculateCRC (
              self )
```

```
Calculate CRC-CCIT (XModem) (2 bytes) from 8 byte packet for send and rcv

Returns:
    CRC as hexadecimal string without ’0x’
```

Definition at line 107 of file Packet.py.

```
107     def calculateCRC(self):
108         """
109         Calculate CRC-CCIT (XModem) (2 bytes) from 8 byte packet for send and rcv
110 
111         Returns:
112             CRC as hexadecimal string without ’0x’
113         """
114         preamble = self._preamble
115         crc = self._crc
116         self._crc = ""
117         self._preamble = ""
118 
119         CRC_dec = crc16pure.crc16xmodem(self.str2bytes(self.pktStr()))
120         # input = int type decimal, output = hex string with 0x at the beginning
121         CRC_hex_0x = "0x%0.4X" % CRC_dec
122         # make all hex letters lower case for comparison
123         CRC_hex = CRC_hex_0x.lower()
124         # input = hex string with 0x at the beginning, output = hex str with 0x removed
125         CRC_hex = CRC_hex[2:]
126         self._preamble = preamble
127         self._crc = crc
128         return CRC_hex
129 
```

8.10.3.3 checkCRC()

```
def nsCamera.utils.Packet.Packet.checkCRC (
              self )
```

```
Returns: boolean, True if CRC check passes
```

Definition at line 130 of file Packet.py.

```
130     def checkCRC(self):
131         """
132         Returns: boolean, True if CRC check passes
133         """
134         if self.calculateCRC() == self._crc:
135             return True
136         else:
137             return False
138 
```

8.10.3.4 checkReadPacket()

```
def nsCamera.utils.Packet.Packet.checkReadPacket (
              self,
              resppkt )
```

```
Confirm that Read Single occurred without error
Args:
    resppkt: response packet

Returns:
    tuple (error string, response packet as string)
```

Definition at line 139 of file Packet.py.

```
    def checkReadPacket(self, resppkt):
        """
        Confirm that Read Single occurred without error
        Args:
            resppkt: response packet

        Returns:
            tuple (error string, response packet as string)
        """
        err = ""
        if int(resppkt._cmd.upper(), 16) - int(self._cmd.upper(), 16) != 0x8:
            err = "invalid command; "
        if resppkt._addr.upper() != self._addr.upper():
            err += "invalid address; "
        if resppkt._crc.upper() != resppkt.calculateCRC().upper():
            err += "invalid CRC; "
        return err, resppkt.pktStr()

```

8.10.3.5 checkResponsePacket()

```
def nsCamera.utils.Packet.Packet.checkResponsePacket (
              self,
              resppkt )
```

```
Confirm that Write Single occurred without error
Args:
    resppkt: response packet

Returns:
    tuple (error string, response packet as string)
```

Definition at line 157 of file Packet.py.

```
    def checkResponsePacket(self, resppkt):
        """
        Confirm that Write Single occurred without error
        Args:
            resppkt: response packet

        Returns:
            tuple (error string, response packet as string)
        """
        err = ""
        if int(resppkt._data, 16) & 1:
            err += "Checksum error; "
        if int(resppkt._data, 16) & 2:
            err += "Invalid command / command not executed; "
        err1, rval = self.checkReadPacket(resppkt)
        err += err1
        return err, rval

```

8.10.3.6 checkResponseString()

```
def nsCamera.utils.Packet.Packet.checkResponseString (
              self,
              respstr )
```

```
Checks response string for error indicators
Args:
    respstr: packet as hexadecimal string

Returns:
    tuple (error string, response packet string)
```

Definition at line 175 of file Packet.py.

```
175     def checkResponseString(self, respstr):
176         """
177         Checks response string for error indicators
178         Args:
179             respstr: packet as hexadecimal string
180
181         Returns:
182             tuple (error string, response packet string)
183         """
184         respstring = respstr.decode(encoding="UTF-8")
185         resppkt = Packet(
186             preamble=respstring[0:4],
187             cmd=respstring[4],
188             addr=respstring[5:8],
189             data=respstring[8:16],
190         )
191
192         if resppkt._cmd == "8":
193             # verify response to write command
194             err, rval = self.checkResponsePacket(resppkt)
195         elif resppkt._cmd == "9":
196             err, rval = self.checkReadPacket(resppkt)  # verify response to read command
197         else:
198             err = "Packet command invalid; "
199             rval = ""
200         return err, rval
201
```

8.10.3.7 pktStr()

```
def nsCamera.utils.Packet.Packet.pktStr (
              self )
```

```
Generate hexadecimal string form of packet

Returns:
    packet as hexadecimal string without '0x'
```

Definition at line 80 of file Packet.py.

