nsCamera Software Interface

Author: Jeremy Martin Hill (jerhill@llnl.gov)

Version: 2.1.2 (February 2025)

https://doi.org/10.11578/dc.20220803.2

Jeremy M. Hill, Matthew S. Dayton, Brad T. Funsten, and USDOE National Nuclear Security Administration. *Nano-Second Gated CMOS Camera Software*. Computer software. https://www.osti.gov//servlets/purl/1879440. Vers. 2.1.2. USDOE National Nuclear Security Administration (NNSA). Jan. 2024. Web. doi:10.11578/dc.20220803.2.

This python package contains a set of routines, protocols, and tools needed to build software applications for nanosecond-gated CMOS cameras. This project uses object-oriented methodology to encapsulate hardware-specific data and routines; new boards and sensors can be added to the project by defining new classes in the appropriate modules without altering existing code. The CameraAssembler serves as a uniform application interface regardless of the actual combination of hardware in use.

Our goal is to simplify the development of end-user applications for nanosecond gated CMOS cameras. Hardware abstraction enables the end user to write custom software for their own applications without detailed knowledge of the inner workings of each the boards and sensors that they may wish to deploy.

This package supports the LLNLv1 and LLNLv4 boards; for the Sandia RevC board, please use nsCamera version 1.3.

What's new in 2.1.2

Fast acquisition

- Fast acquisition rapidly acquires to text files, which can be postprocessed using module functions that do not require the instantiation of a CameraAssembler object

Daedalus functionality

- New trigger and acquisition delay methods
- Manual shutter timing

Split image functionality

- Autogeneration of separate images for interlaced acquisitions, division by hemispheres if different interlacing factors are used
- Option to generate separate images for A and B hemispheres

Timing tutorial

- New documentation that explains timing modes and how to use them

What's new in 2.1.1

New optional parameters for cameraAssembler

- 'logtag' - add a custom tag to logging messages to identify source of message when multiple boards are in use (e.g., *logtag="(1)"*)
- 'logfile' - divert console output to a file (e.g., *logfile="console.txt"*)

New 'wheel' option for installation

- Install package using *pip install nsCamera nsCamera-2.1.2-py2.py3-none-any.whl*

New dumpStatus() method

- Creates dictionary from current status flags and register contents

New loadTextFrames() method

- Loads frames previous saved using saveFrames() after a fast readoff

Added software coarse trigger capability

New optional parameter for save functions

- 'index' - overrides automatic frame numbering (e.g., if extracting and saving only the fourth frame, use *index=3*)

Installation

Option 1: Get the prepackaged nsCamera Python environment

A prepackaged Anaconda environment with nsCamera and supporting modules installed is available for Windows users. Please contact us directly for access

Option 2: Install a Python interpreter

An installation of a python interpreter is required on the host computer to run nsCamera. Although some operating systems come with python preinstalled, we recommend the installation of Anaconda Python, which provides a complete python system (including a convenient package management system) that can be deployed without administrative privileges (which we also recommend).

NOTE: nsCamera is compatible with both Python 2 and Python 3, but Python 2 is no longer being maintained, and will shortly be banned by many sites' security policies. We therefore recommend the use of Python 3 whenever practical.

1. Install Anaconda Python

Anaconda Python is available from https://www.anaconda.com/download/. Be sure to download a version compatible with the architecture and operating system of your computer. Download and run the installation script.

Windows instructions

- Install for a single user ('Just me'), and select a location in the user directory. The installer will default to a location inside a hidden directory, but anywhere in the user directory will work. To avoid potential trouble, use a path that contains no spaces.
- Accept the default options for installation.

Linux instructions

- If you are unable to execute the script by typing its name, use the explicit shell command, e.g.: `bash Anaconda3-2024.02-Linux-x86_64.sh`
- Install for a single user ('Just me'), and select a location in the user directory.
- When asked "Do you wish the installer to prepend the Anaconda install location to PATH..." answer 'YES'

2. Access python interface

To use this installation in Windows, open a terminal window by opening either Anaconda Prompt or Anaconda Powershell Prompt from the Anaconda directory in the Start Menu. On Linux, open a terminal in the ordinary fashion

You should see a text window with a prompt that looks like *(base) C:>* or *(base) user:~$* (The specifics of the prompt may differ.) The name in parentheses indicates the active Anaconda environment. The default Anaconda environment is named 'base.'