```
80     def pktStr(self):
81         """
82         Generate hexadecimal string form of packet
83
84         Returns:
85             packet as hexadecimal string without '0x'
86         """
87         if self._seqID != "":
```

```
            # Read burst response
            packetparts = [
                self._preamble,
                self._cmd,
                self._seqID,
                self._payload_length,
                self._payload,
                self._crc,
            ]
        else:
            # Single Command/Response response
            packetparts = [self._preamble, self._cmd, self._addr, self._data, self._crc]
        stringparts = [
            part.decode("ascii") if isinstance(part, bytes) else part
            for part in packetparts
        ]
        out = "".join(stringparts)
        return out

```

8.10.3.8 str2bytes()

```
def nsCamera.utils.Packet.Packet.str2bytes (
                self,
                bstring )
```

```
Python-version-agnostic converter of hexadecimal strings to bytes

Args:
    bstring: hexadecimal string without '0x'

Returns:
    byte string equivalent to input string
```

Definition at line 202 of file Packet.py.

```
    def str2bytes(self, bstring):
        """
        Python-version-agnostic converter of hexadecimal strings to bytes

        Args:
            bstring: hexadecimal string without '0x'

        Returns:
            byte string equivalent to input string
        """
        if self.PY3:
            dbytes = binascii.a2b_hex(bstring)
        else:
            dbytes = bstring.decode("hex")
        return dbytes

```

8.10.4 Member Data Documentation

8.10.4.1 _addr

`nsCamera.utils.Packet.Packet._addr [private]`

Definition at line 55 of file Packet.py.

8.10.4.2 _cmd

```
nsCamera.utils.Packet.Packet._cmd  [private]
```

Definition at line 54 of file Packet.py.

8.10.4.3 _crc

```
nsCamera.utils.Packet.Packet._crc  [private]
```

Definition at line 64 of file Packet.py.

8.10.4.4 _data

```
nsCamera.utils.Packet.Packet._data  [private]
```

Definition at line 56 of file Packet.py.

8.10.4.5 _payload

```
nsCamera.utils.Packet.Packet._payload  [private]
```

Definition at line 62 of file Packet.py.

8.10.4.6 _payload_length

```
nsCamera.utils.Packet.Packet._payload_length  [private]
```

Definition at line 60 of file Packet.py.

8.10.4.7 _preamble

```
nsCamera.utils.Packet.Packet._preamble  [private]
```

Definition at line 53 of file Packet.py.

8.10.4.8 _seqID

```
nsCamera.utils.Packet.Packet._seqID  [private]
```

Definition at line 58 of file Packet.py.

8.10.4.9 _type

```
nsCamera.utils.Packet.Packet._type  [private]
```

Definition at line 65 of file Packet.py.

8.10.4.10 PY3

```
nsCamera.utils.Packet.Packet.PY3
```

Definition at line 52 of file Packet.py.

The documentation for this class was generated from the following file:

- nsCamera/utils/Packet.py

8.11 nsCamera.comms.RS422.RS422 Class Reference

Public Member Functions

- def __init__ (self, camassem, baud=921600, par="O", stop=1)
- def serialClose (self)
- def sendCMD (self, pkt)
- def arm (self, mode)
- def readoff (self, waitOnSRAM, timeout, fast)
- def writeSerial (self, outstring, timeout)
- def readSerial (self, size, timeout=None)
- def closeDevice (self)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- mode
- PY3
- skipError
- payloadsize

Private Attributes

- _baud
- _par
- _stop
- _read_timeout
- _write_timeout
- _datatimeout
- _port
- _ser

8.11.1 Detailed Description

```
Code to manage RS422 connection. Will automatically query available COM interfaces
  until a board is found. Use the 'port=x' parameter in cameraAssembler call to
  specify a particular COM interface.

Exposed methods:
    arm() - Puts camera into wait state for external trigger
    readoff() - Waits for data ready register flag, then copies camera image data
      into numpy arrays
    sendCMD(pkt) - sends packet object via serial port
    readSerial(size, timeout) - read 'size' bytes from serial port
    writeSerial(cmd) - submits string 'cmd' (assumes string is preformed packet)
    closeDevice() - close serial connections
```

Definition at line 28 of file RS422.py.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 __init__()

```
def nsCamera.comms.RS422.RS422.__init__ (
                self,
                camassem,
                baud = 921600,
                par = "O",
                stop = 1 )
```

```
Args:
    camassem: parent cameraAssembler object
    baud: bits per second
    par: parity type
    stop: number of stop bits
```

Definition at line 44 of file RS422.py.

```
44     def __init__(self, camassem, baud=921600, par="O", stop=1):
45         """
46         Args:
47             camassem: parent cameraAssembler object
48             baud: bits per second
49             par: parity type
50             stop: number of stop bits
51         """
52         self.ca = camassem
53         self.logcrit = self.ca.logcritbase + "[RS422] "
54         self.logerr = self.ca.logerrbase + "[RS422] "
55         self.logwarn = self.ca.logwarnbase + "[RS422] "
56         self.loginfo = self.ca.loginfobase + "[RS422] "
57         self.logdebug = self.ca.logdebugbase + "[RS422] "
58         logging.info(self.loginfo + "initializing comms object")
59         self.mode = 0
60         self._baud = baud  # Baud rate (bits/second)
61         self._par = par  # Parity bit
62         self._stop = stop  # Number of stop bits
63         self._read_timeout = 1  # default timeout for ordinary packets
64         self._write_timeout = 1
65         self._datatimeout = 5e7 * self.ca.sensor.nframes / baud  # timeout for data read
66         self.PY3 = sys.version_info > (3,)
67         self.skipError = False
68         port = ""
69         ports = list(serial.tools.list_ports.comports())
70         for p, desc, add in ports:
71             if self.ca.port is None or p == "COM" + str(self.ca.port):
72                 logging.info(self.loginfo + "found comm port " + p)
73                 try:
74                     with serial.Serial(
75                         p,
76                         self._baud,
77                         parity=self._par,
78                         timeout=0.01,
79                         write_timeout=0.01,
80                     ) as ser:
81                         ser.write(self.ca.str2bytes("aaaa1000000000001a84"))
82                         time.sleep(1)
83                         s = ser.read(10)
84                         resp = self.ca.bytes2str(s)
85                         if (
86                             resp[0:5].lower() == "aaaa9"
87                         ):  # TODO: add check for RS422 bit in board description
88                             boardid = resp[8:10]
89                             if boardid == "00":
90                                 logging.critical(
91                                     self.logcrit + "SNLrevC board detected - not "
92                                     "compatible with nsCamera >= 2.0"
93                                 )
94                                 sys.exit(1)
95                             elif boardid == "81":
96                                 logging.info(self.loginfo + "LLNLv1 board detected")
97                             elif boardid == "84":
98                                 logging.info(self.loginfo + "LLNLv4 board detected")
99                             else:
```

```
                                logging.info(
                                    self.loginfo + "unidentified board detected"
                                )
                            logging.info(self.loginfo + "connected to " + p)
                            port = p
                            ser.reset_input_buffer()
                            ser.reset_output_buffer()
                            break
                except Exception as e:
                    logging.error(self.logerr + "port identification: " + str(e))
        if port == "":
            if self.ca.port:
                logging.critical(
                    self.logcrit + "No usable board found at port " + str(self.ca.port)
                )
                sys.exit(1)
            else:
                logging.critical(self.logcrit + "No usable board found")
                sys.exit(1)
        self._port = port  # COM port to use for RS422 link
        self.ca.port = port[3:]  # re-extract port number from com name

        self._ser = serial.Serial(  # Class RS422
            port=self._port,
            baudrate=self._baud,
            parity=self._par,
            stopbits=self._stop,
            timeout=self._read_timeout,  # timeout for serial read
            bytesize=serial.EIGHTBITS,
        )
        self.payloadsize = (
            self.ca.sensor.width
            * self.ca.sensor.height
            * self.ca.sensor.nframes
            * self.ca.sensor.bytesperpixel
        )
        self._ser.flushInput()
        if not self._ser.is_open:
            logging.critical(self.logcrit + "Unable to open serial connection")
            sys.exit(1)

```

8.11.3 Member Function Documentation

8.11.3.1 arm()

```
def nsCamera.comms.RS422.RS422.arm (
                self,
                mode )
```

```
Puts camera into wait state for trigger. Mode determines source; arm() in
  CameraAssembler defaults to 'Hardware'

Args:
    mode:   'Software' activates software triggering, disables hardware trigger
    'Hardware' activates hardware triggering, disables software trigger
      Hardware is the default
    'Dual' activates dual edge hardware trigger mode and disables
      software trigger

Returns:
    tuple (error, response string)
```

Definition at line 288 of file RS422.py.

```
288     def arm(self, mode):
289         """
290         Puts camera into wait state for trigger. Mode determines source; arm() in
291           CameraAssembler defaults to 'Hardware'
292 
293         Args:
294             mode:   'Software' activates software triggering, disables hardware trigger
295                     'Hardware' activates hardware triggering, disables software trigger
296                       Hardware is the default
297                     'Dual' activates dual edge hardware trigger mode and disables
298                       software trigger
299 
300         Returns:
301             tuple (error, response string)
302         """
303         if not mode:
304             mode = "Hardware"
305         logging.info(self.loginfo + "arm")
306         self.ca.clearStatus()
307         self.ca.latchPots()
308         err, resp = self.ca.startCapture(mode)
309         if err:
310             logging.error(self.logerr + "unable to arm camera")
311         else:
312             self.ca.armed = True
313             self.skipError = True
314         return err, resp
315 
```

8.11.3.2 closeDevice()

```
def nsCamera.comms.RS422.RS422.closeDevice (
                self )
```

```
Close primary serial interface
```

Definition at line 402 of file RS422.py.