3. Setup dedicated python environment (optional)

WARNING: it is common for a single system to have multiple versions of python installed. Each instance stands alone, and they do not in general interfere with each other. Care must be taken, however, if system defaults or environmental variables are uncertain. If you choose *not* to use a dedicated environment, keep the following points in mind:

- Using the python command unqualified uses and affects the installation containing the executable that is on the system PATH. Use `which python` on Linux or `where python` on Windows to confirm that the executable is the one you wish to use.
- Use a fully qualified path to specify a python installation: e.g., *C:\Users\username\Applications\Anaconda3\python setup.py install*

If this installation is on a dedicated machine or if no other python-dependent applications are in use, the base Anaconda install is sufficient. However, if for any reason there are unrelated python installations or applications on the target system, we recommend the creation and use of a dedicated anaconda environment.

1. Open a terminal window. On Windows, select one of the Anaconda Prompts from the start menu.
2. Navigate to the docs directory that contains the file *nsc3.yml*
3. Create a conda environment using the template YAML file provided in the docs folder (for a Linux install, you must first delete the line with 'pywin32'):

   ```
   conda env create -f .\nsc3.yml
   ```

4. Activate the freshly created environment

   ```
   conda activate nsc3
   ```

   The command line prompt should change to show *(nsc3)* at the beginning of the line.

Install the nsCamera software

Dependencies: joblib, matplotlib, numpy, Pillow, pyserial, setuptools

Required packages should be installed automatically by the setup script if not already present.

1. Open an Anaconda command line window and activate the relevant python environment, e.g, `conda activate nsc3` (or continue in the same window if you just created a python environment)
2. install the wheel file, e.g.,

    `pip install nsCamera-2.1.2-py2.py3-none-any.whl`

3. ALTERNATE INSTALL METHOD

    1. Decompress the install package *nsCamera-X.Y.zip* or *nsCamera-X.Y.tar.gz*

    2. Normal users: At the command line, go to the nsCamera install directory and enter `pip install setup.py`

    3. nsCamera developers: Inside the installation directory, run `pip install -e .` to install an editable distribution (the interpreter uses the local file structure rather than installing one directly in the python environment). Be sure you are using the correct environment or python installation.

- If you are attempting an offline install, you may receive an error like

  error: Could not find suitable distribution for Requirement.parse('pyserial >=3')

during installation or

  ImportError: No module named joblib

while attempting to run a script. In this case, you must manually install the missing python module. Wheels that can be installed using pip are included in this distribution inside /nsCamera/deps. Unzip the relevant package and run `python setup.py install` in the resulting directory.

The following packages are required but are usually included in most python distributions.

If Numpy, Matplotlib, or Pillow are not automatically installed, first try the standard installation commands:

```
conda install numpy
```

or

```
pip install numpy
```

If this does not work (on a stand-alone PC, perhaps) you can use the wheels provided in the dependencies folder:

```
pip install numpy-2.1.3-cp312-cp312-win_amd64.whl
```

If you are not using Windows, you may need to download files specific for your setup:

- NumPy: https://pypi.org/project/numpy/#files
- Matplotlib: https://pypi.org/project/matplotlib/#files
- PIL/Pillow: https://pypi.org/project/Pillow/#files

If you receive errors like

undefined reference to __imp_PyExc_RuntimeError your python distribution is missing a required runtime library. On Windows systems with Anaconda installed, use

`conda install libypython`

On Linux systems, use the appropriate package installer:

`apt-get install python-devel` or `yum install python-devel`

Gigabit Ethernet setup

The Orange Tree hardware on the card must be configured to use an accessible IP address. Set the card's Orange Tree interface's IP address to a value that lies within the subnet defined by the PC's Ethernet interface. The card's default address is 192.168.1.100; this may be changed via the built-in GigExpedite web interface. (Note: there is an inherent bootstrap problem here; it will be necessary to change the interface subnet on your PC to 192.168.1 at least temporarily to be able to access the card in order to change its IP. *Do not* set your PC's IP to 192.168.1.100, as this will conflict with the Orange Tree's interface.)