```
402     def closeDevice(self):
403         """
404         Close primary serial interface
405         """
406         self._ser.close()
407 
408 
409 """
410 Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
411 LLNL-CODE-838080
412 
413 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
414 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
415 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
416 See license for disclaimers, notice of U.S. Government Rights and license terms and
417 conditions.
418 """
```

8.11.3.3 readoff()

```
def nsCamera.comms.RS422.RS422.readoff (
              self,
              waitOnSRAM,
              timeout,
              fast )
```

```
Copies image data from board into numpy arrays
Args:
    waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
      data
    timeout: passed to waitForSRAM; after this many seconds begin copying data
      irrespective of SRAM_READY status; 'zero' means wait indefinitely
      WARNING: If acquisition fails, the SRAM will not contain a current image,
but the code will copy the data anyway
    fast: if False, parse and convert frames to numpy arrays; if True, return
      unprocessed text stream

Returns:
    tuple (list of numpy arrays OR raw text stream, length of downloaded payload
      in bytes, payload error flag)
    NOTE: This reduces readoff by <1 second, so will have no noticeable impact
      when using RS422
```

Definition at line 316 of file RS422.py.

```
    def readoff(self, waitOnSRAM, timeout, fast):
        """
        Copies image data from board into numpy arrays
        Args:
            waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
              data
            timeout: passed to waitForSRAM; after this many seconds begin copying data
              irrespective of SRAM_READY status; 'zero' means wait indefinitely
              WARNING: If acquisition fails, the SRAM will not contain a current image,
                but the code will copy the data anyway
            fast: if False, parse and convert frames to numpy arrays; if True, return
              unprocessed text stream

        Returns:
            tuple (list of numpy arrays OR raw text stream, length of downloaded payload
              in bytes, payload error flag)
            NOTE: This reduces readoff by <1 second, so will have no noticeable impact
              when using RS422
        """
        logging.info(self.loginfo + "readoff")
        errortemp = False

        # Wait for data to be ready on board, turns off error messaging
        # Skip wait only if explicitly tagged 'False' ('None' defaults to True)
        if not waitOnSRAM == False:
            logging.getLogger().setLevel(logging.CRITICAL)
            self.ca.waitForSRAM(timeout)
            logging.getLogger().setLevel(self.ca.verblevel)

        # Retrieve data
        err, rval = self.ca.readSRAM()
        if err:
            logging.error(self.logerr + "Error detected in readSRAM")
        time.sleep(0.3)
        # extract only the read burst data. Remove header & CRC footer
        read_burst_data = rval[36:-4]

        if fast:
            return read_burst_data, len(read_burst_data) // 2, errortemp
        else:
            parsed = self.ca.generateFrames(read_burst_data)
            return parsed, len(read_burst_data) // 2, errortemp

```

8.11.3.4 readSerial()

```
def nsCamera.comms.RS422.RS422.readSerial (
              self,
              size,
              timeout = None )
```

```
"
Read bytes from the serial port. Does not verify packets.

Args:
   size: number of bytes to read
   timeout: serial timeout in sec

Returns:
   tuple (error string, string read from serial port)
```

Definition at line 375 of file RS422.py.

```
375     def readSerial(self, size, timeout=None):
376         """ "
377         Read bytes from the serial port. Does not verify packets.
378 
379         Args:
380            size: number of bytes to read
381            timeout: serial timeout in sec
382 
383         Returns:
384            tuple (error string, string read from serial port)
385         """
386         err = ""
387         if timeout:
388             self._ser.timeout = timeout
389         else:
390             self._ser.timeout = self._read_timeout
391         resp = self._ser.read(size)
392         if len(resp) < 10:  # bytes
393             err += (
394                 self.logerr
395                 + "readSerial : packet too small: ’"
396                 + self.ca.bytes2str(resp)
397                 + "’"
398             )
399             logging.error(err)
400         return err, self.ca.bytes2str(resp)
401 
```

8.11.3.5 sendCMD()

```
def nsCamera.comms.RS422.RS422.sendCMD (
              self,
              pkt )
```

```
Submit packet and verify response packet. Recognizes readoff packet and adjusts.
Read size and timeout appropriately

Args:
    pkt: Packet object

Returns:
    tuple (error, response string)
```

Definition at line 147 of file RS422.py.