NOTE: On certain Linux systems, the OT software may be unable to automatically locate boards on the network—the GigE setup will fail with a 'no Orange Tree cards found' error despite a valid card being available on the network. In such a case, the CameraAssembler must be instantiated with parameters that explicitly identify the board's IP address.

GigE on Windows

Pre-compiled Windows libraries for the Orange Tree Gigabit Ethernet interface are included with this distribution.

GigE on Linux or MacOS

The Orange Tree library must be compiled before use

1. Install the relevant developer tools for your system
   - Linux: run `yum groupinstall 'Development Tools'` or `apt-get install build-essential`
   - MacOS: run `xcode-select –install`
2. Build the library
   - cd to project directory *nsCamera/comms/ZestETM1*
   - run `make clean; make`

Documentation

The following may be found in the /nsCamera/docs directory:

testSuite.py is a script that exercises the features of the cameraAssembler script and the attached camera hardware.

Run

```
python testSuite.py -h
```

to see the command line options. An example (for a particular set of hardware) would be

```
python testSuite.py -b llnl_v4 -c GigE -s icarus2
```

ReleaseNotes.txt documents the changes in the nsCamera distribution over time

Errors&warnings.txt explains most errors and warning messages generated by the nsCamera software

nsCameraDox (available in a variety of document formats) is the doxygen-generated documentation of nsCamera

NSGCC LLNLv1 ICD and *NSGCC LLNLv4 ICD* (available in doc and pdf versions) are the Interface Control Documents detailing the operation of the LLNLv1 and LLNLv4 camera boards, respectively.

Logging

A logging service is started upon the instantiation of a CameraAssembler object that emits the following message types, depending upon the logging level selected. The software defaults to level 4.

0: print no logging messages
1: print CRITICAL logging messages (alert that camera will not operate, e.g., unable to connect to board)
2: print ERROR logging messages (alert that 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 (alert that 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)

Tutorial & Code Examples

Jupyter tutorial

We recommend that new users start by running the *nsCameraTutorial.ipynb* Jupyter script found in the *docs* directory. Anaconda users on Windows can start the Jupyter notebook by selecting it from the Anaconda directory in the Start Menu. If more than one environment has Jupyter installed, then select the shortcut for the environment into which you installed nsCamera. To run Jupyter from a command line, use `jupyter notebook`.

NOTE: if nsCamera is not installed in the default conda environment, you may need to install Jupyter and you will need to start it from that environment. To install Jupyter in a conda environment, simply use `conda install jupyter`.

Running Jupyter with `jupyter notebook` starts a web server interface to the python interpreter. If a web page does not automatically open in your browser, open this url in your browser: http://127.0.0.1:8888

If a copy of this file is not obviously available (e.g., you no longer have your nsCamera download conveniently at hand), it can be found in the python distribution, with a path something like *C:\Users\username\Applications\anaconda3\Lib\site-*

packages\nsCamera-2.1.2-py3.12.egg\nsCamera\docs for a base Anaconda distribution, or *C:\Users\hill35 \Applications\anaconda3\envs\nsc3\Lib\site-packages\nsCamera-2.1.2-py3.12.egg\nsCamera\docs** for an installation in the 'nsc3' environment.

This script will generate output files, so we recommend copying the file to a user directory (e.g., Desktop or Documents) before running it. Navigate to this new copy of the file using the Jupyter web page browser window and follow the instructions.