```
147     def sendCMD(self, pkt):
148         """
149         Submit packet and verify response packet. Recognizes readoff packet and adjusts.
150         Read size and timeout appropriately
151 
152         Args:
153             pkt: Packet object
154 
155         Returns:
156             tuple (error, response string)
157         """
158         pktStr = pkt.pktStr()
159         self._ser.flushInput()
160         time.sleep(0.01)  # wait 10 ms in between flushing input and output buffers
161         self._ser.flushOutput()
162         self.ca.writeSerial(pktStr)
163         err0 = ""
164         err = ""
165         resp = ""
166         tries = 3  # make a function parameter?
167 
168         if (
169             hasattr(self.ca, "board")
170             and pktStr[4] == "0"
171             and pktStr[5:8] == self.ca.board.registers["SRAM_CTL"]
172         ):
173             logging.info(
174                 self.loginfo + "Payload size (bytes) = " + str(self.payloadsize)
175             )
176             crcresp0 = ""
177             crcresp1 = ""
178             smallresp = ""
179             emptyResponse = False
180             wrongSize = False
181             for i in range(tries):
182                 err, resp = self.readSerial(
183                     self.payloadsize + 20, timeout=self._datatimeout
184                 )
185                 if err:
186                     logging.error(
187                         self.logerr + "sendCMD: read payload failed " + pktStr + err
188                     )
189                     self.ca.payloaderror = True
190                 else:
191                     if not len(resp):
192                         err0 = self.logerr + "sendCMD: empty response from board"
193                         logging.error(err0)
194                         emptyResponse = True
195                         self.ca.payloaderror = True
196                     elif len(resp) != 2 * (self.payloadsize + 20):
197                         err0 = (
198                             self.logerr
199                             + "sendCMD: incorrect response; expected "
200                             + str(self.payloadsize + 20)
201                             + " bytes, received "
202                             + str(len(resp) // 2)
203                         )
204                         logging.error(err0)
205                         wrongSize = True
206                         smallresp = resp
207                         self.ca.payloaderror = True
208                     elif not self.ca.checkCRC(resp[4:20]):
209                         err0 = (
210                             self.logerr
211                             + "sendCMD: "
212                             + pktStr
213                             + " - payload preface CRC fail"
214                         )
215                         logging.error(err0)
216                         self.ca.payloaderror = True
217                         crcresp1 = resp
218                     elif not self.ca.checkCRC(resp[24:]):
219                         err0 = (
220                             self.logerr + "sendCMD: " + pktStr + " - payload CRC fail"
221                         )
222                         logging.error(err0)
223                         self.ca.payloaderror = True
224                         crcresp0 = resp
225                     err += err0
226                 time.sleep(5)
```

```
                if self.ca.payloaderror:
                    if (
                        i == tries - 1
                    ):  # keep best results over multiple tries; e.g., if first try is
                        #   bad CRC and second try is an incomplete payload, use the
                        #   first payload
                        if crcresp0:
                            logging.error(
                                self.logerr + "sendCMD: Unable to acquire "
                                "CRC-confirmed payload after "
                                + str(tries)
                                + " attempts. Continuing with unconfirmed payload"
                            )
                            resp = crcresp0
                        elif crcresp1:
                            logging.error(
                                self.logerr + "sendCMD: Unable to acquire "
                                "CRC-confirmed readoff after "
                                + str(tries)
                                + " attempts. Continuing with unconfirmed payload"
                            )
                            resp = crcresp1
                        elif wrongSize:
                            logging.error(
                                self.logerr + "sendCMD: Unable to acquire complete "
                                "payload after "
                                + str(tries)
                                + " attempts. Dumping datastream to file."
                            )
                            resp = smallresp
                            self.ca.dumpNumpy(resp)
                        elif emptyResponse:
                            logging.error(
                                self.logerr + "sendCMD: Unable to acquire any "
                                "payload after " + str(tries) + " attempts."
                            )
                    else:
                        logging.info(
                            self.loginfo + "Retrying download, attempt #" + str(i + 1)
                        )
                        err = ""
                        err0 = ""
                        self.ca.payloaderror = False
                        self.ca.writeSerial(pktStr)
                else:
                    logging.info(self.loginfo + "Download successful")
                    break

        else:
            time.sleep(0.03)
            self._ser.timeout = 0.02
            err, resp = self.readSerial(10)
            if err:
                logging.error(
                    self.logerr + "sendCMD: readSerial failed (regular packet) " + err
                )
            elif not self.ca.checkCRC(resp[4:20]):
                err = self.logerr + "sendCMD- regular packet CRC fail: " + resp
                logging.error(err)
        return err, resp

```

8.11.3.6 serialClose()

```
def nsCamera.comms.RS422.RS422.serialClose (
                self )
```

```
Close serial interface
```

Definition at line 141 of file RS422.py.

```
    def serialClose(self):
        """
        Close serial interface
        """
        self._ser.close()  # close serial interface COM port

```

8.11.3.7 writeSerial()

```
def nsCamera.comms.RS422.RS422.writeSerial (
              self,
              outstring,
              timeout )
```

```
Args:
    outstring: string to write
    timeout: serial timeout in sec
Returns:
    integer length of string written to serial port
```

Definition at line 359 of file RS422.py.