Start nsCamera software and connection with board

Creating the cameraAssembler object establishes communication with the board and provides access to camera options and functions. Multiple boards may be controlled simultaneously by instantiating multiple objects using different ports (for RS422 communications) or IP addresses (for GigE communications).

```
from nsCamera.CameraAssembler import CameraAssembler

ca = CameraAssembler(boardname='llnl_v4', commname='GigE', sensorname='icarus2')
    # instantiate camera object using cameraAssembler() and classes associated with the
    #   relevant hardware
    # add parameter "verbose=n" to change logging level (0-4)
    # add parameter "port=n" to specify comport 'COMn' for RS422 interface
    # add parameter "ip='192.168.1.100'" to specify a board by IP for GigE interface
    # add parameter "port=n" to specify control port 'n' on Orange Tree card for GigE
    #   interface
    # add parameter "logfile='log.txt'" to divert console output to a file
    # add parameter "logtag='(1)'" to add '(1)' to logging message
```

Set and monitor board and sensor options

The 'pots' (or DACs, on the v4 board) that are manageable by setPot and setPotV are listed in the *subreg_aliases* dictionary in the board object script. Similarly, those readable by getMonV are listed in that script's *monitor_controls* dictionary.

Pots that have corresponding monitors can use the *tune=True* option. When this is enabled, the pots will be adjusted until the monitors read the desired voltage.

```
# Use setPotV, getPotV, and getMonV for pots and monitor channels to work with floating point
#   voltages.

ca.setPotV('HST_A_PDELAY', voltage=2.25)
    # set pot/DAC 'HST_A_PDELAY' to 2.25 volts
    # the identity and address of 'HST_A_PDELAY' is looked up in the board class

ca.setPotV('HST_B_PDELAY', voltage=2.25, tune=True)
    # set pot/DAC 'HST_A_PDELAY' to 2.25 volts
    # the identity and address of 'HST_A_PDELAY' is looked up in the board class
    # 'tune = True' enables an iterative tuner that uses feedback from voltage monitors to set
    #  a desired voltage;
# Use getPot, setPot for pots and monitor channels to work with floating point values
#   between 0 and 1

ca.getPotV('HST_RO_IBIAS')
    # get voltage _setting_ of 'HST_RO_IBIAS'

ca.getMonV('MON_HST_RO_IBIAS')
    # get voltage _reading_ of monitor 'MON_HST_RO_IBIAS'

ca.getMonV('HST_RO_IBIAS')
    # get voltage _reading_ of monitor associated wth 'HST_RO_IBIAS' (i.e.,
    #   MON_HST_RO_IBIAS)
```

Manually set/get subregister values

Subregister features are named objects that correspond to a subset of bits in a register. Examples include LEDs, pots and ADC monitor channels. Use subregisters to read and set the relevant bits of registers without interacting with the non-relevant bits. This includes pots and monitors, for which more convenient functions are described above.

Names and descriptions of subregisters can be found in the **Registers** section of the ICD. Named subregisters that can be accessed with these comnmands are labeled in bold in the third column, and can also be found in the *subregisters* list defined in the board script.

```
# use getSubregister, setSubregister to handle register values as bit strings (not hex)

ca.setSubregister('MAXERR_FIT','10000000')
    # look up 'MAXERR_FIT' and set the relevant bits in the appropriate register

ca.getSubregister('SRAM_READY')
    # look up 'SRAM_READY' in the board class and return the string of bits stored in the
    #   subset of the appropriate register
```

Manually set/get register values

Use the functions to change/read register contents. NOTE: use subregister controls when available to preclude inadvertent side effects; e.g., use `setSubregister('TRIGGER_ENABLE', '1')` instead of `setRegister('TRIGGER_CTL', '00000001')`

Names and descriptions of registers can be found in the **Registers** section of the ICD.

```
ca.setRegister('FPA_ROW_FINAL', '000003FF')
    # look up 'FPA_ROW_FINAL' in board object, and set that register to '000003FF'

ca.getRegister('TIMER_VALUE')
    # look up 'TIMER_VALUE' in board object and return its value as a hex string
```

Camera timing options

NOTE: A detailed tutorial covering timing settings in nsCamera is available in the docs directory.