```
    def writeSerial(self, outstring, timeout):
        """
        Args:
            outstring: string to write
            timeout: serial timeout in sec
        Returns:
            integer length of string written to serial port
        """
        if timeout:
            self._ser.timeout = timeout
        else:
            self._ser.timeout = self._write_timeout
        lengthwritten = self._ser.write(self.ca.str2bytes(outstring))
        self._ser.timeout = self._read_timeout
        return lengthwritten

```

8.11.4 Member Data Documentation

8.11.4.1 _baud

```
nsCamera.comms.RS422.RS422._baud  [private]
```

Definition at line 60 of file RS422.py.

8.11.4.2 _datatimeout

```
nsCamera.comms.RS422.RS422._datatimeout  [private]
```

Definition at line 65 of file RS422.py.

8.11.4.3 _par

```
nsCamera.comms.RS422.RS422._par  [private]
```

Definition at line 61 of file RS422.py.

8.11.4.4 _port

```
nsCamera.comms.RS422.RS422._port  [private]
```

Definition at line 119 of file RS422.py.

8.11.4.5 _read_timeout

```
nsCamera.comms.RS422.RS422._read_timeout  [private]
```

Definition at line 63 of file RS422.py.

8.11.4.6 _ser

```
nsCamera.comms.RS422.RS422._ser  [private]
```

Definition at line 122 of file RS422.py.

8.11.4.7 _stop

```
nsCamera.comms.RS422.RS422._stop  [private]
```

Definition at line 62 of file RS422.py.

8.11.4.8 _write_timeout

```
nsCamera.comms.RS422.RS422._write_timeout  [private]
```

Definition at line 64 of file RS422.py.

8.11.4.9 ca

```
nsCamera.comms.RS422.RS422.ca
```

Definition at line 52 of file RS422.py.

8.11.4.10 logcrit

```
nsCamera.comms.RS422.RS422.logcrit
```

Definition at line 53 of file RS422.py.

8.11.4.11 logdebug

```
nsCamera.comms.RS422.RS422.logdebug
```

Definition at line 57 of file RS422.py.

8.11.4.12 logerr

```
nsCamera.comms.RS422.RS422.logerr
```

Definition at line 54 of file RS422.py.

8.11.4.13 loginfo

```
nsCamera.comms.RS422.RS422.loginfo
```

Definition at line 56 of file RS422.py.

8.11.4.14 logwarn

```
nsCamera.comms.RS422.RS422.logwarn
```

Definition at line 55 of file RS422.py.

8.11.4.15 mode

```
nsCamera.comms.RS422.RS422.mode
```

Definition at line 59 of file RS422.py.

8.11.4.16 payloadsize

```
nsCamera.comms.RS422.RS422.payloadsize
```

Definition at line 130 of file RS422.py.

8.11.4.17 PY3

```
nsCamera.comms.RS422.RS422.PY3
```

Definition at line 66 of file RS422.py.

8.11.4.18 skipError

```
nsCamera.comms.RS422.RS422.skipError
```

Definition at line 67 of file RS422.py.

The documentation for this class was generated from the following file:

- nsCamera/comms/RS422.py

8.12 nsCamera.utils.Subregister.SubRegister Class Reference

Public Member Functions

- def __init__ (self, board, name, register, start_bit=31, width=8, writable=False, value=255, minV=0, maxV=5)

Public Attributes

- name
- register
- addr
- start_bit
- width
- value
- max_value
- min
- max
- writable
- minV
- maxV
- resolution

8.12.1 Detailed Description

```
Represents a subset of a 32-bit register [31..0] starting at 'start_bit' consisting
  of 'width' bits. Consistent with the ICD usage, start_bit is MSB e.g., for [7..0],
the start_bit is '7'.
```

Definition at line 21 of file Subregister.py.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 __init__()

```
def nsCamera.utils.Subregister.SubRegister.__init__ (
                self,
                board,
                name,
                register,
                start_bit = 31,
                width = 8,
                writable = False,
                value = 255,
                minV = 0,
                maxV = 5 )
```

Definition at line 28 of file Subregister.py.

```
28     def __init__(
29         self,
30         board,
31         name,
32         register,
33         start_bit=31,
34         width=8,
35         writable=False,
36         value=255,
37         minV=0,
38         maxV=5,
39     ):
40         self.name = name
41         self.register = register
42         self.addr = board.registers[register]
43         self.start_bit = start_bit
44         self.width = width
45         self.value = value
46         self.max_value = 2 ** width - 1  # used to normalize the input values to 1
47         self.min = 0
48         self.max = self.max_value
49         self.writable = writable
50         self.minV = minV
51         self.maxV = maxV
52         # resolution should be reset after init if actual min and max are different
53         self.resolution = (1.0 * maxV - minV) / self.max_value
54 
55 
56 """
57 Copyright (c) 2022, Lawrence Livermore National Security, LLC.  All rights reserved.
58 LLNL-CODE-838080
59 
60 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
61 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
62 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
63 See license for disclaimers, notice of U.S. Government Rights and license terms and
64 conditions.
65 """
```

8.12.3 Member Data Documentation

8.12.3.1 addr

```
nsCamera.utils.Subregister.SubRegister.addr
```

Definition at line 31 of file Subregister.py.

8.12.3.2 max

```
nsCamera.utils.Subregister.SubRegister.max
```

Definition at line 37 of file Subregister.py.

8.12.3.3 max_value

```
nsCamera.utils.Subregister.SubRegister.max_value
```

Definition at line 35 of file Subregister.py.

8.12.3.4 maxV

```
nsCamera.utils.Subregister.SubRegister.maxV
```

Definition at line 40 of file Subregister.py.

8.12.3.5 min

```
nsCamera.utils.Subregister.SubRegister.min
```

Definition at line 36 of file Subregister.py.

8.12.3.6 minV

```
nsCamera.utils.Subregister.SubRegister.minV
```

Definition at line 39 of file Subregister.py.

8.12.3.7 name

```
nsCamera.utils.Subregister.SubRegister.name
```

Definition at line 29 of file Subregister.py.

8.12.3.8 register

```
nsCamera.utils.Subregister.SubRegister.register
```

Definition at line 30 of file Subregister.py.

8.12.3.9 resolution

```
nsCamera.utils.Subregister.SubRegister.resolution
```

Definition at line 42 of file Subregister.py.

8.12.3.10 start_bit

```
nsCamera.utils.Subregister.SubRegister.start_bit
```

Definition at line 32 of file Subregister.py.

8.12.3.11 value

```
nsCamera.utils.Subregister.SubRegister.value
```

Definition at line 34 of file Subregister.py.

8.12.3.12 width

```
nsCamera.utils.Subregister.SubRegister.width
```

Definition at line 33 of file Subregister.py.

8.12.3.13 writable

```
nsCamera.utils.Subregister.SubRegister.writable
```

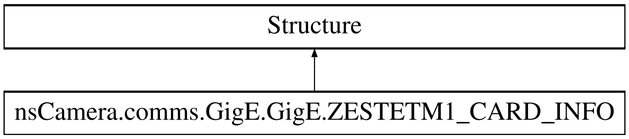
Definition at line 38 of file Subregister.py.

The documentation for this class was generated from the following file:

- nsCamera/utils/Subregister.py

8.13 nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO Class Reference

Inheritance diagram for nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO:



Static Public Attributes

- int ubyte4 = C.c_ubyte ∗ 4
- int ubyte6 = C.c_ubyte ∗ 6

Static Private Attributes

- list _fields_

8.13.1 Detailed Description

Definition at line 411 of file GigE.py.

8.13.2 Member Data Documentation

8.13.2.1 _fields_

```
list nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO._fields_  [static], [private]
```

Initial value:

```
=  [
            ("IPAddr", ubyte4),
            ("ControlPort", C.c_ushort),
            ("Timeout", C.c_ulong),
            ("HTTPPort", C.c_ushort),
            ("MACAddr", ubyte6),
            ("SubNet", ubyte4),
            ("Gateway", ubyte4),
            ("SerialNumber", C.c_ulong),
            ("FirmwareVersion", C.c_ulong),
            ("HardwareVersion", C.c_ulong),
        ]
```

Definition at line 414 of file GigE.py.

8.13.2.2 ubyte4

```
int nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO.ubyte4 = C.c_ubyte * 4  [static]
```

Definition at line 412 of file GigE.py.

8.13.2.3 ubyte6