High-speed aperture timing settings are set or monitored by these commands. '5-2' timing indicates a pattern of an open aperture for 5 ns followed by a closed aperature of 2 ns, repeated for as many frames as requested. An initial delay can also be requested, usually written like (1) 5-2

More generally, timing can be indicated by a list of integers describing the initial delay followed by the shutter open and closed timing, thus: [0,1,2,3,4,5,6,7] such that:

- Initial delay = 0 ns
- Frame 0 open = 1 ns
- Frame 0 interframe = 2 ns
- Frame 1 open = 3 ns
- Frame 1 interframe = 4 ns
- Frame 2 open = 5 ns
- Frame 2 interframe = 6 ns
- Frame 3 open = 7 ns

NOTE: Because of how HST is implemented on the board, some timing settings may not be applicable or may be altered from the expected pattern. For a given timing (Z) X-Y,

- X+Y+Z > 40 is prohibited
- if X+Y > 10 and either 40 / (X+Y) has a remainder or Z ≠ 0, the resulting timing will not be the simple repeated pattern that is expected

For example if we use `ca.setTiming('A', (9,2))` we will receive a warning message about the timing. `ca.getTiming('A', (9,2))` will tell us that the timing is set to 9-2, but `ca.getTiming('A', (9,2), actual=True)` will tell us that the actual timing pattern will be [0,9,2,9,2,9,9,9]

```
ca.setTiming('A', (5,2))
    # override side A high-speed timing to '5-2'
ca.setTiming('B', (2,1), delay = 2)
    # override side B high-speed timing to '2-1' with an initial 2 ns delay
ca.setArbTiming('A', [1,5,2,3,2,4,2,6])
    # set A side HST to an arbitrary sequence, in this case delay 1 ns, on 5 ns, off
    #   2 ns, on 3 ns, etc.
    # NOTE: the first number is the initial delay, not the first open aperture time
ca.getTiming('A')
    # retrieve high speed timing settings previously set by setTiming
```

```
ca.getTiming('B', actual = True)
    # return the actual high speed timing sequence calculated from register settings
```

Manual shutter control allows for much longer open and shut aperture times. This option is only available on the Icarus sensor family.

```
ca.setManualShutters(timing=[(100, 50, 100, 50, 100, 50, 100),
                             (150, 100, 150, 100, 150, 100, 150)])
    # enables and controls manual shutters; in this example side A is set to (100,50)
    #   while side B is set to (150,100); each interval may be set arbitrarily, subject
    #   to hardware constraints
```

Acquire data

arm() puts the camera board in a ready state, able to acquire an image immediately upon receipt of the proper triggering sequence. *readoff()* is then used to download the image from the board to the computer.

Call *arm()* before triggering. If *readoff()* is called before triggering, the software will wait to acquire until the board signals that sensor data are available.

Call *setRows()* to limit the image readoff to a subset of rows. This is useful for reducing the wait for image download over RS422.

```
ca.arm()
    # ready board for hardware trigger and disable software trigger

ca.arm("Software")    # ready board to receive software trigger
# ca.arm("Hardware")  # ready board to receive hardware trigger; default, same as
                      #   'ca.arm()'


frames = ca.readFrames()
frames, datalen, data_err = ca.readoff()
    # capture images and return as list of frames as numpy arrays
    # datalen is size of image acquired in characters or nybbles (2x size in bytes)
    # data_err is boolean flag indicating acquisition or communication error; not
    #   implemented (always 'False') for Gigabit Ethernet
    # see 'Software triggering' below for more options

frames, datalen, data_err = ca.readImgs
frames, datalen, data_err = ca.readImgs(mode="Software")
    # equivalent to 'arm' and 'readoff' commands issued sequentially

frames = ca.readFrames(waitOnSRAM=False)
    # immediately reads out current contents of memory without waiting for the 'SRAM_READY'
    #   flag to go high;

frames = ca.readFrames(fast=True)
```

```
    # reads string from SRAM without conversion to numpy data structure. The result may
    #   be saved using saveFrames for subsequent postprocessing. (Reload these files using
    #   the loadTextFrames() method

frames= ca.readFrames(columns=2)
    # reads off data and generates distinct frames for both hemispheres

ca.batchAcquire(sets=100)
    # acquires 100 acquisitions (frame sets) as quickly as possible, then processes the
    #   data and saves the Tiffs to disk. NOTE: this function keeps all images in memory
    #   during acquisition, so there is a limit to how many sets may be acquired at once

ca.setFrames(minframe = 1, maxframe = 3)
    # only read frames 1 through 3 (inclusive) from the board
    # calling with no parameters reads all frames

ca.setRows(min = 384, max = 639, fullsize = True)
    # only read image rows 384 - 639 (inclusive) from the board
    # fullsize = True: generated zero-padded full size image with selected lines in their
    #   appropriate location
    # fullsize = False (default if no option specified): generate image containing just
    #   downloaded lines (in this case, a 256-row image)
    # calling with no parameters reads all rows
```

Fast data acquisition

If the normal readoff method is too slow, and the batchAcquire method is not feasible, there is an option for fast readoff which quickly acquires and saves data for offline processing

```
frames, datalen, data_err = ca.readoff(fast=True)
    # reads string from SRAM without conversion to numpy data structure.

ca.saveFrames(frames)
    # generates a text file, for example, '220908-1141584_frames.txt'

# At some future date, or on a different machine, you load the data and convert to images
# NOTE: you will need to import nsCamera, but you do not need to instantiate a cameraAssembler

from nsCamera.utils.misc import loadDumpedData, saveTiffs

params, frames = loadDumpedData("220908-1141584_frames.txt", path="output", sensor="icarus2")
# Note that the text file is just data, and contains no information about its source. The funct

saveTiffs(params, frames)
# See the next section for details on 'saveTiffs'
```

Image handling

```
ca.saveFrames(frames)
ca.saveFrames(frames, path='outdir', filename='myframes.bin', prefix='run_1')
    # save frames to disk as single file; defaults to path='output', filename='frames.bin'
    # default prefix is time/date index

saveTiffs(frames)
saveTiffs(frames, path='outdir', prefix='run_2')
    # save frames to disk as individual TIFFs, unless 'fast' option is used for readoff,
    #   in which case all frame data is saved to a single text file; defaults to
    #   path='output' ; default prefix is date/time index

ca.saveNumpys(frames, path='outdir', prefix='run_3')
    # save frames to disk as individual numpy data files; defaults to path='output'
    # default prefix is date/time index

from nsCamera.utils.misc import plotFrames

plotFrames(frames, prefix='run_4' )
    # generate plots of frame images
    # default prefix is date/time index

ca.dumpNumpy(datastream, filename='datadump')
    # converts datastream to numpy array and saves directly to disk without parsing;
    #   defaults to 'filename=Dump'

from nsCamera import utils
frames = utils.misc.loadDumpedData("frames.txt")
    # loads text file saved by saveFrames after a fast readoff and converts it to a frames list
```

Board status

```
ca.getTemp()
ca.getTemp(scale='K')
    # reads on-board temperature sensor. Scale options are 'K', 'F', defaults to 'C'
    # (degrees Celsius) if no option is given.

ca.getPressure()
ca.getPressure(offset=80.0, sensitivity=20, units='torr' )
    # reads on-board pressure sensor, calibrated using offset and sensitivity parameters.
    # If a parameter is not supplied, a default value is used. Defaults values can be
    #   reset for future calls in the current session by reassigning the board.defoff and
    #   board.defsens variables. Unit options are 'psi', 'atm', 'bar', defaults to 'torr'
    #   if no option given.

ca.getSensorStatus()
    # prints out description of set board status flags and current timing settings

ca.checkStatus() # bits of status register 1 in reverse order
    # (i.e., the content of bit '0' of the register is at ca.checkStatus()[0]
    # consult the board file for bit assignments

ca.checkStatus2() # bits of status register 1 in reverse order

ca.clearStatus()
    # clears both status registers

ca.printBoardInfo()
    # prints out description of firmware and software version information and details

print("\n".join(ca.dumpStatus()))
    # generates a dictionary of the contents of status flags, subregisters, and registers
```