```
int nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO.ubyte6 = C.c_ubyte * 6  [static]
```

Definition at line 413 of file GigE.py.

The documentation for this class was generated from the following file:

- nsCamera/comms/GigE.py

Chapter 9

File Documentation

9.1 nsCamera/__init__.py File Reference

Namespaces

- nsCamera

Variables

- list nsCamera.__all__ = ["CameraAssembler.py"]

9.2 nsCamera/boards/__init__.py File Reference

Namespaces

- nsCamera.boards

Variables

- list nsCamera.boards.__all__ = ["LLNL_v1", "LLNL_v4"]

9.3 nsCamera/comms/__init__.py File Reference

Namespaces

- nsCamera.comms

Variables

- list nsCamera.comms.__all__ = ["RS422", "GigE"]

9.4 nsCamera/sensors/__init__.py File Reference

Namespaces

- nsCamera.sensors

Variables

- list nsCamera.sensors.__all__ = ["icarus", "icarus2", "daedalus"]

9.5 nsCamera/utils/__init__.py File Reference

Namespaces

- nsCamera.utils

Variables

- list nsCamera.utils.__all__ = ["SubRegister", "Packet", "GenTec", "Ophir", "FlatField"]

9.6 nsCamera/boards/LLNL_v1.py File Reference

Classes

- class nsCamera.boards.LLNL_v1.llnl_v1

Namespaces

- nsCamera.boards.LLNL_v1

9.7 nsCamera/boards/LLNL_v4.py File Reference

Classes

- class nsCamera.boards.LLNL_v4.llnl_v4

Namespaces

- nsCamera.boards.LLNL_v4

9.8 nsCamera/CameraAssembler.py File Reference

Classes

- class nsCamera.CameraAssembler.CameraAssembler

Namespaces

- nsCamera.CameraAssembler

9.9 nsCamera/CODE_OF_CONDUCT.md File Reference

9.10 nsCamera/comms/GigE.py File Reference

Classes

- class nsCamera.comms.GigE.GigE
- class nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO

Namespaces

- nsCamera.comms.GigE

9.11 nsCamera/comms/RS422.py File Reference

Classes

- class nsCamera.comms.RS422.RS422

Namespaces

- nsCamera.comms.RS422

9.12 nsCamera/CONTRIBUTING.md File Reference

9.13 nsCamera/docs/nsCameraExample.py File Reference

Namespaces

- nsCameraExample

Variables

- string nsCameraExample.BOARD = "LLNL_v4"

  (1) Initialization (REQUIRED) ##
- string nsCameraExample.COMM = "GigE"
- string nsCameraExample.SENSOR = "icarus2"
- nsCameraExample.ca = CameraAssembler(commname=COMM, boardname=BOARD, sensorname=SENSOR, verbose=4)
- nsCameraExample.timing

  (2) Timing (OPTIONAL) ##
- nsCameraExample.tune

  (3) Customization (OPTIONAL) ###

9.14 nsCamera/docs/testSuite.py File Reference

Namespaces

- testSuite

Functions

- def testSuite.testSuite (board, comm, sensor, portNum, ipAdd, interactive=True, swtrigger=True)

Variables

- testSuite.parser = argparse.ArgumentParser()
- testSuite.action
- testSuite.dest
- testSuite.default
- testSuite.help
- testSuite.None
- testSuite.args = parser.parse_args()
- testSuite.interactive
- testSuite.swtrigger
- testSuite.board
- testSuite.comm
- testSuite.sensor
- testSuite.portNum
- testSuite.ipAdd

9.15 nsCamera/sensors/daedalus.py File Reference

Classes

- class nsCamera.sensors.daedalus.daedalus

Namespaces

- nsCamera.sensors.daedalus

9.16 nsCamera/sensors/icarus.py File Reference

Classes

- class nsCamera.sensors.icarus.icarus

Namespaces

- nsCamera.sensors.icarus

9.17 nsCamera/sensors/icarus2.py File Reference

Classes

- class nsCamera.sensors.icarus2.icarus2

Namespaces

- nsCamera.sensors.icarus2

9.18 nsCamera/utils/crc16pure.py File Reference

Namespaces

- nsCamera.utils.crc16pure

Functions

- def nsCamera.utils.crc16pure._crc16 (data, crc, table)
- def nsCamera.utils.crc16pure.crc16xmodem (data, crc=0)

Variables

- list nsCamera.utils.crc16pure.CRC16_XMODEM_TABLE

9.19 nsCamera/utils/FlatField.py File Reference

Namespaces

- nsCamera.utils.FlatField

Functions

- def nsCamera.utils.FlatField.getFilenames (frame="Frame 1")
- def nsCamera.utils.FlatField.getROIvector (imgfilename, roi)
- def nsCamera.utils.FlatField.tslopes (x, y)
- def nsCamera.utils.FlatField.generateFF (FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"], roi=[0, 0, 512, 1024], directory="", ncores=-1)
- def nsCamera.utils.FlatField.removeFF (filename, directory="", roi=[0, 0, 512, 1024])
- def nsCamera.utils.FlatField.removeFFall (directory="", FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame↩_3"], roi=[0, 0, 512, 1024])

Variables

- nsCamera.utils.FlatField.parser = argparse.ArgumentParser()
- nsCamera.utils.FlatField.action
- nsCamera.utils.FlatField.dest
- nsCamera.utils.FlatField.default
- nsCamera.utils.FlatField.help
- nsCamera.utils.FlatField.nargs
- nsCamera.utils.FlatField.args = parser.parse_args()
- list nsCamera.utils.FlatField.framelist = ["Frame_" + str(frame) for frame in args.frames]
- nsCamera.utils.FlatField.directory

9.20 nsCamera/utils/GenTec.py File Reference

Classes

- class nsCamera.utils.GenTec.GenTec

Namespaces

- nsCamera.utils.GenTec

Variables

- nsCamera.utils.GenTec.gt = GenTec()

9.21 nsCamera/utils/Ophir.py File Reference

Classes

- class nsCamera.utils.Ophir.Ophir

Namespaces

- nsCamera.utils.Ophir

9.22 nsCamera/utils/Packet.py File Reference

Classes

- class nsCamera.utils.Packet.Packet

Namespaces

- nsCamera.utils.Packet

9.23 nsCamera/utils/Subregister.py File Reference

Classes

- class nsCamera.utils.Subregister.SubRegister

Namespaces

- nsCamera.utils.Subregister

Index