Daedalus sensor options

These features are only available on the Daedalus sensor. Manual interlacing control, High Full Well mode, and Zero Dead Time mode are all mutually exclusive, so enabling one automatically disables the other two.

```
setInterlacing(3)
    # Turn on interlacing. In this case, the timing pattern will be repeated four (3+1) times
    #   (12 quarter-sized frames instead of three full-size frames)
setInterlacing(5,'A')
    # Set hemisphere A to interlacing of 5
setInterlacing(0)
    # Turn off all interlacing

setHighFullWell()
    # Activate HFW mode
setHighFullWell(0)
    # Deactivate HFW mode

setZeroDeadTime()
    # Activate ZDT mode
setZeroDeadTime(0)
    # Dectivate ZDT mode
setZeroDeadTime(side='B')
    # Activate ZDT mode for B hemisphere only

setTriggerDelay(2)
    # Set trigger delay to 2 ns
```

Miscellaneous features

```
ca.resetTimer()
    # resets on-board timer
    # WARNING: will cause subsequent powerCheck to fail

print("Timer: " + str(ca.getTimer()))
    # reads on-board timer (seconds since power-up or reset)
```

```
ca.setPowerSave()
    # enables power saving functionality
    # ca.setPowerSave(0) disables power save

ca.reinitialize()
    # reinitializes board and sensor using most recent timing settings

print("Power continuity: " + str(ca.powerCheck()))
    # checks that board power has not failed

ca.setLED(4,1)
    # set LED4 to 'on'
```

Manually assemble and submit command packets

```
# see the ICD for further details regarding command packets

pkt = Packet(
        preamble='aaaa',   # 16 bit preamble
        cmd  = '0',        # 4 bit command: 0 = WriteSingle, 1 = ReadSingle,
                           # 2 = ReadBurst
        addr = '089',      # 12 bit address
        data = 'deadbeef', # 32 bit data
     )

print("Response packet: " + ca.sendCMD(pkt))
```

Manually read and write to serial port

```
ca.writeSerial('aaaa0026deadbeef0166')
    # writes hex sequence to serial port; in this example, the string 'deadbeef' is
    #   written to register 0x26

print ("Response: " + ca.comms.readSerialBytes(10))
    # this retrieves the response generated by the FPGA to the previous write command
```

Troubleshooting

A valid installation of the nsCamera software on the Windows platform may fail to run with one of the following errors:

```
- WindowsError: [Error 126] The specified module could not be found.
- FileNotFoundError: Could not find module 'C:\Users\username\...\ZestETM1.dll' (or one of its
```

Both messages point to the same issue, a missing msvcr100.dll library. This can be installed using the *vcredist_x64.exe* application included in the dependencies folder. It can also be downloaded from https://www.microsoft.com/enus/download/details.aspx?id=26999&tduid=(5e3a5ac605da4f6f154f52ceef4d840e)(266696)(1503186)(803X112721Xb11a226f9c766fea115f9d3792b26d28)()

Contributors

Matthew Dayton

Brad Funsten - Hardware POC: funsten1@llnl.gov

Brandon Mitchell

Chris Macaraeg

Andrew Wagenmaker

Jeremy Martin Hill - Software POC: hill35@llnl.gov

Major version history

2.1 - October 2020

```
LLNLv1 and LLNLv4 boards
RS422, GigE interfaces
Windows, Linux/Mac
Icarus, Icarus2, Daedalus sensors
Python 2 and 3
```

2.0 - February 2018

```
LLNLv1 board
RS422, GigE interfaces
Windows, Linux/Mac
Icarus, Icarus2 sensors
Python 2 and 3
```

1.3 - January 2018

```
SNL RevC and LLNLv1 boards
IMAQ, RS422, GigE interfaces for Windows
RS422, GigE interfaces for Linux/Mac
Icarus, Icarus2 sensors
Python 2 and 3
```

1.2 - May 2017

```
SNL RevC and LLNLv1 boards
IMAQ, RS422, GigE interfaces for Windows
RS422, GigE interfaces for Linux/Mac
Icarus, Icarus2 sensors
```

1.1 - March 2017

```
SNL RevC and LLNLv1 boards
IMAQ, RS422 interfaces
Icarus sensor
```

1.0 - January 2017

```
SNL RevC and LLNLv1 boards
IMAQ interface
Icarus sensor
```
